Lógica Aplicada à Computação

Newton José Vieira*
Departamento de Ciência da Computação
Instituto de Ciências Exatas
Universidade Federal de Minas Gerais

Logic is the most important theoretical tool we possess but, as with all tools, one must know how and when to use it. (John A. Paulos, 1945–)

* http://www.dcc.ufmg.br/~nvieira
email: nvieira@dcc.ufmg.br

Sumário

Capítulo 1

Modelagem e Formalização

Men content themselves with the same words as other people use, as if the very sound necessarily carried the same meaning. John Locke (1632–1704)

Um ser humano procura observar e compreender a realidade que o cerca utilizando os sentidos, o raciocínio, a intuição etc., munido dos poderes a eles conferidos pela natureza, mas também limitado pelas imperfeições inerentes aos mesmos. Os sentidos são os meios através dos quais o ser humano tem contato com objetos ou acontecimentos externos. Em tal contato ele nunca tem uma compreensão exata daquilo que está sendo apreendido, seja porque os sentidos são imperfeitos, incapazes de captar a realidade como ela é em todos os seus detalhes, seja porque os interesses, objetivos, emoções etc. do indivíduo têm influência sobre o que ele efetivamente assimila ou não. Durante a assimilação de conhecimento, o raciocínio, a intuição etc. são utilizados, em maior ou menor grau, na tentativa de compreender o que se passa e também para decidir o que é importante e o que não é, o que vale a pena reter e o que não vale.

O processo de assimilação do conhecimento correspondente a uma porção da realidade, descrito acima de forma bastante sucinta e imperfeita, está presente quando um profissional de computação se envolve na atividade de construir um algoritmo, um sistema de informação, uma base de dados ou uma base de conhecimentos para certa aplicação. Mas, além das limitações impostas pelos sentidos, raciocínio, intuição, interesses pessoais etc. o profissional se depara também com restrições ditadas pelo empregador, cliente, ou o que seja, que influenciam diretamente em como ele deve "ver" a realidade, de maneira que possa levar a bom termo a atividade sob sua responsabilidade. Tal processo, nesse contexto, é denominado *modelagem*. O programador ou analista obtém, a partir da realidade correspondente à aplicação que tem em vista, um modelo conceitual que, não apenas inclua todos os detalhes necessários para a satisfação dos requisitos da aplicação, mas também que não inclua detalhes em excesso, que possam comprometer a eficiência do tratamento computacional da mesma. Nesse contexto, a "realidade" não é necessariamente constituída de objetos e acontecimentos "reais", objetos e acontecimentos que efetivamente tenham existido, existam ou vão existir. Ela pode conter, além de entidades concretas, entidades abstratas, fictícias, ou o que quer que seja "conveniente" considerar como povoando a realidade na aplicação

específica.

Vários tipos de entidades matemáticas podem ser utilizados no processo de modelagem como, por exemplo, conjuntos, funções, relações, grafos, autômatos finitos, equações diferenciais etc. Depois de obtido um *modelo conceitual* em termos de algumas entidades matemáticas como essas, um programador ou analista deve codificar o conhecimento modelado em uma *linguagem formal* ou mais de uma, para que a aplicação possa ser tratada computacionalmente ou simplesmente para atingir um maior rigor ou precisão. Exemplos de linguagens formais são as linguagens de programação (procedurais, funcionais, orientadas a objeto ou baseadas em lógica), linguagens de especificação formal de sistemas, linguagens de bases de dados (como as baseadas na álgebra ou no cálculo relacional), e linguagens lógicas para expressão de conhecimento (como as lógicas proposicional, de primeira ordem, de segunda ordem, não monotônicas, paraconsistentes, probabilísticas) etc. O processo de codificar o conhecimento em uma linguagem formal a partir de um modelo conceitual (explícito ou não) é denominado *formalização* e o resultado é um *sistema formal*, sendo que este último pressupõe, além da linguagem formal propriamente dita, algum mecanismo de processamento do conhecimento expresso.[1]

A Figura 1.1 mostra de forma esquemática os processos mencionados anteriormente. As caixas representam os três de tipos de entidades envolvidas: a realidade (domínio da aplicação), o modelo conceitual e o sistema formal. As setas duplas representam processos que levam de um tipo de entidade a outro. Assim, a figura mostra que o processo de modelagem parte da realidade em questão e produz como resultado um modelo conceitual, e o processo de formalização parte do modelo conceitual e obtém um sistema formal como resultado. Com isto, pode-se dizer que existem três níveis, correspondentes aos três tipos de entidades envolvidas: o *nível objeto*, em que se considera a realidade, o *nível conceitual*, em que se expressa o modelo conceitual, e o *nível formal*, em que se trata do sistema formal. Do lado direito de cada caixa, na figura, estão anotados exemplos de elementos utilizados na composição do conhecimento relativo às entidades de cada um dos níveis.

Na atividade de modelagem é importante, não apenas a capacidade de distinguir os detalhes importantes e expressá-los por meio das entidades matemáticas escolhidas, mas também a de desprezar detalhes irrelevantes, principalmente aqueles que possam comprometer o processamento posterior à formalização. Assim, pode-se dizer que a atividade de modelagem é uma atividade de *abstração*: aos elementos da realidade correspondem elementos abstratos (matemáticos), os quais não só são frequentemente incapazes de capturar todos os aspectos, mas também devem ser utilizados (intencionalmente) de forma a deixar de capturar vários deles. Já na atividade de formalização, deve-se utilizar uma linguagem formal a partir da qual seja possível expressar tudo

[1] Como será visto no final do capítulo, em sua definição padrão um sistema formal consta, basicamente, de uma linguagem formal e um conjunto de regras de derivação. Estas últimas definem, implicitamente, um mecanismo não determinístico de geração de expressões na linguagem. A concepção de sistema formal como constituído de linguagem formal mais um "mecanismo de processamento" é consistente com a definição padrão, embora possa ser um pouco mais "permissiva" se se flexibilizar a noção de mecanismo de processamento.

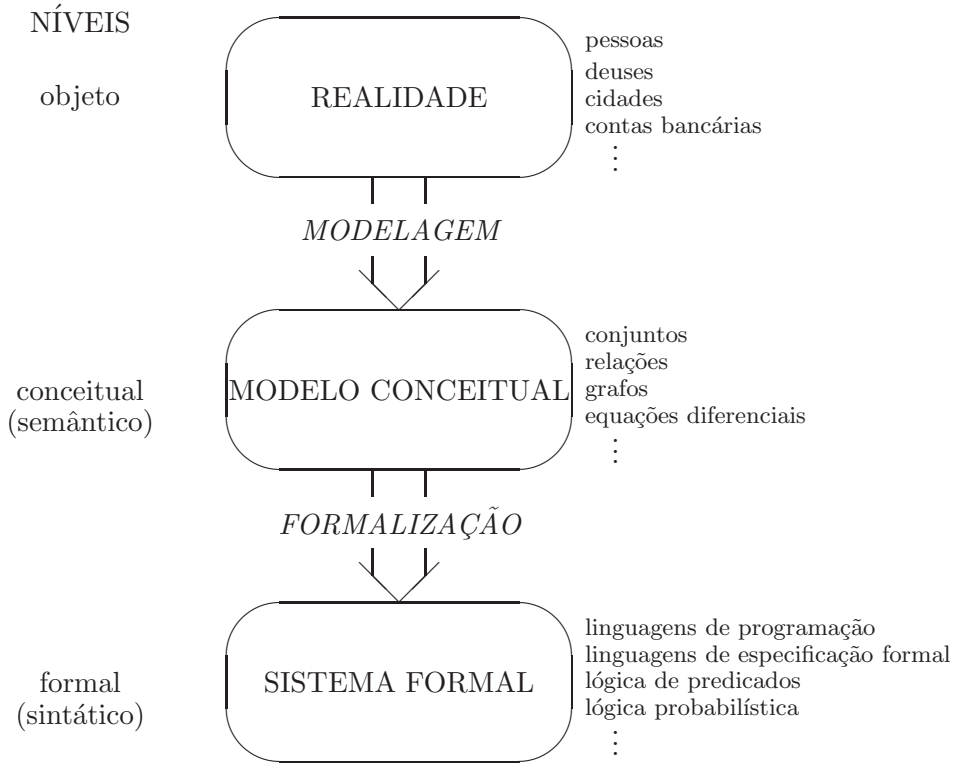


Figura 1.1: Os processos de modelagem e formalização.

aquilo que está presente no modelo conceitual concebido. E o mesmo princípio de "economia" que aconselha o desprezo de detalhes irrelevantes durante o processo de modelagem pode ser utilizado na etapa de formalização: normalmente, é conveniente utilizar uma linguagem formal apenas suficiente para expressar tudo que está previsto pela modelagem efetuada, pois uma linguagem que permita expressar mais do que o necessário pode levar a ineficiências indesejáveis ou, mesmo, insuperáveis.

Na Figura 1.2 está mostrado o *triângulo do significado*,[2] que explicita claramente a relação indireta que existe entre um símbolo e o objeto que ele representa. O símbolo, na verdade, diz respeito a um conceito que, por sua vez, é uma versão abstrata de um objeto. O ser humano apreende e raciocina com conceitos, que ele interpreta como referindo-se a elementos da realidade. Havendo a necessidade de expressão dos conceitos por meio de uma linguagem, como no caso de uma formalização, entram em cena os símbolos. Assim, um símbolo representa um objeto indiretamente, denotando um conceito que deve ser interpretado como uma versão abstrata do objeto. Evidentemente, o triângulo do significado destaca os elementos fundamentais de cada um dos três níveis explicitados na Figura 1.1. Um exemplo bem simples mostrando entidades que povoam os três níveis: a *idade de uma pessoa* (no nível objeto) pode ser modelada (abstraída)

[2]Adaptado de Ogden, C.K., Richards, I.A. *The meaning of meaning*, Mariner Books, 1989.

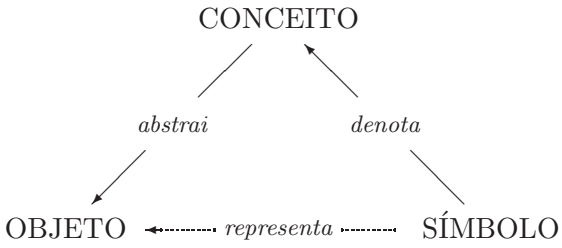


Figura 1.2: O triângulo do significado.

como um *número inteiro positivo* (no nível conceitual) que expresse o número de anos que a pessoa tem, desprezando-se os meses, dias etc. excedentes; tal número pode ser formalizado (denotado) por meio de uma sequência de dígitos na *notação decimal* (no nível formal). Veja como são três tipos de entidades diferentes: a idade (número de anos da pessoa), o número (entidade matemática abstrata) e a sequência de dígitos decimais.

É interessante observar que cada um dos vértices pode estar ausente em certa situação. Por exemplo, uma pessoa, ao ver um objeto desconhecido, pode formar uma idéia (conceito) do mesmo, mas não ter uma denominação para ele. A palavra "lobisomem" denota certa entidade (conceito) que não corresponde a um objeto real. Uma pessoa pode encontrar uma palavra para a qual não conhece significado (conceito) ou a que objeto ela se refere; pode inventar um conceito para o qual ainda não formulou uma denominação e que não designa (ainda) qualquer objeto; pode conviver com um objeto para o qual não tem denominação nem faz idéia do que seja.

Um texto, tanto o escrito em uma linguagem formal quanto o escrito em uma língua natural, se situa no nível formal, ou seja, simbólico. Com isto, quando uma pessoa escreve uma sentença em português, por exemplo, ela diz alguma coisa a respeito de uma certa realidade (na verdade, conforme o entendimento que *ela* tem da realidade). Deve-se enfatizar que a sentença usa *nomes* para as entidades às quais a pessoa se refere, não as próprias entidades nem os conceitos que ela tem das mesmas. Por exemplo, quando se diz

O céu é azul.

não se está usando as entidades *céu* e *azul* na sentença, mas nomes (ou seja, símbolos, palavras) para elas. Uma forma de se referir ao nome de uma entidade em uma sentença, portanto colocando-o no nível da realidade da qual se fala, é usar como nome do nome o próprio entre aspas. Com esse artifício, a sentença

(a) *A palavra "céu" tem três letras.*

está correta e diz uma verdade, enquanto que a sentença

(b) *A palavra céu tem três letras.*

não diz uma verdade: afinal, o *céu* não é uma palavra! Porém, é comum interpretar (b) como dizendo o que diz a sentença (a). É como se, implicitamente, se fizesse uma conversão de uma na outra, restabelecendo-se a correção. Esse tipo de atitude é comum e, na verdade, é a mais comum, mesmo em textos em que se preza mais o uso de formalismos, como os de matemática e os de lógica. Mas, nestes, é extremamente importante que o leitor tenha consciência disso, pois muitas vezes a própria linguagem é objeto de discurso. Por exemplo, em matemática, quando se diz *a equação* $x^2 - x - 1 = 0$ *tem duas raízes complexas*, o que se está dizendo, na verdade, é que *a equação "*$x^2 - x - 1 = 0$*" tem duas raízes complexas*. Uma outra maneira de dizer a mesma coisa é: *existem dois números complexos* x *tais que* $x^2 - x - 1 = 0$ (aqui usa-se a equação para expressar uma proposição sobre números, e não como um objeto do qual se esteja falando; logo, não há aspas).

Quando se estuda lógica, a distinção mencionada anteriormente é particularmente importante porque ela tem como componente uma linguagem formal. Por causa disso, é útil também introduzir mais um conceito: o de *metalinguagem*. Quando se está discursando sobre uma linguagem, a *linguagem objeto*, usa-se para falar dela uma linguagem, a metalinguagem, que pode ser a mesma ou não. Por exemplo, em livros de ensino da língua portuguesa, a própria língua portuguesa é usada como metalinguagem para falar sobre ela mesma. Já no caso de lógica, costuma-se utilizar português suplementado com uma simbologia matemática (como de costume em matemática em geral).

Os processos inversos aos da modelagem e da formalização estão ilustrados na Figura 1.3. O caminho inverso ao da modelagem é percorrido por meio de uma *interpretação*. Em uma interpretação, elementos das estruturas matemáticas utilizadas para compor um modelo conceitual são vistos como abstraindo elementos respectivos da realidade. Esta associação entre o modelo conceitual e a realidade só existe na cabeça de quem interpreta. Na verdade, qualquer estrutura matemática pode ser aplicada a múltiplas realidades. Pode-se imaginar que tais realidades têm em comum a "estrutura" capturada por meio do modelo matemático em questão. Para que pessoas diferentes possam interpretar um modelo de uma maneira próxima daquela intencionada pelo seu criador, o mesmo deve documentá-lo convenientemente, por exemplo, lançando mão de descrições complementares. Assim, o trabalho de mostrar que um modelo conceitual captura uma porção de realidade é um trabalho de *convencimento*, de mostrar com argumentos intuitivos que as estruturas matemáticas empregadas capturam adequadamente o que se precisa.

O caminho inverso ao da formalização é percorrido por meio de uma *interpretação formal*. Em uma interpretação formal, os elementos linguísticos utilizados na formalização são vistos como denotando elementos respectivos nas estruturas matemáticas utilizadas na modelagem. Esta associação entre a linguagem formal e o modelo conceitual é dada pela *semântica* da linguagem. A grande vantagem de se utilizar métodos formais está justamente aqui: a semântica é bem definida, de tal maneira que as sentenças tenham um significado preciso em termos das estruturas matemáticas usadas na modelagem. Pessoas diferentes não podem fazer "interpretações formais" diferentes.[3]

[3]Na verdade, podem existir estruturas matemáticos distintas a partir dos quais se pode ter inter-

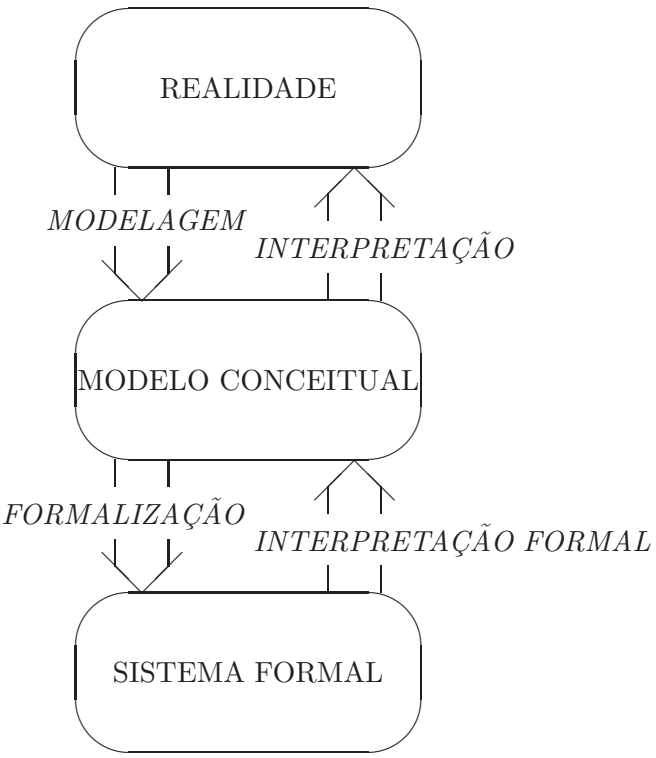


Figura 1.3: Os dois processos de interpretação.

É interessante notar que os objetos concretos, presentes em nosso dia-a-dia, não podem povoar (eles próprios) o nível conceitual e que os conceitos, como manifestações mentais que são, não podem estar (eles próprios) no nível formal. Em certo sentido, é essa impossibilidade mesma que induz a criação dos dois níveis artificiais, o conceitual, para que se possa organizar o (pouco que seja) que se consiga abstrair do mundo, e o formal, para que se possa atingir um máximo de rigor e precisão (evidentemente, quando for possível e necessário). Por outro lado, a "realidade", como considerada aqui, não tem que ser constituída apenas por coisas concretas. Por exemplo, na metamatemática o objeto de estudo é a própria matemática. Em computação, na área de construção de compiladores, a "realidade" tratada contém uma entidade formal: a linguagem de programação a compilar.

Em uma aplicação mais simples ou em que o analista ou programador tenha intimidade suficiente com certo tipo de sistema formal, pode ser conveniente realizar os processos de modelagem e formalização quase que concomitantemente, obtendo-se diretamente uma expressão formal da realidade, saltando-se, ou quase, a etapa de composição do modelo conceitual. Por exemplo, um programador pode às vezes produzir um algoritmo diretamente em uma linguagem de programação, sem passar pela etapa

pretações formais alternativas, mas, fixada a estrutura, a interpretação formal é única.

de produzir um algoritmo em uma pseudolinguagem (esta última livre de restrições e idiossincrasias existentes em linguagens de programação reais, que, normalmente, dificultam o processo de modelagem em si).

Nestas notas estaremos interessados na expressão e manipulação de conhecimento utilizando como linguagem formal a da *lógica matemática*,[4] sendo que daremos uma ênfase especial à manipulação por meio de computadores, ou seja, de algoritmos. Os algoritmos serão escritos em uma linguagem procedural estilo Pascal. Acredita-se que o leitor não terá dificuldade em entendê-los, já que as construções menos usais serão explicadas ao longo do texto na medida em que forem aparecendo. De qualquer forma, o Apêndice apresenta uma descrição sucinta das construções da linguagem algorítmica usada.

Embora a lógica matemática tenha sido inicialmente desenvolvida no intuito de prover uma fundamentação para o raciocício correto (principalmente o matemático), ela teve e continua tendo sucesso, talvez até maior do que o daquele verificado para o propósito original, na expressão e manipulação de conhecimento em várias áreas da computação como bancos de dados, linguagens de programação e inteligência artificial. As versões que serão aqui estudadas, a *lógica proposicional* e a *lógica de predicados de primeira ordem*, são suficientes, apesar da simplicidade e economia conceitual de ambas, para a expressão e manipulação de uma parte substancial da matemática e também do conhecimento tipicamente tratado em computação. Além disso, elas podem ser usadas como bases a partir das quais se considera extensões, como lógicas de ordem superior, lógicas modais etc., ou alternativas, como lógicas probabilísticas, intuicionistas, não monotônicas, paraconsistentes etc.

Daqui para frente o termo "lógica" será usado como sinônimo de "lógica matemática".

As aplicações mais importantes das diversas versões de lógica estão ligadas à possibilidade de determinar se a veracidade de uma certa afirmativa segue ou não da veracidade de um conjunto de afirmativas, as quais compõem uma teoria, ou uma base de conhecimentos, ou a especificação de um sistema etc. Essa determinação é o que faz, por exemplo, um matemático quando demonstra um teorema α a partir de um conjunto de hipóteses H (incluindo-se em tais hipóteses todo o conhecimento necessário para a demonstração do teorema). É também o que faz um programa para prova automática de teoremas que, a partir de um conjunto de afirmativas H, demonstra uma afirmativa α. Algumas diferenças básicas entre estas duas atividades são:

- O programa lida com uma linguagem formal, tipicamente da lógica proposicional ou de predicados, enquanto que o matemático usa o português intercalado, em maior ou menor grau, com uma simbologia típica da área em que se situa o teorema.

- O funcionamento do programa é determinado por um algoritmo que, por sua vez, baseia-se em um sistema dedutivo formal. Já o matemático, como um ser humano que é (brincadeiras à parte...), utiliza intuição, raciocínio, estratégias etc.

[4] Também chamada de lógica formal ou lógica simbólica.

- As demonstrações produzidas por um programa são deduções formais, enquanto que as produzidas por um matemático são informais. Mas é importante ressaltar que ambos se baseiam nos mesmos *princípios*. Por exemplo, tanto um passo de uma dedução formal, quanto um passo de uma dedução informal (embora a granularidade do primeiro costume ser menor) devem ser *logicamente corretos* (em um sentido a ser esclarecido a seguir).

Assim, dadas a primeira e a terceira observações anteriores, existem duas coisas importantes no *nível formal* que devem ser abordadas com relação a uma lógica:

- a linguagem para expressão do conhecimento;
- a noção de dedução.

Correspondentemente, existem duas coisas importantes no *nível conceitual* que devem ser abordadas com relação a uma lógica, quais sejam, estruturas matemáticas a partir das quais seja possível definir:

- a semântica da linguagem para expressão do conhecimento; e
- a semântica da noção de dedução.

A ligação entre a parte formal e a conceitual pode ser dada, no primeiro caso, por exemplo, por meio de uma função denominada *função de valoração* que especifica em que condições cada sentença da linguagem é verdadeira ou falsa. No segundo caso, a ligação pode ser dada, por exemplo, por meio da equivalência entre a relação de *dedutibilidade* (do nível formal) e a de *consequência lógica* (do nível conceitual).

A noção de consequência lógica, que será a mesma para as lógicas proposicional e de predicados, pode ser informalmente definida assim: dado um conjunto de afirmativas H e uma afirmativa α, diz-se que α é consequência lógica de H se, e somente se, α é verdadeira sempre que todas as afirmativas em H sejam verdadeiras. Para dizer de forma abreviada que α é consequência lógica de H, escreve-se: $H \models \alpha$. Observe que esta definição depende do significado de "α é verdadeira", ou seja, da interpretação formal que se associe a sentenças α e da conceituação (modelagem) de "ser verdade".

A contrapartida formal da consequência lógica é a dedutibilidade. Será usada a notação $H \vdash \alpha$ para dizer que a afirmativa α é dedutível a partir do conjunto de afirmativas H. Idealmente, é interessante que se tenha:

$$H \vdash \alpha \text{ se e somente se } H \models \alpha,$$

que expressa a equivalência anteriormente referida entre as relações de dedutibilidade e de equivalência lógica. Em termos da Figura 1.3, pode-se dizer que $\models$ é a interpretação formal de $\vdash$.

Adiante será apresentada a noção de *sistema formal*, que, em particular, permite uma formalização da noção de dedutibilidade mediante um sistema axiomático. A obtenção de um sistema formal (no sentido a ser apresentado) para uma determinada lógica, que garanta a equivalência referida acima, é extremamente importante. Se por

um lado, é uma tarefa considerada absolutamente essencial do ponto de vista matemático, por mostrar que o conceito de consequência lógica em questão pode ser capturado formalmente, por outro pode também servir como base para a construção de algoritmos, os denominados *procedimentos de prova*. A facilidade com que se pode obter um procedimento de prova mais ou menos eficiente, mais ou menos produtor de deduções legíveis etc. depende do sistema formal específico no qual ele se baseia. Para a mesma lógica podem existir sistemas formais com características diferentes, que sejam mais adequados para uma coisa do que para outra.

O conceito de sistema formal é bem geral, podendo ser utilizado para a formalização de lógicas, mas também para outros fins.[5] Assim, será feita a seguir uma revisão bem sucinta do que vem a ser sistema formal. A idéia central é a geração de expressões a partir de expressões. O objetivo é obter de forma puramente mecânica (formal, sintática) resultados que, sob interpretação, correspondam àqueles intencionados no nível conceitual.

Um *sistema formal* é uma tripla (L, R, A), em que L é uma *linguagem*, R um conjunto finito de *regras de derivação* (ou, simplesmente, regras) e A é um conjunto de elementos de L denominados *axiomas*. A linguagem, como usual, é um conjunto de expressões (palavras, cadeias, *strings*), podendo ser definida por meio de uma gramática, definição recursiva etc. Já uma regra $r \in R$ é uma relação binária em que cada par $(P, c) \in r$ explicita que a expressão $c \in L$ (denominada *conclusão*) pode ser derivada a partir do conjunto finito de expressões $P \subseteq L$ (cada uma das quais denominada *premissa*). É comum especificar uma regra r por meio de *esquemas* que, sob instanciação, produzem exatamente os pares $(P, c) \in r$.[6] Um exemplo a seguir dará uma idéia de como isso pode ser feito.

Dado um sistema formal $\mathcal{S} = (L, R, A)$ e $\Sigma \subseteq L$, diz-se que de Σ *deriva-se* x (ou x é derivável a partir de Σ), escreve-se $\Sigma \Rightarrow x$, se e somente se existe uma sequência de expressões de L, $x_1, x_2, \ldots, x_n$ tal que $x_n = x$, e para $i = 1, \ldots n$:

- $x_i \in A$, ou
- $x_i \in \Sigma$, ou
- $(P, x_i) \in r$, para alguma regra $r \in R$, e cada expressão de P é alguma x_j para $j < i$.

O conjunto Σ é denominado *conjunto de partida* e a sequência $x_1, x_2, \ldots, x_n$ é denominada uma *derivação* de x (a partir de Σ). O *comprimento da derivação* é $n - 1$. Se Σ for um conjunto unitário $\{w\}$, os parênteses serão omitidos, escrevendo-se $w \Rightarrow x$ em vez de $\{w\} \Rightarrow x$. Evidentemente, o significado das expressões depende da semântica de L e o uso de $\mathcal{S}$ depende do significado associado às derivações.

[5] Na verdade, muitos autores concebem sistema formal como sendo a mesma coisa que sistema dedutivo. Nestas notas, o termo sistema dedutivo será reservado para se referir a sistema formal para uma lógica.

[6] É comum exigir-se que o conjunto R seja, não apenas recursivo (computável), mas que cada regra seja "imediatamente" identificável como podendo ser aplicada ou não a partir de uma dada expressão. Este será o caso para as duas lógicas a serem estudadas.

Nada impede que um sistema formal possa ser utilizado para múltiplos propósitos, dependentes dos significados que se associe às expressões. A adequação de um sistema formal se mede por dois atributos do mesmo *com relação ao propósito* que se tenha em mente:

- *Correção*: um sistema formal é dito correto, com relação a um propósito, se qualquer expressão derivável denota uma afirmativa (conceitualmente) correta.
- *Completude*: um sistema formal é dito completo, com relação a um propósito, se qualquer afirmativa (conceitualmente) correta é denotada por uma expressão derivável.

Com isto, em um sistema formal correto e completo são deriváveis exatamente as expressões que denotam, em conjunto, tudo aquilo que é correto afirmar no nível conceitual.

No pequeno exemplo a seguir, para cada regra r, seus conjuntos de premissas serão sempre unitários. Cada regra será representada por $f_p \mapsto f_c$, f_p especificando o formato da premissa e f_c o da conclusão.[7] Além disso, a aplicação da regra terá o efeito de substituir uma *subexpressão*, especificada em f_p, por outra, especificada em f_c. Por causa disso, em vez de f_p e f_c especificarem expressões, vão especificar subexpressões. Note que pode haver mais de uma subexpressão no formato f_p; apenas uma é substituída como resultado da aplicação da regra. Esse tipo de regra é denominado *regra de reescrita*, e sistemas formais com esse tipo de regra costumam ser chamados de *sistemas de reescrita*.

Exemplo 1 Segue um sistema formal $\mathsf{SN} = (E, S, \emptyset)$ em que cada elemento de E é uma expressão com símbolos para sucessor, soma e zero, e S tem por objetivo produzir um numeral que denote o número natural denotado por uma expressão:

- E é o conjunto de expressões no alfabeto $\{\mathsf{0}, \mathsf{s}, (,), +\}$ gerado pela gramática cujas regras são:

  $$T \;\rightarrow\; (T{+}T) \,|\, \mathsf{s}T \,|\, \mathsf{0}$$

- S é constituído das regras de reescrita:

  (1) $(m{+}\mathsf{0}) \mapsto m$
  (2) $(m{+}\mathsf{s}n) \mapsto \mathsf{s}(m{+}n)$

  sendo m e n expressões geradas pela gramática acima.

São exemplos de expressões: $\mathsf{0}$, $\mathsf{ss0}$, $(\mathsf{s0{+}ss0})$, $\mathsf{s}(\mathsf{s0{+}ss0})$, $(\mathsf{s}(\mathsf{s0{+}ss0}){+}\mathsf{s0})$ etc. Em particular, uma expressão como as duas primeiras, com apenas os símbolos s e $\mathsf{0}$ é denominada *numeral*, sendo da forma $\mathsf{ss}\ldots\mathsf{s0}$ (n símbolos s em sequência, $n \geq 0$, seguidos de $\mathsf{0}$); o propósito aqui é que tal numeral denote o número n.

A partir da expressão $(\mathsf{s}(\mathsf{s0{+}ss0}){+}\mathsf{s0})$, por exemplo, pode-se obter a seguinte derivação:

[7] Neste texto serão empregadas também outras notações para expressar regras, normalmente aquelas mais usuais nos contextos em questão.

1. (s(s0+ss0)+s0) (de Σ)
2. s(s(s0+ss0)+0) (regra 2)
3. ss(s0+ss0) (regra 1)
4. sss(s0+s0) (regra 2)
5. ssss(s0+0) (regra 2)
6. sssss0 (regra 1)

o que mostra que (s(s0+ss0)+s0) $\Rightarrow$ sssss0. (Note que existem outras derivações de sssss0 a partir de (s(s0+ss0)+s0.) Interpretando-se 0 como zero, s como sucessor (soma com 1) e + como soma, cada expressão da derivação denota o número 5; em particular, a última é o numeral respectivo. □

Seja $\eta(m)$ o número representado pela expressão m no sistema formal SN do Exemplo 1, que foi construído com o objetivo de avaliar expressões envolvendo zero, sucessor e soma, de modo a obter o numeral que denote o número natural denotado pela expressão. Para se certificar de que o sistema formal do Exemplo 1 atinge tal objetivo, deve-se demonstrar que ele tem as seguintes propriedades:

- é correto, ou seja, se $m \Rightarrow n$, então $\eta(m) = \eta(n)$, e
- é completo, ou seja, se n é o numeral tal que $\eta(m) = \eta(n)$, então $m \Rightarrow n$.

Observe que para uma expressão inicial podem existir múltiplas derivações, o que está de acordo com o caráter puramente declarativo de um sistema formal: não existe uma ordem para a aplicação de regras. Embora para o exemplo anterior quaisquer derivações confluam para o resultado pretendido (que é único e denota um número natural), em geral isso pode não acontecer. Para alguns sistemas formais, para certas expressões iniciais podem ou não existir derivações que levem a um resultado desejado; e, existindo, ele pode não ser único. O problema de construir um algoritmo que seja capaz de determinar se existe ou não uma derivação partindo de certa expressão e que chegue a um resultado intencionado pode até ser insolúvel; ou seja, para alguns problemas pode existir sistema formal, mas não existir algoritmo. Este é o caso, por exemplo, de qualquer sistema formal que seja um sistema dedutivo para lógica de primeira ordem.

No sistema formal apresentado, o lado esquerdo de cada regra especifica o formato de uma subexpressão. A subexpressão especificada pelo lado direito substitui apenas uma *subexpressão* que satisfaça o lado esquerdo. Como foi dito, esse tipo de regra é chamado regra de reescrita. Um outro tipo de regra é aquela em que o lado esquerdo especifica, não o formato de uma subexpressão, mas de uma expressão inteira ou (se tem várias premissas) conjunto de expressões inteiras. Adiantando, as regras em sistemas formais cujo propósito é a formalização da noção de consequência lógica são normalmente desse último tipo. No entanto, serão vistos também sistemas formais com regras de reescrita para propósitos mais específicos.

No próximo capítulo será abordada uma versão limitada da lógica clássica, denominada lógica proposicional, cuja linguagem é um subconjunto daquela da lógica de

predicados de primeira ordem. Existem duas razões para se apresentar essa lógica mais simples. Uma delas é possibilitar a introdução de alguns conceitos importantes em um contexto mais simples, já que os mesmos são preservados na lógica de primeira ordem, e mesmo em várias outras versões. A outra razão é que ela é suficiente para a expressão e processamento de conhecimento em uma gama apreciável de aplicações práticas.

Exercícios

1. Identifique, dentre as atividades a seguir, as que envolvem modelagem e as que envolvem formalização. A mesma atividade pode envolver ambas.

a) Obtenção um algoritmo em pseudo-liguagem.

b) Obtenção do modelo conceitual de um banco de dados.

c) Obtenção da especificação de um problema em termos de grafos, para posterior resolução por aplicação de algum algoritmo clássico já existente.

d) Codificação de um algoritmo em uma linguagem de programação.

e) Determinação de um autômato finito para solucionar um certo problema de decisão.

f) Expressão de um certo corpo de conhecimento em uma certa lógica.

2. Nas sentenças a seguir, dentre os termos sublinhados, coloque entre aspas aqueles que denotem a si mesmos. Por exemplo, nas sentenças

- Adoro azucrinar meu chefe.
- Azucrinar é um verbo transitivo.

apenas o da última deve ser colocado entre aspas.

a) O planeta terra é azul, visto do espaço.

b) Azul tem quatro letras.

c) Azul está localizada na página 98 do dicionário.

d) Minha cor preferida é azul.

e) Azul é um adjetivo.

f) Adjetivo é um substantivo masculino.

g) O adjetivo qualifica o substantivo a que estiver ligado.

h) Se $x^2 = -1$, então x não é um número real.

i) A equação $x^2 = -1$ tem raiz no domínio dos complexos.

j) Se $x^2 = -1$ é verdadeira, então x não é um número real.

3. O que há de errado com o seguinte raciocínio?

> Joãozinho adora pirulito. Pirulito é uma palavra de oito letras. Portanto, Joãozinho adora uma palavra de oito letras.

4. O que há de errado com o seguinte raciocínio?

> Pedrinho pensava que Manaus ficava em Minas Gerais. Manaus é a capital do Amazonas. Logo, Pedrinho pensava que a capital do Amazonas ficava em Minas Gerais.

5. Pesquise quais são os principais tipos de semânticas formais (aquelas que fazem o mapeamento correspondente a interpretações formais) propostas para:

a) Bancos de dados.

b) Linguagens de programação.

6. Faça uma definição recursiva de $\eta(n)$, o número denotado por uma expressão n do conjunto E do sistema formal SN do Exemplo 1.

7. Obtenha todas as derivações possíveis de sssss0 a partir de (s(s0+ss0)+s0) no sistema formal SN do Exemplo 1. *Dica*: desenhe um grafo em que os vértices sejam as expressões obtidas a partir de (s(s0+ss0)+s0) e em que há uma aresta (e_1, e_2) se, e somente se e_2 pode ser obtida de e_1 aplicando-se uma das duas regras de SN.

8. Modifique o sistema formal SN do Exemplo 1 para incluir um operador para multiplicação.

9. Prove que o sistema formal SN do Exemplo 1 é correto: se $m \Rightarrow n$, então $\eta(m) = \eta(n)$.

10. Prove que o sistema formal do Exemplo 1 SN é completo: se n é o numeral tal que $\eta(m) = \eta(n)$, então $m \Rightarrow n$.

Capítulo 2

Lógica Proposicional

A good notation sets the mind free to think about really important things.
Alfred N. Whitehead (1861–1947)

Em vez de partir diretamente para uma apresentação de como formalizar conhecimento em lógica proposicional, é feita inicialmente uma introdução à natureza da "realidade" que se pode formalizar, assim como dos conceitos abstraídos a partir dessa realidade.

2.1 A realidade sob a ótica da lógica proposicional

Que tipo de realidade pode ser formalizada através da lógica proposicional? Informalmente, a realidade é vista como composta de *proposições* e *argumentos* que envolvem tais proposições. Uma proposição é um enunciado susceptível de ser *verdadeiro* ou *falso*. Dois princípios importantes assumidos na lógica clássica são:

- princípio da *não contradição*: uma proposição não pode ser verdadeira e falsa ao mesmo tempo;
- princípio do *terceiro excluído*: toda proposição é verdadeira ou falsa.

Existem dois tipos de proposições: as *simples* e as *compostas*. Qualquer enunciado susceptível de ser verdadeiro ou falso pode ser considerado como uma proposição simples na atividade de modelagem; basta que ela possa ser considerada como *indivisível*, ou seja, não constituída de proposições mais simples. Observe que um mesmo enunciado pode ser considerado simples em uma aplicação e composto em outra. A escolha dos enunciados a serem considerados simples é uma das tarefas da (arte da) modelagem conceitual.

Exemplo 2 Podem ser consideradas proposições simples:

- *Belo Horizonte é a capital de Minas Gerais*;
- *Graciliano Ramos escreveu "Memórias do Cárcere"*;

- $2 + 2 = 5$;
- 2 *é um número primo*;
- *Elvis não morreu.*

Já expressões do tipo "*a capital de Minas Gerais*" ou "2+2" não podem ser consideradas proposições (nem simples, nem compostas), visto que não são susceptíveis de serem verdadeiras ou falsas. □

Uma proposição composta é um enunciado visto como constituído de outras proposições mais simples de forma a expressar uma:

- *Negação.* A proposição afirma que certa proposição não é verdadeira, ou seja, é falsa (pelo princípio do terceiro excluído). Exemplo: *a lua não está visível hoje* diz que a proposição *a lua está visível hoje* não é verdadeira, ou seja, é falsa.
- *Conjunção.* Afirma que duas proposições são verdadeiras. Exemplo: *o dia está lindo, embora esteja nublado* diz que *o dia está lindo* e *está nublado.*
- *Disjunção.* Afirma que pelo menos uma dentre duas proposições é verdadeira. Exemplo: *Godofredo estuda muito ou é inteligente* diz que *Godofredo estuda muito* ou *Godofredo é inteligente* ou *Godofredo estuda muito e é inteligente.*
- *Condicional.* Afirma que caso uma certa proposição (antecedente) seja verdadeira, uma outra (consequente) também é, ou seja, não é o caso que a primeira possa ser verdadeira e a outra falsa. Exemplo: *se chover hoje, não vou ao cinema* diz que se a proposição *hoje vai chover* for verdadeira, então a proposição *hoje não vou ao cinema* é verdadeira.
- *Bicondicional.* Afirma que uma certa proposição é verdadeira exatamente nos casos em que uma outra também é. Exemplo: *o número é par se, e somente se, seu quadrado é par* diz que as proposições *o número é par* e *o quadrado do número é par* são ambas verdadeiras ou ambas falsas.

Em português, existem muitas formas de expressar esses tipos de composições. Por outro lado, uma sentença em português normalmente apresenta um conteúdo que não pode ser totalmente expresso utilizando apenas tais tipos de composições. Às vezes a "aproximação" obtida ao se expressar o que é possível expressar via lógica proposicional é bastante crua... Mas, se ela for suficiente para os propósitos de uma aplicação específica, por que não? Economia conceitual pode levar a economia formal, que pode levar a maior eficiência (de processamento, do entendimento do cerne de uma questão etc.).

Exemplo 3 Seguem mais exemplos de proposições que podem ser vistas como compostas:

- *O número 5 não é negativo.* É a negação da proposição *o número 5 é negativo.*

- *O número 1 é ímpar e o 2 é par.* É a conjunção de duas proposições: *o número 1 é ímpar* e *o número 2 é par.*

- *Mário é arquiteto ou engenheiro.* É a disjunção de duas proposições: *Mário é arquiteto* e *Mário é engenheiro.*

- *Se José é engenheiro, então ele sabe matemática.* É uma proposição condicional: é verdade que *José sabe matemática*, se for verdade que *José é engenheiro.* Em outras palavras: não é possível (é falso) que *José é engenheiro* seja verdadeira e *José sabe matemática* seja falsa.

- *Se o Sr. Silva está feliz, então a Sra. Silva não está feliz, e se o Sr. Silva não está feliz, então a Sra. Silva está feliz.* É a conjunção de duas proposições condicionais: (1) *se o Sr. Silva está feliz, então a Sra. Silva não está feliz* e (2) *se o Sr. Silva não está feliz, então a Sra. Silva está feliz.* A proposição condicional (1) diz que a negação de *a Sra. Silva está feliz* é verdadeira caso *o Sr. Silva está feliz* seja verdadeira. E a proposição condicional (2) diz que a proposição *a Sra. Silva está feliz* é verdadeira caso a negação de *o Sr. Silva está feliz* seja verdadeira.

Observe que qualquer uma dessas proposições pode ser considerada uma proposição simples. Por exemplo, se a primeira proposição acima for considerada simples (na etapa de modelagem), não mais será possível distinguir que ela é a *negação* da proposição *o número 5 é negativo.* □

A disjunção, como apresentada acima, é a denominada disjunção *inclusiva*: ela afirma que uma dentre duas proposições é verdadeira *ou ambas.* Um outro tipo de disjunção frequentemente empregado é a disjunção *exclusiva*, que afirma que uma dentre duas proposições é verdadeira *mas não ambas.* Assim, a proposição *passarei o próximo sábado em Paris ou em New York* pode ser considerada, em níveis crescentes de detalhes:

- uma proposição simples, não se distinguindo a proposição como uma composição de outras;

- uma disjunção inclusiva de *passarei o próximo sábado em Paris* e *passarei o próximo sábado em New York*, não descartando a possibilidade de que aconteçam ambos os eventos;

- uma disjunção exclusiva de *passarei o próximo sábado em Paris* e *passarei o próximo sábado em New York*, descartando a possibilidade de que aconteçam ambos os eventos.

Note que, assim como não é incorreto considerar a proposição anterior uma proposição simples, também não é incorreto considerá-la uma disjunção inclusiva: neste caso, apenas deixa-se de capturar uma informação *adicional*, qual seja, que *passarei o próximo*

sábado em Paris e *passarei o próximo sábado em New York* não podem ser ambas verdadeiras.

Vendo o princípio da não contradição como uma *metaproposição*, ele diz com relação a qualquer proposição α: a negação da conjunção de α e a negação de α deve ser verdadeira. E vendo o princípio do terceiro excluído como uma *metaproposição*, ele diz com relação a qualquer proposição α: a disjunção de α e a negação de α deve ser verdadeira. As duas metaproposições dizem em conjunto, com relação a qualquer proposição α: a disjunção *exclusiva* de α e a negação de α deve ser verdadeira.

Uma simplificação digna de nota, inerente à lógica matemática, é que a veracidade de uma proposição composta, para todos os tipos de composição apresentados, depende única e exclusivamente da veracidadade das proposições componentes. Com isto, por exemplo, a conjunção de duas proposições deve ser considerada verdadeira se, e somente se, ambas forem verdadeiras, a disjunção de duas proposições será verdadeira se, e somente se, uma delas ou ambas forem verdadeiras, e assim por diante. Em outras palavras, o significado de cada tipo de composição pode ser modelado mediante uma função (como será visto na Seção 2.3). Por outro lado, não há limite para a construção de uma proposição composta com relação ao uso dos vários tipos e ao número de níveis de composição. Embora exemplos disso tenham que esperar pela introdução de símbolos para expressar proposições simples, o que será feito na próxima seção, deve-se ressaltar que as proposições resultantes podem ser extremamente complexas de serem enunciadas em linguagem natural.

As proposições são os componentes básicos dos argumentos. Um argumento, tipicamente, diz que uma certa proposição (a *conclusão*) é verdadeira caso as proposições de um certo conjunto (as *premissas*) sejam verdadeiras. Segue um exemplo.

Exemplo 4 Segue um argumento do tipo que pode ser modelado e formalizado mediante a lógica proposicional.

> Supondo que:
>
> - *meu chapéu é vermelho ou branco*;
> - *se meu chapéu é vermelho, então o dele é branco*; e
> - *o chapéu dele não é branco*;
>
> pode-se concluir que *meu chapéu é branco*.

Neste argumento, as três primeiras proposições são premissas, e a última, *meu chapéu é branco*, é a conclusão. Deve-se notar que dificilmente esse argumento fará sentido se alguma das três primeiras proposições for considerada uma proposição simples. Se a primeira for considerada uma disjunção, a segunda uma condicional e a terceira uma negação, aí sim, é o que basta para alguém se convencer que o argumento é "correto" (tente!). □

Se, de fato, a conclusão do argumento for verdadeira sempre que todas as premissas o forem, diz-se que o argumento é *correto* ou *válido*, ou ainda, é impossível que todas

as premissas sejam verdadeiras e a conclusão falsa. Se for possível as premissas serem verdadeiras e a conclusão falsa, o argumento é dito *incorreto* ou *inválido*. Por exemplo, o argumento

> Supondo que:
>
> - *se Mário é engenheiro então ele sabe matemática*; e
> - *Mário é engenheiro*;
>
> pode-se concluir que *Mário sabe matemática*.

é um argumento correto (válido), enquanto que o o argumento

> Supondo que:
>
> - *se Mário é engenheiro então ele sabe matemática*; e
> - *Mário sabe matemática*;
>
> pode-se concluir que *Mário é engenheiro*.

é um argumento incorreto (inválido). Para explicar por que assim é, precisa-se definir com precisão uma conceituação para as formas de composição usadas (no caso, condicional) e para a noção de consequência, o que será feito adiante.

No nível formal, uma proposição será representada por meio de uma *fórmula*, sendo que para cada tipo de composição de proposições será utilizado um *conectivo lógico*. Uma característica fundamental da lógica é a separação cuidadosa dos níveis formal e conceitual, de forma a se obter uma semântica precisa.

Os conceitos correspondentes a "verdadeiro" e "falso" serão aqui referidos por V e F. Assim, V será usado no nível conceitual para denotar que uma certa proposição é *verdadeira* e F será usado para denotar que ela é *falsa*. Como já ressaltado anteriomente, esta associação entre V e verdadeiro e entre F e falso só existe em nossa mente. Uma "função de valoracão", que fará a ponte entre os níveis formal e conceitual, será definida adiante de tal maneira que, dada uma fórmula, ela associará à mesma um dos dois valores, V ou F. O uso de uma função é condizente com o princípio já referido da *não contradição*: uma proposição não pode ser verdadeira e falsa ao mesmo tempo; ou seja, a valoração deve ter realmente a propriedade de *unicidade* inerente a uma função. Além disso, coerente com o princípio do *terceiro excluído* (toda proposição é verdadeira ou falsa), a imagem da função de valoração deve ser o conjunto $\{V, F\}$.

Além de V e F, os valores passíveis de serem atribuídos como significado de uma fórmula pela função de valoração, o nível conceitual conterá os elementos necessários para a definição dessa função: as *interpretações* de cada um dos conectivos lógicos.

Definida a função de valoração, que dará então a semântica das fórmulas (que representam proposições), será possível também definir em que condições um argumento é correto, como será visto mais adiante.

Após essa descrição bastante sucinta do que se espera modelar e formalizar (proposições e argumentos) e da estrutura matemática a ser utilizada na modelagem (basicamente, os dois valores, V e F, e funções desses dois valores) na próxima seção sobe-se

diretamente para o nível formal, apresentando-se a linguagem proposicional. Em seguida, é apresentada a semântica da linguagem proposicional, descendo-se, portanto, para o nível conceitual esboçado anteriormente.

Exercícios

1. Identifique as proposições simples que compõem as proposicões a seguir, assim como os tipos de composição. Procure identificar o máximo possível de proposições simples, isto é, considere uma proposição como simples somente se não for possível considerá-la composta. Há certos aspectos que não são representáveis em lógica proposicional; ignore-os.

a) João é político mas é honesto.

b) João é honesto, mas seu irmão não é.

c) Virão à festa João ou sua irmã, além da mãe.

d) A estrela do espetáculo não canta, dança, nem representa.

e) Sempre que o trem apita, João sai correndo.

f) Caso João não perca o dinheiro no jogo, ele vai à feira.

g) Uma condição suficiente para que n seja ímpar, é que seja primo.

h) João vai ao teatro somente se estiver em cartaz uma comédia.

i) Se João for ao circo ou ao futebol hoje, ele estará sem dinheiro amanhã.

j) Se você for brasileiro, gosta de futebol, desde que não torça para o Tabajara.

k) João vai ser multado, a menos que diminua a velocidade ou a rodovia não tenha radar.

l) A propina será paga exatamente nas situações em que o deputado votar como instruído pelo João.

m) Se, mas somente se, João não peder o dinheiro no jogo, ele irá pagar o que lhe deve.

n) Se João testemunhar e disser a verdade, ele será absolvido, mas se João não testemunhar, ele não será absolvido.

2. Que tipo de disjunção está envolvida em cada uma das proposições a seguir?

a) Vou viajar pela Europa se herdar uma fortuna ou ganhar na loteria.

b) Brasil: ame-o ou deixe-o.

c) $n \geq 2$.

d) Ao refém restava trair o seu país ou morrer por ele.

e) Quando estou com dor-de-cotovelo ou meu time perde ou empata, procuro ir ao cinema para me distrair.

3. Considere o seguinte conjunto de proposições:

> O mordomo é inocente ou culpado. Se o mordomo é inocente, então a testemunha mentiu. Se o mordomo tem um cúmplice, então a testemunha não mentiu. A testemunha tendo mentido ou não, o mordomo tem um cúmplice.

Supondo que tais proposições sejam verdadeiras, responda: o mordomo é culpado ou é inocente? Explique sua resposta de uma forma coloquial, sem apelar para os conceitos e formalismos lógicos que porventura conheça.

4. Considere os argumentos:

a) Em Frivolândia, todo dia tem futebol, festa ou circo. E quando tem futebol, tem festa. Como hoje não tem festa, então tem circo.

b) Se meu time perde, fico infeliz. E quando estou feliz, vou passear. Logo, se meu time perde, não vou passear.

Para cada um, verifique se ele está correto. Explique sua resposta de uma forma coloquial, sem apelar para os conceitos e formalismos lógicos que porventura conheça.

5. Seja o seguinte argumento referente a um número natural n:

> Se n for par, então $n > 300$. Se n for ímpar, então $100 \leq n \leq 200$. Sabe-se que n não é maior que 200. Portanto, $100 \leq n \leq 200$.

Para explicar a correção desse argumento, deve-se usar algumas proposições implícitas da aritmética. Explique porque o argumento é válido, explicitando também tais proposições implícitas.

2.2 A linguagem proposicional

O alfabeto da linguagem consta de três tipos de símbolos:

- *Variáveis proposicionais*: símbolos usados para representar proposições simples. O conjunto das variáveis proposicionais é o conjunto *infinito* constituído de p_0, p_1, p_2, ... Normalmente, em aplicações, usa-se palavras (ou mesmo frases), em vez desses símbolos, que lembrem as proposições representadas. Isso será feito nos exemplos deste texto. O conjunto das variáveis será denotado por $\mathcal{V}$.

- *Conectivos lógicos*: símbolos usados para representar os tipos de composição. São eles: $\top$ (*verum*), $\bot$ (*falsum*), $\neg$ (negação), $\wedge$(conjunção), $\vee$ (disjunção), $\rightarrow$ (condicional) e $\leftrightarrow$ (bicondicional).

- *Símbolos auxiliares*: abre e fecha parênteses.

A linguagem nada mais é do que um conjunto, $\mathcal{F}$, de sequências de símbolos, denominadas *fórmulas* (ou fórmulas bem formadas ou sentenças), que pode ser definido recursivamente como segue.

Definição 1 *O conjunto de* fórmulas, $\mathcal{F}$, *que constitui a linguagem da lógica proposicional pode ser assim definido:*

a) $\mathcal{V} \subseteq \mathcal{F}$, *ou seja, cada variável proposicional é uma fórmula, denominada* fórmula atômica *ou* átomo*;*

b.1) $\top$ *e* $\bot$ *são fórmulas;*

b.2) se α *é uma fórmula, então* $(\neg\alpha)$ *é uma fórmula;*

b.3) se α *e* β *são fórmulas, então* $(\alpha\wedge\beta)$, $(\alpha\vee\beta)$, $(\alpha \to \beta)$ *e* $(\alpha \leftrightarrow \beta)$ *são fórmulas;*

c) só são fórmulas as palavras que podem ser obtidas como indicado em (a) e (b.1) a (b.3). □

Em uma fórmula $\gamma = (\alpha c \beta)$, sendo c um dos conectivos binários, c é dito ser o *conectivo principal* de γ. Já se a fórmula for $\top$, $\bot$ ou da forma $(\neg\alpha)$, terá como conectivo principal $\top$, $\bot$ ou $\neg$, respectivamente.

Exemplo 5 Considere o Exemplo 3. Seguem algumas representações de proposições simples por variáveis proposicionais (tal associação só existe na nossa cabeça):

- *5neg*: *o número 5 é negativo*;
- *1ímpar*: *o número 1 é ímpar*;
- *2par*: *o número 2 é par*;
- *arq*: *Mário é arquiteto*;
- *eng*: *Mário é engenheiro.*

Eis algumas fórmulas seguidas, entre parênteses, pelas proposições que representam:

- *5neg* (*o número 5 é negativo*);
- $(\neg$*5neg*$)$ (*o número 5 não é negativo*);
- $($*1ímpar* $\wedge$ *2par*$)$ (*o número 1 é ímpar e o 2 é par*);
- $($*arq* $\vee$ *eng*$)$ (*Mário é arquiteto ou engenheiro*);
- $(($*arq* $\wedge$ *eng*$) \to$ *5neg*$)$ (*se Mário é arquiteto e engenheiro, então o número 5 é negativo*).

Esta última fórmula exemplifica como se pode expressar coisas que em linguagem natural soam absurdas (ou estranhas). O conectivo principal dela é $\to$. □

Cada fórmula não atômica receberá a mesma designação que é dada à proposição que ela representa. Assim, uma fórmula da forma $(\neg\alpha)$ é dita ser a *negação* de α; $(\alpha \wedge \beta)$ é a *conjunção* de α e β, $(\alpha \vee \beta)$ é a *disjunção* de α e β, $(\alpha \rightarrow \beta)$ é uma fórmula *condicional* e $(\alpha \leftrightarrow \beta)$ uma *bicondicional*. No caso da condicional $(\alpha \rightarrow \beta)$, α é dita ser o *antecedente* e β o *consequente*.

Na verdade, poderiam ser introduzidos outros conectivos além daqueles explicitados na Definição 1. Por outro lado, como será visto posteriormente, após se considerar a semântica dos conectivos, o conjunto de conectivos adotado naquela definição é redundante, no sentido de alguns podem ser expressos por meio de outros. Por que, então, a escolha do conjunto de conectivos específico da Definição 1? A resposta é que tal conjunto é o *mais conveniente* para uso nas aplicações que temos em mente, basicamente o do uso de lógica como formalismo para expressão e/ou manipulação de conhecimento em geral. Em outros tipos de aplicação, como, por exemplo, especificação e concepção de circuitos digitais, um outro conjunto de conectivos é mais adequado; por exemplo, nesse contexto costuma haver conectivos para ou exclusivo, negação da conjunção e negação da disjunção.

Os conectivos $\top$ e $\bot$, em particular, são dispensáveis para expressão de conhecimento. No entanto, eles estão sendo introduzidos porque podem ser úteis do ponto de vista de manipulação no nível formal, como será visto mais à frente.

Nos exemplos em que não for importante que proposições as variáveis proposicionais representam, serão usados como variáveis as letras p, q, r e s, com ou sem índices. Note que, ainda assim, elas continuam sendo símbolos de $\mathcal{V}$ (não são "variáveis" sintáticas).

Na medida em que as fórmulas se tornam maiores, elas podem ficar bem ilegíveis. Por isso, será admitida, daqui para frente, a omissão de parênteses. Neste caso, assume-se que os parênteses são repostos de acordo com as duas regras:

- os conectivos têm a seguinte ordem de precedência, da maior para a menor: $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$.
- para conectivos idênticos, faz-se a associação à direita.

Veja exemplo a seguir.

Exemplo 6 A expressão $p_1 \wedge p_2 \wedge p_3 \rightarrow \neg q \vee r$ abrevia a fórmula $((p_1 \wedge (p_2 \wedge p_3)) \rightarrow ((\neg q) \vee r))$, assim obtida, passo a passo:

- $p_1 \wedge p_2 \wedge p_3 \rightarrow (\neg q) \vee r$ (precedência de $\neg$);
- $(p_1 \wedge (p_2 \wedge p_3)) \rightarrow (\neg q) \vee r$ (precedência de $\wedge$ e associação à direita);
- $(p_1 \wedge (p_2 \wedge p_3)) \rightarrow ((\neg q) \vee r)$ (precedência de $\vee$);
- $((p_1 \wedge (p_2 \wedge p_3)) \rightarrow ((\neg q) \vee r))$ (parênteses externos).

Veja que o conectivo principal de $p_1 \wedge p_2 \wedge p_3 \rightarrow \neg q \vee r$ é $\rightarrow$. □

Note que a fórmula $((p_1 \wedge p_2) \wedge p_3)$ pode ser abreviada como $(p_1 \wedge p_2) \wedge p_3$, mas não como $p_1 \wedge p_2 \wedge p_3$. Esta última abrevia $(p_1 \wedge (p_2 \wedge p_3))$. As duas regras fornecem um critério puramente sintático de eliminação de parênteses.

Uma coisa que se deve ter em mente é que o estoque de variáveis, $\mathcal{V}$, é considerado infinito. Evidentemente, em uma aplicação é usado apenas um subconjunto finito de variáveis. O fato de $\mathcal{V}$ ser infinito não complica as coisas, visto que a verdade de uma fórmula não depende de variáveis que não ocorram na mesma, como será visto.

Exercícios

1. Represente as proposições do Exercício 1 da seção anterior por meio de fórmulas. Ao fazê-lo, indique que variáveis proposicionais representam que proposições simples. Como lá, procure usar o máximo possível de conectivos, isto é, considere uma proposição como simples somente se não for possível considerá-la composta. Aspectos não representáveis em lógica proposicional devem ser ignorados.

2. Represente as proposições do Exercício 3 da seção anterior por meio de fórmulas.

3. Formalize os dois argumentos do Exercício 4 da seção anterior.

4. Formalize o argumento do Exercício 5 da seção anterior, incluindo as proposições implícitas necessárias para tornar o argumento válido.

5. Reponha os parênteses das seguintes fórmulas de acordo com as regras enunciadas na seção:

a) $p \to q \to \bot$

b) $\neg p_1 \wedge p_2 \to q_1 \vee q_2 \vee q_3$

c) $p_1 \vee p_2 \leftrightarrow \neg q_1 \wedge q_2$

d) $p_1 \leftrightarrow p_2 \to p_3 \vee p_4 \wedge p_5$

e) $\neg(\top \wedge \neg\bot) \leftrightarrow \top \to \bot \vee \top$

2.3 Semântica da linguagem proposicional

A semântica das fórmulas será dada pela *função de valoração* $v^i : \mathcal{F} \to \{V, F\}$, uma função que dá, para uma certa interpretação formal i das variáveis proposicionais, o valor V ou F. Como já ressaltado anteriormente, V pode ser imaginado como *verdadeiro* e F como *falso*. Assim, dada uma fórmula $\alpha \in \mathcal{F}$, $v^i(\alpha)$ será V ou F. Veja que $v^i(\alpha)$ não é a proposição representada por α; esta só existe (se existir) na nossa cabeça. Em outras palavras, $v^i(\alpha)$ é a interpretação formal de α, V ou F, e da proposição correspondente a α (no nível da realidade) só é capturado matematicamente o fato de que ela é V (verdadeira) ou F (falsa).

2.3.1 O significado das fórmulas

Truth is agreement of thought with its object. (Aristóteles, 384 AC–322 AC)

A função de valoração é definida levando-se em consideração os significados (interpretações formais) das variáveis proposicionais e dos conectivos lógicos. Tais significados serão dados, cada um por uma função específica. Para os conectivos $\top$ e $\bot$, o significado será dado por duas funções sem argumentos (constantes, portanto), para o conectivo $\neg$, ele será dado por uma função unária, e para os outros conectivos será dado por funções binárias. O significado das variáveis proposicionais será dado por meio de uma função i que associa a cada variável ν o valor $\nu^i \in \{V, F\}$. Pode-se imaginar que a função i dá uma *interpretação* (formal) para cada variável proposicional. Para n variáveis proposicionais, existem 2^n funções i distintas, já que $\nu^i \in \{V, F\}$.[1]

Exemplo 7 Sejam as seguintes variáveis proposicionais, com as respectivas proposições que elas representam (só na nossa cabeça):

- p: *a terra gira em torno do sol*;
- q: *Paris é a capital da França*;
- r: $\sqrt{2}$ *é um número racional*.

Então, de maneira consistente com a realidade em que vivemos, as duas primeiras proposições são verdadeiras e a terceira é falsa:

- $p^i = V$;
- $q^i = V$;
- $r^i = F$.

Se na nossa "realidade" tudo que interessa pode ser expresso a partir destas três proposições simples, então ν^i é irrelevante para ν diferente de p, q e r, mas é considerado como sendo um dos dois valores: V ou F. □

Cada conectivo é interpretado como uma *função booleana*. Em particular, $\top$ e $\bot$ são interpretados como as constantes (booleanas) $\textcircled{\top} = V$ e $\textcircled{\bot} = F$. O conectivo $\neg$ é interpretado como uma função unária $\textcircled{\neg} : \{V, F\} \to \{V, F\}$ e os outros são interpretados como funções binárias $\textcircled{c} : \{V, F\} \times \{V, F\} \to \{V, F\}$, $c \in \{\wedge, \vee, \to, \leftrightarrow\}$, sendo tais funções como definido a seguir.[2]

[1] Na verdade, o suprimento de variáveis proposicionais é infinito. O abuso de linguagem aqui faz sentido se se considerar que o valor lógico de uma fórmula α só depende dos valores associados (por i) às n variáveis que ocorrem em α.

[2] Por meio de $\textcircled{c}$, sendo c um conectivo, denota-se a função correspondente ao conectivo c. Assim, $\textcircled{\neg}$ é a função que dá o significado do conectivo $\neg$.

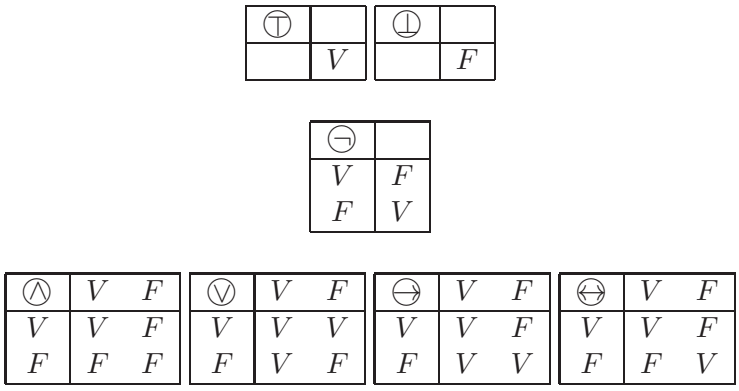


Figura 2.1: Significados dos conectivos.

Definição 2 *As interpretações de* $\top$, $\bot$, $\neg$ *e de cada conectivo* $c \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$ *são, respectivamente, as funções* $\textcircled{\top}$, $\textcircled{\bot}$, $\textcircled{\neg} : \{V, F\} \rightarrow \{V, F\}$ *e* $\textcircled{c} : \{V, F\} \times \{V, F\} \rightarrow \{V, F\}$ *tais que:*

- $\textcircled{\top} = V$;
- $\textcircled{\bot} = F$;
- $\textcircled{\neg}(V) = F$ *e* $\textcircled{\neg}(F) = V$;
- $\textcircled{\wedge}(x, y) = V$ *se* $x = V$ *e* $y = V$, *e* $\textcircled{\wedge}(x, y) = F$ *nos outros três casos;*
- $\textcircled{\vee}(x, y) = F$ *se* $x = F$ *e* $y = F$, *e* $\textcircled{\vee}(x, y) = V$ *nos outros três casos;*
- $\textcircled{\Rightarrow}(x, y) = F$ *se* $x = V$ *e* $y = F$, *e* $\textcircled{\Rightarrow}(x, y) = V$ *nos outros três casos;*
- $\textcircled{\Leftrightarrow}(x, y) = V$ *se* $x = y$, *e* $\textcircled{\Leftrightarrow}(x, y) = F$ *se* $x \neq y$.

A Figura 2.1 apresenta as funções em formato tabular. □

A seguir, expressões envolvendo as funções booleanas serão apresentadas na notação infixada. Para isto, será assumida para cada função a mesma precedência que o conectivo correspondente. Por exemplo, $\textcircled{\neg}V\textcircled{\Rightarrow}F\textcircled{\vee}V$ irá significar o mesmo que $\textcircled{\Rightarrow}(\textcircled{\neg}(V), \textcircled{\vee}(F, V))$.

A interpretação (formal) de uma fórmula é a valoração lógica $v^i : \mathcal{F} \rightarrow \{V, F\}$, que pode ser definida recursivamente utilizando-se as interpretações dos símbolos do alfabeto, como segue.

Definição 3 *A* valoração lógica *de uma fórmula,* $v^i : \mathcal{F} \rightarrow \{V, F\}$, *é definida recursivamente a partir de* i*:*

- $v^i(\nu) = \nu^i$ *para cada* $\nu \in \mathcal{V}$;
- $v^i(\top) = \textcircled{\top} = V$ *e* $v^i(\bot) = \textcircled{\bot} = F$;

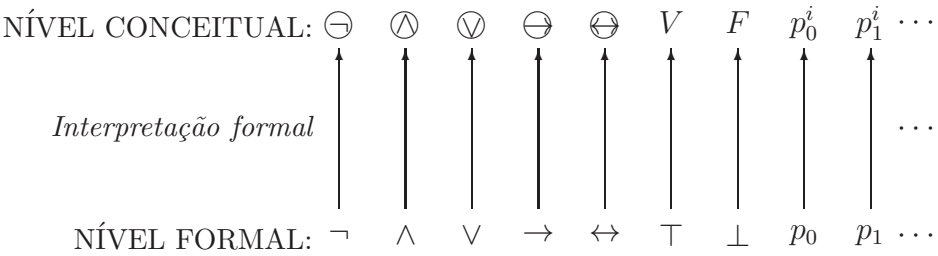


Figura 2.2: Interpretação dos símbolos da linguagem proposicional.

- *se* $\alpha \in \mathcal{F}$*, então* $v^i(\neg\alpha) = \ominus v^i(\alpha)$*;*
- *se* $\alpha, \beta \in \mathcal{F}$*, então*
  $v^i(\alpha \wedge \beta) = v^i(\alpha) \oslash v^i(\beta)$,
  $v^i(\alpha \vee \beta) = v^i(\alpha) \oslash v^i(\beta)$,
  $v^i(\alpha \rightarrow \beta) = v^i(\alpha) \ominus v^i(\beta)$,
  $v^i(\alpha \leftrightarrow \beta) = v^i(\alpha) \ominus v^i(\beta)$. □

Em síntese, o mapeamento entre a linguagem, no nível formal, e a estrutura matemática constituída dos dois valores, V e F, no nível conceitual, é feito por meio da função de valoração v^i. Ou seja, $v^i(\alpha)$ é a interpretação formal da fórmula α. Tal função é definida a partir das interpretações formais dos conectivos, dadas pelas funções booleanas $\oplus$, $\oplus$, $\ominus$, $\oslash$, $\oslash$, $\ominus$, $\ominus$, e dos valores das variáveis, p_0^i, p_1^i, p_2^i etc., sendo que cada um destes é V ou F. Tal mapeamento está esquematizado na Figura 2.2, em que são mostrados apenas os símbolos do alfabeto e seus significados. A partir deles, a definição de v^i mapeia qualquer sentença α no nível formal para seu significado $v^i(\alpha) \in \{V, F\}$ no nível conceitual. Observe que $v^i(\alpha)$ só depende dos valores p_0^i, p_1^i, ... relativos às variáveis proposicionais, visto que as funções que interpretam os conectivos são predeterminadas (é como se $\top^i = V$, $\bot^i = F$, $\neg^i = \ominus$, $\wedge^i = \oslash$ etc., para toda interpretação i). Diz-se que os conectivos, tendo significados predeterminados, são *símbolos lógicos*, enquanto que as variáveis proposicionais, não os tendo, são *símbolos não lógicos*.[3]

Exemplo 8 A valoração lógica de $(p \rightarrow q) \leftrightarrow (\neg p \vee \bot)$ para i tal que $p^i = V$ e $q^i = F$ será calculada conforme a Definição 3:

$$\begin{aligned} v^i((p \rightarrow q) \leftrightarrow (\neg p \vee \bot)) &= (p^i \ominus q^i) \ominus (\ominus p^i \oslash \oplus) \\ &= (V \ominus F) \ominus (\ominus V \oslash F) \\ &= F \ominus (F \oslash F) \\ &= F \ominus F \\ &= V \end{aligned}$$

[3]Em inglês, *nonlogical symbols*.

Note que, como as regras de precedência das funções booleanas são as mesmas dos conectivos, o processo pode ser iniciado pela substituição literal dos conectivos pelos nomes das respectivas funções e de cada variável ν por ν^i. □

Como dito na seção anterior, outros conectivos além dos previstos nas Definições 1 e 3 poderiam ser introduzidos (como um para disjunção exclusiva, por exemplo). Além disso, aquele grupo de conectivos é redundante. Por exemplo, a bicondicional pode ser expressa utilizando-se conjunção e condicional, já que $\circledcirc(x,y) = \circledwedge(\circledrightarrow(x,y), \circledrightarrow(y,x))$, como pode ser verificado. O uso dos conectivos específicos previstos na Definição 3 é justificado por sua conveniência para os propósitos que se tem em mente.

Se, por um lado, a veracidade de uma fórmula depende de i, a veracidade de ν^i só pode ser conhecida consultando-se a "realidade" para saber se a proposição representada por ν é verdadeira ou falsa. Como será visto posteriormente, no nível formal a única forma de circunscrever os valores lógicos que um conjunto de variáveis proposicionais pode assumir é afirmar (ou assumir) que certas fórmulas em que elas participam são verdadeiras (ou seja, têm valor V). Assim, por exemplo, quando se afirma (ou se supõe) no nível formal a fórmula $p \wedge (p \rightarrow (q \vee r))$, afirma-se (conclui-se) no nível conceitual que $v^i(p \wedge (p \rightarrow (q \vee r))) = V$; com isto, $v^i(p) = V$ e $v^i(p \rightarrow (q \vee r)) = V$ e, como consequência, $v^i(q)$ e $v^i(r)$ não podem ser ambos F.

Na próxima subseção será abordado o problema de, dadas uma fórmula α e uma interpretação i, determinar $v^i(\alpha)$. Lá será enfatizado o estilo de trabalhar diretamente com as entidades do nível conceitual, no caso, as funções booleanas. Em uma seção posterior, esse mesmo problema será abordado trabalhando no nível formal, ou seja com base em sistema formal.

Exercícios

1. Prove que $v^i(\alpha \leftrightarrow \beta) = v^i((\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha))$.

2. A disjunção vista nesta seção é a chamada disjunção *inclusiva*. Uma outra, menos usada em matemática (mas mais usada em outros contextos) é a disjunção *exclusiva*, em que se considera "α ou β" verdadeira apenas se α é verdadeira ou β verdadeira, mas não ambas. Como seria a função booleana para *ou exclusivo*? Use o símbolo $\oplus$ como conectivo para *ou exclusivo*. Mostre como expressar *ou exclusivo* sem um símbolo específico para tal, ou seja, usando apenas os conectivos já vistos: expresse $\circledplus$ como uma composição das outras funções boolenas vistas (como foi feito no texto para a bicondicional).

3. Quantos conectivos constantes, unários e binários são possíveis? (Ou seja, quantas funções de zero, um e dois argumentos existem sobre $\{V, F\}$?)

4. Para cada fórmula, determine seu valor lógico para a interpretação i em que $p^i = V$, $q^i = F$ e $r^i = V$:

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

b) $(\neg p \rightarrow \neg\neg p) \rightarrow p$

c) $((q \rightarrow p) \leftrightarrow q) \rightarrow p$

d) $((p \rightarrow q) \vee r) \rightarrow (\neg q \wedge (p \vee r))$

5. Suponha que $v^i(p \rightarrow \alpha) = V$ e $v^i(\neg p \rightarrow \beta) = V$ para certas fórmulas α e β. Prove que, neste caso, $v^i(\alpha \vee \beta) = V$.

6. Para cada fórmula a seguir, determine interpretações para suas variáveis que as façam receber o valor V, se possível, e interpretações que as façam receber o valor F, se possível.

a) $(p \rightarrow \bot) \vee (p \rightarrow \top)$

b) $\neg(p \vee q) \rightarrow (p \rightarrow \bot)$

c) $(p \vee q) \rightarrow (p \rightarrow q)$

d) $(p \wedge q) \rightarrow (p \leftrightarrow q)$

e) $(p \leftrightarrow q) \rightarrow (p \wedge q)$

7. Mostre como os significados dos conectivos presentes na Definição 2 podem ser definidos a partir dos significados de:

a) $\vee$ e $\neg$

b) $\wedge$ e $\neg$

c) $\rightarrow$ e $\bot$

2.3.2 Avaliação de fórmulas no nível conceitual

Um algoritmo óbvio para, dadas uma fórmula α e uma interpretação i, calcular $v^i(\alpha)$ é aquele extraído diretamente da Definição 3. Tal algoritmo é recursivo como a definição citada e trabalha do conectivo principal em direção às variáveis proposicionais da fórmula. Veja a Figura 2.3.

O método mais tradicional de se calcular $v^i(\alpha)$ é o da *tabela da verdade*, que trabalha na direção inversa: partindo-se das variáveis proposicionais, os valores lógicos de cada subfórmula vão sendo calculados até se chegar ao de α. Antes de mais nada, será visto um exemplo em que o método é apresentado informalmente.

Exemplo 9 Sejam as seguintes proposições simples, juntamente com variáveis proposicionais que as representam:

- *ch*: *está chovendo*;
- *ce*: *há um cano estourado*;
- *rm*: *a rua está molhada*;
- *re*: *a rua está escorregadia*.

proc avalia(*fórmula* α, *interpretação* i) **retorna** $\{V, F\}$:
  caso α **seja**
    var: **retorne** α^i
    $\top$: **retorne** V
    $\bot$: **retorne** F
    $\neg\beta$: **retorne** $\ominus$avalia(β, i)
    $\beta \wedge \gamma$: **retorne** avalia$(\beta, i) \oslash$ avalia(γ, i)
    $\beta \vee \gamma$: **retorne** avalia$(\beta, i) \oplus$ avalia(γ, i)
    $\beta \rightarrow \gamma$: **retorne** avalia$(\beta, i) \ominus$ avalia(γ, i)
    $\beta \leftrightarrow \gamma$: **retorne** avalia$(\beta, i) \otimes$ avalia(γ, i)
  fimcaso
fim avalia

Figura 2.3: Avaliação de uma fórmula.

Suponha uma situação em que:

- $ch^i = V$ (*está chovendo*);
- $ce^i = F$ (*não há um cano estourado*);
- $rm^i = V$ (*a rua está molhada*); e
- $re^i = V$ (*a rua está escorregadia*).

Seja $\alpha = ((ch \vee ce) \rightarrow rm) \wedge (rm \rightarrow re)$ (*se está chovendo ou há um cano estourado, a rua está molhada; e se a rua está molhada, então está escorregadia*). Para calcular $v^i(\alpha)$ pelo método da tabela da verdade, inicialmente obtém-se as subfórmulas de α em uma ordem tal que todas as subfórmulas de uma subfórmula β figurem antes de β. Para este exemplo, pode ser:

ch	ce	rm	re	$ch \vee ce$	$rm \rightarrow re$	$(ch \vee ce) \rightarrow rm$	α

Em seguida, anota-se os valores ν^i para as variáveis proposicionais ν embaixo das mesmas:

ch	ce	rm	re	$ch \vee ce$	$rm \rightarrow re$	$(ch \vee ce) \rightarrow rm$	α
V	F	V	V				

Agora calcula-se os valores lógicos $v^i(\beta)$ das subfórmulas β restantes da esquerda para a direita, obtendo-se:

ch	ce	rm	re	$ch \vee ce$	$rm \rightarrow re$	$(ch \vee ce) \rightarrow rm$	α
V	F	V	V	V	V	V	V

Portanto, $v^i(\alpha) = V$. Por outro lado, mantendo-se os mesmos valores para ch^i, ce^i e rm^i e fazendo-se $re^i = F$, obtém-se:

ch	ce	rm	re	$ch \vee ce$	$rm \rightarrow re$	$(ch \vee ce) \rightarrow rm$	α
V	F	V	F	V	F	V	F

E, neste caso, $v^i(\alpha) = F$. □

Define-se agora, com precisão, o conceito de subfórmula.

Definição 4 *O conjunto das* subfórmulas *de uma fórmula α, $subf(\alpha)$, pode ser assim definido:*

a) se $\alpha \in \mathcal{V} \cup \{\top, \bot\}$, então $subf(\alpha) = \{\alpha\}$;

b) se $\alpha = (\neg\beta)$, então $subf(\alpha) = \{\alpha\} \cup subf(\beta)$;
se $\alpha = (\beta c \gamma)$, em que $c \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$, $subf(\alpha) = \{\alpha\} \cup subf(\beta) \cup subf(\gamma)$.

No item (b), β e γ são ditas ser subfórmulas imediatas *de α, sendo que se $\alpha = (\beta c \gamma)$, β é uma subfórmula imediata* à esquerda *e γ uma subfórmula imediata* à direita. □

Exemplo 10 As subfórmulas de $\alpha = (p \rightarrow q) \wedge (\neg q \vee p)$, podem ser assim calculadas como indicado na Definição 4:

$$\begin{aligned} subf(\alpha) &= \{\alpha\} \cup subf(p \rightarrow q) \cup subf(\neg q \vee p) \\ &= \{p \rightarrow q, \neg q \vee p, \alpha\} \cup subf(p) \cup subf(q) \cup subf(\neg q) \cup subf(p) \\ &= \{p, q, \neg q, p \rightarrow q, \neg q \vee p, \alpha\} \cup subf(q) \\ &= \{p, q, \neg q, p \rightarrow q, \neg q \vee p, \alpha\} \end{aligned}$$

Note que as subfórmulas imediatas de α são $p \rightarrow q$ e $\neg q \vee p$, sendo $p \rightarrow q$ à esquerda e $\neg q \vee p$ à direita. □

A relação binária *subfórmula imediata* restrita ao conjunto $subf(\alpha)$ será designada por $si(\alpha)$. Segue uma definição.

Definição 5 *Seja α uma fórmula. A relação $si(\alpha) \subseteq subf(\alpha) \times subf(\alpha)$ é tal que para quaisquer $\beta, \gamma \in subf(\alpha)$, $(\beta, \gamma) \in si(\alpha)$ se, e somente se β é subfórmula imediata de γ.*

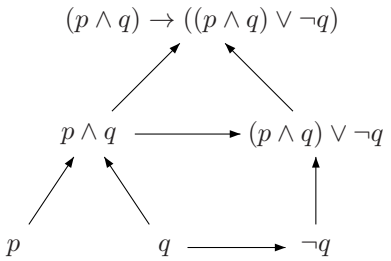
Uma forma alternativa de dizer $(\beta, \gamma) \in si(\alpha)$ é dizer que "β é uma subfórmula imediata *de γ em α".* □

Exemplo 11 Seja $\alpha = (p \rightarrow q) \wedge (\neg q \vee p)$. Como mostrado no Exemplo 10, $subf(\alpha) = \{p, q, \neg q, p \rightarrow q, \neg q \vee p, \alpha\}$. Aplicando-se a Definição 5, vê-se que:

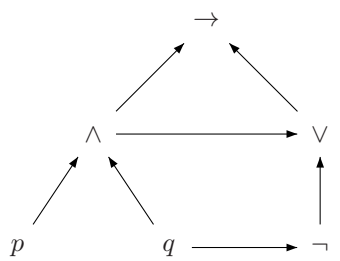
- $p \rightarrow q$ e $\neg q \vee p$ são subfórmulas imediatas de α em α;
- p e q são subfórmulas imediatas de $p \rightarrow q$ em α;
- $\neg q$ e p são subfórmulas imediatas de $\neg q \vee p$ em α; e
- q é subfórmula imediata de $\neg q$ em α. □

Observe que p é subfórmula imediata à esquerda de $p \rightarrow q$ e é uma subfórmula imediata à direita de $\neg q \vee p$. □

O par $(subf(\alpha), si(\alpha))$ é um grafo dirigido acíclico, aqui denominado *o grafo de α*.



(a) O grafo da fórmula.



(b) Forma simplicada do grafo da fórmula.

Figura 2.4: Grafo de $(p \wedge q) \rightarrow ((p \wedge q) \vee \neg q)$.

Definição 6 *O* grafo de uma fórmula α *é o grafo dirigido que tem* $subf(\alpha)$ *como seu conjunto de vértices e* $si(\alpha)$ *como seu conjunto de arestas, sendo que cada aresta* (β, γ) *tem associada a informação se* β *é subfórmula imediata à esquerda ou à direita de* γ*, quando pertinente.* □

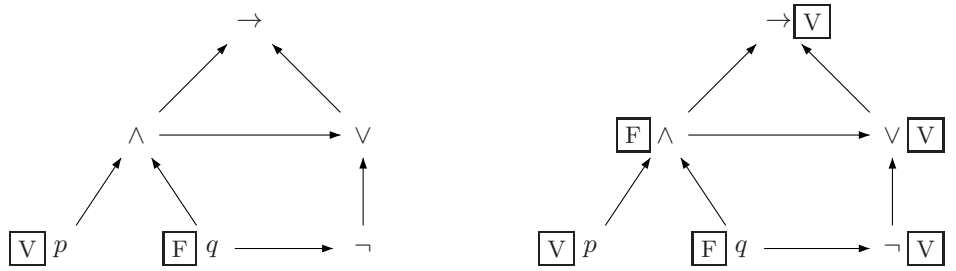
Deve-se notar que α é o único vértice do grafo de α que não tem sucessor: não existe β tal que $(\alpha, \beta) \in si(\alpha)$. Além disso, se $\alpha \in \mathcal{V} \cup \{\top, \bot\}$, α não é sucessor de qualquer outro vértice: não existe β tal que $(\beta, \alpha) \in si(\alpha)$. O próximo exemplo mostra o desenho do grafo de uma fórmula.

Exemplo 12 O grafo da fórmula $(p \wedge q) \rightarrow ((p \wedge q) \vee \neg q)$ está desenhado na Figura 2.4(a). A informação de que uma aresta representa uma subfórmula imediata à esquerda é colocada desenhando-a à esquerda daquela que representa uma subfórmula imediata à direita da mesma fórmula. □

Uma forma simplificada de desenhar o grafo de uma fórmula é explicitar apenas o conectivo principal de cada subfórmula, visto que a subfórmula pode ser recuperada dos subgrafos antecessores, como mostra o exemplo a seguir.

Exemplo 13 A Figura 2.4(b) mostra o desenho simplificado do grafo da fórmula $(p \wedge q) \rightarrow ((p \wedge q) \vee \neg q)$, que fora desenhado na Figura 2.4(a). Veja que a subfórmula que constitui o vértice rotulado com $\wedge$ é a concatenação da que constitui o vértice do antecessor imediato da esquerda, com $\wedge$, com a subfórmula que constitui o vértice do antecessor imediato da direita: $p \wedge q$. As outras subfórmulas são obtidas, uma de cada vértice, de forma similar. □

No método da tabela da verdade, o valor $v^i(\alpha)$ é obtido a partir dos valores das variáveis de α propagando-se estes últimos de tal forma que $v^i(\beta)$, para $\beta \in subf(\alpha)$, é determinado a partir das subfórmulas imediatas de β. A seguir é mostrado um exemplo usando o grafo da fórmula, em vez da notação tabular, de forma a se ter uma visão abrangente de todas as subfórmulas envolvidas e seus interrelacionamentos.



(a) O grafo com valores anotados para as variáveis. (b) O grafo com valores propagados.

Figura 2.5: Propagação de valores no grafo de $(p \wedge q) \rightarrow ((p \wedge q) \vee \neg q)$.

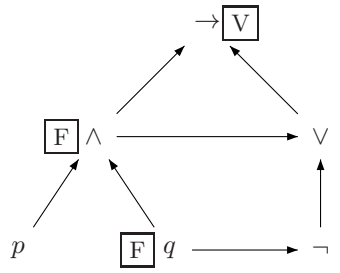
Exemplo 14 Seja novamente o grafo simplificado da fórmula $(p \wedge q) \rightarrow ((p \wedge q) \vee \neg q)$, que está desenhado na Figura 2.4(b). Na Figura 2.5(a) estão mostrados os valores lógicos $p^i = V$ e $q^i = F$ para as duas variáveis da fórmula, anotados ao lado dos vértices correspondentes, dentro de quadrados. Na Figura 2.5(b) estão mostrados os valores lógicos propagados, obtidos assim:

- como $v^i(p) = V$ e $v^i(q) = F$, $v^i(p \wedge q) = F$ e, portanto, a subfórmula $p \wedge q$ recebe a anotação $\boxed{\text{F}}$;
- como $v^i(q) = F$, $v^i(\neg q) = V$, e $\neg q$ recebe a anotação $\boxed{\text{V}}$;
- como $v^i(p \wedge q) = F$ e $v^i(\neg q) = V$, $v^i((p \wedge q) \vee \neg q) = V$ e, assim, a subfórmula $(p \wedge q) \vee \neg q$ recebe a anotação $\boxed{\text{V}}$;
- finalmente, $(p \wedge q) \rightarrow ((p \wedge q) \vee \neg q)$ recebe a anotação $\boxed{\text{V}}$, visto que $v^i(p \wedge q) = F$ e $v^i((p \wedge q) \vee \neg q) = V$.

Conclui-se que $v^i((p \wedge q) \rightarrow ((p \wedge q) \vee \neg q)) = V$ para $p^i = V$ e $q^i = F$. □

Algumas vezes, é possível fazer a propagação de valores lógicos de forma que $v^i(\alpha)$ seja determinado sem a necessidade de determinar valores para todas as subfórmulas; algumas vezes, inclusive, sem utilizar valores para todas as variáveis de α. Isso se deve ao fato de que o resultado da aplicação de uma função booleana algumas vezes pode ser determinado ao se conhecer apenas um dos argumentos (neste caso, não havendo a necessidade de avaliar o outro). Por exemplo, quaisquer que sejam as fórmulas α e β, se $v^i(\alpha) = V$, então $v^i(\alpha \vee \beta) = V$, não havendo, portanto a necessidade de determinar $v^i(\beta)$. Veja o exemplo a seguir.

Exemplo 15 Considerando o grafo da Figura 2.5, veja que o mesmo resultado pode ser obtido propagando-se apenas o valor $v^i(q) = F$:



já que $v^i(p \wedge q) = F$ se $v^i(q) = F$, e $v^i((p \wedge q) \rightarrow ((p \wedge q) \vee \neg q)) = V$ se $v^i(p \wedge q) = F$. □

O Exercício 4 pede a construção de um algoritmo para avaliação de uma fórmula para uma certa interpretação utilizando o método esboçado nos últimos exemplos.

Assim como em português, existem inúmeras formas de se dizer a mesma coisa na linguagem da lógica proposicional. Para ver isto, basta considerar que os significados de α e $\neg\neg\alpha$ são o mesmo, qualquer que seja a fórmula α: $v^i(\alpha) = v^i(\neg\neg\alpha)$ para toda i. Para fixar esse conceito de "fórmulas com o mesmo significado", será usado o conceito de "equivalência lógica", definido na próxima seção.

Exercícios

1. Para cada fórmula a seguir, determine seu conjunto de subfórmulas:

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

b) $(\neg p \rightarrow \neg\neg p) \rightarrow p$

c) $((p \rightarrow q) \wedge q) \rightarrow p$

d) $((p \rightarrow q) \vee r) \rightarrow (\neg q \wedge (p \vee r))$

2. Desenhe o grafo de cada fórmula do Exercício 1.

3. Para cada grafo do exercício anterior, avalie a fórmula respectiva para a interpretação i tal que $p^i = V$, $q^i = F$ e $r^i = V$. Faça como no Exemplo 15, apenas para a sequência que propicie um mínimo de anotações V/F para os vértices. Além disso, não faça anotações desnecessárias.

4. Faça um algoritmo que avalie uma fórmula para uma dada interpretação utilizando o método *bottom-up* da tabela da verdade tomando como base o grafo da fórmula, como foi feito nos Exemplos 14 e 15.

2.3.3 Equivalência lógica

Duas fórmulas podem ter exatamente o mesmo "significado", por mais díspares que forem sintaticamente. A definição a seguir captura o que vem a ser fórmulas de mesmo significado por meio do conceito de equivalência lógica.

Definição 7 *Duas fórmulas α e β são* logicamente equivalentes *se, e somente se, $v^i(\alpha) = v^i(\beta)$ para toda i.* □

Será usada a notação $\alpha \equiv \beta$ para dizer que α e β são logicamente equivalentes. Note que "$\equiv$" não é um conectivo e "$\alpha \equiv \beta$" não é uma fórmula. A expressão "$\alpha \equiv \beta$" é apenas uma forma abreviada de dizer "a fórmula α é logicamente equivalente à fórmula β".

Exemplo 16 Observe como as colunas relativas às fórmulas $\neg p$ e $p \to \bot$ são idênticas na seguinte tabela da verdade:

p	$\bot$	$\neg p$	$p \to \bot$
V	F	F	F
F	F	V	V

Isso prova que $v^i(\neg p) = v^i(p \to \bot)$ para toda i. Conclui-se, portanto, que $\neg p \equiv p \to \bot$.

Exemplo 17 $p \to q \equiv \neg p \vee q$. Para ver isso, basta avaliar $v^i(p \to q)$ e $v^i(\neg p \vee q)$ para cada i possível. Usando tabela da verdade:

p	q	$p \to q$	$\neg p$	$\neg p \vee q$
V	V	V	F	V
V	F	F	F	F
F	V	V	V	V
F	F	V	V	V

Como as colunas relativas a $v^i(p \to q)$ e $v^i(\neg p \vee q)$ são idênticas, $v^i(p \to q) = v^i(\neg p \vee q)$ para cada i. Portanto, $p \to q \equiv \neg p \vee q$. □

O teorema a seguir permite generalizar resultados como os dos exemplos anteriores: na verdade, $\neg\alpha \equiv \alpha \to \bot$ para qualquer fórmula α, não apenas para $\alpha = p$; e $\alpha \to \beta \equiv \neg\alpha \vee \beta$ para quaisquer fórmulas α e β, não apenas para $\alpha = p$ e $\beta = q$. De forma geral: se duas fórmulas são logicamente equivalentes, se forem aplicadas as mesmas substituições de variáveis por fórmulas a ambas, elas continuam logicamente equivalentes. Antes de enunciar o teorema, será apresentada a definição de substituição de *toda* ocorrência de uma subfórmula por outra.

Definição 8 *A* substituição *de uma subfórmula σ por uma fórmula φ em α, $\alpha[\sigma/\varphi]$, pode ser assim definida:*

a) se $\sigma \notin subf(\alpha)$, então $\alpha[\sigma/\varphi] = \alpha$;

b) se $\sigma \in subf(\alpha)$, então

- *se $\alpha = \sigma$, então $\alpha[\sigma/\varphi] = \varphi$;*
- *se $\alpha = (\neg\beta)$, então $\alpha[\sigma/\varphi] = (\neg\beta[\sigma/\varphi])$,*
- *se $\alpha = (\beta c \gamma)$, em que $c \in \{\wedge, \vee, \to, \leftrightarrow\}$, $\alpha[\sigma/\varphi] = (\beta[\sigma/\varphi] c \gamma[\sigma/\varphi])$.* □

O próximo exemplo apresenta algumas substituições.

Exemplo 18 Seja $\alpha = (p \to q) \wedge (\neg p \vee (q \leftrightarrow \neg r))$. Então:

- $\alpha[\neg q/p \vee r] = \alpha$;
- $\alpha[p \to q/\neg r] = \neg r \wedge (\neg p \vee (q \leftrightarrow \neg r))$;
- $\alpha[q \leftrightarrow \neg r/q] = (p \to q) \wedge (\neg p \vee q)$;
- $\alpha[p/r \vee \neg q] = ((r \vee \neg q) \to q) \wedge (\neg(r \vee \neg q) \vee (q \leftrightarrow \neg r))$. □

Segue o teorema aludido anteriormente, sem demonstração.

Teorema 1 *Sejam $\alpha, \beta, \varphi \in \mathcal{F}$ e $\nu \in \mathcal{V}$. Se $\alpha \equiv \beta$, então $\alpha[\nu/\varphi] \equiv \beta[\nu/\varphi]$.*

A veracidade de tal teorema é bastante intuitiva se for considerado que *todas* as ocorrências de ν são substituídas pela *mesma* fórmula φ e que o valor lógico de todas as ocorrências de uma variável é o mesmo para a mesma interpretação i. Assim, dado que $v^i(\alpha) = v^i(\beta)$ independente do valor de $v^i(\nu)$, então $v^i(\alpha[\nu/\varphi]) = v^i(\beta[\nu/\varphi])$ independente do valor de $v^i(\varphi)$.

Exemplo 19 No exemplo anterior, mostrou-se que $p \to q \equiv \neg p \vee q$. O Teorema 1 permite concluir que para quaisquer fórmula α, $(p \to q)[p/\alpha] \equiv (\neg p \vee q)[p/\alpha]$, ou seja, $\alpha \to q \equiv \neg\alpha \vee q$. Pelo mesmo teorema, tem-se então que para qualquer fórmula β, $(\alpha \to q)[q/\beta] \equiv (\neg\alpha \vee q)[q/\beta]$, ou seja, $\alpha \to \beta \equiv \neg\alpha \vee \beta$ para quaisquer $\alpha, \beta \in \mathcal{F}$. Em particular, fazendo-se $\alpha = (p \to q)$ e $\beta = \neg p$, tem-se que $(p \to q) \to \neg p \equiv \neg(p \to q) \vee \neg p$. Aplicando-se novamente o Teorema 1, este último resultado pode ser generalizado para $(\alpha \to \beta) \to \neg\alpha \equiv \neg(\alpha \to \beta) \vee \neg\alpha$. □

Na Figura 2.6 estão exibidas algumas equivalências que podem ser demonstradas facilmente por meio de tabelas da verdade e aplicação do Teorema 1, como feito nos Exemplos 17 e 19 para $\alpha \to \beta \equiv \neg\alpha \vee \beta$.

Como era de se esperar, fórmulas logicamente equivalentes podem ser substituídas umas pelas outras sem mudança de significado, como mostrado a seguir.

Teorema 2 *Sejam $\alpha, \beta, \gamma \in \mathcal{F}$ e α' o resultado de substituir em α uma ou mais ocorrências de β por γ. Se $\beta \equiv \gamma$, então $\alpha \equiv \alpha'$.*

Prova

Suponha que $\beta \equiv \gamma$, ou seja, $v^i(\beta) = v^i(\gamma)$ para toda i. Será provado por indução sobre o número de conectivos de α que $\alpha \equiv \alpha'$. Seja $n \geq 0$ e suponha, como hipótese de indução, que $\alpha \equiv \alpha'$ para α com menos de n conectivos. No caso em que $n = 0$, $\alpha \in \mathcal{V}$. Neste caso, se $\alpha = \beta$, então $\alpha' = \gamma$ e, assim, para qualquer i $v^i(\alpha') = v^i(\gamma) = v^i(\beta) = v^i(\alpha)$; logo, $\alpha \equiv \alpha'$. E se $\alpha \neq \beta$, $\alpha' = \alpha$ e, assim, $\alpha' \equiv \alpha$. O caso em que $n > 0$ pode ser divido nos subcasos:

$$\neg\top \equiv \bot \qquad \neg\bot \equiv \top$$
$$\alpha \wedge \top \equiv \alpha \qquad \alpha \vee \bot \equiv \alpha$$
$$\alpha \wedge \bot \equiv \bot \qquad \alpha \vee \top \equiv \top$$
$$\alpha \wedge \alpha \equiv \alpha \qquad \alpha \vee \alpha \equiv \alpha$$
$$\alpha \wedge \neg\alpha \equiv \bot \qquad \alpha \vee \neg\alpha \equiv \top$$
$$\alpha \wedge \beta \equiv \beta \wedge \alpha \qquad \alpha \vee \beta \equiv \beta \vee \alpha$$
$$(\alpha \wedge \beta) \vee \alpha \equiv \alpha \qquad (\alpha \vee \beta) \wedge \alpha \equiv \alpha$$
$$\alpha \wedge (\alpha \vee \beta) \equiv \alpha \qquad \alpha \vee (\alpha \wedge \beta) \equiv \alpha$$
$$\alpha \wedge (\beta \wedge \gamma) \equiv (\alpha \wedge \beta) \wedge \gamma \qquad \alpha \vee (\beta \vee \gamma) \equiv (\alpha \vee \beta) \vee \gamma$$
$$\alpha \wedge (\beta \vee \gamma) \equiv (\alpha \wedge \beta) \vee (\alpha \wedge \gamma) \qquad \alpha \vee (\beta \wedge \gamma) \equiv (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$$
$$\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta \qquad \neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$$
$$\neg\neg\alpha \equiv \alpha$$
$$\alpha \to \beta \equiv \neg\beta \to \neg\alpha$$
$$\alpha \to \beta \equiv \neg\alpha \vee \beta$$
$$\neg(\alpha \to \beta) \equiv \alpha \wedge \neg\beta$$
$$\alpha \leftrightarrow \beta \equiv (\alpha \to \beta) \wedge (\beta \to \alpha)$$
$$\alpha \leftrightarrow \beta \equiv (\alpha \wedge \beta) \vee (\neg\alpha \wedge \neg\beta)$$
$$\neg(\alpha \leftrightarrow \beta) \equiv (\alpha \wedge \neg\beta) \vee (\neg\alpha \wedge \beta)$$
$$\neg(\alpha \leftrightarrow \beta) \equiv (\alpha \vee \beta) \wedge (\neg\alpha \vee \neg\beta)$$

Figura 2.6: Algumas equivalências básicas.

1. $\alpha = \top$ ou $\alpha = \bot$. Este caso é similar àquele em que α é variável ($n = 0$).

2. $\alpha = \neg\phi$. Então $\alpha' = \neg\phi'$, em que ϕ' recebe as substituições de β por γ. Pela hipótese de indução, $\phi \equiv \phi'$. Segue-se que, para qualquer i, $v^i(\alpha) = \ominus v^i(\phi) = \ominus v^i(\phi') = v^i(\neg\phi') = v^i(\alpha')$. Logo, $\alpha' \equiv \alpha$.

3. $\alpha = \phi c \delta$, sendo c um dos conectivos binários. Então $\alpha' = \phi' c \delta'$, em que ϕ' e/ou δ' recebem substituições de β por γ. Pela hipótese de indução, $\phi \equiv \phi'$ e $\delta \equiv \delta'$. O resultado é imediato, já que para qualquer i $v^i(\alpha) = v^i(\phi) \textcircled{c} v^i(\delta) = v^i(\phi') \textcircled{c} v^i(\delta') = v^i(\phi' c \delta') = v^i(\alpha')$. □

Munido de um arsenal de equivalências como as da Figura 2.6, e outras que se pode demonstrar mediante o uso de tabela da verdade e do Teorema 1, pode-se demonstrar novas equivalências aplicando-se o Teorema 2, como exemplificado a seguir. Evidentemente, essas novas equivalências podem, também, ser usadas na demonstração de outras. À direita em cada linha das demonstrações é anotada, entre colchetes, a equivalência na qual ela se baseia.

Exemplo 20 Segue uma demonstração de $(p \to q) \to p \equiv p$:

$$\begin{aligned}(p \to q) \to p &\equiv \neg(p \to q) \vee p && [\alpha \to \beta \equiv \neg\alpha \vee \beta]\\ &\equiv (p \wedge \neg q) \vee p && [\neg(\alpha \to \beta) \equiv \alpha \wedge \neg\beta]\\ &\equiv p. && [(\alpha \wedge \beta) \vee \alpha \equiv \alpha]\end{aligned}$$

Usando-se o Teorema 1, pode-se concluir, portanto, que $(\alpha \to \beta) \to \alpha \equiv \alpha$ para quaisquer fórmulas α e β. □

A seguir, mais um exemplo.

Exemplo 21 Segue uma demonstração de $\alpha \to (\beta \to \alpha) \equiv \top$ para quaisquer fórmulas α e β:

$$\begin{array}{lll} \alpha \to (\beta \to \alpha) \equiv \neg\alpha \vee (\neg\beta \vee \alpha) & [\gamma \to \delta \equiv \neg\gamma \vee \delta, \text{ 2 vezes}] \\ \equiv (\neg\beta \vee \alpha) \vee \neg\alpha & [\gamma \vee \delta \equiv \delta \vee \gamma] \\ \equiv \neg\beta \vee (\alpha \vee \neg\alpha) & [(\gamma \vee \delta) \vee \phi \equiv \gamma \vee (\delta \vee \phi)] \\ \equiv \neg\beta \vee \top & [\gamma \vee \neg\gamma \equiv \top] \\ \equiv \top. & [\gamma \vee \top \equiv \top] \end{array}$$

□

Nas onze primeiras linhas da Figura 2.6, as fórmulas da esquerda do símbolo de equivalência da primeira e segunda colunas são o que se denomina fórmulas *duais*, assim como as fórmulas da direita do símbolo de equivalência da primeira e segunda colunas. Veja definição a seguir.

Definição 9 *Seja α uma fórmula que, caso tenha conectivos, os mesmos estão restritos ao conjunto $\{\top, \bot, \neg, \wedge, \vee\}$. Então a* dual *de α, α^d, é obtida substituindo-se $\top$ por $\bot$, e vice-versa, e $\wedge$ por $\vee$ e vice-versa. Recursivamente::*

- $\nu^d = \nu$, *se ν é variável proposicional;*
- $\top^d = \bot$;
- $\bot^d = \top$;
- $(\neg\alpha)^d = \neg\alpha^d$;
- $(\alpha \wedge \beta)^d = \alpha^d \vee \beta^d$;
- $(\alpha \vee \beta)^d = \alpha^d \wedge \beta^d$. □

O *princípio de dualidade*, apresentado aqui sem demonstração, diz que se $\alpha \equiv \beta$, então $\alpha^d \equiv \beta^d$.

Exemplo 22 Sabendo-se que $(p \vee q) \wedge \neg(p \wedge q) \equiv (\neg p \wedge q) \vee (p \wedge \neg q)$, pode-se concluir que $[(p \vee q) \wedge \neg(p \wedge q)]^d \equiv [(\neg p \wedge q) \vee (p \wedge \neg q)]^d$, ou seja, $(p \wedge q) \vee \neg(p \vee q) \equiv (\neg p \vee q) \wedge (p \vee \neg q)$. □

Na próxima subseção serão introduzidos outros conceitos importantes no nível semântico, conceitos esses definidos, assim como o de equivalência lógica, a partir da interpretação formal de fórmulas (dada por v^i).

Exercícios

1. Mostre, a partir da definição de equivalência lógica, que:

a) $p \leftrightarrow q \equiv (p \wedge q) \vee (\neg p \wedge \neg q)$
b) $p \leftrightarrow q \equiv (\neg p \vee q) \wedge (p \vee \neg q)$
c) $\neg(p \leftrightarrow q) \equiv (\neg p \wedge q) \vee (p \wedge \neg q)$
d) $\neg(p \leftrightarrow q) \equiv (p \vee q) \wedge (\neg p \vee \neg q)$
e) $p \leftrightarrow (p \rightarrow q) \equiv (p \wedge q)$

2. Mostre que são logicamente equivalentes utilizando equivalências mais simples, como no Exemplo 20:

a) $(p \rightarrow \bot) \vee (p \rightarrow \top)$ e $\top$
b) $(p \rightarrow q) \wedge p \wedge \neg q$ e $\bot$
c) $p \vee q$ e $\neg(\neg p \wedge \neg q)$
d) $(p \rightarrow q) \rightarrow r$ e $(\neg p \rightarrow r) \wedge (q \rightarrow r)$
e) $\neg(p \leftrightarrow q)$ e $(p \vee q) \wedge \neg(p \wedge q)$

Use apenas equivalências mostradas na Figura 2.6, se possível.

3. Diga quais são a duais de:

a) $\neg(\bot \wedge \top) \vee \neg p$
b) $\neg(p \wedge q) \vee (\neg p \wedge q)$
c) $(\neg p \vee \bot) \wedge \neg(p \vee q)$

4. Mostre que se $\alpha \equiv \beta$, então $\alpha^d \equiv \beta^d$.

2.3.4 Satisfabilidade, tautologias e contradições

> *To say of what is that it is not, or of what is not that it is, is false, while to say of what is that it is, and of what is not that it is not, is true. (Aristóteles, 384 AC–322 AC)*

Nesta subseção serão tratados alguns conceitos extremamente importantes, não apenas na versão proposicional da lógica, mas também na versão com predicados.

Definição 10 *Uma interpretação i é um* modelo *para uma fórmula α se, e somente se, $v^i(\alpha) = V$. E i é um modelo para um conjunto de fórmulas H se, e somente se, é modelo para toda $\alpha \in H$.* □

Assim, os modelos para uma fórmula correspondem às linhas da tabela da verdade para as quais ela é verdadeira. Em particular, qualquer i é modelo para $\top$ e nenhum é modelo para $\bot$. No caso de um conjunto de fórmulas H, se ele for finito i será um modelo para H se, e somente se, for um modelo para a conjunção das fórmulas em H.

Exemplo 23 Sejam as variáveis proposicionais e respectivas proposições:

- *lf*: *lógica é fácil*;
- *mf*: *matemática é fácil*;
- *gl*: *muitos gostam de lógica*.

Seja H o conjunto das três fórmulas:

- gl
  (*muitos gostam de lógica*);
- $mf \rightarrow lf$
  (*se matemática é fácil, lógica é fácil*);
- $\neg lf \vee \neg gl$
  (*lógica não é fácil ou não é verdade que muitos gostam de lógica*).

A seguinte tabela da verdade mostra na última coluna a conjunção das três fórmulas de H:

lf	mf	gl	$\neg lf$	$\neg gl$	$mf \rightarrow lf$	$\neg lf \vee \neg gl$	$gl \wedge (mf \rightarrow lf) \wedge (\neg lf \vee \neg gl)$
V	V	V	F	F	V	F	F
V	V	F	F	V	V	V	F
V	F	V	F	F	V	F	F
V	F	F	F	V	V	V	F
F	V	V	V	F	F	V	F
F	V	F	V	V	F	V	F
$\boxed{F}$	$\boxed{F}$	$\boxed{V}$	V	F	V	V	$\boxed{V}$
F	F	F	V	V	V	V	F

Note que há uma única linha (interpretação i) para a qual as três fórmulas de H são verdadeiras e, consequentemente, a conjunção das três também é verdadeira: a penúltima. Nessa linha estão marcados os valores para *lf*, *mf* e *gl*. Vê-se, então, que todo modelo i para H deve ser tal que:

- $lf^i = F$: *lógica não é fácil*;
- $mf^i = F$: *matemática não é fácil*;
- $gl^i = V$: *muitos gostam de lógica*. □

Uma outra forma de dizer que α tem modelo é dizer que α é *satisfatível*.

Definição 11 *Uma fórmula α é* satisfatível *se, e somente se, α tem um modelo. Um conjunto de fórmulas que tenha um modelo também é dito* satisfatível. *E se α não é satisfatível (não tem modelo), diz-se que é* insatisfatível *(ou uma* contradição*). E um conjunto de fórmulas que não tenha um modelo também é dito* insatisfatível. □

Logo, pela definição anterior, uma fórmula é satisfatível se, e somente se $v^i(\alpha) = V$ para alguma i; e é insatisfatível (uma contradição) se, e somente se, $v^i(\alpha) = F$ para toda i. Assim, em particular, $\bot$ é insatisfatível. Observe que o conjunto $\mathcal{F}$ de todas as fórmulas é particionado nos dois conjuntos: o conjunto das fórmulas satisfatíveis e o das contradições.

Ainda, pela Definição 11, um conjunto de fórmulas H é satisfatível se, e somente se, existe i tal que $v^i(\alpha) = V$ para toda $\alpha \in H$; e é insatisfatível (contraditório) se, e somente se, para toda i existe $\alpha \in H$ tal que $v^i(\alpha) = F$. Note que $\emptyset$ é satisfatível por vacuidade.

Exemplo 24 O conjunto H do Exemplo 23 é satisfatível, como mostra a sétima linha da tabela da verdade lá costruída. Considere a fórmula $\beta = gl \wedge (mf \rightarrow lf) \wedge (\neg lf \vee \neg gl)$ que expressa a conjunção das três fórmulas de H, e $\gamma = \beta \wedge mf$. No exemplo anterior ficou constatado que $v^i(\beta) = V$ somente se $mf^i = F$. Logo, $v^i(\beta \wedge mf) = F$ para qualquer i e, portanto, γ é insatisfatível. Naturalmente, o conjunto $H \cup \{mf\}$ é insatisfatível. □

Se i é um modelo para uma fórmula (ou conjunto de fórmulas), diz-se também que i *satisfaz* a fórmula (ou conjunto de fórmulas).

Como uma fórmula α é uma contradição se, e somente se, $v^i(\alpha) = F$ para toda i, segue-se que todas as contradições são logicamente equivalentes. Em particular, para qualquer contradição α, $\alpha \equiv \bot$.

Definição 12 *Uma fórmula α é* falseável *se, e somente se, $v^i(\alpha) = F$ para alguma i. E se α não é falseável, diz-se que é* tautológica *(ou uma* tautologia*).* □

Assim, pela Definição 12, α é uma tautologia se, e somente se, $v^i(\alpha) = V$ para toda i. Com isso, por exemplo, $\top$ é uma tautologia. Dada a noção de equivalência lógica, segue-se que todas as tautologias são logicamente equivalentes. Em particular, $\alpha \equiv \top$ para qualquer tautologia α. Observe que o conjunto $\mathcal{F}$ de todas as fórmulas é particionado nos dois conjuntos: o conjunto das tautologias e das fórmulas falseáveis.

Exemplo 25 Considere novamente a fórmula $\beta = gl \wedge (mf \rightarrow lf) \wedge (\neg lf \vee \neg gl)$ que expressa a conjunção das três fórmulas de H do Exemplo 23, e seja $\delta = \beta \rightarrow \neg mf$. No Exemplo 23 ficou constatado que $v^i(\beta) = V$ somente se $mf^i = F$. Com isto, não se pode ter $v^i(\beta) = V$ e $v^i(\neg mf) = F$, pois neste último caso ter-se-ia $mf^i = V$. Logo, $v^i(\delta) = V$ para qualquer i e, portanto, δ é uma tautologia (é tautológica). □

O teorema a seguir expõe uma relação importante entre $\equiv$ e o conectivo $\leftrightarrow$.

Teorema 3 *$\alpha \equiv \beta$ se, e somente se, $\alpha \leftrightarrow \beta$ é uma tautologia.*

Prova

Por definição, $\alpha \equiv \beta$ se, e somente se, $v^i(\alpha) = v^i(\beta)$ para toda i. E $\alpha \leftrightarrow \beta$ é uma tautologia se, e somente se $v^i(\alpha \leftrightarrow \beta) = V$ para toda i, ou seja, por definição, $v^i(\alpha) \leftrightarrows v^i(\beta) = V$ para toda i, ou ainda, $v^i(\alpha) = v^i(\beta)$ para toda i. □

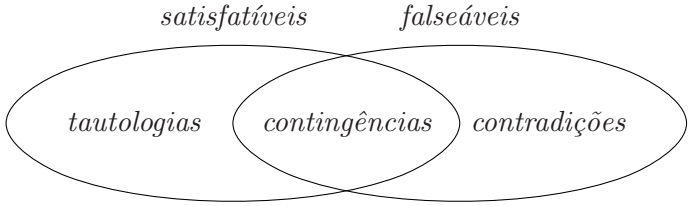


Figura 2.7: Relacionamentos entre tipos de fórmulas.

Assim, para mostrar que $\neg p \vee q \equiv p \rightarrow q$, basta mostrar que $(\neg p \vee q) \leftrightarrow (p \rightarrow q)$ é uma tautologia, o que pode ser feito usando-se a tabela da verdade, de forma similar à empregada no Exemplo 17, só que com uma coluna a mais para a fórmula $(\neg p \vee q) \leftrightarrow (p \rightarrow q)$: sendo as colunas para as subfórmulas $\neg p \vee q$ e $p \rightarrow q$ idênticas, segue-se que a coluna para $(\neg p \vee q) \leftrightarrow (p \rightarrow q)$ contém apenas Vs: a fórmula é uma tautologia; portanto, $\neg p \vee q \equiv p \rightarrow q$.

Para completar o quadro da terminologia relativa aos vários tipos de fórmulas, define-se o que é uma contingência.

Definição 13 *Uma fórmula α é uma* contingência *se e somente se α é satisfatível e falseável.* □

Ou seja, α é uma contingência se, e somente se, não é tautologia (visto que é falseável) e não é contradição (visto que é satisfatível).

A Figura 2.7 apresenta esquematicamente o relacionamento entre as cinco classes de fórmulas definidas anteriormente. Na figura ficam bem claras as relações existentes entre as várias classes de fórmulas, como, por exemplo:

- $\mathcal{F}$ é particionada em três classes: tautologias, contingências e contradições.
- $\mathcal{F}$ é particionada em duas classes: tautologias e fórmulas falseáveis.
- $\mathcal{F}$ é particionada em duas classes: contradições e fórmulas satisfatíveis.
- A classe das contingências é a interseção entre a classe das satisfatíveis e a das falseáveis. Ou ainda: é a classe das satisfatíveis subtraída da classe das tautologias. Ou ainda: é a classe das falseáveis subtraída da classe das contradições.

Considere o seguinte problema:

> *Suponha que α pertença a uma das cinco classes de fórmulas mencionadas na Figura 2.7. Neste caso, a que classe pertence $\neg\alpha$? E se $\neg\alpha$ pertencer a uma das cinco classes, a que classe pertencerá α?*

As respostas para tais perguntas seguem quase que diretamente das definições das classes:

- A negação de uma fórmula satisfatível dá uma falseável e vice-versa:

$$\alpha \text{ é satisfatível se e somente se } \neg\alpha \text{ é falseável} \tag{2.1}$$
$$\neg\alpha \text{ é satisfatível se e somente se } \alpha \text{ é falseável} \tag{2.2}$$

  Para ver a veracidade de 2.1, suponha que α seja satisfatível e seja i tal que $v^i(\alpha) = V$. Segue-se que $v^i(\neg\alpha) = \ominus v^i(\alpha) = \ominus V = F$ e, portanto, $\neg\alpha$ é falseável. Por outro lado, se $\neg\alpha$ é falseável, existe i tal que $v^i(\neg\alpha) = F$, seguindo-se que $v^i(\alpha) = V$ e, portanto, α é satisfatível.

  De 2.1 segue-se que $\neg\alpha$ é satisfatível se e somente se $\neg\neg\alpha$ é falseável, de onde se conclui 2.2.

- A negação de uma tautologia dá uma contradição e vice-versa:

$$\alpha \text{ é tautologia se e somente se } \neg\alpha \text{ é contradição}$$
$$\neg\alpha \text{ é tautologia se e somente se } \alpha \text{ é contradição.}$$

  A primeira:

  α é tautologia
  se e somente se α não é falseável (por definição)
  se e somente se $\neg\alpha$ não é satisfatível (por 2.2)
  se e somente se $\neg\alpha$ é contradição (por definição).

  A segunda:

  $\neg\alpha$ é tautologia
  se e somente se $\neg\alpha$ não é falseável (por definição)
  se e somente se α não é satisfatível (por 2.1)
  se e somente se α é contradição (por definição).

- A negação de uma contingência dá uma contingência:

$$\alpha \text{ é contingência se e somente se } \neg\alpha \text{ é contingência}$$

  pois:

  α é contingência
  se e somente se α é satisfatível e α é falseável (por definição)
  se e somente se $\neg\alpha$ é falseável e $\neg\alpha$ é satisfatível (por 2.1 e 2.2)
  se e somente se $\neg\alpha$ é contingência (por definição).

A Figura 2.8 mostra esquematicamente os relacionamentos expressos anteriormente entre fórmulas e suas negações.

O acréscimo ou retirada de uma fórmula de um conjunto pode fazer com que seu *status* de satisfatível ou não se modifique ou permaneça o mesmo. O seguinte teorema apresenta um exemplo. Outros são propostos como exercícios ao final da seção.

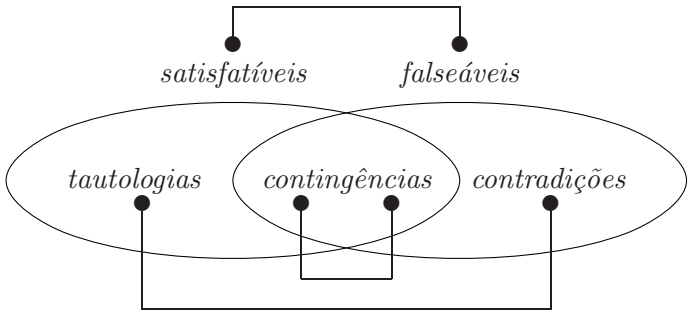


Figura 2.8: Relacionamentos entre fórmulas e suas negações.

Teorema 4 *Se H é um conjunto de fórmulas insatisfatível e α é uma tautologia, então $H - \{\alpha\}$ é insatisfatível.*

Prova

Suponha que H seja insatisfatível e α uma tautologia. Seja i uma interpretação arbitrária. Como H é insatisfatível, $v^i(\gamma) = F$ para alguma $\gamma \in H$. Visto que α é tautologia, $v^i(\alpha) = V$ e, portanto, $\gamma \neq \alpha$. Logo, $\gamma \in H - \{\alpha\}$. Assim, como há uma fórmula $\gamma \in H - \{\alpha\}$ tal que $v^i(\gamma) = F$ para i arbitrária, segue-se que $H - \{\alpha\}$ é insatisfatível. □

Considere agora os problemas de, dada uma fórmula, descobrir se ela pertence a uma dada classe. Dados os relacionamentos entre as classes, vê-se que os problemas podem todos ser reduzidos ao de verificar se uma fórmula pertence a uma das quatro classes: satisfatíveis, falseáveis ou as complementares a estas (contradições e tautologias). Por exemplo, eles podem ser reduzidos ao de verificar se uma fórmula é satisfatível:

- α é falseável se, e somente se, $\neg\alpha$ é satisfatível;
- α é contradição se, e somente se, α não é satisfatível;
- α é tautologia se, e somente se, $\neg\alpha$ não é satisfatível;
- α é contingência se, e somente se, α e $\neg\alpha$ são satisfatíveis.

Um exercício ao final da seção pede as reduções para as outras classes.

Antes de terminar esta seção, será apresentado um tipo de problema para o qual os conceitos aqui apresentados podem ser usados para análise e solução. Em seu livro *Logical Labyrinths*[4] Raymond Smullyan mostra como formalizar e solucionar problemas do tipo "*Knights and Knaves*". Tais problemas versam sobre os habitantes de uma ilha em que cada um deles, ou *sempre* diz a verdade (é um *knight*) ou *sempre* mente (é um *knave*). Em cada problema, deve-se descobrir quem fala a verdade e/ou quem mente a partir do que os habitantes falam. Um exemplo bem simples (adaptado do livro citado):

[4]Smullyan, R.M., *Logical Labyrinths*, A K Peters Ltd., 2009. É um livro que aborda as lógicas proposicional e de primeira ordem a partir de quebra-cabeças lógicos introduzidos por Smullyan em suas publicações anteriores.

> Um visitante se encontra com dois habitantes, A e B, da ilha. A afirma: "Ambos somos mentirosos". O problema é determinar para cada um, A e B, se é mentiroso ou não.

Smullyan propõe uma abordagem para esse tipo de problema como mostrada a seguir. Suponha que A_1, A_2, ..., A_n sejam os habitantes da ilha, $n \geq 2$. Sejam n variáveis proposicionais v_1, v_2, ..., v_n em que v_k denota que A_k sempre fala a verdade. Com isto, se A_k afirma uma proposição denotada por uma fórmula α, sabe-se que $v_k \leftrightarrow \alpha$ é verdadeira, ou seja, A_k sempre fala a verdade se, e somente se, o que ele falou é verdade. O problema, então, torna-se o de encontrar interpretações que satisfaçam (ou seja, modelos para) o conjunto de fórmulas bicondicionais assim obtidas. O problema costuma ser particulamente interessante quando o modelo tem alguma característica peculiar como, por exemplo, o de ser único. Segue um exemplo.

Exemplo 26 Seja o problema referido acima:

> Um visitante se encontra com dois habitantes, A e B, da ilha. A afirma: "Ambos somos mentirosos". Determinar se A e B são mentirosos ou não.

Usando a formalização sugerida anteriormente, o que A disse pode ser expresso por $\neg v_A \wedge \neg v_B$; já B não disse nada. A solução para o problema são, então, os modelos para a fórmula $v_A \leftrightarrow \neg v_A \wedge \neg v_B$. Tal modelo é único como mostra a tabela da verdade:

v_A	v_B	$\neg v_A$	$\neg v_B$	$\neg v_A \wedge \neg v_B$	$v_A \leftrightarrow \neg v_B \wedge \neg v_B$
V	V	F	F	F	F
V	F	F	V	F	F
F	V	V	F	F	V
F	F	V	V	V	F

Segue-se que o único modelo para $v_A \leftrightarrow \neg v_A \wedge \neg v_B$ é aquele em que $v_A^i = F$ e $v_B^i = V$. Em outras palavras, $(v_A \leftrightarrow \neg v_A \wedge \neg v_B) \rightarrow (\neg v_A \wedge v_B)$ é uma tautologia. Ou ainda, $(v_A \leftrightarrow \neg v_A \wedge \neg v_B) \wedge \neg(\neg v_A \wedge v_B)$ é insatisfatível. Logo, a resposta para o problema é: A mente e B fala a verdade. □

O Exercício 6 pede a solução de problemas adicionais do tipo exemplificado acima.

Para a solução automática de problemas como os mencionados acima, assim como de inúmeros outros (como ficará claro ao longo do texto), usa-se como base um algoritmo que seja capaz de determinar interpretações que satisfazem uma fórmula. O chamado *problema da satisfabilidade* é o de, dada uma fórmula, determinar se ela é satisfatível. Nos algoritmos usuais, para fazer essa determinação é construída, explícita ou implicitamente, uma interpretação específica que satisfaz a fórmula. Como visto anteriormente, os problemas de determinar se uma fórmula pertence a uma das classes exibidas na Figura 2.7 podem todos ser reduzidos ao da satisfabilidade. A seguir será mostrado como ele pode ser resolvido no nível conceitual.

Exercícios

1. Descubra, para cada fórmula abaixo, se ela é tautológica, contraditória ou contingente. Explique da maneira mais sucinta que puder a sua resposta:

a) $\neg(\top \wedge \neg\bot) \leftrightarrow (\top \rightarrow \bot \vee \top)$

b) $(p \rightarrow q) \wedge (\neg p \rightarrow q) \rightarrow q$

c) $(p \rightarrow q) \wedge \neg(q \rightarrow p)$

d) $(p \rightarrow q) \wedge \neg(\neg q \rightarrow \neg p)$

e) $\neg p \wedge (q \rightarrow p) \rightarrow \neg q$

2. Considere as fórmulas obtidas para o Exercício 2 da Seção 2.2. Construa um modelo para tal conjunto. Este modelo diz o que, caso se interprete as variáveis proposicionais de acordo com as proposições do Exercício 3 da Seção 2.1?

3. Explique como os problemas de determinar se uma fórmula pertence a cada uma das classes de fórmulas exibidas na Figura 2.7 pode ser reduzidos ao de determinar se uma fórmula é:

a) falseável;

b) tautologia;

c) contradição.

4. Mostre que sim ou que não:

a) Se $\alpha \wedge \beta$ é satisfatível, então β é satisfatível.

b) Se $\alpha \wedge \beta$ é tautologia, então β é tautologia.

c) Se $\alpha \vee \beta$ é satisfatível, então β é satisfatível.

d) Se $\alpha \vee \beta$ é tautologia, então β é satisfatível.

e) Se $\alpha \rightarrow \beta$ e α são tautologias, então β é tautologia.

f) Se $\alpha \rightarrow \beta$ é tautologia e α é satisfatível, então β é satisfatível.

g) Se $\alpha \rightarrow \beta$ e α são satisfatíveis, então β é satisfatível.

5. Suponha que (a) o conjunto de fórmulas H possa ser satisfatível ou insatisfatível, (b) a fórmula α possa ser tautologia, contradição ou ser uma fórmula qualquer, e (c) α deva ser acrescentada a H ou retirada de H. Para cada um dos 12 casos, diga se o conjunto resultante é satisfatível, insatisfatível ou não se sabe.

6. Solucione os seguintes problemas do tipo *Knights and Knaves*, de forma análoga ao que foi feito no Exemplo 26 (lembre-se: cada habitante, ou sempre diz a verdade, ou sempre mente):

a) Um visitante se encontra com dois habitantes, A e B. A afirma: "Pelo menos um de nós dois é mentiroso". Determinar se A e B são mentirosos ou não.

b) Um visitante se encontra com dois habitantes, A e B. A afirma: "Nós somos do mesmo tipo, ou seja, somos ambos mentirosos ou não mentirosos". Determinar se A e B são mentirosos ou não.

c) Um visitante se encontra com três habitantes, A, B e C. Ele pergunta a A: "Você mente ou não?". E não consegue entender a resposta de A. Então, pergunta a B: "O que A disse?". Ao que B responde: "Ele disse que mente". Aí C intervém e diz: "Não acredite em B, ele está mentindo!". Determinar se C é mentiroso ou não.

7. Em seu livro *The Lady or the Tiger?*[5] Raymond Smullyan formula problemas nos quais um rapaz (originalmente, um prisioneiro) deve escolher entre duas salas, uma que contém uma moça, e outra que contém um tigre. Se o rapaz escolher a moça, ele tem o direito de se casar com ela, e se escolher o tigre, o tigre tem o direito de comê-lo. Em um desses problemas, o rapaz deve escolher entre duas salas numeradas, I e II, em cujas portas estão gravadas as seguintes afirmativas:

Porta I: Nesta sala há uma moça, e na outra há um tigre.

Porta II: Em uma destas salas há uma moça, e na outra há um tigre.

Sabendo que uma das afirmativas é verdadeira e que a outra é falsa, em qual sala está a moça?

Assim como para os problemas anteriores, resolva este problema encontrando um modelo para certo conjunto de fórmulas.

2.3.5 Verificação de satisfabilidade e falseabilidade no nível conceitual

Os problemas de determinar se uma fórmula é satisfatível e de determinar se uma fórmula é falseável são sabidamente NP-completos, sendo que o problema da satisfabilidade foi o primeiro problema de decisão a ser provado NP-completo. A seguir, são apresentados alguns algoritmos *não determinísticos* de complexidade polinomial, assim como um algoritmo determinístico de complexidade exponencial, para o problema da satisfabilidade. Algoritmos análogos podem ser trivialmente obtidos para o problema da falseabilidade. Os algoritmos apresentados trabalham no nível conceitual. Em seções posteriores serão apresentados também algoritmos que trabalham no nível formal. O intuito é propiciar uma maior intimidade com o problema, dada a importância do mesmo de ambos os pontos de vista, prático e teórico. A abordagem mais utilizada na prática, particularmente para a resolução de problemas altamente complexos, será vista separadamente na Seção 2.6.

Uma primeira idéia para obter um algoritmo para determinar se uma fórmula α é satisfatível é testar cada uma das interpretações relativas às variáveis que ocorrem em α, via o algoritmo da Figura 2.3 ou um algoritmo baseado no método da tabela da verdade. Este último é proposto como exercício ao final da seção.

[5] Smullyan, R.M., *The Lady or the Tiger? And Other Logic Puzzles*, Times Books, 1982.

A seguir, são apresentados dois algoritmos não determinísticos que trabalham a partir do conectivo principal da fórmula em direção às variáveis. A verificação de satisfabilidade funciona através da "dissecação" da fórmula e ações apropriadas dependentes dos conectivos que vão sendo encontrados. Por exemplo, na tentativa de mostrar que $\alpha \wedge \beta$ é satisfatível, basta encontrar uma interpretação que satisfaça α **e** β; na tentativa de mostrar que $\alpha \vee \beta$ é satisfatível, basta encontrar uma interpretação que satisfaça α **ou** β; e assim por diante, assegurando a *mesma interpretação* para toda ocorrência de uma *mesma subfórmula*, o que é conseguido garantindo-se a mesma interpretação para toda ocorrência da *mesma variável*.

Na Figura 2.9 está um algoritmo, satA-nd, *não determinístico* que procura por uma interpretação que satisfaça uma fórmula. A variável s contém V se o objetivo é fazer a subfórmula atual, γ, valer V, ou contém F se o objetivo é fazê-la valer F. É utilizado $\overline{\mathsf{s}}$ para significar F, se o conteúdo de s for V, ou V, se o conteúdo for F. A variável A contém o conjunto de pares $\mathsf{s}\alpha$ tais que ainda falta fazer com que $v^i(\alpha)$ seja o valor em s, sendo inicializada com $\{V\gamma\}$, em que γ é a fórmula passada como argumento. A interpretação que vai sendo construída é colocada na variável Σ, cujos elementos são da forma $s\nu$, em que $s \in \{V, F\}$ e $\nu \in \mathcal{V}$, de tal forma que se $s\nu \in \Sigma$ então $\nu^i = s$. Com isto, ao final de uma computação de sucesso, Σ contém uma descrição de modelos para a fórmula. Assim, em particular, se Σ terminar com $\emptyset$, qualquer interpretação é modelo para a fórmula.

Daqui para frente, os pares da forma $s\alpha$, em que $s \in \{V, F\}$ e $\alpha \in \mathcal{F}$, serão denominados *fórmulas marcadas*; e caso α seja uma variável, $s\alpha$ será denominado também *variável marcada*. Em um dado instante, em satA-nd, se uma fórmula marcada $V\alpha \wedge \beta$, contida em A, for a escolhida via o pseudocomando escolha(A), ela é substituída pelas duas fórmulas marcadas $V\alpha$ e $V\beta$, significando que para fazer uma interpretação que satisfaça $\alpha \wedge \beta$ é preciso que ela satisfaça ambas, α e β. Se a fórmula marcada escolhida for $F\alpha \wedge \beta$, ela pode ser substituída, *não deterministicamente* por $F\alpha$ *ou* por $F\beta$, significando que para fazer uma interpretação que falseie $\alpha \wedge \beta$ é preciso que ela falseie, ou α ou β (qualquer uma das duas). As outras (sub)fórmulas marcadas são processadas de fórmula análoga.

Note que na escolha da próxima fórmula marcada em A, propiciada pelo pseudocomando escolha(A), tem-se um não determinismo especial: a satisfação ou não da fórmula independe da sequência de escolhas efetuadas, já que *todas* as fórmulas em A devem ser escolhidas e satisfeitas para que haja sucesso. Diferentemente desse não determinismo, existe aquele em que as escolhas podem redundar em sucesso ou fracasso. Este é o caso das escolhas expressas mediante o pseudomando escolha-nd(C), sendo C um conjunto. Aqui uma sequência de escolhas pode resultar em *sucesso* ou *fracasso* de uma computação! Está-se usando **proc-nd** para ressaltar que o procedimento é *não determinístico* (o não determinismo sendo propiciado pelo pseudocomando *escolha-nd*) no sentido de que: uma chamada é dita ter sucesso se, e somente se, **existe** uma computação que resulta em *sucesso*. Deve-se ressaltar que um procedimento não determinístico, aqui começado por **proc-nd**, só pode ser terminado com emissão de um dos pseudocomandos *sucesso* ou *fracasso*, diferentemente de um procedimento determinístico, começado

```
proc-nd satA-nd(fórmula γ):
    conjunto de fórmulas marcadas A;
    conjunto de variáveis marcadas Σ;
    elemento de {V, F} s;
    Σ = ∅;
    A := {Vγ};
    repita
        sγ := escolha(A); A := A − {sγ};
        caso sγ seja
          ∗variável:    se s̄γ ∈ Σ então fracasso senão Σ := Σ ∪ {sγ} fimse
          Vτ,Fτ:        { continue }
          Vτ,Fτ:        fracasso
          Vα:           A := A ∪ {Fα}
          Fα:           A := A ∪ {Vα}
          Vα ∧ β:       A := A ∪ {Vα, Vβ}
          Fα ∧ β:       A := A ∪ {escolha-nd({Fα, Fβ})} { escolha não determinística }
          Vα ∨ β:       A := A ∪ {escolha-nd({Vα, Vβ})} { escolha não determinística }
          Fα ∨ β:       A := A ∪ {Fα, Fβ}
          Vα → β:       A := A ∪ {escolha-nd({Fα, Vβ})} { escolha não determinística }
          Fα → β:       A := A ∪ {Vα, Fβ}
          Vα ↔ β:       A := A ∪ {escolha-nd({{Vα, Vβ}, {Fα, Fβ}})} { escolha não determinística }
          Fα ↔ β:       A := A ∪ {escolha-nd({{Vα, Fβ}, {Fα, Vβ}})} { escolha não determinística }
        fimcaso
    até A = ∅;
    sucesso
fim satA-nd
```

Figura 2.9: Satisfabilidade por construção de modelo.

por **proc**, que deve terminar via um **retorne**.[6]

Nos exemplos, as computações possíveis do algoritmo são mostradas por meio de uma *árvore de computação*, na qual as bifurcações correspondem a escolhas não determinísticas. Em geral, um vértice em uma árvore de computação está associado ao estado atual de execução do procedimento não determinístico (valores de todas as variáveis), sendo que a cada ramo da árvore[7] corresponde uma computação. Uma árvore de computação correspondente a uma chamada de satA-nd será apresentada exibindo-se em cada vértice os conteúdos de Σ e A, *sempre que um deles for modificado*, sendo que a raiz é rotulada com os valores iniciais de Σ e A. Se A contiver mais de um elemento, aquele escolhido por meio do comando escolha será sublinhado. Se um ramo corresponder a uma computação de fracasso, será escrito *fracasso* no final do ramo. No caso em que A atingir o valor $\emptyset$, Σ conterá a descrição de um modelo e a computação terminará com sucesso, sendo escrito *sucesso* no final do ramo. Na verdade, para cada sequência de escolhas via o comando escolha, há uma árvore de computação distinta. No entanto, como as soluções para as diversas árvores são as mesmas, nos exemplos

[6] Caso o leitor tenha dúvidas sobre este ponto, pode consultar, por exemplo, Horowitz, E., Sahni, S. *Fundamentals of Computer Algorithms*, Computer Science Press, 1984, pág. 502–510

[7] Um *ramo* de uma árvore é um caminho que inicia na raiz dela e termina em uma folha.

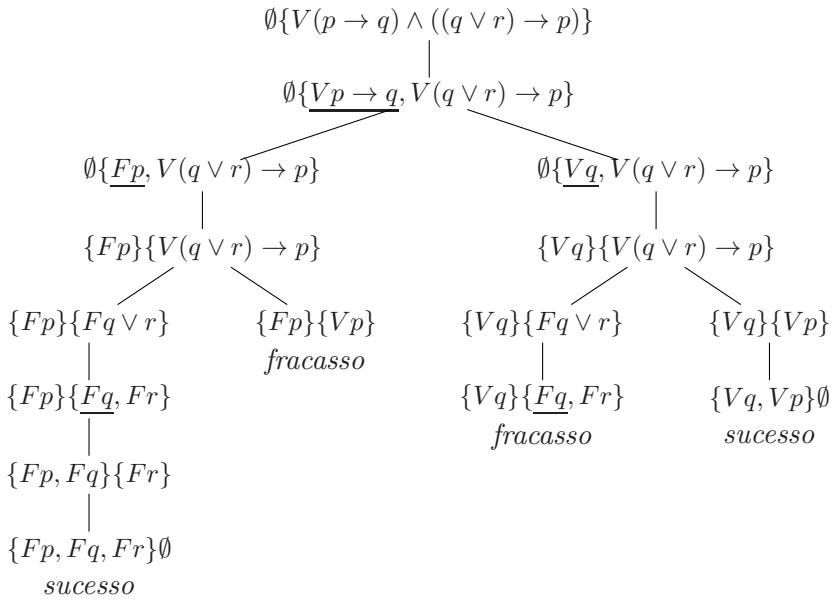


Figura 2.10: Árvore de computação para satA-nd$((p \to q) \wedge ((q \vee r) \to p))$.

será apresentada apenas uma das árvores.

Exemplo 27 Uma árvore de computação correspondente à chamada satA-nd$((p \to q) \wedge ((q \vee r) \to p))$, obtida como explicado anteriormente, está mostrada na Figura 2.10. Vê-se, então, que existem quatro computações possíveis, duas de sucesso e duas de fracasso. Os dois ramos da árvore que correspondem às de sucesso mostram as interpretações que satisfazem $(p \to q) \wedge ((q \vee r) \to p)$ no Σ final:

- $\Sigma = \{Fp, Fq, Fr\}$ implica que interpretações em que $p^i = q^i = r^i = F$ são modelos; e
- $\Sigma = \{Vp, Vq\}$ implica que interpretações em que $p^i = q^i = V$ são modelos. □

Um algoritmo não determinístico para testar se uma fórmula γ é falseável seria o próprio satA-nd, substituindo-se o pseudomando A $:= \{V\gamma\}$ no início por A $:= \{F\gamma\}$.[8] Outra possibilidade seria chamar o algoritmo satA-nd passando $\neg\gamma$ como argumento: γ é falseável se e somente se satA-nd$(\neg\gamma)$ resulta em *sucesso*.

Para que o algoritmo satA-nd verifique se um *conjunto de fórmulas* H, passado como argumento, é satisfatível, basta substituir o pseudocomando A $:= \{V\gamma\}$ por A $:= \{V\gamma \,|\, \gamma \in H\}$.

O procedimento satA-nd será refeito de forma a descartar a necessidade do prefixo s, que indica o que fazer (satisfazer ou falsear). O nome do procedimento resultante será satB-nd. O artifício para obtenção satB-nd a partir de satA-nd baseia-se no fato de que para falsear α, basta satisfazer $\neg\alpha$. Tendo isso em vista, no lugar do par $V\alpha$

[8] Evidentemente, além da mudança assinalada, seria conveniente alterar o nome do procedimento para, por exemplo, falA-nd.

```
proc-nd satB-nd(fórmula γ):
    conjunto de fórmulas A;
    conjunto de literais Σ;
    Σ = ∅;
    A := {γ};
    repita
        γ := escolha(A); A := A − {γ};
        caso γ seja
          literal:      se γ̄ ∈ Σ então fracasso senão Σ := Σ ∪ {γ} fimse
          ⊤,¬⊥:         { continue }
          ⊥,¬⊤:         fracasso
          ¬¬α:          A := A ∪ {α}
          α ∧ β:        A := A ∪ {α, β}
          ¬(α ∧ β):     A := A ∪ {escolha-nd({¬α, ¬β})} { escolha não determinística }
          α ∨ β:        A := A ∪ {escolha-nd({α, β})} { escolha não determinística }
          ¬(α ∨ β):     A := A ∪ {¬α, ¬β}
          α → β:        A := A ∪ {escolha-nd({¬α, β})} { escolha não determinística }
          ¬(α → β):     A := A ∪ {α, ¬β}
          α ↔ β:        A := A ∪ {escolha-nd({{α, β}, {¬α, ¬β}})} { escolha não determinística }
          ¬(α ↔ β):     A := A ∪ {escolha-nd({{α, ¬β}, {¬α, β}})} { escolha não determinística }
        fimcaso
    até A = ∅;
    sucesso
fim satB-nd
```

Figura 2.11: Satisfabilidade por construção de modelo sem a variável s.

usa-se α e no lugar de $F\alpha$, usa-se $\neg\alpha$; agora, em todos os pontos a satisfação de uma subfórmula é obtida pela satisfação (nunca falseamento) de suas subfórmulas, como mostra a Figura 2.11. No algoritmo, e também em vários locais daqui para frente, é usado o conceito de *literal*. Um literal é uma variável precedida ou não pelo conectivo da negação. O *complemento* do literal l, denotado por $\overline{l}$, é assim definido: seja ν a variável de l; se $l = \nu$, então $\overline{l} = \neg\nu$, e se $l = \neg\nu$, então $\overline{l} = \nu$.

Agora, quando A atinge $\emptyset$, Σ indica um modelo tal que, para uma variável ν, se $\nu \in \Sigma$ então $v^i(\nu) = V$, e se $\neg\nu \in \Sigma$ então $v^i(\nu) = F$.

O Exemplo 27 é remodelado a seguir para retratar as mudanças anteriores.

Exemplo 28 Uma árvore de computação correspondente à chamada satB-nd($(p \rightarrow q) \wedge ((q \vee r) \rightarrow p)$) está mostrada na Figura 2.12. Note como ela é parecida com a da Figura 2.10 do Exemplo 27. Novamente, como no exemplo anterior, as interpretações que satisfazem a fórmula podem ser facilmente recuperáveis da árvore. □

Um algoritmo não determinístico para testar se uma fórmula γ é falseável seria o próprio satB-nd, substituindo-se o pseudomando A := $\{\gamma\}$ no início por A := $\{\neg\gamma\}$. [9]

[9]Evidentemente, além da mudança assinalada, seria conveniente alterar o nome do procedimento para, por exemplo, falB-nd.

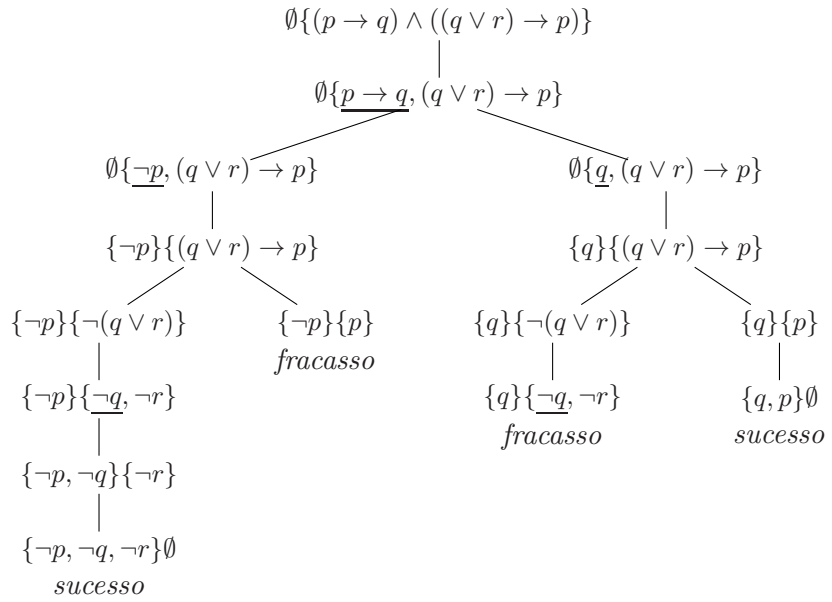


Figura 2.12: Árvore de computação para satB-nd$((p \to q) \wedge ((q \vee r) \to p))$.

Outra possibilidade seria chamar o algoritmo satB-nd passando $\neg\gamma$ como argumento: γ é falseável se e somente se satB-nd$(\neg\gamma)$ resulta em *sucesso*.

Para que satB-nd verifique a satisfabilidade de um *conjunto de fórmulas* H, passado como argumento, basta substituir o pseudocomando A := $\{\gamma\}$ por A := H.

Na Figura 2.13 é apresentada uma versão determinística do algoritmo satA-nd da Figura 2.9, denominado satA. Um algoritmo determinístico correspondente a satB-nd pode ser obtido de maneira análoga.

Uma estratégia muito usada para lidar com não determinismo é a de *backtracking*, que mantém ativa em determinado instante apenas uma das computações, estendendo-a enquanto possível, até o *sucesso*; caso haja *fracasso*, o estado mais recente da computação para o qual haja alternativa ainda não testada é restaurado e prossegue-se a computação a partir do estado restaurado. O procedimento satA apresenta um esquema mais geral do que esse: as múltiplas computações de satA-nd são representadas em LA, de tal forma que qualquer uma delas pode ser estendida em dado momento. Cada computação é representada em LA pelo par (Σ, A), em que Σ contém as variáveis marcadas que representam o modelo parcialmente construído até o momento, e A contém o conjunto dos pares $s\gamma$ como em satA-nd. Se LA for implementada como uma pilha, obtém-se a estratégia de busca em profundidade na árvore de computação, que é justamente a de *backtracking*. Se LA for implementada como uma fila, obtém-se a estratégia de busca em largura. O algoritmo, como formulado, mediante o uso do pseudocomando escolha, não se compromete com nenhuma estratégia de escolha, tanto da computação a estender (escolha do próximo elemento de LA), quanto da próxima subfórmula a tentar satisfazer (escolha do próximo elemento de A); ou seja, nestes pontos ele continua "não determinístico".

Com relação à complexidade do algoritmo, quaisquer estratégias de escolha proje-

```
proc satA(fórmula γ) retorna {verdadeiro, falso}:
    conjunto de pares (conjunto de variáveis marcadas, conjunto de fórmulas marcadas) LA;
    conjunto de fórmulas marcadas A;
    conjunto de variáveis marcadas Σ;
    elemento de {V, F} s;
    LA := {(∅, {Vγ})};
    repita
        (Σ, A) := escolha(LA); LA := LA − (Σ, A);
        sγ := escolha(A); A := A − {sγ};
        caso sγ seja
            *var:       se s̄γ ∉ Σ então
                            Σ := Σ ∪ {sγ};
                            se A = ∅ então retorne verdadeiro fimse
                            LA := LA ∪ {(Σ, A)}
                        fimse
                        { caso contrário, fracasso; continue }
            Vα∧β:       se A = ∅ então retorne verdadeiro fimse;
                        LA := LA ∪ {(Σ, A)}
            Vα∨β:       { fracasso; continue }
            Vα→β:       LA := LA ∪ {(Σ, A ∪ {Fα})}
            V¬α:        LA := LA ∪ {(Σ, A ∪ {Vα})}
            F¬α:        LA := LA ∪ {(Σ, A ∪ {Vα, Vβ})}
            Fα∧β:       LA := LA ∪ {(Σ, A ∪ {Fα}), (Σ, A ∪ {Fβ})}
            Vα∧β:       LA := LA ∪ {(Σ, A ∪ {Vα}), (Σ, A ∪ {Vβ})}
            Fα∨β:       LA := LA ∪ {(Σ, A ∪ {Fα, Fβ})}
            Vα→β:       LA := LA ∪ {(Σ, A ∪ {Fα}), (Σ, A ∪ {Vβ})}
            Fα→β:       LA := LA ∪ {(Σ, A ∪ {Vα, Fβ})}
            Vα↔β:       LA := LA ∪ {(Σ, A ∪ {Vα, Vβ}), (Σ, A ∪ {Fα, Fβ})}
            Fα↔β:       LA := LA ∪ {(Σ, A ∪ {Fα, Vβ}), (Σ, A ∪ {Vα, Fβ})}
        fimcaso;
    até LA = ∅;
    retorne falso
fim satA
```

Figura 2.13: Satisfabilidade por construção de modelo determinística.

tadas até hoje nos dois casos citados acima, não evitam um gasto exponencial no pior caso. Isto é, há sempre entradas que provocam a exploração de uma quantidade exponencial de vértices da árvore de computação do algoritmo não determinístico (a qual é percorrida, mesmo que implicitamente).

Um algoritmo determinístico para testar se uma fórmula γ é falseável seria o próprio satA, substituindo-se o pseudomando LA $:= \{(\emptyset, \{V\gamma\})\}$ no início por LA $:= \{(\emptyset, \{F\gamma\})\}$.[10] Outra possibilidade seria chamar o algoritmo satA passando $\neg\gamma$ como argumento.

Para que satA verifique a satisfabilidade de um *conjunto de fórmulas* H, passado como argumento, basta substituir o pseudocomando LA $:= \{(\emptyset, \{V\gamma\})\}$ por LA $:= \{(\emptyset, \{V\gamma \mid \gamma \in H\})\}$.

[10] Novamente, seria conveniente alterar o nome do procedimento para, por exemplo, falB.

Exercícios

1. Utilize um algoritmo que propague o valor de uma subfórmula em direção ao conectivo principal da fórmula, como o pedido no Exercício **??** da Seção 2.3.2, em um algoritmo *não determinístico* baseado no método da tabela da verdade para determinar se uma fórmula é satisfatível. Ele deverá usar o pseudomando escolha-nd($\{V, F\}$)) para escolher, não deterministicamente, um valor a ser atribuído a uma variável.

2. Para cada fórmula a seguir faça árvores de computação correspondentes às execuções de satA-nd e satB-nd. Explicite também os modelos associados aos ramos de sucesso.

a) $(p \vee \neg p) \rightarrow \bot$

b) $(p \vee q) \wedge \neg q$

c) $(p \wedge \neg p) \rightarrow \bot$.

d) $(q \rightarrow p) \vee ((p \rightarrow q) \wedge \neg q)$

e) $(p \rightarrow \bot) \leftrightarrow (p \vee \top)$

3. Para as fórmulas da questão anterior, faça árvores de computação correspondentes às execuções de falA-nd e falB-nd, obtidos como explicado no texto pela substituição de A := $\{V\gamma\}$ por A := $\{F\gamma\}$ num caso e de A := $\{V\gamma\}$ por A := $\{\neg\gamma\}$ no outro. Explicite também os contramodelos associados aos ramos de sucesso.

4. Faça uma versão determinística do algoritmo satB-nd.

2.3.6 Verificação de tautologias e contradições no nível conceitual

Nenhum dos algoritmos não determinísticos, satA-nd ou satB-nd, pode ser usado para determinar se uma fórmula é insatisfatível. Da mesma forma, algoritmos não determinísticos análogos para determinar se uma fórmula é falseável não podem ser usados para determinar se uma fórmula é tautológica (isto é, não falseável). Um algoritmo não determinístico não serve para solucionar o problema complementar: trocando-se *sucesso* por *fracasso* e vice-versa, não se obtém um *algoritmo não determinístico* para o problema complementar.[11] Os problemas complementares aos da classe NP (como os problemas da satisfabilidade e da falseabilidade) são ditos ser da classe coNP. Pensa-se, em geral, que os problemas coNP são mais difíceis, embora não haja ainda uma prova de que NP $\neq$ coNP.[12] Assim, não se conhece algoritmo polinomial para insatisfabilidade, mesmo não determinístico, mas em geral pensa-se que não existe.

No entanto, o algoritmo *determinístico* satA da Figura 2.13 pode ser usado para verificar se uma fórmula α é insatisfatível: α é insatisfatível se, e somente se, satA(α) =

[11] Na verdade, um algoritmo não determinístico *correto* pode até mesmo entrar em *loop* para instâncias negativas.

[12] Sabe-se, ainda, que se NP $\neq$ coNP, então P $\neq$ NP.

falso. De forma análoga, um algoritmo determinístico falA para determinar se uma fórmula α é falseável pode ser usado para verificar se ela é tautológica: α é tautológica se, e somente se, $\mathsf{falA}(\alpha) = \mathit{falso}$.

Nas seções seguintes serão apresentados algoritmos similares aos já vistos para avaliação de fórmulas, determinação de satisfabilidade e falseabilidade e verificação de contradições e tautologias, só que trabalhando em nível formal. Sistemas formais nos quais eles se baseiam serão apresentados explicitamente.

Exercícios

1. Como o algoritmo *determinístico* satA da Figura 2.13 pode ser usado para verificar se uma fórmula α é tautológica?

2. Como um algoritmo *determinístico* falA, para verificar se uma fórmula é falseável, pode ser usado para determinar se uma fórmula é insatisfatível?

3. Como modificar satA para determinar se um conjunto é insatisfatível?

4. Como modificar satA para determinar se um conjunto não é falseável, ou seja, que toda interpretação satisfaz pelo menos uma fórmula do conjunto?

2.4 Avaliação, satisfabilidade, falseabilidade, tautologias e contradições no nível formal

Agora serão vistos sistemas formais para os importantes problemas de avaliação de fórmulas dada uma interpretação, de determinação de satisfabilidade e falseabilidade e de geração de tautologias e contradições.

2.4.1 Avaliação de fórmulas no nível formal

Nos exemplos da Seção 2.3.2 mostrou-se como avaliar uma fórmula α para uma interpretação i a partir de avaliações de suas subfórmulas, tanto trabalhando do conectivo principal em direção às variáveis (via aplicação direta da Definição 3) quanto na direção oposta (via aplicação do método da tabela da verdade). Tal avaliação pode ser imaginada como ocorrendo no nível conceitual, já que ela é feita diretamente a partir das funções booleanas que dão significado aos conectivos.

A seguir, será apresentada uma outra abordagem, que faz a avaliação no nível formal. A avaliação é dita formal porque é baseada apenas na obtenção de expressões a partir de expressões, cujos significados não são usados diretamente. Em outras palavras, ela é baseada em um sistema formal. Com isto, um algoritmo neste nível é basicamente uma *estratégia* para exploração do espaço de busca definido pelo sistema formal subjacente. Apesar de poderem existir múltiplos sistemas formais para o mesmo problema, uns podem ser mais adequados do que outros para servir como base para a construção de um algoritmo. De qualquer forma, a especificação de um sistema formal

tem uma importância ímpar, não apenas pela sua adequação (ou não) para o propósito em vista, como pelo fato de que podem existir múltiplos algoritmos alternativos baseados no mesmo sistema formal.[13] Tais algoritmos herdam, em maior ou menor grau, características importantes do sistema formal, como a correção e a completude. Por outro lado, um sistema formal constuma ser mais adequado do que um algoritmo (mesmo baseado nele) para análise matemática.

Serão apresentados três sistemas formais para o presente problema, sendo que os dois últimos serão aproveitados também para o importante problema da satisfabilidade na próxima seção.

Para a especificação do primeiro sistema formal, será utilizado o conceito de substituição de *uma* ocorrência de uma subfórmula por outra (de forma similar àquela apresentada no Capítulo 1). O próximo exemplo apresenta algumas substituições.

Exemplo 29 Seja $\gamma = (p \to q) \wedge (\neg p \vee (q \leftrightarrow \neg r))$. Então:

- substituindo-se $q \leftrightarrow \neg r$ por q em γ obtém-se $(p \to q) \wedge (\neg p \vee q)$;
- substituindo-se a primeira ocorrência de p por $r \vee \neg q$ em γ obtém-se $((r \vee \neg q) \to q) \wedge (\neg p \vee (q \leftrightarrow \neg r))$.
- substituindo-se a segunda ocorrência de p por $r \vee \neg q$ em γ obtém-se $(p \to q) \wedge (\neg(r \vee \neg q) \vee (q \leftrightarrow \neg r))$. □

Se α é uma fórmula e i uma interpretação para as variáveis, $\alpha[i]$ irá denotar a fórmula obtida pela substituição de cada variável ν em α por $\top$ se $\nu^i = V$ ou por $\bot$ se $\nu^i = F$. Por exemplo, se $p^i = V$ e $q^i = F$, então $(p \to q) \vee \neg(q \wedge p)[i] = (\top \to \bot) \vee \neg(\bot \wedge \top)$.

Será apresentado, a seguir, um sistema formal em que $\top$ é derivável a partir de $\alpha[i]$ se, e somente se, $v^i(\alpha) = V$ e $\bot$ é derivável a partir de $\alpha[i]$ se, e somente se, $v^i(\alpha) = F$. Tal sistema formal poderia ser feito de maneira que cada regra correspondesse a uma aplicação de uma função booleana a argumentos específicos. Por exemplo à aplicação $V \bigoplus V = V$ corresponderia uma regra $\top \to \top \mapsto \top$; analogamente, existiriam as regras $\top \to \bot \mapsto \bot$, $\bot \to \top \mapsto \top$ etc. Em vez disso, será apresentado um sistema formal que permite "passos maiores" de simplificação; por exemplo, com regra do tipo $\bot \to \alpha \mapsto \top$, sendo α uma fórmula qualquer.

O sistema formal é $\mathcal{S}_0 = (L_0, R_0, \emptyset)$, em que L_0 é o subconjunto das fórmulas em $\mathcal{F}$ que não contêm nenhuma variável proposicional e R_0 é o conjunto de regras de reescrita da Figura 2.14. Uma regra $x \mapsto y$ é aplicável a uma fórmula γ (sem nenhuma variável) se existe $\alpha \in \mathit{subf}(\gamma)$ que satisfaça o esquema x e, neste caso, o resultado da aplicação é o resultado de substituir uma das ocorrências de α em γ pela fórmula especificada por y. Por exemplo a regra conj1 diz que uma subfórmula $\top \wedge \alpha$ (em que α é qualquer fórmula) pode ser substituída por α. Já a regra conj3 diz que $\bot \wedge \alpha$ pode ser substituída por $\bot$. Como foi visto no Capítulo 1, diz-se que γ_n *é derivável a partir de* γ_0, isto é, $\gamma_0 \Rightarrow \gamma_n$, se e somente se existe uma sequência $\gamma_0, \ldots, \gamma_n$ tal que cada γ_i, $0 < i \leq n$,

[13] Por exemplo, as várias abordagens para prova automática de teoremas são comumente como da classe dos sistemas formais subjacentes.

Negação				Conjunção				Disjunção			
neg1: $\neg\top$	$\mapsto$	$\bot$		conj1: $\top \wedge \alpha$	$\mapsto$	α		disj1: $\top \vee \alpha$	$\mapsto$	$\top$	
neg2: $\neg\bot$	$\mapsto$	$\top$		conj2: $\alpha \wedge \top$	$\mapsto$	α		disj2: $\alpha \vee \top$	$\mapsto$	$\top$	
				conj3: $\bot \wedge \alpha$	$\mapsto$	$\bot$		disj3: $\bot \vee \alpha$	$\mapsto$	α	
				conj4: $\alpha \wedge \bot$	$\mapsto$	$\bot$		disj4: $\alpha \vee \bot$	$\mapsto$	α	

Condicional				Bicondicional			
cond1: $\top \to \alpha$	$\mapsto$	α		bic1: $\top \leftrightarrow \alpha$	$\mapsto$	α	
cond2: $\alpha \to \top$	$\mapsto$	$\top$		bic2: $\alpha \leftrightarrow \top$	$\mapsto$	α	
cond3: $\bot \to \alpha$	$\mapsto$	$\top$		bic3: $\alpha \leftrightarrow \alpha$	$\mapsto$	$\top$	

Figura 2.14: Regras do sistema formal $\mathcal{S}_0$.

é obtido de γ_{i-1} pela aplicação de uma das regras. Segue um exemplo de derivação em $\mathcal{S}_0$.

Exemplo 30 Seja $\alpha = (p \to q) \vee ((\neg r \to q) \wedge \neg r)$ e i tal que $p^i = V$, $q^i = F$, $r^i = V$. Segue-se uma derivação a partir de $\alpha[i]$:

1. $(\top \to \bot) \vee ((\neg\top \to \bot) \wedge \neg\top)$ $(\alpha[i]))$
2. $\bot \vee (\neg\top \to \bot) \wedge \neg\top$ (cond1)
3. $(\neg\top \to \bot) \wedge \neg\top$ (disj3)
4. $(\bot \to \bot) \wedge \neg\top$ (neg1)
5. $\top \wedge \neg\top$ (cond3)
6. $\neg\top$ (conj1)
7. $\bot$ (neg1)

Portanto, $\alpha[i] \Rightarrow \bot$. Note que a derivação acima não é a única que leva a $\bot$ a partir de $\alpha[i]$. □

No Exercício 4 pede-se para mostrar que o sistema formal $\mathcal{S}_0$ tem as seguintes propriedades:

1. *correção*: se $\gamma[i] \Rightarrow \top$ então $v^i(\gamma) = V$ e se $\gamma[i] \Rightarrow \bot$ então $v^i(\gamma) = F$; e

2. *completude*: se $v^i(\gamma) = V$ então $\gamma[i] \Rightarrow \top$ e se $v^i(\gamma) = F$ então $\gamma[i] \Rightarrow \bot$.

O algoritmo avalA, mostrado na Figura 2.15, é baseado em $\mathcal{S}_0$. Ele apenas simplifica uma fórmula sem variáveis proposicionais por meio das regras de $\mathcal{S}_0$ até obter $\top$ ou $\bot$. A chamada avalA($\alpha[i]$) retorna $\top$, se $\alpha[i] \Rightarrow \top$, ou retorna $\bot$, se $\alpha[i] \Rightarrow \bot$. O processo realmente termina com a produção de $\top$ ou $\bot$, visto que em cada passo do comando **enquanto**, caso α não seja $\top$ nem $\bot$:

- alguma regra é aplicável;
- a aplicação de uma regra provoca a obtenção de uma fórmula menor.

proc avalA(*fórmula sem variáveis proposicionais* α) **retorna** $\{\top, \bot\}$:
  enquanto *alguma regra da Figura 2.14 é aplicável a* α **faça**
    $\alpha :=$ *resultado de aplicar uma regra a* α
  fimenquanto;
  retorne α
fim avalA

Figura 2.15: Avaliação formal de uma fórmula.

Como cada α' obtido após a aplicação de uma regra a uma fórmula α é tal que $v^i(\alpha') = v^i(\alpha)$, segue-se que o resultado é $\top$ se, e somente se, $v^i(\alpha) = V$.

Deve-se notar que o algoritmo avalA é *não determinístico*, visto que não especifica a ordem de aplicação das regras. No entanto, tal não determinismo é especial: qualquer que seja a sequência de escolhas de regras, haverá *sucesso*, com o algoritmo produzindo a resposta desejada. A cada sequência possível de escolhas de regras corresponde uma derivação diferente, mas cada uma delas "demonstra" o (mesmo) resultado da avaliação.

Exemplo 31 Seja, novamente, o problema do Exemplo 9 de determinar $v^i(\alpha)$ para $\alpha = ((ch \vee ce) \rightarrow rm) \wedge (rm \rightarrow re)$ em uma situação em que: $ch^i = V$, $ce^i = F$, $rm^i = V$ e $re^i = F$. A seguinte derivação seria uma que corresponderia a uma sequência de escolhas levando ao resultado ($\bot$, no caso):

1. $((\top \vee \bot) \rightarrow \top) \wedge (\top \rightarrow \bot)$ $(\alpha[i]))$
2. $\top \wedge (\top \rightarrow \bot)$ (cond2))
3. $\top \rightarrow \bot$ (conj1))
4. $\bot$ (cond1)

Portanto, $v^i(\alpha) = F$. □

O conjunto de regras da Figura 2.14 é redundante, podendo várias regras se aplicarem para a mesma subfórmula como enfatizado no Exercício 5 no final desta seção. O critério principal utilizado na obtenção das mesmas foi: propiciar o máximo de simplificações possível. Assim, para uma subfórmula $\gamma_1 \wedge \gamma_2$, caso γ_1 ou γ_2 seja $\top$ ou $\bot$, há simplificação (se nenhum for $\top$ ou $\bot$, obviamente γ_1 ou γ_2 deve ser simplificada antes); em particular, se uma das duas (γ_1 ou γ_2) for $\bot$ ou $\top$, a simplificação pode ser bem grande, caso a outra seja uma subfórmula bem grande. Por outro lado, uma regra da forma cond4: $\alpha \rightarrow \bot \mapsto \neg\alpha$ não foi incluída, visto que forçaria a avaliação da subfórmula α de qualquer forma para que fosse possível depois a aplicação da regra neg1 ou neg2.

Os dois sistemas formais mostrados adiante, além de servirem como base para o problema de determinar $v^i(\alpha)$, dados α e i, servirão também para o problema de determinar i tal que $v^i(\alpha) = V$ (o famoso problema da satisfabilidade) na Seção 2.4.2.

No próximo sistema formal, para codificar diretamente o fato de que $v^i(\alpha) = V$ ou $v^i(\alpha) = F$, serão usadas as expressões $V\alpha$ e $F\alpha$, respectivamente, de forma análoga à empregada na Seção 2.3.5. Para isto, será usada a linguagem das *fórmulas marcadas*:

Verum e falsum

$$\mathsf{verp}: \frac{}{V\top} \qquad \mathsf{faln}: \frac{}{F\bot}$$

Negação

$$\mathsf{negp}: \frac{F\alpha}{V\neg\alpha} \qquad \mathsf{negn}: \frac{V\alpha}{F\neg\alpha}$$

Conjunção

$$\mathsf{conjp}: \frac{V\alpha \;\; V\beta}{V\alpha\wedge\beta} \qquad \mathsf{conjn1}: \frac{F\alpha}{F\alpha\wedge\beta} \qquad \mathsf{conjn2}: \frac{F\beta}{F\alpha\wedge\beta}$$

Disjunção

$$\mathsf{disjp1}: \frac{V\alpha}{V\alpha\vee\beta} \qquad \mathsf{disjp2}: \frac{V\beta}{V\alpha\vee\beta} \qquad \mathsf{disjn}: \frac{F\alpha \;\; F\beta}{F\alpha\vee\beta}$$

Condicional

$$\mathsf{condp1}: \frac{F\alpha}{V\alpha\to\beta} \qquad \mathsf{condp2}: \frac{V\beta}{V\alpha\to\beta} \qquad \mathsf{condn}: \frac{V\alpha \;\; F\beta}{F\alpha\to\beta}$$

Bicondicional

$$\mathsf{bicp1}: \frac{V\alpha \;\; V\beta}{V\alpha\leftrightarrow\beta} \qquad \mathsf{bicp2}: \frac{F\alpha \;\; F\beta}{V\alpha\leftrightarrow\beta} \qquad \mathsf{bicn1}: \frac{V\alpha \;\; F\beta}{F\alpha\leftrightarrow\beta} \qquad \mathsf{bicn2}: \frac{F\alpha \;\; V\beta}{F\alpha\leftrightarrow\beta}$$

Figura 2.16: Regras do sistema formal $\mathcal{S}_A$.

$\mathcal{FM} = \{s\alpha \,|\, s \in \{V, F\} \text{ e } \alpha \in \mathcal{F}\}$. Note que o alfabeto da linguagem $\mathcal{FM}$ é igual ao alfabeto de $\mathcal{F}$ acrescido dos dois símbolos "V" e "F".[14] Haverá dois tipos de regras para cada conectivo: se α não for variável, haverá, para o conectivo principal de α, regra(s) do tipo p (positivo) para derivar $V\alpha$ e do tipo n (negativo) para derivar $F\alpha$. Por exemplo, para o conectivo $\wedge$:

1. como se $v^i(\alpha) = V$ e $v^i(\beta) = V$, então $v^i(\alpha \wedge \beta) = V$, haverá a regra conjp que permite concluir $V\alpha \wedge \beta$ a partir de $V\alpha$ e $V\beta$:

$$\mathsf{conjp}: \frac{V\alpha \;\; V\beta}{V\alpha\wedge\beta}$$

2. como se $v^i(\alpha) = F$ ou $v^i(\beta) = F$, então $v^i(\alpha\wedge\beta) = F$, haverá duas regras, a regra conjn1 que permite concluir $F\alpha \wedge \beta$ a partir de $F\alpha$ e a regra conjn2 que permite concluir $F\alpha \wedge \beta$ a partir de $F\beta$:

$$\mathsf{conjn1}: \frac{F\alpha}{F\alpha\wedge\beta} \qquad \mathsf{conjn2}: \frac{F\beta}{F\alpha\wedge\beta}$$

De forma análoga, chega-se às regras para os outros conectivos. O sistema formal é $\mathcal{S}_A = (\mathcal{FM}, R_A, \emptyset)$, sendo R_A o conjunto de regras da Figura 2.16. As regras de

[14] V e F usados nesse contexto não são os conceitos V e F. Eles são símbolos da linguagem formal.

$\mathcal{S}_A$, por si só, não garantem necessariamente que a *mesma interpretação* que satisfaça as premissas também satisfaz a conclusão. Por exemplo, pela regra conjp, de Vp e $V\neg p$, pode-se concluir $Vp \wedge \neg p$. A garantia é dada pela definição de *conjuntos de partida* permissíveis. Dada uma interpretação i, o conjunto de partida respectivo Σ_i é $\{s\nu \,|\, \nu^i = s\}$.[15] O sistema $\mathcal{S}_A$ é tal que para todo $\alpha \in \mathcal{F}$ e toda i,

$$\Sigma_i \Rightarrow s\alpha \text{ se, e somente se, } v^i(\alpha) = s.$$

Isto expressa a correção e completude de $\mathcal{S}_A$ com relação a avaliação de fórmulas a partir de interpretações.

O problema de determinar $v^i(\alpha)$, dados α e i, tomando-se $\mathcal{S}_A$ como sistema formal subjacente, torna-se o de encontrar uma derivação de $s\alpha$ a partir de Σ_i. Encontrada a derivação, a correção de $\mathcal{S}_A$ garante que $v^i(\alpha) = s$.

O exemplo a seguir mostra o uso de $\mathcal{S}_A$ para demonstrar que $v^i(\alpha) = F$ para a fórmula α e interpretação i do Exemplo 30.

Exemplo 32 Seja, novamente, a fórmula $\alpha = (p \to q) \vee ((\neg r \to q) \wedge \neg r)$ e i tal que $p^i = V$, $q^i = F$ e $r^i = V$. Segue uma derivação de $F\alpha$ a partir de Σ_i em $\mathcal{S}_A$:

1. Vp	(Σ_i)
2. Fq	(Σ_i)
3. Vr	(Σ_i)
4. $F(p \to q)$	(condn 1,2)
5. $F\neg r$	(negn 3)
6. $F((\neg r \to q) \wedge \neg r)$	(conjn2 5)
7. $F(p \to q) \vee ((\neg r \to q) \wedge \neg r)$	(disjn 4,6)

Logo, pela correção de $\mathcal{S}_A$, $v^i(\alpha) = F$. □

No exemplo anterior, fica claro que de Σ_i só são usadas como premissas variáveis marcadas $s\nu$ em que ν é variável da fórmula em questão. Embora neste exemplo tenham sido usadas todas as variáveis da fórmula derivada, isso nem sempre é necessário. Por exemplo, a derivação

1. Fp	(Σ_i)
4. $Fp \wedge q$	(conjn1 1)

mostra que $\Sigma_i \Rightarrow Fp \wedge q$ e, portanto, $v^i(p \wedge q) = F$ para qualquer i tal que $p^i = F$.

Uma demonstração da correção de $\mathcal{S}_A$ é apresentada a seguir.

Teorema 5 *(Correção de $\mathcal{S}_A$) Para todo $\alpha \in \mathcal{F}$ e toda i, se $\Sigma_i \Rightarrow s\alpha$, então $v^i(\alpha) = s$.*

Prova

Sejam $\alpha \in \mathcal{F}$ e i arbitrários. Será mostrado por indução sobre o comprimento da derivação de $s\alpha$ a partir de Σ_i que se $\Sigma_i \Rightarrow s\alpha$ então $v^i(\alpha) = s$. Seja $n \geq 0$ e suponha,

[15] Aqui, e também em alguns pontos à frente, comete-se o abuso de usar s com dois sentidos: como significando um conceito em $\nu^i = s$ e um símbolo em $s\nu$. A justificativa é que isto propicia uma maior concisão.

como hipótese de indução, que isto valha para derivações de comprimento menor que n. Seja uma derivação de $s\alpha$ a partir de Σ_i de comprimento n. Se $n = 0$, então $s\alpha \in \Sigma_i$ e, portanto, por definição de Σ_i, $v^i(\alpha) = s$. Considere agora uma derivação de $s\alpha$ a partir de Σ_i de comprimento $n > 0$. Neste caso, no último passo da derivação obtém-se $s\alpha$ por meio de uma das regras da Figura 2.16:

1. Regra verp. Então $s = V$ e $\alpha = \top$. Logo, $v^i(\alpha) = \textcircled{T} = V$, como requerido.

2. Regra faln. Então $s = F$ e $\alpha = \bot$. Logo, $v^i(\alpha) = \textcircled{\bot} = F$, como requerido.

3. Regra negp. Então $s = V$ e $\alpha = \neg\beta$ e existe uma derivação de $F\beta$ a partir de Σ_i de comprimento $n - 1$. Pela hipótese de indução, $v^i(\beta) = F$. Logo, $v^i(\alpha) = \ominus v^i(\beta) = \ominus F = V$, como requerido.

4. Regra negn. Então $s = F$ e $\alpha = \neg\beta$ e existe uma derivação de $V\beta$ a partir de Σ_i de comprimento $n - 1$. Pela hipótese de indução, $v^i(\beta) = V$. Logo, $v^i(\alpha) = \ominus v^i(\beta) = \ominus V = F$, como requerido.

5. Regra conjp. Então $s = V$, $\alpha = \beta \wedge \gamma$ e existem derivações de $V\beta$ e de $V\gamma$ a partir de Σ_i de comprimentos menores que n. Pela hipótese de indução, $v^i(\beta) = V$ e $v^i(\gamma) = V$. Logo, $v^i(\alpha) = v^i(\beta) \textcircled{\wedge} v^i(\gamma) = V \textcircled{\wedge} V = V$, como requerido.

6. Regra conjn1. Então $s = F$, $\alpha = \beta \wedge \gamma$ e existe uma derivação de $F\beta$ a partir de Σ_i de comprimento $n - 1$. Pela hipótese de indução, $v^i(\beta) = F$. Logo, $v^i(\alpha) = v^i(\beta) \textcircled{\wedge} v^i(\gamma) = F \textcircled{\wedge} v^i(\gamma) = F$, como requerido.

7. Regra conjn2. Então $s = F$, $\alpha = \beta \wedge \gamma$ e existe uma derivação de $F\gamma$ a partir de Σ_i de comprimento $n - 1$. Pela hipótese de indução, $v^i(\gamma) = F$. Logo, $v^i(\alpha) = v^i(\beta) \textcircled{\wedge} v^i(\gamma) = vi(\beta) \textcircled{\wedge} F = F$, como requerido.

Os casos restantes, que envolvem as regras para disjunção, condicional e bicondicional, são deixados como exercício. Conclui-se, então, que se $\Sigma_i \Rightarrow s\alpha$, então $v^i(\alpha) = s$, para α e i arbitrários, de onde segue o resultado. □

Agora vem uma demonstração da completude de $\mathcal{S}_A$. Será usada a notação $\overline{s}$ para significar F, se $s = V$, ou V, se $s = F$.

Teorema 6 *(Completude de $\mathcal{S}_A$) Para todo $\alpha \in \mathcal{F}$ e toda i, se $v^i(\alpha) = s$, então $\Sigma_i \Rightarrow s\alpha$.*

Prova

Sejam $\alpha \in \mathcal{F}$ e i arbitrários. Será mostrado por indução sobre o número de conectivos de α que se $v^i(\alpha) = s$, então $\Sigma_i \Rightarrow s\alpha$. Seja $n \geq 0$ e suponha, como hipótese de indução, que isto valha para fórmulas com menos de n conectivos. Seja $\alpha \in \mathcal{F}$ com n conectivos e *suponha que* $v^i(\alpha) = s$. Se $n = 0$, então α é uma variável proposicional e, por definição, $s\alpha \in \Sigma_i$. Logo, $\Sigma_i \Rightarrow s\alpha$. Considere agora α com $n > 0$ conectivos. Para cada possível conectivo principal de α considera-se um caso:

1. $\alpha = \top$. Então $v^i(\alpha) = V$ e, pela regra verp, $\Sigma_i \Rightarrow V\alpha$.

2. $\alpha = \bot$. Então $v^i(\alpha) = F$ e, pela regra faln, $\Sigma_i \Rightarrow F\alpha$.

3. $\alpha = \neg\beta$. Então $v^i(\alpha) = \ominus v^i(\beta)$ e, assim, $v^i(\beta) = \overline{s}$. Como β tem menos de n conectivos, pela hipótese de indução, $\Sigma_i \Rightarrow \overline{s}\beta$. Pelas regras negp e negn, segue-se então que $\Sigma_i \Rightarrow s\neg\beta$.

4. $\alpha = \beta \wedge \gamma$. Então $v^i(\alpha) = v^i(\beta) \oslash v^i(\gamma)$. Como $v^i(\alpha) = s$, tem-se 3 casos: (a) $s = V = v^i(\beta) = v^i(\gamma)$ ou (b) $s = F = v^i(\beta)$ ou (c) $s = F = v^i(\gamma)$. Como β e γ têm menos de n conectivos, pela hipótese de indução, se $s = V$ (caso a), então $\Sigma_i \Rightarrow s\beta$ e $\Sigma_i \Rightarrow s\gamma$, e pela regra conjp, segue-se que $\Sigma_i \Rightarrow s\beta \wedge \gamma$; por outro lado, se $s = F$, então $\Sigma_i \Rightarrow s\beta$ (no caso b) ou $\Sigma_i \Rightarrow s\gamma$ (no caso c), e pela regra conjn1 (caso b) ou conjn2 (caso c), tem-se que $\Sigma_i \Rightarrow s\beta \wedge \gamma$.

Os casos restantes, para α das formas $\beta \vee \gamma$, $\beta \rightarrow \gamma$ e $\beta \leftrightarrow \gamma$, são deixados como exercício. □

Um sistema formal não força que algoritmos nele baseados trabalhem das variáveis em direção ao conectivo principal ou na direção oposta; podem ser construídos algoritmos de ambos os tipos. Para o presente caso, no entanto, parece ser mais eficiente produzir uma derivação de $s\alpha$, para algum α e alguma i, partindo-se de Σ_i e, sendo guiado pela estrutura de α (isto é, pelo grafo de α), ir aplicando as regras até atingir uma fórmula marcada $s\alpha$. A Figura 2.17 apresenta um algoritmo baseado nessa idéia. A variável Σ nesse algoritmo é inicializada com os elementos de Σ_i que podem ser utilizados em uma derivação (aqueles restritos às variáveis da fórmula em questão e em conformidade com i). Já a variável D, inicializada com $\emptyset$, contém no final todas as fórmulas marcadas da derivação encontrada. O procedimento propA, apresentado na Figura 2.18, realiza implicitamente a aplicação das regras de $\mathcal{S}_A$, enquanto caminha no grafo da fórmula, podendo ser modificado facilmente para anotar as regras aplicadas de modo a propiciar a apresentação das derivações no estilo daquelas vistas nos exemplos.

O terceiro sistema formal é $\mathcal{S}_B = (\mathcal{F}, R_B, \emptyset)$, em que R_B pode ser obtido de R_A eliminando-se o símbolo V e substituindo-se o símbolo F por $\neg$. Agora, o fato de que $\nu^i = V$ ou $\nu^i = F$ será afirmado pela presença, em Σ_i, de ν (no lugar de $V\nu$) ou $\neg\nu$ (no lugar de $F\nu$). As regras em R_B são as da Figura 2.19. Note que os nomes das regras são os mesmos que os usados para as regras de $\mathcal{S}_A$, e que em $\mathcal{S}_B$ não existe a regra negp, que teria premissa e conclusão idênticas ($\neg\alpha$).

O teorema da completude para o sistema $\mathcal{S}_B$ é analogo ao de $\mathcal{S}_A$: para todo $\alpha \in \mathcal{F}$ e toda i,

$$\Sigma_i \Rightarrow \alpha \text{ se, e somente se, } v^i(\alpha) = V.$$

Observe que desta e do fato de que $v^i(\neg\alpha) = V$ se, e somente se, $v^i(\alpha) = F$, segue-se que:

$$\Sigma_i \Rightarrow \neg\alpha \text{ se, e somente se, } v^i(\alpha) = F.$$

O exemplo a seguir mostra o uso de $\mathcal{S}_B$ para a mesma fórmula do Exemplo 32.

proc avalA(*fórmula* α, *interpretação* i) **retorna** $\{V, F\}$:
  conjunto de fórmulas marcadas D;
  conjunto de variáveis marcadas Σ;
  elemento de $\{V, F\}$ s;
  variável ν;
  $\Sigma := \{s\nu \mid \nu \in \mathcal{V} \cap subf(\alpha) \text{ e } \nu^i = s\}$;
  $D := \emptyset$;
  se $\top \in subf(\alpha)$ **então** propA($V\top$) **fimse**; { regra verp }
  se $\bot \in subf(\alpha)$ **então** propA($F\bot$) **fimse**; { regra faln }
  enquanto *não existe* s *tal que* $s\alpha \in D$ **faça**
    $s\nu :=$ escolha(Σ);
    $\Sigma := \Sigma - \{s\nu\}$;
    propA($s\nu$)
  fimenquanto;
  retorne s
fim avalA

Figura 2.17: Avaliação formal de uma fórmula.

Exemplo 33 Seja $\alpha = (p \to q) \vee ((\neg r \to q) \wedge \neg r)$ e i tal que $p^i = V$, $q^i = F$ e $r^i = V$. Segue uma derivação de $\neg\alpha$ a partir de Σ_i em $\mathcal{S}_B$:

1. p (Σ_i)
2. $\neg q$ (Σ_i)
3. r (Σ_i)
4. $\neg(p \to q)$ (condn 1,2)
5. $\neg\neg r$ (negn 3)
6. $\neg((\neg r \to q) \wedge \neg r)$ (conjn2 5)
7. $\neg[(p \to q) \vee ((\neg r \to q) \wedge \neg r)]$ (disjn 6,5)

Logo, $v^i(\alpha) = F$. Note como esta derivação pode ser obtida daquela do Exemplo 32 pela eliminação do prefixo V e substituição do prefixo F por $\neg$ em cada fórmula marcada. Veja também que essa mesma derivação corresponderia à das 7 fórmulas marcadas da derivação do Exemplo 32 seguidas de

8. $V\neg(p \to q) \vee ((\neg r \to q) \wedge \neg r)$ (negp 7)

□

Derivar $\neg\alpha$ a partir de Σ_i em $\mathcal{S}_B$ tem dois significados: que $v^i(\neg\alpha) = V$ e que $v^i(\alpha) = F$. Com isto, a uma derivação de $\neg\alpha$ em $\mathcal{S}_B$ correspondem duas derivações em $\mathcal{S}_A$, as de $V\neg\alpha$ e $F\alpha$, como visto no exemplo anterior. O passo correspondente à geração de $V\neg\alpha$ por aplicação de negp em $\mathcal{S}_A$ simplesmente não existe em $\mathcal{S}_B$: a aplicação de uma regra negp em $\mathcal{S}_B$ (que apenas substituiria uma fórmula $\neg\alpha$ por ela mesma), é desnecessária. Assim, nem sempre a derivação de $\neg\alpha$ em $\mathcal{S}_B$ pode ser obtida de uma em $\mathcal{S}_A$ pela simples eliminação do prefixo V e substituição do prefixo F por $\neg$ em cada fórmula marcada, como já mostrado no exemplo anterior. O exemplo a seguir também demonstra isso com maior ênfase.

```
proc propA(fórmula marcada sβ):
    fórmula γ, ψ;
    elemento de {V, F} s;
    se sβ ∈ D então retorne fimse
    D := D ∪ {sβ};
    para cada γ tal que (β, γ) ∈ si(α) faça
        caso γ seja
            ¬β: { negação }
                propA(s̄γ)                          { regras negp e negn }
            β ∧ ψ ou ψ ∧ β: { conjunção }
                se s = F então
                    propA(Fγ)                      { regras conjn1 e conjn2 }
                senãose Vψ ∈ D então
                    propA(Vγ)                      { regra conjp }
                fimse
            β ∨ ψ ou ψ ∨ β: { disjunção }
                se s = V então
                    propA(Vγ)                      { regras disjp1 e disjp2 }
                senãose Fψ ∈ D então
                    propA(Fγ)                      { regra disjn }
                fimse
            { ***os outros casos, análogos, ficam como exercício }
        fimcaso
    fimpara;
    retorne
fim propA
```

Figura 2.18: Propagação formal a partir de uma variável.

Exemplo 34 Seguem derivações em $\mathcal{S}_A$ e $\mathcal{S}_B$ que mostram que uma interpretação em que $p^i = F$ é modelo para $\neg\neg\neg p$ e contramodelo para $\neg\neg p$. Em $\mathcal{S}_A$, considere uma derivação constituída das fórmulas marcadas de 1 a 3, e outra com as fórmulas marcadas de 1 a 4:

1. Fp (Σ_i)
2. $V\neg p$ (negp 1)
3. $F\neg\neg p$ (negn 2)
4. $V\neg\neg\neg p$ (negp 3)

Em $\mathcal{S}_B$, uma única "corresponde" às duas acima:

$1'$. $\neg p$ (Σ_i)
$3'$. $\neg\neg\neg p$ (negn $1'$)

Note como $1'$ e $3'$ nesta correspondem, respectivamente, a 1 e 3 nas derivações em $\mathcal{S}_A$; não existem fórmulas correspondentes a 2 e 4 (que seriam obtidas via aplicações de negp; em outras palavras, $2'$ seria idêntica a 2 e $4'$ idêntica a 4). □

A obtenção do algoritmo avalA levou em conta que os *símbolos* V e F das fórmulas marcadas em $\mathcal{S}_A$ podem ser interpretados como os *valores lógicos* V e F atribuídos a

Verum e falsum

$$\mathsf{verp}: \frac{}{\top} \qquad \mathsf{faln}: \frac{}{\neg\bot}$$

Negação

$$\mathsf{negn}: \frac{\alpha}{\neg\neg\alpha}$$

Conjunção

$$\mathsf{conjp}: \frac{\alpha \quad \beta}{\alpha \wedge \beta} \qquad \mathsf{conjn1}: \frac{\neg\alpha}{\neg(\alpha \wedge \beta)} \qquad \mathsf{conjn2}: \frac{\neg\beta}{\neg(\alpha \wedge \beta)}$$

Disjunção

$$\mathsf{disjp1}: \frac{\alpha}{\alpha \vee \beta} \qquad \mathsf{disjp2}: \frac{\beta}{\alpha \vee \beta} \qquad \mathsf{disjn}: \frac{\neg\alpha \quad \neg\beta}{\neg(\alpha \vee \beta)}$$

Condicional

$$\mathsf{condp1}: \frac{\neg\alpha}{\alpha \to \beta} \qquad \mathsf{condp2}: \frac{\beta}{\alpha \to \beta} \qquad \mathsf{condn}: \frac{\alpha \quad \neg\beta}{\neg(\alpha \to \beta)}$$

Bicondicional

$$\mathsf{bicp1}: \frac{\alpha \quad \beta}{\alpha \leftrightarrow \beta} \qquad \mathsf{bicp2}: \frac{\neg\alpha \quad \neg\beta}{\alpha \leftrightarrow \beta} \qquad \mathsf{bicn1}: \frac{\alpha \quad \neg\beta}{\neg(\alpha \leftrightarrow \beta)} \qquad \mathsf{bicn2}: \frac{\neg\alpha \quad \beta}{\neg(\alpha \leftrightarrow \beta)}$$

Figura 2.19: Regras do sistema formal $\mathcal{S}_B$.

subfórmulas da fórmula cujo valor lógico se procura: a partir das *variáveis proposicionais* da fórmula, tais valores podem ser propagados pelas subfórmulas de acordo com as regras, até se atingir a própria fórmula no grafo. Isso nada mais é do que o método da tabela da verdade! Em outras palavras, o algoritmo avalA pode ser visto como baseado no sistema formal $\mathcal{S}_A$, mas também como baseado no método da tabela da verdade. (Esse mesmo algoritmo poderia ser dado como resposta ao Exercício 4 da Seção 2.3.2.)

Já um algoritmo análogo a avalA, porém baseado em $\mathcal{S}_B$, não existe, pois, se uma regra referencia $\neg\alpha$ (como premissa ou conclusão), não necessariamente $\neg\alpha$ é subfórmula da fórmula em questão. Não há como fazer de forma tão simples a propagação referida anteriormente. No entanto, de uma derivação em $\mathcal{S}_A$, pode-se obter uma correspondente em $\mathcal{S}_B$ sem esforço (veja Exercício 12).

A seguir é considerado, no nível formal, o problema de determinar se uma fórmula é satisfatível.

Exercícios

1. Considerando as regras da Figura 2.14, apresente todas as fórmulas γ tais que $(\top \to \bot) \vee ((\neg\top \to \bot) \wedge \neg\top) \Rightarrow \gamma$ pela aplicação de uma única regra.

2. Use o algoritmo da Figura 2.15 (ou seja, o sistema formal $\mathcal{S}_0$) para avaliar cada fórmula abaixo, para uma interpretação i tal que $p^i = V$, $q^i = F$ e $r^i = V$. Faça

como no Exemplo 31, procurando ter o "menor trabalho" possível.

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

b) $(\neg p \rightarrow \neg\neg p) \rightarrow p$

c) $((p \rightarrow q) \wedge q) \rightarrow p$

d) $((p \rightarrow q) \vee r) \rightarrow (\neg q \wedge (p \vee r))$

3. Encontre um formato geral de fórmula de n conectivos e uma sequência de aplicações de regras para os quais o algoritmo da Figura 2.15 tenha: (a) o pior comportamento possível em termos de tempo de execução; (b) o melhor comportamento possível em termos de tempo de execução.

4. Demonstre que o sistema $\mathcal{S}_0$ é:

a) correto: se $\gamma[i] \Rightarrow \top$ então $v^i(\gamma) = V$ e se $\gamma[i] \Rightarrow \bot$ então $v^i(\gamma) = F$; e

b) completo: se $v^i(\gamma) = V$ então $\gamma[i] \Rightarrow \top$ e se $v^i(\gamma) = F$ então $\gamma[i] \Rightarrow \bot$.

5. As regras da Figura 2.14 são redundantes, no sentido de que a mesma subfórmula pode casar com o lado esquerdo de mais de uma regra. Por exemplo, ambas as regras conj1 e conj2 são aplicáveis para a (sub)fórmula $\top \wedge \top$, ambas produzindo $\top$. Modifique o conjunto de regras para eliminar (totalmente) as redundâncias, porém satisfazendo dois requisitos: (a) o número de regras deve ser o *mínimo* possível e (b) a aplicação de uma regra deve levar a uma fórmula com um conectivo a menos (como ocorre com as regras da Figura 2.14). Em seguida, considere a aplicação do algoritmo da Figura 2.15 com esse novo conjunto de regras. Como o comportamento do algoritmo varia de acordo com o conjunto de regras adotado?

6. Encontre uma derivação no sistema formal $\mathcal{S}_A$ que forneça o valor de cada fórmula do Exercicício 2 para uma interpretação i tal que $p^i = V$, $q^i = F$ e $r^i = V$.

7. Repita o exercício anterior para o sistema formal $\mathcal{S}_B$.

8. Complete a demonstração do Teorema 5 com os casos faltantes (referentes às regras da disjunção, condicional e bicondicional).

9. Complete a demonstração do Teorema 6 com os casos faltantes (referentes a fórmulas disjuntivas, condicionais e bicondicionais).

10. Complete o algoritmo propA, da Figura 2.18, para prever todas as regras de $\mathcal{S}_A$.

11. Prove a correção e completude do sistema $\mathcal{S}_B$: para todo $\alpha \in \mathcal{F}$ e toda i, $\Sigma_i \Rightarrow \alpha$ se, e somente se, $v^i(\alpha) = V$. *Dica*: tome como base as demonstrações dos Teoremas 5 e 6 relativos ao sistema formal $\mathcal{S}_A$.

12. Mostre como, de uma derivação em $\mathcal{S}_A$ obter uma correspondente em $\mathcal{S}_B$. Dica: comece analisando os exemplos 33 e 34.

2.4.2 Verificação de satisfabilidade e falseabilidade no nível formal

Os algoritmos da Seção 2.3.5, satA-nd e satB-nd das Figuras 2.9 e 2.11, podem ser vistos como algoritmos para verificação de satisfabilidade baseados, respectivamente, nos sistemas formais $\mathcal{S}_A$ e $\mathcal{S}_B$ da Seção 2.4.1! Assim, tem-se a situação em que *ambos podem ser vistos como* trabalhando no nível conceitual ou formal. Isso, em particular, mostra como os sistemas formais são bastante intuitivos, visto que bem próximos do nível conceitual (suas regras são derivadas diretamente de relações conceituais óbvias).

Na Seção 2.4.1, $\mathcal{S}_A$ e $\mathcal{S}_B$ foram propostos para a avaliação de uma fórmula, dada uma interpretação específica i. No caso de $\mathcal{S}_A$, por exemplo, a derivação de $s\alpha$ a partir de Σ_i demonstra que $v^i(\alpha) = s$. Agora, Σ_i não é mais dado, e o problema é justamente o inverso: encontrar um conjunto Σ tal que haja uma derivação de $s\alpha$ a partir de Σ. É suficiente que Σ seja um subconjunto (finito) de Σ_i que se refira a apenas variáveis de α tal que $v^i(\alpha) = s$. A correção e completude se tornam, no caso de $\mathcal{S}_A$:

$$\Sigma \Rightarrow s\gamma \text{ para algum } \Sigma \text{ se, e somente se } v^i(\gamma) = s \text{ para alguma } i.$$

Um *conjunto de partida* é um conjunto $\Sigma \subseteq \{s\nu \mid s \in \{V, F\} \text{ e } \nu \in \mathcal{V}\}$ tal que $V\nu \notin \Sigma$ ou $F\nu \notin \Sigma$ para toda $\nu \in \mathcal{V}$. Naturalmente, apenas variáveis marcadas de um conjunto de partida podem ser usadas em uma derivação sem terem sido obtidas por aplicação de uma das regras da Figura 2.16.

De forma análoga, os algoritmos falA-nd e falB-nd, que podem ser obtidos como mostrado na Seção 2.3.5, podem ser vistos como algoritmos para verificação de falseabilidade baseados, respectivamente, nos mesmos sistemas formais $\mathcal{S}_A$ e $\mathcal{S}_B$ da Seção 2.4.1! Note que a correção e completude de $\mathcal{S}_A$, tanto para satisfabilidade, quanto para falseabilidade, já estão expressas em "$\Sigma \Rightarrow s\gamma$ para algum Σ se e somente se $v^i(\gamma) = s$ para alguma i". Por outro lado, a correção e completude de $\mathcal{S}_B$ para falseabilidade seria assim expressa, indiretamente: $\Sigma \Rightarrow \neg\gamma$ para algum Σ se e somente se $v^i(\gamma) = F$ para alguma i. Um exercício ao final da seção pede um sistema formal similar a $\mathcal{S}_B$, porém diretamente para falseabilidade, de tal forma que $\Sigma \Rightarrow \gamma$ para algum Σ se e somente se $v^i(\gamma) = F$ para alguma i.

A seguir, um exemplo de derivação que demonstra que uma fórmula é satisfatível.

Exemplo 35 Segue uma derivação de $V(p \to q) \wedge ((q \vee r) \to p))$ a partir do conjunto de partida $\Sigma = \{Fp, Fq, Fr\}$.

1. Fr	(Σ)
2. Fq	(Σ)
3. $Fq \vee r$	(disjn 2,1)
4. $V(q \vee r) \to p$	(condp1 3)
5. Fp	(Σ)
6. $Vp \to q$	(condp1 5)
7. $V(p \to q) \wedge ((q \vee r) \to p)$	(conjp 6,4)

Portanto, $(p \to q) \wedge ((q \vee r) \to p)$ é satisfatível. □

No próximo exemplo, a mesma fórmula do exemplo anterior é mostrada ser falseável usando, de novo, $\mathcal{S}_A$.

Exemplo 36 Segue uma derivação de $F(p \to q) \wedge ((q \vee r) \to p))$ a partir do conjunto de partida $\Sigma = \{Vp, Fq\}$.

1. Vp	(Σ)
2. Fq	(Σ)
3. $Fp \to q$	(condn 1,2)
4. $F(p \to q) \wedge ((q \vee r) \to p)$	(conjn1 3)

Portanto, $(p \to q) \wedge ((q \vee r) \to p)$ é falseável. □

A idéia por trás do uso do sistema formal $\mathcal{S}_A$, neste contexto, é que uma derivação de $V\gamma$ a partir de Σ represente a construção da fórmula $V\gamma$ a partir de expressões (de Σ) que denotem um modelo para γ. E em uma derivação de $F\gamma$, Σ denota um contramodelo, ou seja, uma interpretação que falseia γ. De fato, um conjunto de partida Σ denota qualquer interpretação i tal que

$$\text{se } s\nu \in \Sigma \text{ então } \nu^i = s.$$

Não há conflito para os valores ν^i, visto que para toda $\nu \in \mathcal{V}$, $V\nu \notin \Sigma$ ou $F\nu \notin \Sigma$.

Como dito, um algoritmo baseado em $\mathcal{S}_A$ é o próprio satA-nd da Figura 2.9. Ele é reapresentado na Figura 2.20 com anotações das regras de $\mathcal{S}_A$ que justificam cada passo. Para transformá-lo em um algoritmo que teste falseabilidade, basta substituir o pseudocomando A := $\{V\gamma\}$ por A := $\{F\gamma\}$. Como na Seção 2.3.5, o exemplo a seguir mostra a evolução dos conjuntos Σ e A para todas as computações possíveis, mediante uma árvore de computação que apresenta os conteúdos de Σ e A rotulando seus vértices e em que as bifurcações exibem os pontos de não determinismo. As regras de $\mathcal{S}_A$ são anotadas, mostrando como os pontos de não determinismo correspondem a situações em que mais de uma regra é aplicável pelo sistema formal.

Exemplo 37 Seja novamente determinar a satisfabilidade de $(p \to q) \wedge ((q \vee r) \to p))$. Na Figura 2.21 está mostrada a árvore de computação relativa à aplicação do algoritmo satA-nd, da Figura 2.20, a tal fórmula. A derivação mostrada no Exemplo 35 corresponde ao ramo da esquerda na Figura 2.21. À outra computação de sucesso corresponde esta derivação de $V(p \to q) \wedge ((q \vee r) \to p))$ a partir de $\{Vp, Vq\}$:

1. Vp	(de Σ)
2. $V(q \vee r) \to p$	(condp2 1)
3. Vq	(de Σ)
4. $Vp \to q$	(condp2 3)
5. $V(p \to q) \wedge ((q \vee r) \to p)$	(conjp 4,2)

Às duas computações de fracasso, obviamente não correspondem derivações. □

proc-nd satA-nd(fórmula γ):
  conjunto de fórmulas marcadas A;
  conjunto de variáveis marcadas Σ;
  elemento de $\{V, F\}$ s;
  $\Sigma := \emptyset$;
  A $:= \{V\gamma\}$;
  repita
    $s\gamma :=$ *escolha*(A); { escolhe próxima meta }
    A $:=$ A $- \{s\gamma\}$;
    caso $s\gamma$ **seja**
      $s\nu$: **se** $\overline{s}\nu \in \Sigma$ **então** *fracasso* **senão** $\Sigma := \Sigma \cup \{\gamma\}$ **fimse**
      $V\top$, $F\bot$: { regras verp e faln }
      $V\bot$, $F\top$: *fracasso*
      $V\neg\alpha$: A $:=$ A $\cup \{F\alpha\}$ { regra negp }
      $F\neg\alpha$: A $:=$ A $\cup \{V\alpha\}$ { regra negn }
      $V\alpha \wedge \beta$: A $:=$ A $\cup \{V\alpha, V\beta\}$ { regra conjp }
      $F\alpha \wedge \beta$: A $:=$ A $\cup \{$*escolha-nd*$(\{F\alpha, F\beta\})\}$ { não determinismo: conjn1 ou conjn2 }
      $V\alpha \vee \beta$: A $:=$ A $\cup \{$*escolha-nd*$(\{V\alpha, V\beta\})\}$ { não determinismo: disjp1 ou disjp2 }
      $F\alpha \vee \beta$: A $:=$ A $\cup \{F\alpha, F\beta\}$ { regra disjn }
      $V\alpha \rightarrow \beta$: A $:=$ A $\cup \{$*escolha-nd*$(\{F\alpha, V\beta\})\}$ { não determinismo: condp1 ou condp2 }
      $F\alpha \rightarrow \beta$: A $:=$ A $\cup \{V\alpha, F\beta\}$ { regra condn }
      $V\alpha \leftrightarrow \beta$: A $:=$ A $\cup$ *escolha-nd*$(\{\{V\alpha, V\beta\}, \{F\alpha, F\beta\}\})$ { não determinismo: bicp1 ou bicp2 }
      $F\alpha \leftrightarrow \beta$: A $:=$ A $\cup$ *escolha-nd*$(\{\{V\alpha, F\beta\}, \{F\alpha, V\beta\}\})$ { não determinismo: bicn1 ou bicn2 }
    fimcaso
  até A $= \emptyset$;
  sucesso
fim satA-nd

Figura 2.20: Procedimento satA-nd com anotação das regras de $\mathcal{S}_A$.

Como já foi dito na Seção 2.3.5, para que satA-nd verifique a satisfabilidade de um conjunto de fórmulas H fornecido como argumento, basta substituir o pseudocomando A $:= \{V\gamma\}$ por A $:= \{V\gamma \,|\, \gamma \in H\}$.

Um algoritmo similar a satA-nd é o baseado no sistema formal $\mathcal{S}_B$ apresentado na Seção 2.4.1, como já ressaltado. Neste caso, como já visto, a linguagem do sistema formal é $\mathcal{F}$ e as regras são as da Figura 2.19. No caso, um *conjunto de partida* é um conjunto $\Sigma \subseteq \mathcal{V} \cup \{\neg\nu \,|\, \nu \in \mathcal{V}\}$ tal que não há variável ν tal que, ambos, $\nu \in \Sigma$ e $\neg\nu \in \Sigma$. Novamente, apenas fórmulas de um conjunto de partida podem ser usadas em uma derivação sem terem sido obtidas por aplicação de uma das regras da Figura 2.19.

Exemplo 38 Segue uma derivação de $(p \rightarrow q) \wedge ((q \vee r) \rightarrow p)$ a partir de $\Sigma = \{\neg p, \neg q, \neg r\}$:

1. $\neg r$ (Σ)
2. $\neg q$ (Σ)
3. $\neg(q \vee r)$ (disjn 2,1)
4. $(q \vee r) \rightarrow p$ (condp1 3)
5. $\neg p$ (Σ)
6. $p \rightarrow q$ (condp1 5)

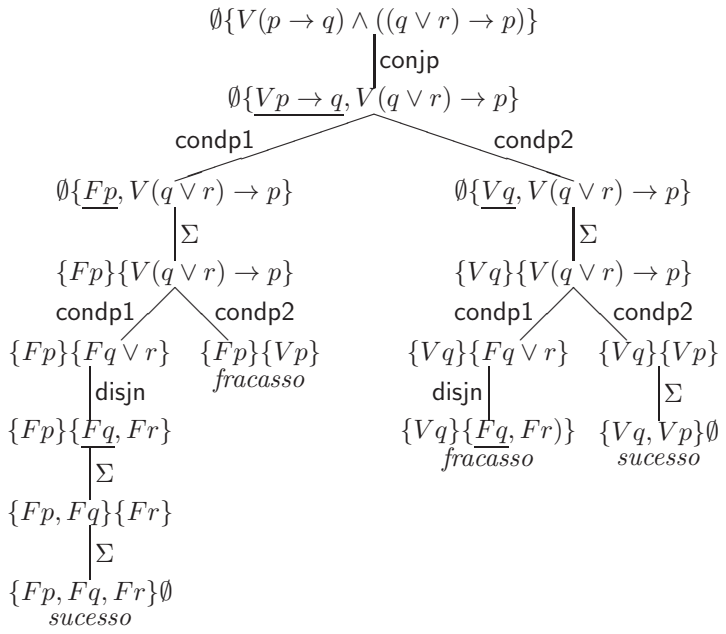


Figura 2.21: Árvore de computação para satA-nd$((p \to q) \wedge ((q \vee r) \to p))$.

7. $(p \to q) \wedge ((q \vee r) \to p)$ (conjp 6,4)

Logo, $(p \to q) \wedge ((q \vee r) \to p)$ é satisfatível. Veja a semelhança entre esta e a derivação do Exemplo 35. Como lá não se usou a regra negp, esta derivação pode ser obtida daquela pela simples eliminação dos símbolos V e substituição dos símbolos F por $\neg$. □

O algoritmo da Figura 2.22 é o mesmo que o da Figura 2.11, agora com anotações das regras de $\mathcal{S}_B$. Aqui também, o algoritmo *pode ser visto como* baseado em atribuições de valores verdade (nível conceitual) ou no sistema formal $\mathcal{S}_B$ (nível formal).

O exemplo a seguir apresenta a árvore de computação similar àquela do Exemplo 37 (Figura 2.21), porém usando satB-nd.

Exemplo 39 Seja novamente determinar a satisfabilidade de $(p \to q) \wedge ((q \vee r) \to p))$. Na Figura 2.23 está mostrada a árvore de computação para o algoritmo satB-nd da Figura 2.22. Ao ramo da esquerda, por exemplo, corresponde a derivação mostrada no Exemplo 38. □

Como dito na Seção 2.3.5, para que satB-nd verifique a satisfabilidade de um conjunto de fórmulas H, basta substituir o pseudomando A := $\{\gamma\}$ por A := H. Para que ele verifique a falseabilidade de uma fórmula γ, basta trocar a inicialização A := $\{\gamma\}$ por A := $\{\neg\gamma\}$.

Será mostrado a seguir que o sistema formal $\mathcal{S}_A$ é correto e completo para os problemas da satisfabilidade e faseabilidade:

proc-nd satB-nd(fórmula γ):
  conjunto de fórmulas A;
  conjunto de literais Σ;
  $\Sigma := \emptyset$;
  A := $\{\gamma\}$;
  repita
    γ := *escolha*(A); { escolhe próxima meta }
    A := A − $\{\gamma\}$;
    caso γ **seja**
      ν: **se** $\neg\nu \in \Sigma$ **então** *fracasso* **senão** $\Sigma := \Sigma \cup \{\nu\}$ **fimse**
      $\neg\nu$: **se** $\nu \in \Sigma$ **então** *fracasso* **senão** $\Sigma := \Sigma \cup \{\neg\nu\}$ **fimse**
      $\top$, $\neg\bot$: { regras verp e faln }
      $\bot$, $\neg\top$: *fracasso*
      $\neg\neg\alpha$: A := A $\cup \{\alpha\}$ { regra negn }
      $\alpha \wedge \beta$: A := A $\cup \{\alpha, \beta\}$ { regra conjp }
      $\neg(\alpha \wedge \beta)$: A := A $\cup \{$*escolha-nd*$(\{\neg\alpha, \neg\beta\})\}$ { não determinismo: conjn1 ou conjn2 }
      $\alpha \vee \beta$: A := A $\cup \{$*escolha-nd*$(\{\alpha, \beta\})\}$ { não determinismo: disjp1 ou disjp2 }
      $\neg(\alpha \vee \beta)$: A := A $\cup \{\neg\alpha, \neg\beta\}$ { regra disjn }
      $\alpha \rightarrow \beta$: A := A $\cup \{$*escolha-nd*$(\{\neg\alpha, \beta\})\}$ { não determinismo: condp1 ou condp2 }
      $\neg(\alpha \rightarrow \beta)$: A := A $\cup \{\alpha, \neg\beta\}$ { regra condn }
      $\alpha \leftrightarrow \beta$: A := A $\cup$ *escolha-nd*$(\{\{\alpha, \beta\}, \{\neg\alpha, \neg\beta\}\})$ { não determinismo: bicp1 ou bicp2 }
      $\neg(\alpha \leftrightarrow \beta)$: A := A $\cup$ *escolha-nd*$(\{\{\alpha, \neg\beta\}, \{\neg\alpha, \beta\}\})$ { não determinismo: bicn1 ou bicn2 }
    fimcaso
  até A = $\emptyset$;
  sucesso
fim satB-nd

Figura 2.22: Procedimento satB-nd com anotações das regras de $\mathcal{S}_B$.

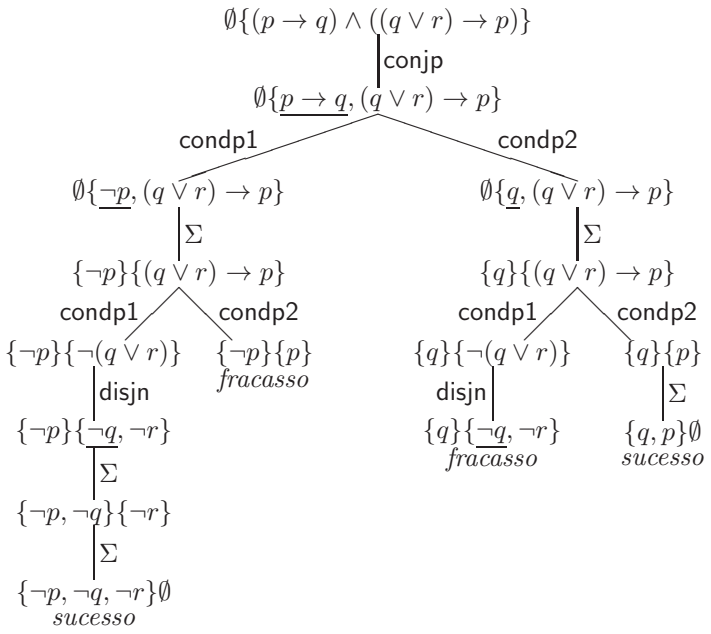


Figura 2.23: Árvore de computação para satB-nd$((p \to q) \wedge ((q \vee r) \to p))$.

- correto: se $\Sigma \Rightarrow s\gamma$ para algum conjunto de partida Σ, então existe i tal que $v^i(\gamma) = s$; e
- completo: se existe i tal que $v^i(\gamma) = s$, então $\Sigma \Rightarrow s\gamma$ para algum conjunto de partida Σ.

Para isto serão usados os Teoremas 5 e 6. As demonstrações da correção e da completude de $\mathcal{S}_B$ podem ser feitas de forma análoga. Nas demonstrações será usado o conjunto Σ_i assim definido: dada uma interpretação i, $\Sigma_i = \{s\nu \,|\, \nu^i = s\}$. Note que com isso, para toda variável $\nu \in \mathcal{V}$, $s\nu \in \Sigma_i$ se, e somente se, $\overline{s}\nu \notin \Sigma_i$. Note também que se Σ é um conjunto de partida tal que $s\nu \notin \Sigma$, $\overline{s}\nu \notin \Sigma$ e $\Sigma \Rightarrow s\gamma$, sendo ν uma variável, então ambos $\Sigma \cup \{s\nu\} \Rightarrow \gamma$ e $\Sigma \cup \{\overline{s}\nu\} \Rightarrow \gamma$. Pode-se mostrar também que se Σ é um conjunto de partida qualquer e $\Sigma \Rightarrow s\gamma$, então existe i tal que $\Sigma \subseteq \Sigma_i$ e $\Sigma_i \Rightarrow s\gamma$ (por exemplo, aquele i tal que $\nu^i = F$ para toda $\nu \in \mathcal{V}$ tal que $s\nu \notin \Sigma$ e $\overline{s}\nu \notin \Sigma$).

Teorema 7 *(Correção) Se existe um conjunto de partida Σ tal que $\Sigma \Rightarrow s\gamma$, então existe i tal que $v^i(\gamma) = s$.*

Prova

Suponha que existe um conjunto de partida Σ tal que $\Sigma \Rightarrow s\gamma$. Seja uma interpretação i_0 tal que $\Sigma \subseteq \Sigma_{i_0}$. Então $\Sigma_{i_0} \Rightarrow s\gamma$. Pelo Teorema 5 segue-se, então, que $v^{i_0}(\gamma) = s$. Logo, existe i tal que $v^i(\gamma) = s$. Portanto, se existe um conjunto de partida Σ tal que $\Sigma \Rightarrow \gamma$, então existe i tal que $v^i(\gamma) = s$. □

Agora a demonstração do teorema da completude.

Teorema 8 *(Completude) Se existe i tal que $v^i(\gamma) = s$, então existe um conjunto de partida Σ tal que $\Sigma \Rightarrow s\gamma$.*

Prova

Suponha que existe i tal que $v^i(\gamma) = s$. Seja i_0 uma tal interpretação. Então, pelo Teorema 6, segue-se que $\Sigma_{i_0} \Rightarrow s\gamma$. Logo, existe um conjunto de partida Σ tal que $\Sigma \Rightarrow s\gamma$. Portanto, se existe i tal que $v^i(\gamma) = s$, então existe um conjunto de partida Σ tal que $\Sigma \Rightarrow s\gamma$. □

Na Figura 2.13 da Seção 2.3.5 foi apresentada uma versão determinística de satA-nd, que determina se uma fórmula é satisfatível por construção de modelo. O mesmo algoritmo pode ser visto como baseado no uso do sistema formal $\mathcal{S}_A$. Assim como para os algoritmos não determinísticos satA-nd e satB-nd, os determinísticos respectivos praticamente não apresentam diferença, não apenas em termos de desempenho, mas também em termos de apresentação.

Uma característica importante em sistemas formais é que repetições de expressões em derivações são absolutamente desnecessárias: mantendo-se apenas uma e substituindo-se todas as referências às outras por referências à mantida, obtém-se uma derivação legítima mais curta. Chega-se, então, à conclusão: *existe uma derivação de uma fórmula satisfatível α, em $\mathcal{S}_B$, cujo comprimento é, no máximo, igual a $|subf(\alpha)|$*. O mesmo resultado se aplica a $\mathcal{S}_A$, visto que para cada $\beta \in subf(\alpha)$, se $s\beta$ é derivável, então $\overline{s}\beta$ não é (pela correção de $\mathcal{S}_A$). Da Definição 4, segue-se de imediato que $|subf(\alpha)|$ é menor ou igual ao número de *ocorrências* de conectivos e variáveis em α. Como os algoritmos satA-nd e satB-nd trabalham justamente pela obtenção das subfórmulas que vão permitindo o uso das regras, vê-se que suas computações de sucesso têm desempenho linear com relação ao número de *ocorrências* de conectivos e variáveis na fórmula passada como argumento.

Uma fórmula é satisfatível se, e somente se, há uma construção da mesma via $\mathcal{S}_B$ a partir de um conjunto Σ que contenha literais representando um modelo para a fórmula. Consistentemente, um ramo de sucesso na árvore de computação da aplicação do algoritmo satB-nd a uma fórmula α mostra, quando visto do final (a folha, rotulada com $\Sigma\emptyset$) para o início (a raiz, rotulada com $\emptyset\{\alpha\}$, como tal construção é feita passo a passo mediante aplicações das regras de $\mathcal{S}_B$ (veja a Figura 2.23, por exemplo). Se todas as arestas (a_1, a_2) de uma árvore computação rotuladas com Σ forem eliminadas, assim como o vértice a_2, e toda aresta (a_2, b) for substituída por (a_1, b), e, além disso, todo rótulo ΣA de cada aresta restante for substituído por $\Sigma \cup A$, tem-se uma árvore de computação *simplificada* que ainda mostra como um conjunto Σ evolui passo a passo, pela aplicação das regras, em direção à fórmula que rotula a raiz. Segue um exemplo.

Exemplo 40 A árvore de computação *simplificada* correspondente à da Figura 2.23 está mostrada na Figura 2.24. Veja como o conjunto $\Sigma = \{\neg p, \neg q, \neg r\}$ evolui para $\{(p \rightarrow q) \wedge ((q \vee r) \rightarrow p)\}$ no ramo de sucesso da esquerda invertido, sendo que à direita de cada conjunto está a regra que o originou:

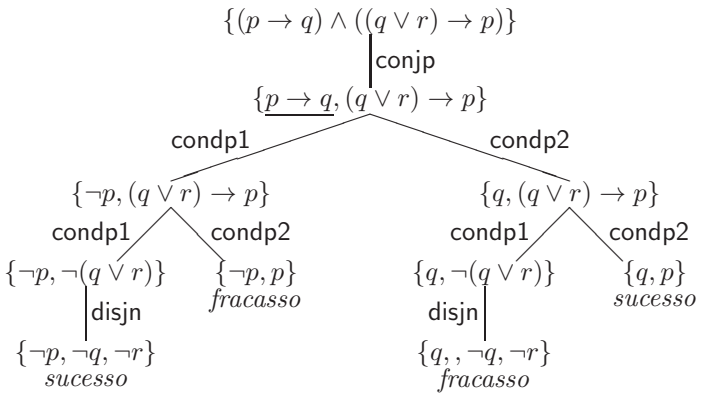


Figura 2.24: Árvore de computação simplificada para satB-nd$((p \to q) \wedge ((q \vee r) \to p))$.

1. $\{\neg p, \neg q, \neg r\}$ (Σ)
2. $\{\neg p, \neg(q \vee r)\}$ (disjn)
3. $\{\neg p, (q \vee r) \to p\}$ (condp1)
4. $\{p \to q, (q \vee r) \to p\}$ (condp1)
5. $\{(p \to q) \wedge ((q \vee r) \to p)\}$ (conjp)

Note, ainda, como as premissas para a aplicação de cada regra estão todas no conjunto anterior. □

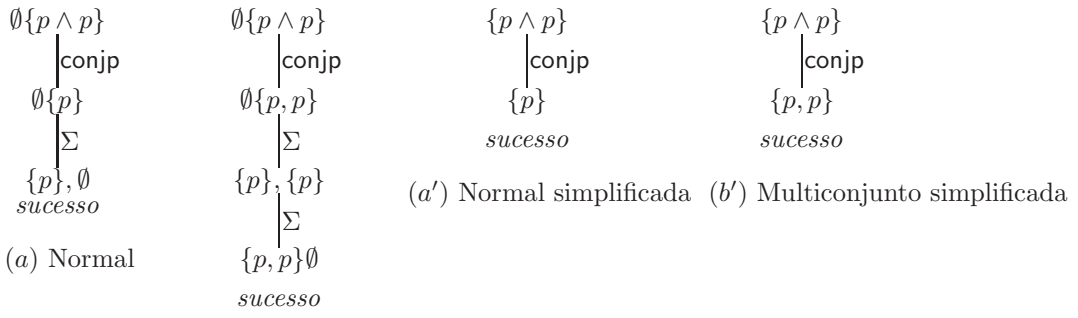
Ora, o exemplo anterior inspira a construção de um novo sistema formal, denominado $\mathcal{S}_C$, que permite construir um conjunto satisfatível de fórmulas a partir de um conjunto (satisfatível) de literais que represente um modelo para o conjunto de fórmulas. Depois, em seções posteriores, sistemas formais similares serão construídos para derivação de contradições e tautologias, assim como para formalização do importante conceito de consequência lógica.

Embora o sistema formal $\mathcal{S}_C$, a ser descrito, seja diferente de $\mathcal{S}_B$, o próprio algoritmo satB-nd[16] *poderá ser visto como* baseado em $\mathcal{S}_C$; ou seja, derivações em $\mathcal{S}_C$ podem ser obtidas por meio de pequenos acréscimos a satB-nd. Antes de mais nada, no entanto, uma correspondência mais direta entre ramos de sucesso e derivações em $\mathcal{S}_C$ é obtida se se considerar os conjuntos A e Σ de satB-nd como *multiconjuntos*;[17] ou seja, se se considerar as operações de união no algoritmo como de união de multiconjuntos. Um pequeno exemplo ilustra o que se acabou de dizer.

Exemplo 41 A Figura 2.25 mostra as árvores de computação para satB-nd$(p \wedge p)$ quando se considera A e Σ como conjuntos (versão (a)) ou como multiconjuntos (versão (b). Mostra também as árvores respectivas no modo simplificado, (a') e (b'). A derivação que se obtém em $\mathcal{S}_B$ corresponde à árvore (a):

[16] Particularmente, aquele modificado para reconhecer a satisfabilidade de um *conjunto* de fórmulas.

[17] Multiconjuntos são conjuntos em que elementos pode aparecer mais de uma vez. Assim, por exemplo, $\{p\} \neq \{p, p\}$.



(*a*) Normal

(*b*) Multiconjuntos

(*a′*) Normal simplificada (*b′*) Multiconjunto simplificada

Figura 2.25: Árvores de computação para satB-nd($p \wedge p$).

1. p (Σ)
2. $p \wedge p$ (conjp 1,1)

A derivação que se obterá em $\mathcal{S}_C$ corresponderá à árvore (b') como ficará claro a seguir. □

Uma primeira providência na construção de $\mathcal{S}_C$ é representar os multiconjuntos $\mathsf{A} \cup \Sigma$ referidos anteriormente por meio de uma palavra (sequência). Assim, a linguagem de $\mathcal{S}_C$ é o conjunto das sequências de fórmulas separadas por vírgulas, incluindo a sequência vazia; mais precisamente, é o conjunto de palavras gerado pelas regras gramaticais:

$$S \rightarrow S_1 \mid \lambda$$
$$S_1 \rightarrow S_1{,}\alpha \mid \alpha$$

em que $\alpha \in \mathcal{F}$. Cada palavra é denominada uma *sequência*; em particular, λ, que é a palavra vazia, é chamada de *sequência vazia*. Denotando-se por $c(\Psi)$ o conjunto das fórmulas de uma sequência Ψ (em particular, $c(\lambda) = \emptyset$ e $c(p, p) = \{p\}$), $\mathcal{S}_C$ será tal que se $\emptyset \Rightarrow \Psi$ então $c(\Psi)$ é satisfatível, o que expressa a *correção* do sistema formal. Por outro lado, valerá também a *completude*: se $c(\Psi)$ for satisfatível, então $\emptyset \Rightarrow \Psi$.

O único axioma de $\mathcal{S}_C$ é a sequência vazia:

$$\mathsf{Ax}: \quad \lambda$$

Portanto, toda derivação começa com λ.

As regras mostram como de uma sequência (premissa) se obtém outra sequência (conclusão). Há dois tipos de regras, uma *estrutural* e as restantes *lógicas*. A regra estrutural, denominada regra da *troca*, faz com que a ordem das fórmulas em uma sequência não tenha relevância, como em multiconjuntos (Ψ_1 e Ψ_2 são subsequências quaisquer):[18]

[18] A vírgula será utilizada, tanto para separar fórmulas em uma sequência, quanto como operador (associativo) para formar uma (sub)sequência a partir de duas outras. Nas regras, ela é utilizada como operador.

$$\mathsf{tr}: \ \frac{\Psi_1, \alpha, \beta, \Psi_2}{\Psi_1, \beta, \alpha, \Psi_2}$$

Com isto, qualquer fórmula de uma sequência pode, por meio de aplicações da regra tr, ser deslocada para qualquer posição da sequência.

Para cada conectivo, e também para as variáveis, há regras (lógicas) que definem como introduzí-los para obter uma conclusão. Tais regras, com exceção de duas regras para produção de literais apresentadas a seguir, podem ser vistas como inspiradas naquelas do sistema $\mathcal{S}_B$, como sugere o Exemplo 40. Só que agora cada regra tem uma única premissa, e a preservação de satisfabilidade pode ser assim enunciada: uma regra de premissa Ψ_1 e conclusão Ψ_2 é tal que $c(\Psi_1)$ é satisfatível se, e somente se, $c(\Psi_2)$ é satisfatível. Com isso, é obtido um sistema formal correto. Por outro lado, sendo o conjunto de regras "exaustivo", prevendo a introdução de um conectivo ou variável positiva (no caso de conectivo, como conectivo principal) e negativamente (no caso de conectivo, como negação de uma fórmula que o tenha como conectivo principal), é obtido um sistema formal completo.

As regras lógicas de $\mathcal{S}_C$ estão mostradas na Figura 2.26. As regras para variáveis dizem em que situação se pode acrescentar um literal l a uma sequência Ψ. Elas se baseiam no fato de que $c(\Psi)$ é satisfatível se e somente se $c(\Psi) \cup \{l\}$ é satisfatível, *caso* Ψ *só contenha literais (ou seja vazia) e* $\overline{l}$ *não ocorra em* Ψ. Note que com elas consegue-se formar (a partir de λ) qualquer conjunto de partida como em $\mathcal{S}_B$. Para que isso aconteça, se lá o conjunto de partida é usado k vezes em uma derivação para acesso aos literais $l_1, \ldots, l_k$ (nao necessariamente distintos), em $\mathcal{S}_C$ a derivação começa com k aplicações das regras varp/varn.

Nenhum conjunto que contenha $\bot$ ou $\neg\top$ é satisfatível. Portanto, não existe regra que permita concluir que um conjunto é satisfatível, caso ele contenha $\bot$ ou $\neg\top$. Por outro lado, conjuntos que contenham $\top$ ou $\neg\bot$ são satisfatíveis, caso seja possível satisfazer o resto dos mesmos, o que justifica as regras para *verum* e *falsum*. Para a negação há também apenas uma regra baseada em que $c(\Psi, \alpha)$ é satisfatível se e somente se $c(\Psi, \neg\neg\alpha)$ é satisfatível. As regras para conclusão de $\Psi, \neg\alpha$, em que o conectivo principal de α não seja $\neg$, $\top$, nem $\bot$, são explicadas a seguir.

Para cada conectivo binário há dois tipos de regras, como no sistema formal $\mathcal{S}_B$, aquelas que permitem concluir uma fórmula com o conectivo como principal (sufixo p) e as que permitem concluir a negação de uma fórmula com o conectivo como principal (sufixo n). Em particular, as regras para a conjunção se baseiam no fato de que (a) $c(\Psi, \alpha \wedge \beta)$ é satisfatível se e somente se $c(\Psi, \alpha, \beta)$ é satisfatível (regra conjp) e (b) $c(\Psi, \neg(\alpha \wedge \beta))$ é satisfatível se e somente se $c(\Psi, \neg\alpha)$ é satisfatível ou $c(\Psi, \neg\beta)$ é satisfatível (regras conjn1 e conjn2). Já as regras para a disjunção se baseiam no fato de que (a) $c(\Psi, \alpha \vee \beta)$ é satisfatível se e somente se $c(\Psi, \alpha)$ é satisfatível ou $c(\Psi, \beta)$ é satisfatível (regras disjp1 e disjp2) e (b) $c(\Psi, \neg(\alpha \vee \beta))$ é satisfatível se e somente se $c(\Psi, \neg\alpha, \neg\beta)$ é satisfatível (regra disjn). De forma similar, as regras para a condicional se baseiam no fato de que (a) $c(\Psi, \alpha \to \beta)$ é satisfatível se e somente se $c(\Psi, \neg\alpha)$ é satisfatível ou $c(\Psi, \beta)$ é satisfatível (regras condp1 e condp2) e (b) $c(\Psi, \neg(\alpha \to \beta))$ é satisfatível se e somente se $c(\Psi, \alpha, \neg\beta)$ é satisfatível (regra condpn). Finalmente, as regras para a

Variáveis

varp : $\dfrac{\Psi}{\Psi, \nu}$ se Ψ é vazia ou contém apenas literais e não contém $\neg\nu$

varn : $\dfrac{\Psi}{\Psi, \neg\nu}$ se Ψ é vazia ou contém apenas literais e não contém ν

Verum e falsum

verp : $\dfrac{\Psi}{\Psi, \top}$ faln : $\dfrac{\Psi}{\Psi, \neg\bot}$

Negação

negn : $\dfrac{\Psi, \alpha}{\Psi, \neg\neg\alpha}$

Conjunção

conjp : $\dfrac{\Psi, \alpha, \beta}{\Psi, \alpha \wedge \beta}$ conjn1 : $\dfrac{\Psi, \neg\alpha}{\Psi, \neg(\alpha \wedge \beta)}$ conjn2 : $\dfrac{\Psi, \neg\beta}{\Psi, \neg(\alpha \wedge \beta)}$

Disjunção

disjp1 : $\dfrac{\Psi, \alpha}{\Psi, \alpha \vee \beta}$ disjp2 : $\dfrac{\Psi, \beta}{\Psi, \alpha \vee \beta}$ disjn : $\dfrac{\Psi, \neg\alpha, \neg\beta}{\Psi, \neg(\alpha \vee \beta)}$

Condicional

condp1 : $\dfrac{\Psi, \neg\alpha}{\Psi, \alpha \to \beta}$ condp2 : $\dfrac{\Psi, \beta}{\Psi, \alpha \to \beta}$ condn : $\dfrac{\Psi, \alpha, \neg\beta}{\Psi, \neg(\alpha \to \beta)}$

Bicondicional

bicp1 : $\dfrac{\Psi, \alpha, \beta}{\Psi, \alpha \leftrightarrow \beta}$ bicp2 : $\dfrac{\Psi, \neg\alpha, \neg\beta}{\Psi, \alpha \leftrightarrow \beta}$ bicn1 : $\dfrac{\Psi, \alpha, \neg\beta}{\Psi, \neg(\alpha \leftrightarrow \beta)}$ bicn2 : $\dfrac{\Psi, \neg\alpha, \beta}{\Psi, \neg(\alpha \leftrightarrow \beta)}$

Figura 2.26: Regras lógicas do sistema formal $\mathcal{S}_C$.

bicondicional se baseiam em que (a) $c(\Psi, \alpha \leftrightarrow \beta)$ é satisfatível se e somente se $c(\Psi, \alpha, \beta)$ é satisfatível (regras bicp1 e bicp2) ou $c(\Psi, \neg\alpha, \neg\beta)$ é satisfatível e (b) $c(\Psi, \neg(\alpha \leftrightarrow \beta))$ é satisfatível se e somente se $c(\Psi, \alpha, \neg\beta)$ é satisfatível ou $c(\Psi, \neg\alpha, \beta)$ é satisfatível (regras bicn1 e bicn2).

Uma característica importante de $\mathcal{S}_C$ é que suas derivações são *lineares*, ou seja, toda sequência em uma derivação, exceto a primeira (λ) é obtida aplicando-se uma regra com premissa sendo a sequência anterior. Por causa disso, nos exemplos apenas o nome da regra será anotado à direita de cada conclusão obtida, ficando implícito que a premissa é a sequência anterior.

As derivações de uma (sequência de uma) fórmula em $\mathcal{S}_C$ são maiores do que as da mesma fórmula em $\mathcal{S}_B$ por três motivos:

- em $\mathcal{S}_C$ os literais que representam o modelo são gerados a partir do axioma λ, enquanto que em $\mathcal{S}_B$ eles são obtidos diretamente do conjunto de partida, o que já dá um passo a mais de derivação;
- em $\mathcal{S}_C$ pode haver aplicações da regra estrutural tr para posicionar a(s) fórmula(s)

para aplicação de uma regra lógica, o que não é necessário em $\mathcal{S}_B$; e

- em $\mathcal{S}_C$ pode haver aplicações de regras para produção de fórmulas já produzidas na mesma sequência, o que não é necessário em $\mathcal{S}_B$.

O próximo exemplo ilustra os dois primeiros aspectos acima.

Exemplo 42 Uma derivação de $(p \to q) \wedge ((q \vee r) \to p)$ a partir de $\Sigma = \{\neg p, \neg q, \neg r\}$ no sistema formal $\mathcal{S}_B$ foi apresentada no Exemplo 38. Segue uma derivação da mesma fórmula no sistema formal $\mathcal{S}_C$:

1. λ	(Ax)
2. $\neg r$	(varn)
3. $\neg q, \neg r$	(varn)
4. $\neg p, \neg q, \neg r$	(varn)
5. $\neg p, \neg(q \vee r)$	(disjn)
6. $\neg p, (q \vee r) \to p$	(condp1)
7. $(q \vee r) \to p, \neg p$	(tr)
8. $(q \vee r) \to p, p \to q$	(condp1)
9. $p \to q, (q \vee r) \to p$	(tr)
10.$(p \to q) \wedge ((q \vee r) \to p)$	(conjp)

Veja a semelhança entre esta e a derivação do Exemplo 38, notando a três aplicações da regra varn no início da derivação, visto que lá o conjunto de partida é referenciado três vezes. Enquanto lá o comprimento da derivação é 7, aqui é 10; a mais: o uso de Ax e dois usos de tr. □

O exemplo a seguir mostra um exemplo de derivação de uma fórmula α em $\mathcal{S}_B$, obtida de uma árvore de computação resultante da execução de satB-nd. Em seguida, produz uma derivação correspondente bem maior no sistema $\mathcal{S}_C$, sendo o aumento devido, não apenas ao uso de Ax e às aplicações de tr, mas também devido a repetições da mesma fórmula em uma sequência, que correspondem ao uso repetido da mesma fórmula em $\mathcal{S}_B$.

Exemplo 43 Uma árvore de computação para satB-nd$((p \wedge (q \vee r)) \wedge ((q \vee r) \vee p))$ está mostrada na Figura 2.27. Correspondendo ao ramo mais a esquerda, tem-se a derivação em $\mathcal{S}_B$:

1. q	(Σ)
2. $q \vee r$	(disjp 1)
3. p	(Σ)
4. $(q \vee r) \vee p$	(disjp 2)
5. $p \wedge (q \vee r)$	(conjp 3,2)
6. $(p \wedge (q \vee r)) \wedge ((q \vee r) \vee p)$	(conjp 5,4)

Veja que a subfórmula $q \vee r$ foi usada duas vezes como premissa, para produção das fórmulas 4 e 5. Em $\mathcal{S}_C$ tem-se a derivação:

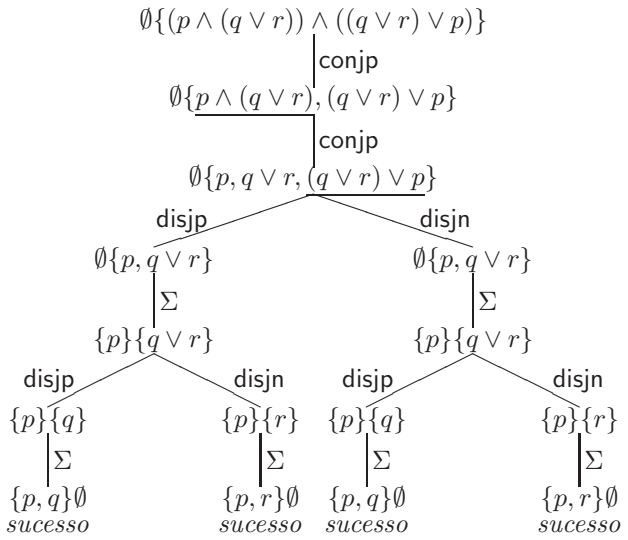


Figura 2.27: Árvore de computação para satB-nd$((p \wedge (q \vee r)) \wedge ((q \vee r) \wedge p))$.

1. λ	(Ax)
2. p	(varp)
3. p, q	(varp)
4. p, q, q	(varp)
5. $p, q, q \vee r$	(disjp)
6. $p, q \vee r, q$	(tr)
7. $p, q \vee r, q \vee r$	(disjp)
8. $p, q \vee r, (q \vee r) \vee p$	(disjp)
9. $p, (q \vee r) \vee p, q \vee r$	(tr)
10.$(q \vee r) \vee p, p, q \vee r$	(tr)
11.$(q \vee r) \vee p, p \wedge (q \vee r)$	(conjp)
12.$p \wedge (q \vee r), (q \vee r) \vee p$	(tr)
13.$(p \wedge (q \vee r)) \wedge ((q \vee r) \vee p)$	(conjp)

A produção de fórmulas repetidas nessa derivação é feita via varp, para produzir nova fórmula q na sequência 4, e disjp para produzir nova fórmula $q \vee r$ na sequência 7. Somando-se essas duas operações com o uso de Ax e as quatro trocas, tem-se o excesso com relação à derivação em $\mathcal{S}_B$. □

O seguinte lema, que expressa as justificativas dadas anteriormente para as regras da Figura 2.26, é utilizado nas demonstrações da correção e completude de $\mathcal{S}_C$.

Lema 1 *Seja* Ψ *uma sequência.*

1. *Se l é um literal, Ψ só contém literais (ou é vazia) e $\overline{l}$ não ocorre em Ψ, então $c(\Psi, l)$ é satisfatível se e somente se $c(\Psi)$ é satisfatível.*

2. *$c(\Psi, \top)$ é satisfatível se e somente se $c(\Psi)$ é satisfatível, e $c(\Psi, \neg\bot)$ é satisfatível se e somente se $c(\Psi)$ é satisfatível.*

3. *$c(\Psi, \neg\neg\alpha)$ é satisfatível se e somente se $c(\Psi, \alpha)$ é satisfatível.*

4. *$c(\Psi, \alpha \wedge \beta)$ é satisfatível se e somente se $c(\Psi, \alpha, \beta)$ é satisfatível.*

5. *$c(\Psi, \neg(\alpha \wedge \beta))$ é satisfatível se e somente se $c(\Psi, \neg\alpha)$ **ou** $c(\Psi, \neg\beta)$ é satisfatível.*

6. *$c(\Psi, \alpha \vee \beta)$ é satisfatível se e somente se $c(\Psi, \alpha)$ **ou** $c(\Psi, \beta)$ é satisfatível.*

7. *$c(\Psi, \neg(\alpha \vee \beta))$ é satisfatível se e somente se $c(\Psi, \neg\alpha, \neg\beta)$ é satisfatível.*

8. *$c(\Psi, \alpha \rightarrow \beta)$ é satisfatível se e somente se $c(\Psi, \neg\alpha)$ **ou** $c(\Psi, \beta)$ é satisfatível.*

9. *$c(\Psi, \neg(\alpha \rightarrow \beta))$ é satisfatível se e somente se $c(\Psi, \alpha, \neg\beta)$ é satisfatível.*

10. *$c(\Psi, \alpha \leftrightarrow \beta)$ é satisfatível se somente se $c(\Psi, \alpha, \beta)$ **ou** $c(\Psi, \neg\alpha, \neg\beta)$ é satisfatível.*

11. *$c(\Psi, \neg(\alpha \leftrightarrow \beta))$ é satisfatível se e somente se $c(\Psi, \alpha, \neg\beta)$ **ou** $c(\Psi, \neg\alpha, \beta)$ é satisfatível.*

Prova

1. Sejam l um literal, Ψ só com literais (ou vazia) e suponha que $\overline{l}$ não ocorre em Ψ. Naturalmente, toda interpretação que satisfaz $c(\Psi, l)$, satisfaz $c(\Psi)$. Por outro lado, se uma interpretação i satisfaz $c(\Psi)$, como $\overline{l} \notin c(\Psi)$, fazendo-se i' tal que $i'(\nu) = i(\nu)$ para toda variável ν que ocorra em Ψ e $v^{i'}(l) = V$ tem-se i' que satisfaz $c(\Psi, l)$.

2. Se uma interpretação satisfaz $c(\Psi, \top)$ ou $c(\Psi, \neg\bot)$, claramente satisfaz $c(\Psi)$. E se uma interpretação i satisfaz $c(\Psi)$, i satisfaz $c(\Psi, \top)$ e $c(\Psi, \neg\bot)$, pois $v^i(\top) = v^i(\neg\bot) = V$.

3. Segue do fato que i satisfaz $c(\Psi, \neg\neg\alpha)$ se e somente se i satisfaz $c(\Psi, \alpha)$, pois $v^i(\alpha) = v^i(\neg\neg\alpha)$.

4. Segue do fato que i satisfaz $c(\Psi, \alpha \wedge \beta)$ se e somente se i satisfaz $c(\Psi, \alpha, \beta)$.

5. $(\rightarrow)$
   Suponha que $c(\Psi, \neg(\alpha \wedge \beta))$ é satisfatível. Seja i tal que i satisfaz Ψ e $v^i(\neg(\alpha \wedge \beta)) = V$. Então $v^i(\alpha \wedge \beta) = F$ e, portanto, $v^i(\alpha) = F$ ou $v^i(\beta) = F$; ou ainda, $v^i(\neg\alpha) = V$ ou $v^i(\neg\beta) = V$. Logo, $c(\Psi, \neg\alpha)$ é satisfatível ou $c(\Psi, \neg\beta)$ é satisfatível.
   $(\leftarrow)$

Suponha que $c(\Psi, \neg\alpha)$ é satisfatível ou $c(\Psi, \neg\beta)$ é satisfatível. Seja i tal que i satisfaz $c(\Psi)$ e $v^i(\neg\alpha) = V$, ou ainda, $v^i(\alpha) = F$; neste caso, $v^i(\neg(\alpha \wedge \beta)) = \ominus(v^i(\alpha \wedge \beta)) = \ominus(F) = V$; e de forma similar se $v^i(\neg\beta) = V$, então $v^i(\neg(\alpha \wedge \beta)) = V$. Logo, $c(\Psi, \neg(\alpha \wedge \beta))$ é satisfatível.

O resto da demonstração é sugerido como exercício. □

O teorema da correção mostra que se $\emptyset \Rightarrow \Psi$, então $c(\Psi)$ é satisfatível.

Teorema 9 *(Teorema da correção) se* $\emptyset \Rightarrow \Psi$ *então* $c(\Psi)$ *é satisfatível.*

Prova

Por indução no comprimento da derivação.[19] Suponha que o resultado valha para derivações com menos de n sequências. Seja $n \geq 0$ e suponha que exista uma derivação de comprimento n de Ψ. Deve-se mostrar que Ψ é satisfatível. Se $n = 0$, se $\lambda \Rightarrow \Psi$, então Ψ é axioma, ou seja, $\Psi = \lambda$; e $c(\lambda) = \emptyset$ e $\emptyset$ é satisfatível. Por outro lado, se $n > 0$, o último passo da derivação terá sido dado aplicando-se uma das regras:

- varp. Então a premissa utilizada no último passo é da forma Ψ', sendo $\Psi = \Psi', \nu$ e Ψ' só contém literais e não contém $\neg\nu$. Como a derivação de Ψ' tem menos de n sequências, pela hipótese de indução $c(\Psi')$ é satisfatível. Conclui-se então, pelo Lema 1(1), que $c(\Psi', \nu)$ é satisfatível.

- varn. Então a premissa utilizada no último passo é da forma Ψ', sendo $\Psi = \Psi', \neg\nu$ e Ψ' só contém literais e não contém ν. Como a derivação de Ψ' tem menos de n sequências, pela hipótese de indução $c(\Psi')$ é satisfatível. Daí conclui-se, pelo Lema 1(1), que $c(\Psi', \nu)$ é satisfatível.

- verp. Então a premissa utilizada no último passo é da forma Ψ', sendo $\Psi = \Psi', \top$. Como a derivação de Ψ' tem menos de n sequências, pela hipótese de indução $c(\Psi')$ é satisfatível. Daí conclui-se, pelo Lema 1(2), que $c(\Psi', \top)$ é satisfatível.

- faln. Então a premissa utilizada no último passo é da forma Ψ', sendo $\Psi = \Psi', \neg\bot$. Como a derivação de Ψ' tem menos de n sequências, pela hipótese de indução $c(\Psi')$ é satisfatível. Segue-se então, pelo Lema 1(2), que $c(\Psi', \neg\bot)$ é satisfatível.

- negn. Então a premissa utilizada no último passo é da forma Ψ', sendo $\Psi = \Psi', \neg\neg\alpha$. Como a derivação de Ψ' tem menos de n sequências, pela hipótese de indução $c(\Psi')$ é satisfatível. De onde se conclui, pelo Lema 1(3), que $c(\Psi', \neg\neg\alpha)$ é satisfatível.

E assim por diante, até esgotar as regras. □

O teorema da completude, que diz que se $c(\Psi)$ é satisfatível então $\emptyset \Rightarrow \Psi$, vem a seguir. Sua demonstração usa indução sobre o grau $g(\Psi)$, de uma sequência Ψ, definido a partir do grau $g(\alpha)$ de uma fórmula α.

[19] Em conformidade com o dito do Capítulo 1, o comprimento de uma derivação é o número de expressões (no caso, sequências) da mesma menos um.

Definição 14 *O grau de uma fórmula é assim definido:*

(a) $g(\alpha) = 0$ *se* α *é literal,* $\top$, $\neg\top$, $\bot$ *ou* $\neg\bot$*;*

(b) $g(\neg\neg\alpha) = g(\neg\alpha) + 1$*;*

(c) $g(\alpha c\beta) = g(\alpha) + g(\beta) + 1$*, para todo conectivo binário* c*.*

(d) $g(\neg(\alpha c\beta)) = g(\neg\alpha) + g(\neg\beta) + 1$*, para todo conectivo binário* c*.*

Já o grau de uma sequência é assim definido (sobrecarregando g*):*

(a) $g(\lambda) = 0$*;*

(b) $g(\Psi, \alpha) = g(\Psi) + g(\alpha)$*.*

Exemplo 44 como $g(p) = 0$, $g(\neg p \vee q) = g(\neg p) + g(q) + 1 = 0 + 0 + 1 = 1$ e $g(\neg(p \leftrightarrow \neg q)) = g(\neg p) + g(\neg\neg q) + 1 = 0 + g(\neg q) + 1 + 1 = 0 + 0 + 2 = 2$, segue-se que o grau da sequência $\neg(p \leftrightarrow \neg q), p, \neg p \vee q$ é $g(\neg(p \leftrightarrow \neg q)) + g(p) + g(\neg p \vee q) = 2 + 0 + 1 = 3$.

Teorema 10 *(Teorema da completude) Se* $c(\Psi)$ *é satisfatível, então* $\lambda \Rightarrow \Psi$*.*

Prova

Por indução no grau de Ψ. Suponha que o resultado vale se o grau de Ψ é menor que n. Seja Ψ uma sequência de grau $n \geq 0$ e suponha que $c(\Psi)$ seja satisfatível. Deve-se mostrar que $\lambda \Rightarrow \Psi$. Se $n = 0$, tem-se os casos:

- $\Psi = \lambda$. $c(\lambda) = \emptyset$, que é satisfatível. Como λ é axioma, é derivável.
- Ψ só contém literais e/ou $\top$s e/ou $\neg\top$s e/ou $\bot$s e/ou $\neg\bot$s (no mínimo um). Como $c(\Psi)$ é satisfatível, Ψ não pode conter literais complementares, nem $\neg\top$s nem $\bot$s. Logo, é possível construir uma derivação de Ψ a partir de λ aplicando-se em sequência: as regras varp e/ou varn de modo a acrescentar os literais um a um, depois as regras verp, uma vez para cada $\top$, e finalmente as regras faln, uma vez para cada $\neg\bot$.

Logo, nos dois casos $\lambda \Rightarrow \Psi$. Por outro lado, se $n > 0$, seja $\gamma \in c(\Psi)$ uma fórmula tal que $g(\gamma) > 0$ e suponha que $c(\Psi)$ é satisfatível. Seja Ψ' com exatamente as mesmas fórmulas de Ψ, mas com γ no final da sequência. Se existem k fórmulas após γ em Ψ, então Ψ' pode ser derivada a partir de Ψ aplicando-se a regra tr k vezes. Como $c(\Psi') = c(\Psi)$, $c(\Psi')$ é satisfatível. Como $g(\gamma) > 0$, tem-se os casos:

- $\gamma = \neg\neg\alpha$. Seja Ψ'' tal que $\Psi' = \Psi'', \neg\neg\alpha$. Como $c(\Psi')$ é satisfatível, pelo Lema 1(3) $c(\Psi'', \alpha)$ é satisfatível. Como $g(\Psi'', \alpha) < g(\Psi'', \neg\neg\alpha) = n$, pela hipótese de indução, segue-se que $\lambda \Rightarrow \Psi'', \alpha$. Usando-se a regra negn, estende-se a derivação de Ψ'', α obtendo-se $\Psi'', \neg\neg\alpha$.

- $\gamma = \alpha \wedge \beta$. Então seja Ψ'' tal que $\Psi' = \Psi'', \alpha \wedge \beta$. Como $c(\Psi')$ é satisfatível, pelo Lema 1(4) $c(\Psi'', \alpha, \beta)$ é satisfatível. Como esta última sequência tem grau menor que n (pois $g(\alpha, \beta) = g(\alpha \wedge \beta) - 1$), pela hipótese de indução, $\lambda \Rightarrow \Psi'', \alpha, \beta$. A partir da derivação de Ψ'', α, β, usando-se a regra regra conjp, obtém-se uma derivação de $\Psi'', \alpha \wedge \beta$.

- $\gamma = \neg(\alpha \wedge \beta)$. Então seja Ψ'' tal que $\Psi' = \Psi'', \neg(\alpha \wedge \beta)$. Como $c(\Psi')$ é satisfatível, pelo Lema 1(5) segue-se que $c(\Psi'', \neg\alpha)$ ou $c(\Psi'', \neg\beta)$ é satisfatível. Como as sequências $\Psi'', \neg\alpha$ e $\Psi'', \neg\beta$ têm grau menor que n (pois $g(\neg(\alpha \wedge \beta)) = g(\neg\alpha) + g(\neg\beta) + 1$), pela hipótese de indução $\lambda \Rightarrow \Psi'', \neg\alpha$ ou $\lambda \Rightarrow \Psi'', \neg\beta$. A partir da derivação de $\Psi'', \neg\alpha$ (se $c(\Psi'', \neg\alpha)$ é satisfatível) ou de $\Psi'', \neg\beta$ (se $c(\Psi'', \neg\beta)$ é satisfatível), usando-se a regra conjn1 ou conjn2, obtém-se uma derivação de $\Psi'', \neg(\alpha \wedge \beta)$.

E assim por diante, até esgotar todos os casos. □

Como visto no Exemplo 43, às vezes a mesma subfórmula deve aparecer mais de uma vez em uma mesma sequência, o que pode ocasionar um aumento da derivação. Tal aumento pode ser evitado (caso a fórmula repetida não seja literal) se for acrescentada a $\mathcal{S}_C$ uma regra estrutural que permita a *duplicação* de uma fórmula:

$$\mathsf{dup}: \ \frac{\Psi_1, \alpha}{\Psi_1, \alpha, \alpha}$$

Tendo em vista as regras varp e varn, pode-se exigir que α não seja literal.

O exemplo a seguir mostra o uso de dup para obter uma derivação mais curta que a do Exemplo 43.

Exemplo 45 Uma derivação de $(p \wedge (q \vee r)) \wedge ((q \vee r) \vee p)$ em $\mathcal{S}_C$ com a regra dup:

1. λ	(Ax)
2. p	(varp)
3. p, q	(varp)
4. $p, q \vee r$	(disjp)
5. $p, q \vee r, q \vee r$	(dup)
6. $p, q \vee r, (q \vee r) \vee p$	(disjp)
7. $p, (q \vee r) \vee p, q \vee r$	(tr)
8. $(q \vee r) \vee p, p, q \vee r$	(tr)
9. $(q \vee r) \vee p, p \wedge (q \vee r)$	(conjp)
10. $p \wedge (q \vee r), (q \vee r) \vee p$	(tr)
11. $(p \wedge (q \vee r)) \wedge ((q \vee r) \vee p)$	(conjp)

Note como as repetições na derivação sem a regra dup no Exemplo 43, via varp (para produzir nova fórmula q) e disjp (para produzir nova fórmula $q \vee r$), são economizadas na derivação com dup, que produz uma cópia de $q \vee r$ diretamente. Encarando as sequências como multiconjuntos, de modo a evitar a aplicação explícita de tr:

1. λ	(Ax)
2. p	(varp)
3. p, q	(varp)
4. $p, q \vee r$	(disjp)
5. $p, q \vee r, q \vee r$	(dup)
6. $p, q \vee r, (q \vee r) \vee p$	(disjp)
7. $(q \vee r) \vee p, p \wedge (q \vee r)$	(conjp)
8. $(p \wedge (q \vee r)) \wedge ((q \vee r) \vee p)$	(conjp)

Assim, encarando sequências como multiconjuntos e permitindo duplicações, obtém-se uma derivação em $\mathcal{S}_C$ de comprimento maior do que em $\mathcal{S}_B$ em apenas duas unidades, devido, essencialmente, ao uso do axioma de $\mathcal{S}_C$ e à necessidade de uma duplicação de subfórmula. □

Agora é importantíssimo ressaltar: assim como o algoritmo satB-nd pode ser reformulado, através do acréscimo de pseudocomandos, de forma que produza uma derivação em $\mathcal{S}_B$, ele pode também ser reformulado para produzir uma derivação em $\mathcal{S}_C$ com ou sem a aplicação da regra dup. Além disso, como é de se esperar, a partir de uma derivação em $\mathcal{S}_B$, pode-se produzir uma em $\mathcal{S}_C$ e vice-versa (veja exercício ao final da seção).

Como já dito, a transformação de um algoritmo não determinístico em determinístico pelo mecanismo padrão de *backtracking* leva a um comportamento exponencial no pior caso. Assim, para que o algoritmo atinja um desempenho que permita a resolução de problemas de interesse prático, que, de outra forma, não poderiam ser resolvidos, é importante, por exemplo, o uso de boas heurísticas de escolha, a proposição de métodos de *backtracking* mais "inteligentes" e de técnicas de recuperação das informações necessárias em caso de *backtracking*. Heurísticas de escolha e métodos de *backtracking* têm sido desenvolvidos para obtenção de algoritmos determinísticos, tanto para o problema da satisfabilidade, quanto para prova automática, em contextos mais simples propiciados por fórmulas em forma normal, tipicamente *forma normal conjuntiva* ou *forma normal disjuntiva*. Por exemplo, o algoritmo DPLL (para satisfabilidade) trabalha com fórmulas na forma normal conjuntiva. E a maioria dos algoritmos determinísticos para satisfabilidade[20] são baseados em DPLL e propõem técnicas para lidar com os problemas mencionados. Antes da apresentação de métodos para a obtenção das principais formas normais e do algoritmo DPLL, serão apresentados, na próxima seção, sistemas formais para gerar contradições e tautologias em estilo parecido com aquele de $\mathcal{S}_C$.

Exercícios

1. Para cada árvore de computação obtida no Exercício 2 da Seção 2.3.5:

a) rotule as arestas da árvore relativa a satA-nd com as regras de S_A; escolha um dos ramos de *sucesso* mais curtos, se houver algum ramo de sucesso, e exiba a derivação em $\mathcal{S}_A$ que demonstre que a fórmula é satisfatível;

[20] Os denominados solucionadores SAT (*SAT solvers*).

b) rotule as arestas da árvore relativa a satB-nd com as regras de S_B; escolha um dos ramos de *sucesso* mais curtos, se houver algum ramo de sucesso, e exiba a derivação em $\mathcal{S}_B$ que demonstre que a fórmula é satisfatível.

2. Para cada árvore de computação obtida no Exercício 3 da Seção 2.3.5:

a) rotule as arestas da árvore relativa a falA-nd com as regras de S_A; escolha um dos ramos de *sucesso* mais curtos, se houver algum ramo de sucesso, e exiba a derivação em $\mathcal{S}_A$ que demonstre que a fórmula é falseável;

b) rotule as arestas da árvore relativa a falB-nd com as regras de S_B; escolha um dos ramos de *sucesso* mais curtos, se houver algum ramo de sucesso, e exiba a derivação em $\mathcal{S}_B$ que demonstre que a fórmula é falseável.

3. Quantas computações a mais terá a execução do algoritmo satA-nd caso ele faça

$$\mathsf{A} := \mathsf{A} \cup \{\textsf{escolha-nd}(\{V\alpha, F\beta\}), \textsf{escolha-nd}(\{(F,\alpha),(V,\beta)\})\})$$

no caso em que $\mathsf{s}\gamma = V\alpha \leftrightarrow \beta$, em vez de

$$\mathsf{A} := \mathsf{A} \cup \{\textsf{escolha-nd}(\{\{V\alpha, V\beta\}\{F\alpha, F\beta\}\})\}$$

considerando apenas a subárvore de computação rotulada $V\alpha \leftrightarrow \beta$?

4. Altere as demonstrações dos teoremas da correção e da completude do sistema formal $\mathcal{S}_A$ para satisfabilidade para o sistema formal $\mathcal{S}_B$.

5. Anote as regras do sistema formal $\mathcal{S}_A$ no texto do algoritmo determinístico satA da Figura 2.13 e escreva também uma versão determinística de satB-nd.

6. Supondo a existência de um algoritmo determinístico satisfaz que, dada uma fórmula, determina se ela é satisfatível, escreva procedimentos que chamem satisfaz para determinar se uma fórmula é:

(a) falseável;

(b) tautológica;

(c) contraditória;

(d) contingente.

7. Complete a demonstração do Lema 1.

8. Complete a demonstração do Teorema 9.

9. Complete a demonstração do Teorema 10.

10. Obtenha um sistema formal similar a $\mathcal{S}_B$, porém para falseabilidade, de tal forma que $\Sigma \Rightarrow \gamma$ para algum Σ se e somente se $v^i(\gamma) = F$ para alguma i.

11. Para cada fórmula do Exercício 1 exiba a dedução em $\mathcal{S}_C$ correspondente à dedução lá exibida no sistema $\mathcal{S}_B$

12. Mostre que os conjuntos a seguir são satisfatíveis encontrando uma dedução de uma sequência das fórmulas do mesmo em $\mathcal{S}_C$:

a) $\{\top, p \vee \neg q, p \to \bot\}$

b) $\{p \leftrightarrow q, p \to \neg q\}$

c) $\{\top \to r, p \to \bot, p \vee (r \to q)\}$

13. Mostre como, a partir de uma derivação em $\mathcal{S}_C$ de sequência de uma única fórmula, se produz uma derivação em $\mathcal{S}_B$. Em particular, como obter uma mais curta?

14. Mostre como, a partir de uma derivação em $\mathcal{S}_B$, se produz uma em $\mathcal{S}_C$.

2.4.3 Verificação de tautologias e contradições no nível formal

Para saber se uma fórmula α é insatisfatível, basta usar o algoritmo determinístico satA (ou satB). Por exemplo: α é insatisfatível se e somente se $\mathsf{satA}(\alpha) = \mathit{falso}$. Já em termos das versões não determinísticas, α é insatisfatível se e somente se a árvore de computação para α só contém computações de fracasso. A análise das árvores de computação cujas computações são todas de fracasso leva à concepção de um sistema formal para derivação de fórmulas insatisfatíveis como explicado a seguir.

Primeiramente, a partir de uma árvore de computação obtém-se uma outra *simplificada*, como mostrado na Seção 2.4.2, da seguinte forma: cada vértice da árvore original é rotulado com $\Sigma \cup \mathsf{A}$, no lugar do par $\Sigma\mathsf{A}$; em seguida, toda sequência linear de vértices idênticos, aqueles ligados por arestas rotuladas com Σ (ou seja em que apenas há transferência de literal de A para Σ), viram apenas um.[21]

Exemplo 46 Uma árvore de computação simplificada, como definida na Seção 2.4.2, correspondente à execução de satB-nd com argumento $\neg((p \leftrightarrow (\neg q \vee r)) \to (\neg p \to q))$ está mostrada na Figura 2.28. Como os dois ramos da árvore representam computações de fracasso, conclui-se que $\neg((p \leftrightarrow (\neg q \vee r)) \to (\neg p \to q))$ é insatisfatível. Este exemplo será usado a seguir para auxiliar algumas explicações. □

Como no sistema $\mathcal{S}_C$ da seção anterior, os conjuntos $\mathsf{A} \cup \Sigma$ do algoritmo satB-nd serão expressos, no sistema formal ora em construção, como *sequências*. Assim, como em $\mathcal{S}_C$, a linguagem do sistema será o conjunto das sequências de fórmulas (separadas por vírgulas). Para que uma sequência dessas represente um multiconjunto, também como em $\mathcal{S}_C$, o sistema formal conterá a *regra estrutural* de *troca*, aqui replicada:

$$\mathsf{tr}: \ \frac{\Psi_1, \alpha, \beta, \Psi_2}{\Psi_1, \beta, \alpha, \Psi_2}$$

[21] Tais árvores de computação simplificadas correspondem aos *tableaux fechados* nos denominados sistemas de *tableaux semânticos*.

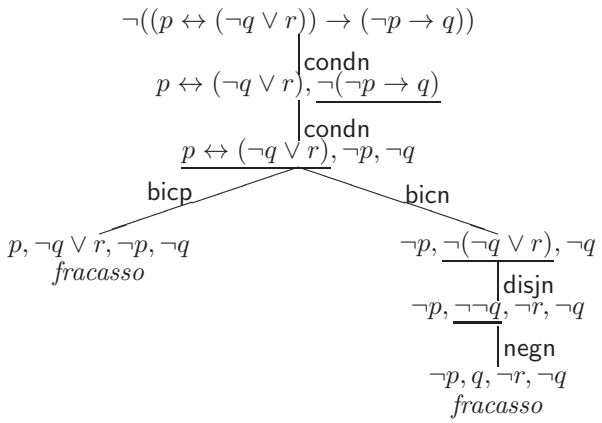


Figura 2.28: Execução de satB-nd para $\neg((p \leftrightarrow (\neg q \vee r)) \rightarrow (\neg p \rightarrow q))$.

Assim como na seção anterior, dada uma sequência Ψ da linguagem, o conjunto das fórmulas em Ψ será referido por $c(\Psi)$ (em particular, $c(\lambda) = \emptyset$). Assim, por exemplo, $c(\neg p, p \vee q, \neg p) = \{\neg p, p \vee q\}$. Aqui também o termo Ψ, com ou sem índice, será usado para significar uma sequência. Com isto, por exemplo, se $\alpha, \beta \in \mathcal{F}$, uma sequência Ψ, α representa o conjunto $c(\Psi, \alpha) = c(\Psi) \cup \{\alpha\}$, enquanto que $\Psi, \alpha, \neg\beta$ representa $c(\Psi, \alpha, \neg\beta) = c(\Psi) \cup \{\alpha, \neg\beta\}$, e assim por diante.

Observando-se a árvore da Figura 2.28, a ideia é novamente que ela represente uma derivação constituída das sequências que rotulam seus vértices, sendo a sequência final aquela que rotula a raiz. Note que o rótulo de uma folha Ψ da árvore de computação contém um literal e seu complemento, sendo que, por isso, $c(\Psi)$ é insatisfatível. No sistema formal haverá, consequentemente, um esquema de axiomas que diz exatamente isso (ν é variável):

$$\mathsf{Ax}: \quad \Psi, \nu, \neg\nu$$

No caso da árvore da Figura 2.28, os axiomas são as sequências $p, \neg q, \vee r, \neg p, \neg q$ e $\neg p, q, \neg r, \neg q$. Dada a regra tr, qualquer permutação das fórmulas constituintes desses axiomas é derivável.

Notando-se como o(s) filho(s) de um vértice em uma árvore de computação são construídos, verifica-se que as sequências que rotulam os filhos representam conjuntos insatisfatíveis se, e somente se, a que rotula o pai é insatisfatível, o que segue do Lema 1, como explicado mais abaixo; isso *propaga* a insatisfabilidade das folhas (axiomas) para a raiz. Por exemplo, as regras para a conjunção podem ser assim justificadas, através da preservação de insatisfabilidade (as premissas são insatisfatíveis se e somente se a conclusão também é): (a) $c(\Psi, \alpha \wedge \beta)$ é insatisfatível se e somente se $c(\Psi, \alpha, \beta)$ é insatisfatível e (b) $c(\Psi, \neg(\alpha \wedge \beta))$ é insatisfatível se e somente se $c(\Psi, \neg\alpha)$ e $c(\Psi, \neg\beta)$ são insatisfatíveis:

$$\mathsf{conjp}: \ \frac{\Psi, \alpha, \beta}{\Psi, \alpha \wedge \beta} \qquad \mathsf{conjn}: \ \frac{\Psi, \neg\alpha \quad \Psi, \neg\beta}{\Psi, \neg(\alpha \wedge \beta)}$$

Verum e falsum

$$\mathsf{falp}: \frac{}{\Psi, \bot} \qquad \mathsf{vern}: \frac{}{\Psi, \neg\top}$$

Negação

$$\mathsf{negn}: \frac{\Psi, \alpha}{\Psi, \neg\neg\alpha}$$

Conjunção

$$\mathsf{conjp}: \frac{\Psi, \alpha, \beta}{\Psi, \alpha \wedge \beta} \qquad \mathsf{conjn}: \frac{\Psi, \neg\alpha \quad \Psi, \neg\beta}{\Psi, \neg(\alpha \wedge \beta)}$$

Disjunção

$$\mathsf{disjp}: \frac{\Psi, \alpha \quad \Psi, \beta}{\Psi, \alpha \vee \beta} \qquad \mathsf{disjn}: \frac{\Psi, \neg\alpha, \neg\beta}{\Psi, \neg(\alpha \vee \beta)}$$

Condicional

$$\mathsf{condp}: \frac{\Psi, \neg\alpha \quad \Psi, \beta}{\Psi, \alpha \to \beta} \qquad \mathsf{condn}: \frac{\Psi, \alpha, \neg\beta}{\Psi, \neg(\alpha \to \beta)}$$

Bicondicional

$$\mathsf{bicp}: \frac{\Psi, \alpha, \beta \quad \Psi, \neg\alpha, \neg\beta}{\Psi, \alpha \leftrightarrow \beta} \qquad \mathsf{bicn}: \frac{\Psi, \alpha, \neg\beta \quad \Psi, \neg\alpha, \beta}{\Psi, \neg(\alpha \leftrightarrow \beta)}$$

Figura 2.29: Regras lógicas do sistema formal $\mathcal{S}_I$.

Chega-se, então, às regras da Figura 2.29. O sistema formal com a linguagem das sequências, axiomas dados por Ax, a regra estrutural tr e as regras lógicas da Figura 2.29 será denominado $\mathcal{S}_I$.

Como mencionado acima, o próprio Lema 1, que justifica as regras de $\mathcal{S}_C$, justifica também as regras de $\mathcal{S}_I$. Para isso ficar claro, basta reexpressar o lema em termos de insatisfabilidade (no lugar de satisfabilidade). Por exemplo, para a conjunção, o Lema 1 diz:

4. $c(\Psi, \alpha \wedge \beta)$ é satisfatível se e somente se $c(\Psi, \alpha, \beta)$ é satisfatível.

5. $c(\Psi, \neg(\alpha \wedge \beta))$ é satisfatível se e somente se $c(\Psi, \neg\alpha)$ **ou** $c(\Psi, \neg\beta)$ é satisfatível.

Reexpressando em termos de insatisfabilidade, tem-se:

4'. $c(\Psi, \alpha \wedge \beta)$ é insatisfatível se e somente se $c(\Psi, \alpha, \beta)$ é insatisfatível.

5'. $c(\Psi, \neg(\alpha \wedge \beta))$ é insatisfatível se e somente se $c(\Psi, \neg\alpha)$ **e** $c(\Psi, \neg\beta)$ são insatisfatíveis.

O que justifica claramente as regras de $\mathcal{S}_I$ para a conjunção.

Às regras da Figura 2.29 podem ser acrescentadas também as regras

$$\mathsf{verp}: \frac{\Psi}{\Psi, \top} \qquad \mathsf{faln}: \frac{\Psi}{\Psi, \neg\bot}$$

mas tais regras são desnecessárias, visto que $\top$ e $\neg\bot$ podem ser introduzidas via axiomas (instâncias de Ax).

Nos exemplos a seguir, as aplicações da regra tr não serão explicitadas; as sequências serão tratadas como se fossem multiconjuntos. Com isso, as sequências obtidas não são exatamente aquelas da árvore de computação de satB-nd, mas da árvore que seria obtida se o algoritmo usasse união de multiconjuntos no lugar de união de conjuntos. No entanto, a derivação é perfeitamente obtenível da árvore obtida pelo uso de satB-nd.

Exemplo 47 Agora a árvore da Figura 2.28 pode ser vista como apresentando uma derivação no sistema formal $\mathcal{S}_I$. No formato usado até agora a derivação pode ser apresentada assim, sem mostrar aplicações da regra tr:

1. $\neg q, \neg q \vee r, p, \neg p$ (Ax)
2. $\neg p, \neg p, \neg r, q, \neg q$ (Ax)
3. $\neg p, \neg p, \neg r, \neg q, \neg\neg q$ (negn 2)
4. $\neg p, \neg p, \neg q, \neg(\neg q \vee r)$ (disjn 3)
5. $\neg p, \neg q, p \leftrightarrow (\neg q \vee r)$ (bicp 1,4)
6. $p \leftrightarrow (\neg q \vee r), \neg(\neg p \to q)$ (condn 5)
7. $\neg((p \leftrightarrow (\neg q \vee r)) \to (\neg p \to q))$ (condn 6)

Veja como aparecem repetições de literais não presentes nos conjuntos da árvore da Figura 2.28. □

Segue um exemplo bem simples mostrando que $\neg(p \to p)$ é uma contradição.

Exemplo 48 Para mostrar que $\neg(p \to p)$ é insatisfatível pelo sistema $\mathcal{S}_I$, deve-se encontrar uma derivação da sequência $\neg(p \to p)$. Existe uma única regra que permite concluir tal sequência, a regra condn com $c(\Psi) = \emptyset$, o que leva à derivação:

1. $p, \neg p$ (Ax)
2. $\neg(p \to p)$ (condn 1)

Isto mostra indiretamente, também, que $p \to p$ é uma tautologia. □

No próximo exemplo, $\neg\bot$ é introduzindo via axioma para mostrar que $\neg((p \to p) \vee \bot)$ é uma contradição.

Exemplo 49 Segue uma derivação que mostra que $\neg((p \to p) \vee \bot)$ é insatisfatível:

1. $\neg\bot, p, \neg p$ (Ax)
2. $\neg\bot, \neg(p \to p)$ (condn 1)
3. $\neg((p \to p) \vee \bot)$ (disjn 2)

□

Agora uma derivação que mostra que $\{p \to (q \to r), q, \neg(p \to r)\}$ é insatisfatível.

Exemplo 50 Para mostrar que $\{p \to (q \to r), q, \neg(p \to r)\}$ é insatisfatível deve ser encontrada uma derivação, em $\mathcal{S}_I$, de $p \to (q \to r), q, \neg(p \to r)$ (ou de alguma sequência com as mesmas fórmulas). Existem duas regras que se aplicadas levam a esta sequência: as regras condn e condp. Para aplicar a primeira (com $c(\Psi) = \{p \to (q \to r), q\}$) a premissa é $p \to (q \to r), q, p, \neg r$. Para deduzir esta, a única regra aplicável é condp com $c(\Psi) = \{q, p, \neg r\}$ e com as premissas $q \to r, q, p, \neg r$ e $q, p, \neg p, \neg r$. A última sequência é uma instância de Ax e o outro pode ser deduzido por aplicação de condp a partir duas instâncias de Ax, como mostra a derivação:

1. $q, p, \neg q, \neg r$	(Ax)
2. $q, p, r, \neg r$	(Ax)
3. $q \to r, q, p, \neg r$	(condp 1,2)
4. $q, p, \neg p, \neg r$	(Ax)
5. $p \to (q \to r), q, p, \neg r$	(condp 3,4)
6. $p \to (q \to r), q, \neg(p \to r)$	(condn 5)

Logo, o conjunto é insatisfatível. □

Embora a derivação encontrada no exemplo anterior não seja lá muito clara, sua obtenção pelo uso do sistema formal $\mathcal{S}_I$ é trivial. A obtenção da derivação é trivial porque existe *uma única* regra aplicável em cada passo na produção da derivação em reverso, isto é, partindo-se da sequência a derivar e indo em direção aos axiomas. Por outro lado, há casos em que ocorre uma *explosão combinatória*, propiciada por uma proliferação de fórmulas pela aplicação das regras de duas premissas conjn, disjp, condp, bicn e bicp. Em outras palavras, o que seria uma árvore de computação para o algoritmo de teste de satisfabilidade satB-nd, agora se torna, toda ela, uma derivação no sistema $\mathcal{S}_I$. Afinal, basta um ramo de sucesso para determinar satisfabilidade, mas é necessário que todo ramo seja de fracasso para atestar a insatisfabilidade.

O sistema $\mathcal{S}_I$ serve para mostrar que um *conjunto de fórmulas* é insatisfatível, podendo ser demonstrado que:

- *é correto*: se $\emptyset \Rightarrow \Psi$ então $c(\Psi)$ é insatisfatível; e
- *é completo*: se $c(\Psi)$ é insatisfatível então $\emptyset \Rightarrow \Psi$.

As demonstrações de correção e completude podem ser feitas com relativa facilidade, inspiradas naquelas do sistema $\mathcal{S}_C$ da seção anterior.

Um algoritmo baseado em $\mathcal{S}_I$ pode ser obtido de uma versão determinística de satB-nd, satB, em que se inverte as respostas verdadeiro (satisfatível)/falso (insatisfatível). (Uma versão análoga, obtida de adaptação de satA-nd, será vista na Seção 2.8.4, Figura 2.45: ela é o mesmo que satA, da Figura 2.13, porém com as saídas verdadeiro/falso invertidas.) Na Figura 2.30 é apresentada uma versão obtida de satB, porém sem o uso de variável para o conjunto Σ. No algoritmo desta figura, a variável Ψ representa, basicamente o mesmo que $\Sigma \cup \mathsf{A}$ do algoritmo satB. Ele deixa transparecer mais claramente o sistema formal subjacente, $\mathcal{S}_I$. Veja que o algoritmo apresentado

```
proc insat(conjunto de fórmulas Ψ) retorna {verdadeiro, falso}:
   conjunto de conjuntos de fórmulas CS;
   CS := {Ψ};
   repita
      Ψ := escolha(CS); CS := CS − {Ψ};
      se Ψ tem um literal e seu complemento então
         { Ψ é axioma; continue }
      senãose Ψ contém fórmula que não é ⊤ ou ¬⊥ então
         γ := escolhanl(Ψ); { escolhe fórmula que não é literal, ⊤ ou ¬⊥ } Ψ := Ψ − {γ};
         caso γ seja
            ¬⊤,⊥:      { regras vern e falp }
            ¬¬α:       CS := CS ∪ {Ψ ∪ {α}} { regra negn }
            α ∧ β:     CS := CS ∪ {Ψ ∪ {α, β}} { regra conjp }
            ¬(α ∧ β):  CS := CS ∪ {Ψ ∪ {¬α}, Ψ ∪ {¬β}} { regra conjn }
            α ∨ β:     CS := CS ∪ {Ψ ∪ {α}, Ψ ∪ {β}} { regra disjp }
            ¬(α ∨ β):  CS := CS ∪ {Ψ ∪ {¬α, ¬β}} { regra disjn }
            α → β:     CS := CS ∪ {Ψ ∪ {¬α}, Ψ ∪ {β}} { regra condp }
            ¬(α → β):  CS := CS ∪ {Ψ ∪ {α, ¬β}} { regra condn }
            α ↔ β:     CS := CS ∪ {Ψ ∪ {α, β}, Ψ ∪ {¬α, ¬β}} { regra bicp }
            ¬(α ↔ β):  CS := CS ∪ {Ψ ∪ {¬α, β}, Ψ ∪ {α, ¬β}} { regra bicn }
         fimcaso;
      senão { Ψ contém só literais e/ou ⊤s e/ou ¬⊥s e não é axioma}
         retorne falso
   fimse
   até CS = ∅;
   retorne verdadeiro
fim insat
```

Figura 2.30: Um algoritmo para verificação de insatisfabilidade.

só apresenta o não determinismo provocado pelas escolhas representadas pelos pseudocomandos *escolha* e *escolhanl*; embora tais escolhas possam influir no desempenho, elas não levam a computações de fracasso. As derivações em $\mathcal{S}_I$ são obtidas pelo algoritmo facilmente em reverso, porque existe *uma única* regra aplicável em cada passo na produção da derivação em reverso. No entanto, há casos em que ocorre uma *explosão combinatória*, propiciada por uma proliferação de sequências pela aplicação das regras de duas premissas, no caso, conjn, disjp, condp, bicp e bicn. O algoritmo constrói implicitamente a árvore de computação para satB-nd (a floresta, no caso de conjunto); enquanto um ramo de sucesso atesta satisfabilidade, é necessário que todo ramo seja de fracasso para atestar que a fórmula (ou conjunto) é insatisfatível.

Um conjunto de fórmulas H é insatisfatível se e somente se qualquer interpretação *falseia* alguma fórmula de H, ou ainda, se e somente se qualquer interpretação *satisfaz* alguma fórmula de $\{\neg\alpha \,|\, \alpha \in H\}$. Em particular, uma fórmula α é insatisfatível se e somente se qualquer interpretação *falseia* α, ou seja, se e somente se qualquer interpretação *satisfaz* $\neg\alpha$, isto é, $\neg\alpha$ é uma tautologia. Assim, se H é finito, $H = \{\alpha_1, \ldots, \alpha_n\}$, H é insatisfatível se e somente se $\alpha_1 \wedge \ldots \wedge \alpha_n$ é insatisfatível se e somente se $\neg\alpha_1 \vee \ldots \vee \neg\alpha_n$ é tautologia se e somente se qualquer interpretação *satisfaz* alguma fórmula de $\{\neg\alpha \,|\, \alpha \in H\}$. A seguir será apresentado um sistema formal, de-

nominado $\mathcal{S}_T$, que trata de conjuntos H para os quais qualquer interpretação *satisfaz* alguma fórmula de H (e, em particular, trata de tautologias). A partir das justificativas das regras de $\mathcal{S}_I$, pode-se obter as justificativas para a regras de $\mathcal{S}_T$ e, obviamente, as próprias regras de $\mathcal{S}_T$. Por exemplo, as justificativas para as regras da conjunção para insatisfabilidade (obtidas a partir do Lema 1) são:

4'. $c(\Psi, \alpha \wedge \beta)$ é insatisfatível se e somente se $c(\Psi, \alpha, \beta)$ é insatisfatível.

5'. $c(\Psi, \neg(\alpha \wedge \beta))$ é insatisfatível se e somente se $c(\Psi, \neg\alpha)$ **e** $c(\Psi, \neg\beta)$ são insatisfatíveis.

Denotando-se por Ψ' uma sequência constituída de, e apenas, fórmulas complementares às de Ψ,[22], tem-se que:

$c(\Psi)$ é insatisfatível se e somente se qualquer interpretação satisfaz alguma fórmula de $c(\Psi')$.

Dadas as equivalências **4'** e **5'**, tem-se então:

4". qualquer interpretação satisfaz alguma fórmula de $c(\Psi, \neg(\alpha \wedge \beta))$ se e somente se qualquer interpretação satisfaz alguma fórmula de $c(\Psi, \neg\alpha, \neg\beta)$.

5". qualquer interpretação satisfaz alguma fórmula de $c(\Psi, \alpha \wedge \beta)$ se e somente se qualquer interpretação satisfaz alguma fórmula de $c(\Psi, \alpha)$ **e** de $c(\Psi, \beta)$.

As equivalências **4"**e **5"** justificam as regras para a conjunção:

$$\mathsf{conjp}: \frac{\Psi, \alpha \quad \Psi, \beta}{\Psi, \alpha \wedge \beta} \qquad \mathsf{conjn}: \frac{\Psi, \neg\alpha, \neg\beta}{\Psi, \neg(\alpha \wedge \beta)}$$

De forma análoga obtém-se o resto das regras mostradas na Figura 2.31.

O esquema de axiomas, embora o mesmo de $\mathcal{S}_I$, diz agora que um conjunto de fórmulas que tenha um literal e seu complemento é tal que qualquer interpretação satisfaz alguma fórmula dele:

$$\mathsf{Ax}: \quad \Psi, \nu, \neg\nu$$

em que ν é uma variável proposicional.

Às regras da Figura 2.31 podem ser acrescentadas também as regras

$$\mathsf{falp}: \frac{\Psi}{\Psi, \bot} \qquad \mathsf{vern}: \frac{\Psi}{\Psi, \neg\top}$$

mas tais regras são desnecessárias, visto que $\bot$ e $\neg\top$ podem ser introduzidas via axiomas.

O exemplo a seguir altera a derivação do Exemplo 47, ainda sem mostrar aplicações da regra tr. Note como todas as fórmulas são negadas e que agora a vírgula em sequências pode ser interpretada como disjunção.

[22] Uma fórmula α é complementar a β se β e´ $\neg\alpha$ ou se α é $\neg\beta$.

Verum e falsum

$$\textsf{verp}: \frac{}{\Psi, \top} \qquad \textsf{faln}: \frac{}{\Psi, \neg\bot}$$

Negação

$$\textsf{negn}: \frac{\Psi, \alpha}{\Psi, \neg\neg\alpha}$$

Conjunção

$$\textsf{conjp}: \frac{\Psi, \alpha \quad \Psi, \beta}{\Psi, \alpha \wedge \beta} \qquad \textsf{conjn}: \frac{\Psi, \neg\alpha, \neg\beta}{\Psi, \neg(\alpha \wedge \beta \ )}$$

Disjunção

$$\textsf{disjp}: \frac{\Psi, \alpha, \beta}{\Psi, \alpha \vee \beta} \qquad \textsf{disjn}: \frac{\Psi, \neg\alpha \quad \Psi\neg\beta}{\Psi, \neg(\alpha \vee \beta)}$$

Condicional

$$\textsf{condp}: \frac{\Psi, \neg\alpha, \beta}{\Psi, \alpha \to \beta} \qquad \textsf{condn}: \frac{\Psi, \alpha \quad \Psi, \neg\beta}{\Psi, \neg(\alpha \to \beta \ )}$$

Bicondicional

$$\textsf{bicp}: \frac{\Psi, \neg\alpha, \beta \quad \Psi, \alpha, \neg\beta}{\Psi, \alpha \leftrightarrow \beta} \qquad \textsf{bicn}: \frac{\Psi, \alpha, \beta \quad \Psi, \neg\alpha, \neg\beta}{\Psi, \neg(\alpha \leftrightarrow \beta)}$$

Figura 2.31: Regras lógicas do sistema formal $\mathcal{S}_T$.

Exemplo 51 Uma demonstração que $(p \leftrightarrow (\neg q \vee r)) \to (\neg p \to q)$ é uma tautologia no sistema $\mathcal{S}_T$ seria dada pela derivação:

1. $\neg p, \neg(\neg q \vee r), p, q$ (Ax)
2. $p, \neg q, r, q$ (Ax)
3. $p, \neg q \vee r, q$ (disjp 2)
4. $\neg(p \leftrightarrow (\neg q \vee r)), p, q$ (bicp 1,3)
5. $\neg(p \leftrightarrow (\neg q \vee r)), \neg p \to q$ (condp 4)
6. $(p \leftrightarrow (\neg q \vee r)) \to (\neg p \to q)$ (condp 5)

As sequências são as obtidas no Exemplo 47 negando-se as fórmulas e as regras são trocadas: as positivas pelas respectivas negativas e vice-versa. □

Agora, $p \to p$, demonstrada indiretamente como tautologia no Exemplo 49, é demonstrada mediante o sistema formal $\mathcal{S}_T$.

Exemplo 52 Para mostrar que $p \to p$ é tautologia pelo sistema $\mathcal{S}_T$, existe uma única regra que permite concluir tal sequência, a regra condp com $\Psi =$ vazia, o que leva à derivação:

1. $\neg p, p$ (Ax)
2. $p \to p$ (condp 1)

□

No Exemplo 50 mostrou-se que o conjunto $\{p \to (q \to r), q, \neg(p \to r)\}$ é insatisfatível usando-se o sistema $\mathcal{S}_I$. Daí se pode concluir que o conjunto de fórmulas complementares $\{\neg(p \to (q \to r)), \neg q, p \to r\}$ é tal que qualquer interpretação satisfaz alguma fórmula dele, ou ainda, que a disjunção de suas fórmulas, $\neg(p \to (q \to r)) \vee \neg q \vee (p \to r)$ é uma tautologia. Como esta é logicamente equivalente a $((p \to (q \to r)) \wedge q) \to (p \to r)$, esta última também é uma tautologia. O exemplo a seguir mostra uma demonstração direta que esta última é uma tautologia via $\mathcal{S}_T$.

Exemplo 53 Existe uma única regra que se aplicada leva à sequência $((p \to (q \to r)) \wedge q) \to (p \to r)$: a regra condp. Para aplicá-la, a premissa é $\neg(p \to (q \to r)) \wedge q), p \to r$. Para derivar esta, tem-se duas possibilidades: aplicar a regra conjn ou a regra condp. Depois de aplicar uma delas aplica-se a outra. Prosseguindo-se chega-se à seguinte derivação obtida de trás para frente:

1. $q, \neg q, \neg p, r$	(Ax)
2. $\neg r, \neg q, \neg p, r$	(Ax)
3. $p, \neg q, \neg p, r$	(Ax)
4. $\neg(q \to r), \neg q, \neg p, r$	(condn 1,2)
5. $\neg(p \to (q \to r)), \neg q, \neg p, r$	(condn 3,4)
6. $\neg((p \to (q \to r)) \wedge q), \neg p, r$	(conjn 5)
7. $\neg((p \to (q \to r)) \wedge q), p \to r$	(condp 6)
8. $((p \to (q \to r)) \wedge q) \to (p \to r)$	(condp 7)

□

Embora as derivações encontradas nos exemplos anteriores não sejam muito claras, suas obtenções pelo uso do sistema formal $\mathcal{S}_I$ e $\mathcal{S}_T$ são triviais, muito mais fáceis do que utilizando um sistema do tipo Hilbert como o da Seção 2.8.2.

O sistema $\mathcal{S}_T$ serve para mostrar que a disjunção das fórmulas em uma sequência é tautológica, podendo ser demonstrado que:

- *é correto*: se $\emptyset \Rightarrow \Psi$, então qualquer interpretação satisfaz alguma fórmula de $c(\Psi)$; e
- *é completo*: se qualquer interpretação satisfaz alguma fórmula de $c(\Psi)$, então $\emptyset \Rightarrow \Psi$.

As demonstrações de correção e completude podem ser feitas com relativa facilidade, inspiradas naquelas do sistema $\mathcal{S}_C$ da seção anterior.

Assim como em $\mathcal{S}_I$, as derivações em $\mathcal{S}_T$ são obtidas facilmente em reverso, porque existe *uma única* regra aplicável em cada passo na produção da derivação em reverso. Aqui, da mesma forma, há casos em que ocorre uma *explosão combinatória*, propiciada por uma proliferação de sequências pela aplicação das regras de duas premissas, no caso, conjp, disjn, condn, bicp e bicn. Assim como o algoritmo baseado em $\mathcal{S}_I$ constrói

implicitamente a árvore de computação para satB-nd, um algoritmo baseado em $\mathcal{S}_T$ deve construir implicitamente a árvore de computação para o análogo de satB-nd que procuraria verificar se a fórmula é *falseável*: a fórmula é tautológica se e somente se ela não é falseável e, portanto, toda computação no teste de falseabilidade resulta em fracasso. Enquanto um ramo de sucesso atesta falseabilidade, é necessário que todo ramo seja de fracasso para atestar que a fórmula é uma tautologia.

Exercícios

1. Refaça a Figura 2.28 supondo que o algoritmo satB-nd usa união de multiconjuntos no lugar de união de conjuntos.

2. Faça uma explicação de cada regra da Figura 2.29 de forma análoga à feita no texto para as regras da conjunção.

3. Refaça a derivação do Exemplo 47 para incluir as aplicações da regra tr.

4. Demonstre a correção de $\mathcal{S}_I$ para efeitos de determinar se um conjunto é insatisfatível.

5. Demonstre a completude de $\mathcal{S}_I$ para efeitos de determinar se um conjunto é insatisfatível.

6. Faça uma explicação de cada regra da Figura 2.31 de forma análoga à feita no texto para as regras da conjunção.

7. Demonstre a correção de $\mathcal{S}_T$ para efeitos de verificar que qualquer interpretação satisfaz alguma fórmula de um conjunto.

8. Demonstre a completude de $\mathcal{S}_T$ para efeitos de verificar que qualquer interpretação satisfaz alguma fórmula de um conjunto.

9. Construa sistemas formais análogos a $\mathcal{S}_I$ e $\mathcal{S}_T$ que usem como linguagem sequências de fórmulas *marcadas*, no lugar de sequências de fórmulas simplesmente. Por exemplo, as regras para a conjunção no caso de sistema análogo a $\mathcal{S}_T$ (para tautologias) seriam:

$$\mathsf{conjp}: \frac{\Psi, V\alpha \quad \Psi, V\beta}{\Psi, V\alpha \wedge \beta} \qquad \mathsf{conjn}: \frac{\Psi, F\alpha, F\beta}{\Psi, F\alpha \wedge \beta}$$

Já os axiomas seriam da forma $\Psi, V\nu, F\nu$.

10. Faça um algoritmo, taut, baseado em $\mathcal{S}_T$, similar a insat, para determinar se um conjunto de fórmulas é tal que qualquer interpretação satisfaz uma de suas fórmulas. O algoritmo deverá determinar também a dedução encontrada, incluindo anotações das regras e das premissas usadas para gerar cada sequência.

(a) eliminação dos conectivos $\rightarrow$ e $\leftrightarrow$:

(a1) $\alpha \rightarrow \beta \mapsto \neg\alpha \vee \beta$

(a2.1) $\alpha \leftrightarrow \beta \mapsto (\neg\alpha \vee \beta) \wedge (\alpha \vee \neg\beta)$ ou (a2.2) $\alpha \leftrightarrow \beta \mapsto (\alpha \wedge \beta) \vee (\neg\alpha \wedge \neg\beta)$

(b) "internalização" da negação:

(b1) $\neg\neg\alpha \mapsto \alpha$

(b2) $\neg(\alpha \wedge \beta) \mapsto \neg\alpha \vee \neg\beta$

(b3) $\neg(\alpha \vee \beta) \mapsto \neg\alpha \wedge \neg\beta$

(c) tratamento de $\top$ e $\bot$:

(c1) $\neg\top \mapsto \bot$

(c2) $\neg\bot \mapsto \top$

(c3) $\top \wedge \alpha \mapsto \alpha$

(c4) $\alpha \wedge \top \mapsto \alpha$

(c5) $\bot \wedge \alpha \mapsto \bot$

(c6) $\alpha \wedge \bot \mapsto \bot$

(c7) $\top \vee \alpha \mapsto \top$

(c8) $\alpha \vee \top \mapsto \top$

(c9) $\bot \vee \alpha \mapsto \alpha$

(c10) $\alpha \vee \bot \mapsto \alpha$

Figura 2.32: Regras para obtenção de fórmulas na FNN.

2.5 Formas normais

O elemento básico de uma fórmula nas formas normais a serem apresentadas é o *literal*. Se $\nu \in \mathcal{V}$, então ν é dito ser um literal *positivo* e $\neg\nu$ é dito ser um literal *negativo*).

Definição 15 *Uma fórmula é dita estar na* forma normal da negação *(FNN) se é $\top$, $\bot$ ou é constituída apenas de literais e (eventualmente) conectivos $\wedge$ e/ou $\vee$.* □

Para qualquer fórmula existe uma fórmula logicamente equivalente na FNN. A Figura 2.32 mostra um conjunto de regras de reescrita que, se aplicadas sobre uma fórmula α até que não seja mais possível aplicar nenhuma delas, propicia a obtenção de uma fórmula logicamente equivalente a α na FNN. Note que para cada regra $\gamma_1 \mapsto \gamma_2$, tem-se que $\gamma_1 \equiv \gamma_2$. Assim, se β é obtida de α pela aplicação de uma regra $\gamma_1 \mapsto \gamma_2$, segue-se, pelo Teorema 2, que $\alpha \equiv \beta$. Assim, se $\alpha \Rightarrow \beta$ (pelo sistema formal $(\mathcal{F}, R, \emptyset)$, em que R é o conjunto de regras da Figura 2.32, segue-se indutivamente que $\alpha \equiv \beta$. Além disso, se nenhuma regra é aplicável a β, então β está na FNN. Isto segue do fato de

que, após a eliminação dos conectivos $\rightarrow$ e $\leftrightarrow$ via as regras (a1) e (a2.1) ou (a2.2), se a fórmula resultante não está na FNN, pela Definição 13 ela necessariamente contém uma subfórmula que é da forma especificada pelo lado esquerdo de alguma das outras regras da Figura 2.32. Resumindo, acabou-se de mostrar que:

se $\alpha \Rightarrow \beta$ e nenhuma regra é aplicável a β, então $\beta \equiv \alpha$ e β está na FNN.

Exemplo 54 Uma fórmula equivalente a $(p \rightarrow q) \wedge ((q \vee r) \rightarrow p)$ na FNN pode ser assim obtida:

1. $(p \rightarrow q) \wedge ((q \vee r) \rightarrow p)$ (fórmula original)
2. $(\neg p \vee q) \wedge ((q \vee r) \rightarrow p)$ (a1)
3. $(\neg p \vee q) \wedge (\neg(q \vee r) \vee p)$ (a1)
4. $(\neg p \vee q) \wedge ((\neg q \wedge \neg r) \vee p)$ (b3)

À direita de cada fórmula está a identificação da regra utilizada para obtê-la. □

A adequação do uso de uma das regras (a2.1) ou (a2.2), depende do que se pretenda com a fórmula na FNN. Note que a primeira alternativa vê uma bicondicional como uma conjunção e a segunda a vê como uma disjunção; mas a negação de uma bicondicional (após o uso de uma das regras) torna a conjunção em disjunção via (b2) ou a disjunção em conjunção via (b3).

Um aspecto importante é que, em geral, em uma fórmula na FNN não se costuma colocar parênteses que especifiquem precedência para uma sequência de "$\vee$s" ou de "$\wedge$s". Com isso, tem-se o que se denomina disjunções e conjunções *generalizadas*. Assim, por exemplo, $p_1 \wedge (q_1 \vee q_2 \vee (r_1 \wedge r_2)) \wedge p_2$ é uma conjunção de três subfórmulas: p_1, $q_1 \vee q_2 \vee (r_1 \wedge r_2)$ e p_2. A segunda destas, por sua vez, é a disjunção de três subfórmulas: q_1, q_2 e $r_1 \wedge r_2$. Além disso, considera-se que o escopo da disjunção e da conjunção generalizadas é *o mais amplo possível*. Assim, por exemplo, a fórmula $(p_1 \vee p_2) \vee (q_1 \vee q_2)$ torna-se $p_1 \vee p_2 \vee q_1 \vee q_2$. A seguir, apresenta-se as alterações das regras da Figura 2.32 para lidar com este novo formato.

As regras (a1), (a2.1) ou (a2.2), (b1), (c1) e (c2) continuam as mesmas. As regras (b2) e (b3) tornam-se, respectivamente ($n \geq 2$):

$$\neg(\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n) \mapsto \neg\alpha_1 \vee \neg\alpha_2 \vee \ldots \vee \neg\alpha_n$$

$$\neg(\alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_n) \mapsto \neg\alpha_1 \wedge \neg\alpha_2 \wedge \ldots \wedge \neg\alpha_n$$

No lugar das regras (c3) e (c4) tem-se ($n \geq 2$):

$$\alpha_1 \wedge \ldots \wedge \alpha_n \mapsto \alpha_1 \wedge \ldots \wedge \alpha_{i-1} \wedge \alpha_{i+1} \wedge \ldots \wedge \alpha_n, \text{ se } \alpha_i = \top \text{ para algum } i.$$

No lugar de (c5) e (c6) tem-se ($n \geq 2$):

$$\alpha_1 \wedge \ldots \wedge \alpha_n \mapsto \bot, \text{ se } \alpha_i = \bot \text{ para algum } i.$$

No lugar de (c7) e (c8) tem-se ($n \geq 2$):

$\alpha_1 \vee \ldots \vee \alpha_n \mapsto \top$, se $\alpha_i = \top$ para algum i.

Finalmente, no lugar de (c9) e (c10) tem-se ($n \geq 2$):

$\alpha_1 \vee \ldots \vee \alpha_n \mapsto \alpha_1 \vee \ldots \vee \alpha_{i-1} \vee \alpha_{i+1} \vee \ldots \vee \alpha_n$, se $\alpha_i = \bot$ para algum i.

Apesar dessas alterações, as regras, nos exemplos, continuarão com os mesmos nomes usados na Figura 2.32. As últimas, que substituem (c3) e (c4), (c5) e (c6), (c7) e (c8), (c9) e (c10), terão os nomes (c3), (c5), (c7) e (c9) respectivamente.

Exemplo 55 Segue a obtenção de uma fórmula equivalente a $\alpha = ((p \vee q) \leftrightarrow r) \wedge \top \wedge \neg q$ na FNN:

1. $((p \vee q) \leftrightarrow r) \wedge \top \wedge \neg q$ (α)
2. $((p \vee q) \leftrightarrow r) \wedge \neg q$ (c3)
3. $(\neg(p \vee q) \vee r) \wedge (p \vee q \vee \neg r) \wedge \neg q$ (a2.1)
4. $((\neg p \wedge \neg q) \vee r) \wedge (p \vee q \vee \neg r) \wedge \neg q$ (b2)

Note como, de 2 para 3, usou-se a generalizada correspondente a (c3) e (c4). Veja que a 4 é $(\neg(p \vee q) \vee r) \wedge (p \vee q \vee \neg r) \wedge \neg q$, e não $((\neg(p \vee q) \vee r) \wedge ((p \vee q) \vee \neg r)) \wedge \neg q$; dois pares de parêntesis foram eliminados para que os escopos ficassem os mais amplos possíveis. Por fim, note que, houvesse sido aplicada a regra (a2.2), a FNN obtida seria outra:

3. $(((p \vee q) \wedge r) \vee (\neg(p \vee q) \wedge \neg r)) \wedge \neg q$ (a2.2)
4. $(((p \vee q) \wedge r) \vee (\neg p \wedge \neg q \wedge \neg r)) \wedge \neg q$ (b3)

□

Uma fórmula na FNN pode ser vista como exibindo uma *estrutura e-ou* de possíveis atribuições de valores verdade a variáveis proposicionais de forma a satisfazer a fórmula (e qualquer outra logicamente equivalente). Por exemplo, a fórmula na FNN $(p \vee q) \wedge \neg p$ obtida de $\alpha = \neg((\neg p \rightarrow q) \rightarrow p)$ (e de muitas outras) exibe mais claramente como satisfazer α: deve-se satisfazer (1) $p \vee q$ **e** (2) $\neg p$; para satisfazer (2), deve-se fazer $p^i = F$; para satisfazer (1), deve-se fazer $p^i = V$ **ou** $q^i = V$; destas duas opções, a primeira não é compatível com a exigida por (2) e a segunda é; logo, deve-se ter $p^i = F$ e $q^i = V$. As coisas não ficam tão simples em geral porque podem existir vários níveis alternados de **e**s e **ou**s. Um algoritmo bastante simples para satisfabilidade é aquele baseado em *computação alternante*. Além de usar o pseudocomando escolha-nd(C) que, para ter sucesso, exige que *exista* uma escolha de elemento de C que leve a sucesso, ele usa o pseudocomando itera(C) que, para ter sucesso, exige que *toda* escolha de elemento de C leve a sucesso. O comando escolha-nd é usado para lidar com disjunções, como já visto, e o itera é usado para lidar com conjunções. Veja a Figura 2.33.

Uma maneira de lidar com o comando itera deterministicamente, é via o uso da lista A, como nos algoritmos satA-nd e satB-nd da Seção 2.4.2 (veja exercício ao final desta seção). Ao se fazer isto, obtém-se, nada mais do que uma adaptação de satB-nd para lidar apenas com fórmulas na FNN com conjunções e disjunções generalizadas.

```
proc-nd satAlt(fórmula na FNN γ):
    conjunto de literais Σ;
    Σ := ∅;
    laço
        caso γ seja
            literal:          se γ̄ ∈ Σ então fracasso senão Σ := Σ ∪ {γ}; sucesso fimse
            α1 ∧ ... ∧ αn:  γ := itera({α1, ..., αn})
            α1 ∨ ... ∨ αn:  γ := escolha-nd({α1, ..., αn})
        fimcaso
    fimlaço
fim satAlt
```

Figura 2.33: Satisfabilidade para FNN com computação alternante.

Existem duas outras notações que encontram aplicações em diversas circunstâncias. São as chamadas *formas normais conjuntiva e disjuntiva*. Para definí-las, serão usados os conceitos de *cláusula* e de *cláusula conjuntiva*. Uma cláusula é uma disjunção de literais $l_1 \vee l_2 \vee \ldots \vee l_k$, $k \geq 0$. No caso em que $k = 0$, tem-se a denominada *cláusula vazia*, que será denotada por $\bot$ (é uma contradição). Já uma cláusula conjuntiva é uma conjunção de literais $l_1 \wedge l_2 \wedge \ldots \wedge l_k$, $k \geq 0$. No caso em que $k = 0$, tem-se a cláusula conjuntiva vazia, que será denotada por $\top$ (é uma tautologia).

Algumas vezes é útil, do ponto de vista pragmático, ver (ou tratar) uma cláusula como um *conjunto* de literais, visto que cláusulas com os mesmos literais são logicamente equivalentes. O mesmo para cláusulas conjuntivas. Embora seja usada a notação já vista, elas serão tratadas aqui, por conveniência, como conjuntos. Assim, por exemplo, $p \vee \neg q$ e $\neg q \vee p$ serão consideradas a *mesma* cláusula, que será denotada também por $\{p, \neg q\}$.

Definição 16 *Uma fórmula é dita estar na* forma normal conjuntiva *(FNC) se for* $\top$, $\bot$ *ou da forma* $\psi_1 \wedge \psi_2 \wedge \ldots \wedge \psi_n$, *para* $n \geq 1$, *sendo cada* ψ_i *uma cláusula não vazia.*

Aqui, novamente, a ordem das subfórmulas (cláusulas) será considerada irrelevante. Observe que, pelas Definições 15 e 16, se uma fórmula está na FNC, então está na FNN, mas não necessariamente vice-versa.

Exemplo 56 São exemplos de fórmulas na FNC:

- $\top$ (não tem nenhuma cláusula);
- $\bot$ (uma cláusula sem literais: a cláusula vazia);
- p (uma cláusula de um literal);
- $\neg p \vee \neg r$ (uma cláusula de dois literais);
- $p \wedge \neg q \wedge r$ (três cláusulas de um literal cada uma);

- $(p_1 \vee \neg r) \wedge \neg q \wedge (r \vee p_2)$ (três cláusulas, uma de um literal e duas de dois literais).

□

A primeira fórmula do exemplo anterior, $\top$, pode ser vista com uma "conjunção de zero cláusulas". E a segunda, $\bot$, como uma "conjunção de uma única cláusula": a cláusula sem nenhum literal. Caso a fórmula na FNC contenha a cláusula vazia, não pode conter nenhuma outra cláusula.

Para obter uma fórmula equivalente na FNC, pode-se obter uma na FNN e, em seguida, aplicar a regra a seguir enquanto for possível. Note que aqui já estão sendo utilizadas a conjunção e a disjunção generalizadas. Além disso, supõe-se, implicitamente, que a ordem dos termos, tanto na disjunção quanto na conjunção, é irrelevante.

(d1) $\alpha_1 \vee \cdots \vee \alpha_m \vee (\beta_1 \wedge \cdots \wedge \beta_n) \mapsto (\alpha_1 \vee \cdots \vee \alpha_m \vee \beta_1) \wedge \cdots \wedge (\alpha_1 \vee \cdots \vee \alpha_m \vee \beta_n)$.

A equivalência lógica que dá suporte a tal regra é a distributividade da disjunção com relação à conjunção.

Exemplo 57 No Exemplo 55 obteve-se uma fórmula equivalente a

$$((p \vee q) \leftrightarrow r) \wedge \top \wedge \neg q$$

na FNN, qual seja:

$$((\neg p \wedge \neg q) \vee r) \wedge (p \vee q \vee \neg r) \wedge \neg q.$$

Segue a obtenção de uma fórmula equivalente na FNC aplicando-se a regra anterior:

1. $((p \vee q) \leftrightarrow r) \wedge \top \wedge \neg q)$
2. $((\neg p \wedge \neg q) \vee r) \wedge (p \vee q \vee \neg r) \wedge \neg q$ (FNN, Ex. 55)
3. $(\neg p \vee r) \wedge (\neg q \vee r) \wedge (p \vee q \vee \neg r) \wedge \neg q$ (d1)

Veja que na aplicação de (d1) a ordem dos termos da disjunção na subfórmula $(\neg p \wedge \neg q) \vee r$ não impediu a aplicação da regra.

A FNN acima, tomada do Exemplo 55, é aquela obtida pela aplicação da regra (a2.1). Se fosse a obtida pela aplicação da regra (a2.2), o processo seria assim:

1. $((p \vee q) \leftrightarrow r) \wedge \top \wedge \neg q$
2. $(((p \vee q) \wedge r) \vee (\neg p \wedge \neg q \wedge \neg r)) \wedge \neg q$ (FNN, Ex. 55)
3. $(p \vee q \vee (\neg p \wedge \neg q \wedge \neg r)) \wedge (r \vee (\neg p \wedge \neg q \wedge \neg r)) \wedge \neg q$ (d1)
4. $(p \vee q \vee \neg p) \wedge (p \vee q \vee \neg q) \wedge (p \vee q \vee \neg r)$
   $\wedge (r \vee (\neg p \wedge \neg q \wedge \neg r)) \wedge \neg q$ (d1)
5. $(p \vee q \vee \neg p) \wedge (p \vee q \vee \neg q) \wedge (p \vee q \vee \neg r)$
   $\wedge (r \vee \neg p) \wedge (r \vee \neg q) \wedge (r \vee \neg r) \wedge \neg q$ (d1)

Esta fórmula contém 3 cláusulas tautológicas: $p \vee q \vee \neg p$, $p \vee q \vee \neg q$ e $r \vee \neg r$. Eliminando-se elas, obtém-se $(p \vee q \vee \neg r) \wedge (r \vee \neg p) \wedge (r \vee \neg q) \wedge \neg q$. □

O exemplo anterior ilustra o fato de que o uso da regra (a2.1) ou (a2.2) tem influência no processamento posterior que se deseje para a FNN. Neste caso específico, há maior facilidade na produção da FNC se a FNN foi produzida pela regra (a2.1). O fato é que, como a FNC é uma conjunção de disjunções, a melhor opção é aquela que obtém uma conjunção de disjunções (a primeira opção), embora aqui a disjunção não seja necessariamente de literais. É fácil notar que as cláusulas de uma FNC obtidas a partir de uma conjunção $\gamma_1 \wedge \gamma_2$ são as obtidas a partir de γ_1 *mais* as obtidas a partir de γ_2, enquanto que as obtidas a partir de uma disjunção $\gamma_1 \vee \gamma_2$ são as que podem ser obtidas unindo-se cada cláusula obtida a partir de γ_1 com cada cláusula obtida a partir de γ_2, o que gera um efeito *multiplicativo*. O aparecimento de cláusulas tautológicas no Exemplo 57 é devido, justamente, a esse efeito multiplicativo. De forma geral: se de $\alpha \leftrightarrow \beta$ aplica-se (a2.2) e obtém-se $(\alpha \wedge \beta) \vee (\neg\alpha \wedge \neg\beta)$, em seguida deriva-se $\underline{(\alpha \vee \neg\alpha)} \vee (\alpha \vee \neg\beta) \wedge (\beta \vee \neg\alpha) \wedge \underline{(\beta \vee \neg\beta)}$; é melhor aplicar (a2.1), obtendo-se $(\alpha \vee \neg\beta) \wedge (\beta \vee \neg\alpha)$ diretamente!

Na verdade, vendo apenas uma subfórmula da forma $\alpha \leftrightarrow \beta$, não há como saber qual é a melhor opção, pois a subfórmula pode estar no escopo de uma negação; e, neste, caso a melhor opção seria (a2.2), que originaria uma derivação da forma

1. $\neg(\alpha \leftrightarrow \beta)$	
2. $\neg((\alpha \wedge \beta) \vee (\neg\alpha \wedge \neg\beta))$	(a2.2)
3. $\neg(\alpha \wedge \beta) \wedge \neg(\neg\alpha \wedge \neg\beta)$	(b3)
4. $(\neg\alpha \vee \neg\beta) \wedge \neg(\neg\alpha \wedge \neg\beta)$	(b2)
5. $(\neg\alpha \vee \neg\beta) \wedge (\neg\neg\alpha \vee \neg\neg\beta)$	(b2)
6. $(\neg\alpha \vee \neg\beta) \wedge (\alpha \vee \neg\neg\beta)$	(b1)
7. $(\neg\alpha \vee \neg\beta) \wedge (\alpha \vee \beta)$	(b1)

enquanto que (a2.1) propiciaria:

1. $\neg(\alpha \leftrightarrow \beta)$	
2. $\neg((\neg\alpha \vee \beta) \wedge (\alpha \vee \neg\beta))$	(a2.1)
3. $\neg(\neg\alpha \vee \beta) \vee \neg(\alpha \vee \neg\beta)$	(b2)
4. $(\neg\neg\alpha \wedge \neg\beta) \vee \neg(\alpha \vee \neg\beta)$	(b3)
5. $(\neg\neg\alpha \wedge \neg\beta) \vee (\neg\alpha \wedge \neg\neg\beta)$	(b3)
6. $(\alpha \wedge \neg\beta) \vee (\neg\alpha \wedge \neg\neg\beta)$	(b1)
7. $(\alpha \wedge \neg\beta) \vee (\neg\alpha \wedge \beta)$	(b1)

Moral da história: vendo já a fórmula na FNN, se a subfórmula que vai resultar da bicondicional tem *polaridadade* positiva, isto é, estiver no escopo de um número par de negações, é melhor usar a regra (a2.1); e se tem *polaridadade* negativa, isto é, estiver no escopo de um número ímpar de negações, é melhor usar a regra (a2.2). O algoritmo da Figura 2.34 leva em conta, implicitamente, a polaridade para decidir qual das duas opções utilizar: na medida em que a fórmula vai sendo processada, é usado o próprio símbolo da negação, $\neg$, para indicar a polaridade de suas subfórmulas. Uma fórmula γ fornecida como argumento tem polaridade positiva. Assim, para se processar uma fórmula α de polaridade negativa, o procedimento deve ser chamado passando-se $\neg\alpha$

proc fnnc(fórmula sem $\top$ nem $\bot$ γ) **retorna** uma fórmula na FNN:
  caso γ **seja**
    literal: **retorne** γ
    $\neg\neg\alpha$: **retorne** fnnc(α) regra (b.1)
    $\alpha \wedge \beta$: **retorne** (fnnc(α) $\wedge$ fnnc(β))
    $\neg(\alpha \wedge \beta)$: **retorne** fnnc($\neg\alpha \vee \neg\beta$) regra (b.2)
    $\alpha \vee \beta$: **retorne** (fnnc(α) $\vee$ fnnc(β))
    $\neg(\alpha \vee \beta)$: **retorne** fnnc($\neg\alpha \wedge \neg\beta$) regra (b.3)
    $\alpha \rightarrow \beta$: **retorne** fnnc($\neg\alpha \vee \beta$) regra (a.1)
    $\neg(\alpha \rightarrow \beta)$: **retorne** fnnc($\alpha \wedge \neg\beta$) regras (a.1), (b.3), (b.1)
    $\alpha \leftrightarrow \beta$: **retorne** fnnc($(\neg\alpha \vee \beta) \wedge (\alpha \vee \neg\beta)$) regra (a2.1)
    $\neg(\alpha \leftrightarrow \beta)$: **retorne** fnnc($(\alpha \vee \beta) \wedge (\neg\alpha \vee \neg\beta)$) regras (a2.2), (b.3), (b.2), (b.2), (b.1), (b.1)
  fimcaso
fim fnnc

Figura 2.34: Procedimento fnnc: FNN para posterior FNC.

como argumento. Se $\gamma = \alpha \leftrightarrow \beta$, então, como a polaridade de γ é positiva, é melhor aplicar a regra (a2.1). Se $\gamma = \neg(\alpha \leftrightarrow \beta)$, então, sendo a polaridade de γ positiva, a de $\alpha \leftrightarrow \beta$ é negativa e, assim, é melhor aplicar a regra (a2.2) a $\alpha \leftrightarrow \beta$; e, como mostrado acima, após a negação, obtém-se o que é usado pelo algoritmo: $(\alpha \vee \beta) \wedge (\neg\alpha \vee \neg\beta)$.

O algoritmo é denominado fnnc, enfatizando que obtém a FNN "mais adequada" para o caso em que esta vá ser tranformada em seguida para a FNC. Para simplificar o algoritmo, supõe-se que $\top$ e $\bot$ não podem ocorrer na fórmula passada como argumento (ou seja, eventuais ocorrências devem ser eliminadas antes e o procedimento só deve ser chamado se a fórmula resultante não é $\top$ nem $\bot$; veja os Exercícios 7 e 8). Supõe-se também que as conjunções e disjunções são as tradicionais (não generalizadas). Nos pseudocomandos **retorne** são usadas concatenações pela simples justaposição das expressões. Por exemplo, **retorne** (fnnc(α)$\vee$fnnc(β)) quer dizer que deve ser retornada a fórmula que é a disjunção das subfórmulas retornadas pelas chamadas fnnc(α) e fnnc(β), ou seja, "(" seguido de fnnc(α), seguido de "$\vee$", seguido de fnnc(β), seguido de ")".

Uma fórmula na FNC é uma tautologia se, e somente se, é $\top$ ou toda cláusula da mesma contém um literal e seu complementar. Isso pode ser testado em tempo linear, mas o problema de obter uma fórmula na FNC é exponencial no pior caso.

Toda cláusula de uma fórmula na FNC, com exceção da cláusula vazia ($\bot$), é satisfatível. Dentre elas, apenas as cláusulas tautológicas não são falseáveis. Com isto, tirando-se a cláusula vazia e as cláusulas tautológicas, toda cláusula é contingente. Pode-se, então, obter facilmente uma interpretação que falseia uma fórmula a partir de uma equivalente na FNC: para uma cláusula *não tautológica* $l_1 \vee l_2 \vee \ldots \vee l_k$ a interpretação tal que $v^i(l_j) = F$, se l_j for um literal positivo, ou $v^i(l_j) = V$, se l_j for um literal negativo, falseia a fórmula. Segue um exemplo.

Exemplo 58 No exemplo anterior, verificou-se que $(\neg p \vee r) \wedge (\neg q \vee r) \wedge (p \vee q \vee \neg r) \wedge \neg q$ é uma fórmula na FNC, sem cláusulas tautológicas, equivalente a $\alpha = ((p \vee q) \leftrightarrow r) \wedge \top \wedge \neg q)$. Cada cláusula dá uma interpretação que falseia α, como dito anteriormente:

- de $\neg p \vee r$ obtém-se $p^i = V, r^i = F$; e
- de $\neg q \vee r$ obtém-se $q^i = V, r^i = F$; e
- de $p \vee q \vee \neg r$ obtém-se $p^i = F, q^i = F, r^i = V$.
- de $\neg q$ obtém-se $q^i = V$. □

Note que se *toda* interpretação falseia alguma cláusula não tautológica de uma fórmula γ na FNC, então γ é insatisfatível e, portanto, logicamente equivalente a $\bot$.

Definição 17 *Uma fórmula é dita estar na* forma normal disjuntiva *(FND) se for* $\bot$, $\top$ *ou da forma* $\psi_1 \vee \psi_2 \vee \ldots \vee \psi_n$, *para* $n \geq 1$, *sendo cada* ψ_i *uma cláusula conjuntiva não vazia.*

Aqui, novamente, a ordem das subfórmulas (cláusulas conjuntivas) será considerada irrelevante.

Exemplo 59 São exemplos de fórmulas na FND:

- $\bot$ (não tem nenhuma cláusula conjuntiva);
- $\top$ (uma cláusula conjuntiva sem literais: a cláusula conjuntiva vazia);
- p (uma cláusula conjuntiva de um literal);
- $\neg p \wedge \neg r$ (uma cláusula conjuntiva de dois literais);
- $p \vee \neg q \vee r$ (três cláusulas conjuntivas de um literal cada uma);
- $(p_1 \wedge \neg r) \vee \neg q \vee (r \wedge p_2)$ (três cláusulas conjuntivas, uma de um literal e duas de dois literais).

□

Note que a primeira fórmula do exemplo anterior, $\bot$, pode ser vista com uma "disjunção de zero cláusulas conjuntivas". E a segunda, $\top$, como uma "disjunção de uma única cláusula conjuntiva": a cláusula conjuntiva sem nenhum literal. Caso a fórmula na FND contenha a cláusula conjuntiva vazia, não pode conter nenhuma outra cláusula.

Para obter uma fórmula equivalente a outra na FND, basta obter uma equivalente na FNN e, em seguida, usar a regra enquanto ela for aplicável:

(d2) $\alpha_1 \wedge \cdots \wedge \alpha_m \wedge (\beta_1 \vee \cdots \vee \beta_n) \mapsto (\alpha_1 \wedge \cdots \wedge \alpha_m \wedge \beta_1) \vee \cdots \vee (\alpha_1 \wedge \cdots \wedge \alpha_m \wedge \beta_n)$.

A equivalência lógica que dá suporte a tal regra é a distributividade da conjunção com relação à disjunção.

Segue um exemplo que mostra que a aplicação da regra (a2.2) para uma ocorrência de bicondicional de polaridade positiva passa a ser melhor na transformação para FND.

Exemplo 60 Segue a obtenção de uma fórmula equivalente à do Exemplo 55,

$$((p \vee q) \leftrightarrow r) \wedge \top \wedge \neg q$$

na FND aplicando-se a regra anterior:

1.	$((p \vee q) \leftrightarrow r) \wedge \top \wedge \neg q$	
2.	$(((p \vee q) \wedge r) \vee (\neg p \wedge \neg q \wedge \neg r)) \wedge \neg q$	(FNN, Ex. 55)
3.	$((p \wedge r) \vee (q \wedge r) \vee (\neg p \wedge \neg q \wedge \neg r)) \wedge \neg q$	(d2)
4.	$(p \wedge r \wedge \neg q) \vee (q \wedge r \wedge \neg q) \vee (\neg p \wedge \neg q \wedge \neg r)$	(d2)
5.	$(p \wedge r \wedge \neg q) \vee (\neg p \wedge \neg q \wedge \neg r)$	(elim. contraditória)

Como feito no último passo acima, para manter as fórmulas mais curtas, conjunções que contenham literais complementares serão imediatamente eliminadas. Segue a obtenção da FND, agora usando-se a FNN obtida após utilização da regra (a2.1).

1.	$((p \vee q) \leftrightarrow r) \wedge \top \wedge \neg q$	
2.	$((\neg p \wedge \neg q) \vee r) \wedge (p \vee q \vee \neg r) \wedge \neg q$	(FNN, Ex. 55)
3.	$((\neg p \wedge \neg q) \vee r) \wedge ((p \wedge \neg q) \vee (q \wedge \neg q) \vee (\neg r \wedge \neg q))$	(d2)
4.	$((\neg p \wedge \neg q) \vee r) \wedge ((p \wedge \neg q) \vee (\neg r \wedge \neg q))$	(elim. contr.)
5.	$[((\neg p \wedge \neg q) \vee r) \wedge (p \wedge \neg q)] \vee [((\neg p \wedge \neg q) \vee r) \wedge (\neg q \wedge \neg r)]$	(d2)
6.	$[(\neg p \wedge \neg q \wedge p) \vee (r \wedge p \wedge \neg q)] \vee [((\neg p \wedge \neg q) \vee r) \wedge (\neg q \wedge \neg r)]$	(d2)
7.	$(r \wedge p \wedge \neg q) \vee [((\neg p \wedge \neg q) \vee r) \wedge (\neg q \wedge \neg r)]$	(elim. contr.)
8.	$(r \wedge p \wedge \neg q) \vee (\neg p \wedge \neg q \wedge \neg r) \vee (r \wedge \neg q \wedge \neg r)$	(d2)
9.	$(r \wedge p \wedge \neg q) \vee (\neg p \wedge \neg q \wedge \neg r)$	(elim. contr.)

O surgimento das cláusulas conjuntivas contraditórias é consequência do efeito multiplicativo propiciado pela aplicação da "regra inadequada"; no caso, (a2.1) em vez de (a2.2). □

Na Figura 2.35 está mostrado um algoritmo para produção de FNN mais adequado para posterior obtençao de FND. Aqui também supõe-se que eventuais $\top$ e/ou $\bot$ sejam eliminados antes e que a fórmula passada como argumento não seja $\top$ nem $\bot$. A única diferença desta versão para a da Figura 2.34 é o tratamento de $\leftrightarrow$.

Se a fórmula for uma contradição, isso irá transparecer claramente: uma fórmula na FND é uma contradição se, e somente se, é $\bot$ ou toda cláusula conjuntiva da mesma contém um literal e seu complementar. Isso pode ser testado em tempo linear, mas o problema de obter uma fórmula na FND é exponencial no pior caso.

Observe, por outro lado, que pode-se obter uma interpretação que satisfaz uma fórmula a partir de uma fórmula equivalente na FND, sem cláusulas conjuntivas contraditórias; basta, a partir de uma cláusula conjuntiva $l_1 \wedge l_2 \wedge \ldots \wedge l_k$, fazer-se para cada literal l_j, $l_j^i = V$, se l_j for um literal positivo, ou $l_j^i = F$, se l_j for um literal negativo.

Note que se *toda* interpretação satisfaz alguma cláusula conjuntiva não contraditória de uma fórmula γ na FND, então γ é tautológica e, portanto, logicamente equivalente a $\top$.

proc fnnd(fórmula sem $\top$ nem $\bot$ γ) **retorna** uma fórmula na FNN:
  caso γ **seja**
    literal: **retorne** γ
    $\neg\neg\alpha$: **retorne** fnnd(α) regra (b.1)
    $\alpha \wedge \beta$: **retorne** (fnnd(α) $\wedge$ fnnd(β))
    $\neg(\alpha \wedge \beta)$: **retorne** fnnd($\neg\alpha \vee \neg\beta$) regra (b.2)
    $\alpha \vee \beta$: **retorne** (fnnd(α) $\vee$ fnnd(β))
    $\neg(\alpha \vee \beta)$: **retorne** fnnd($\neg\alpha \wedge \neg\beta$) regra (b.3)
    $\alpha \rightarrow \beta$: **retorne** fnnd($\neg\alpha \vee \beta$) regra (a.1)
    $\neg(\alpha \rightarrow \beta)$: **retorne** fnnd($\alpha \wedge \neg\beta$)} regras (a.1), (b.3), (b.1)
    $\alpha \leftrightarrow \beta$: **retorne** fnnd($(\alpha \wedge \beta) \vee (\neg\alpha \wedge \neg\beta)$) regra (a2.2)
    $\neg(\alpha \leftrightarrow \beta)$: **retorne** fnnd($(\neg\alpha \wedge \beta) \vee (\alpha \wedge \neg\beta)$) regras (a2.1), (b.2), (b.3), (b.3), (b.1), (b.1)
  fimcaso
fim fnnd

Figura 2.35: Procedimento fnnd: FNN para posterior FND.

A árvore de computação relativa ao algoritmo satB-nd dá uma fórmula equivalente à original na FND, caso escolha(A) só selecione literal em A se A contiver apenas literais: a disjunção das cláusulas conjuntivas constituídas dos conjuntos dos literais que rotulam as folhas. Os conjuntos de literais que rotulam as folhas de ramos relativos a computações de fracasso contêm literais complementares (e, obviamente podem ser descartados), e os conjuntos relativos a computações de sucesso são, naturalmente, aqueles que contribuem efetivamente para obtenção de modelos para a fórmula. Conclusão: os algoritmos satA-nd e satB-nd podem ser vistos como procurando, implicitamente, por alguma cláusula conjuntiva não contraditória da FND. Com isso, algoritmos determinísticos diretamente baseados nos mesmos acabam realizando, também implicitamente, o mesmo trabalho que o da transformação de uma fórmula em uma equivalente na FND (parando ao encontrar uma cláusula conjuntiva não contraditória). A Figura 2.36 mostra uma adaptação de satB-nd para gerar a representação de uma fórmula na FND como uma família de conjuntos de literais: a fórmula é uma disjunção das conjunções dos literais de cada conjunto da família. Em outras palavras: fnd(γ) retorna o conjunto dos conjuntos Σ que rotulariam folhas em todas as computações possíveis de satB-nd, tanto as de sucesso quanto as de fracasso, supondo que as subfórmulas são dissecadas até atingir apenas literais. O algoritmo apresenta dois tipos de operação básicos:

- fnd(α) $\cup$ fnd(β): realiza "soma" de disjunções; por exemplo, se
  - fnd(α) $= \{\{p, q\}, \{\neg p\}\}$, representando $(p \wedge q) \vee \neg p$, e
  - fnd(β) $= \{\{\neg p, r\}, \{p\}\}$, representando $(\neg p \wedge r) \vee p$,

  o resultado é:
  - $\{\{p, q\}, \{\neg p\}, \{\neg p, r\}, \{p\}\}$, representando $(p \wedge q) \vee \neg p \vee (\neg p \wedge r) \vee p$.
- $\{X \cup Y \mid X \in$ fnd(α) e $Y \in$ fnd(β)$\}$: realiza "produto" de disjunções; por exemplo, para os mesmos fnd(α) e fnd(β) acima, o resultado é:

proc fnd(fórmula γ) **retorna** uma família de conjuntos de literais:
  caso γ **seja**
    literal: **retorne** $\{\{\gamma\}\}$
    $\top$, $\neg\bot$: **retorne** $\{\emptyset\}$
    $\bot$, $\neg\top$: **retorne** $\emptyset$
    $\neg\neg\alpha$: **retorne** fnd(α)
    $\alpha \wedge \beta$: **retorne** $\{X \cup Y \mid X \in$ fnd(α) **e** $Y \in$ fnd(β)$\}$
    $\neg(\alpha \wedge \beta)$: **retorne** fnd($\neg\alpha \vee \neg\beta$)
    $\alpha \vee \beta$: **retorne** fnd(α) $\cup$ fnd(β)
    $\neg(\alpha \vee \beta)$: **retorne** fnd($\neg\alpha \wedge \neg\beta$)$\}$
    $\alpha \rightarrow \beta$: **retorne** fnd($\neg\alpha \vee \beta$)
    $\neg(\alpha \rightarrow \beta)$:**retorne** fnd($\alpha \wedge \neg\beta$)$\}$
    $\alpha \leftrightarrow \beta$: **retorne** fnd($(\alpha \wedge \beta) \vee (\neg\alpha \wedge \neg\beta)$)
    $\neg(\alpha \leftrightarrow \beta)$:**retorne** fnd($(\alpha \wedge \neg\beta) \vee (\neg\alpha \wedge \beta)$)
  fimcaso
fim fnd

Figura 2.36: Procedimento fnd.

- $\{\{p, q, \neg p, r\}, \{p, q\}, \{\neg p, r\}, \{\neg p, p\}\}$, representando $(p \wedge q \wedge \neg p \wedge r) \vee (p \wedge q) \vee (\neg p \wedge r) \vee (\neg p \wedge p)$.

Para obter um algoritmo similar para a FNC, basta trocar os papéis de conjunções e disjunções, como mostra a Figura 2.37. Aqui a fórmula na FNC é obtida como uma família de conjuntos de literais de tal maneira que a fórmula é uma conjunção das disjunções dos literais de cada conjunto da família.

Observe, finalmente, que os algoritmos fnc e fnd, ao lidar com a fórmula do conectivo principal em direção aos literais, "empurrando" a negação para o interior da fórmula, têm como escolher a melhor alternativa para lidar com o conectivo $\leftrightarrow$. Assim, na verdade, eles "incorporam" o trabalho efetuado por fnnc e fnnd.

Exercícios

1. Encontre derivações de fórmulas na FNC equivalentes às fórmulas:

a) $((p \wedge q) \vee \neg p) \leftrightarrow (\neg p \vee q)$
b) $\neg(p \rightarrow q) \vee (\neg p \wedge r)$
c) $(\neg(q \vee r) \rightarrow p) \vee (\neg p \rightarrow q)$
d) $p \leftrightarrow (q \vee (\neg p \wedge r))$
e) $(\neg q \rightarrow \neg p) \rightarrow ((\neg q \rightarrow p) \rightarrow q)$

Para encurtar as derivações e também simplificar as fórmulas, pode usar regras adicionais àquelas apresentadas na seção, como $\alpha \vee \neg\alpha \mapsto \top$, $\alpha \leftrightarrow \neg\alpha \mapsto \bot$ etc. Se usá-las, explicite-as e referencie-as. Lembre-se também que a ordem dos termos em conjunções e disjunções é irrelevante: regras de comutação não são necessárias.

proc fnc(fórmula γ) **retorna** uma família de conjuntos de literais:
  caso γ **seja**
    literal: **retorne** $\{\{\gamma\}\}$
    $\top$, $\neg\bot$: **retorne** $\{\emptyset\}$
    $\bot$, $\neg\top$: **retorne** $\emptyset$
    $\neg\neg\alpha$: **retorne** $\mathsf{fnc}(\alpha)$
    $\alpha \wedge \beta$: **retorne** $\mathsf{fnc}(\alpha) \cup \mathsf{fnc}(\beta)$
    $\neg(\alpha \wedge \beta)$: **retorne** $\mathsf{fnc}(\neg\alpha \vee \neg\beta)\}$
    $\alpha \vee \beta$: **retorne** $\{X \cup Y \mid X \in \mathsf{fnc}(\alpha)$ **e** $Y \in \mathsf{fnc}(\beta)\}$
    $\neg(\alpha \vee \beta)$: **retorne** $\mathsf{fnc}(\neg\alpha \wedge \neg\beta)$
    $\alpha \rightarrow \beta$: **retorne** $\mathsf{fnc}(\neg\alpha \vee \beta)\}$
    $\neg(\alpha \rightarrow \beta)$: **retorne** $\mathsf{fnc}(\alpha \wedge \neg\beta)$
    $\alpha \leftrightarrow \beta$: **retorne** $\mathsf{fnc}((\neg\alpha \vee \beta) \wedge (\alpha \vee \neg\beta))$
    $\neg(\alpha \leftrightarrow \beta)$: **retorne** $\mathsf{fnc}((\alpha \vee \beta) \wedge (\neg\alpha \vee \neg\beta))$
  fimcaso
fim fnc

Figura 2.37: Procedimento fnc.

2. Para as fórmulas do exercício anterior, encontre derivações de fórmulas na FND equivalentes. Novamente, pode usar regras adicionais àquelas apresentadas na seção para encurtar as derivações e também simplificar as fórmulas, bastando explicitá-las e referenciá-las.

3. Adapte o algoritmo satB-nd da Figura 2.11 para lidar (apenas) com fórmulas na FNN.

4. O algoritmo satB-nd da Figura 2.22 é idêntico ao da Figura 2.11, tendo sido apenas registradas como comentários identificações das regras do sistema formal $\mathcal{S}_B$ subjacente a satB-nd. De forma similar, especifique o sistema formal subjacente ao algoritmo obtido na questão anterior.

5. Altere o algoritmo satAlt da Figura 2.33 de forma a implementar o comando itera deterministicamente via o uso de uma lista A, como nos algoritmos satA-nd e satB-nd.

6. Faça uma definição recursiva de $\eta(\alpha)$, o número de cláusulas da fórmula na FNC retornada por $\mathsf{fnc}(\alpha)$, a partir da estrutura da fórmula; ou seja, $\eta(\alpha)$ deve ser igual a $|\mathsf{fnc}(\alpha)|$.

7. Na eliminação de $\top$ e $\bot$, por meio de regras de reescrita antes aplicar o algoritmo fnnc da Figura 2.34, as regras (c1) a (c10) da Figura 2.32 devem ser complementadas com outras relativas aos conectivos $\rightarrow$ e $\leftrightarrow$. Faça isso.

8. Modifique o procedimento fnnc da Figura 2.34 para que ele considere fórmulas que contenham $\top$ e $\bot$.

9. Construa um algoritmo para obter a FND em dois passos: no primeiro obtém a FNN, depois a FND propriamente dita.

2.6 O algoritmo DPLL

A grande maioria dos algoritmos determinísticos para o problema da satisfabilidade são baseados no método DPLL, que leva esse nome graças aos pioneiros do mesmo, Davis, Putnam, Logemann e Loveland, que o introduziram em 1962. Ele supõe a fórmula fornecida na FNC. Como dito pelos dois primeiros, essa suposição é útil e prática por dois motivos. Primeiro, por descartar a *complexidade estrutural* das fórmulas em geral, permitindo um tratamento relativamente simples do problema. Assim é que, por exemplo, para se mostrar que uma fórmula α na FNC é satisfatível, basta escolher um literal qualquer, digamos l, e mostrar que $\psi_2' \wedge \gamma$ é satisfatível, sendo $\alpha = \psi_1 \wedge \psi_2 \wedge \gamma$, ψ_1 a conjunção das cláusulas que contêm l, ψ_2 a conjunção das cláusulas que contêm $\overline{l}$ e ψ_2' o resultado de eliminar de ψ_2 o literal $\overline{l}$. Em segundo lugar, como uma base de conhecimentos (finita) pode ser vista como uma conjunção de dezenas, centenas, milhares, ..., de fórmulas, pode-se colocá-la na FNC simplesmente transformando-se fórmula por fórmula, cada uma relativamente pequena, sem o uso explosivo da "multiplicação distributiva" (que seria inevitável, neste caso, na obtenção de uma FND). [23]

No caso geral, a fórmula pode crescer exponencialmente quando transformada para a FNC, como mostra o exemplo seguinte.

Exemplo 61 Transformando a fórmula na FND $(p_1 \wedge q_1) \vee (p_2 \wedge q_2) \vee \cdots \vee (p_n \wedge q_n)$ para a FNC, obtém-se:

$$(p_1 \vee p_2 \vee \cdots \vee p_n) \wedge (p_1 \vee p_2 \vee \cdots \vee q_n) \wedge \cdots \wedge (q_1 \vee q_2 \vee \cdots \vee q_n)$$

que tem um total de 2^n cláusulas! □

Mesmo que, no caso geral, a obtenção de uma fórmula na FNC possa levar a uma explosão combinatória, existe como evitar isso através da obtenção de uma fórmula na FNC não equivalente à original, mas satisfatível se, e somente se, a original for satisfatível. E tal obtenção é linear, tanto no tempo gasto quanto em espaço, sendo conseguida à custa da introdução de novas variáveis proposicionais. A seguir será apresentado um algoritmo para isso, que parte de uma fórmula na FNN, que é baseado no teorema a seguir.

Teorema 11 *Sejam γ e α fórmulas na FNN, sendo que nem γ nem α são variáveis, ν uma variável que não ocorra em γ nem em α. Seja γ' o resultado de substituir zero*

[23] Mesmo na aplicação tradicional de determinar se uma fórmula α segue de um conjunto finito de fórmulas H mediante um algoritmo para satisfabilidade, essa transformação fórmula por fórmula se aplica, já que, neste caso, deve-se verificar a insatisfabilidade de $H \cup \{\neg\alpha\}$.

ou mais ocorrências de α em γ por ν. γ é satisfatível se, e somente se, $\gamma' \wedge (\nu \to \alpha)$ é satisfatível.

Prova

($\to$)[24] Suponha que γ seja satisfatível, e seja i tal que $v^i(\gamma) = V$. Seja a interpretação i' idêntica a i exceto para a variável ν: $i'(x) = x^i$ para toda $x \neq \nu$ e $i'(\nu) = v^i(\alpha)$. Segue-se que $v^{i'}(\gamma') = v^i(\gamma) = V$ e $v^{i'}(\nu \to \alpha) = V$. Portanto, $v^{i'}(\gamma' \wedge (\nu \to \alpha)) = v^{i'}(\gamma') \textcircled{\wedge} v^{i'}(\nu \to \alpha) = V \textcircled{\wedge} V = V$ e, assim, $\gamma' \wedge (\nu \to \alpha)$ é satisfatível.

($\leftarrow$) Suponha que $\gamma' \wedge (\nu \to \alpha)$ seja satisfatível, e seja i tal que $v^i(\gamma' \wedge (\nu \to \alpha)) = V$. Será usada indução sobre o número de conectivos binários ($\wedge$ e/ou $\vee$) de γ para mostrar que $v^i(\gamma) = V$ e, portanto, γ é satisfatível. Suponha, como hipótese de indução, que $v^i(\gamma) = V$ para γ com menos de n conectivos binários. Se $n = 0$: caso α não ocorra em γ, $\gamma' = \gamma$, e conclui-se imediatamente que $v^i(\gamma) = V$; caso α ocorra em γ, $\gamma = \alpha = \neg\nu_1$ para alguma $\nu_1 \in \mathcal{V}$, $\nu_1 \neq \nu$, e $\gamma' = \nu$; como $v^i(\gamma' \wedge (\nu \to \alpha)) = v^i(\nu \wedge (\nu \to \neg\nu_1)) = V$, segue-se que $v^i(\neg\nu_1) = V$, como requerido. Seja agora o caso em que γ tenha $n > 0$ conectivos binários. Como γ está na FNN, tem-se dois casos:
Caso 1. $\gamma = \gamma_1 \wedge \ldots \wedge \gamma_n$. Então $\gamma' = \gamma'_1 \wedge \ldots \wedge \gamma'_n$, sendo que γ'_k, para cada $1 \leq k \leq n$, é o resultado de substituir zero ou mais ocorrências de α em γ_k por ν. Como $v^i(\gamma'_1 \wedge \ldots \wedge \gamma'_n \wedge (\nu \to \alpha)) = V$, $v^i(\gamma'_1) = \ldots = v^i(\gamma'_n) = v^i(\nu \to \alpha) = V$. Aplicando-se a hipótese de indução, deduz-se que $v^i(\gamma_1) = \ldots = v^i(\gamma_n) = V$ e, portanto, $v^i(\gamma) = V$.
Caso 2. $\gamma = \gamma_1 \vee \ldots \vee \gamma_n$. Então $\gamma' = \gamma'_1 \vee \ldots \vee \gamma'_n$, sendo que γ'_k, para cada $1 \leq k \leq n$, é o resultado de substituir zero ou mais ocorrências de α em γ_k por ν. Como $v^i((\gamma'_1 \vee \ldots \vee \gamma'_n) \wedge (\nu \to \alpha)) = V$ segue-se que $v^i(\gamma'_k) = V$ para algum $1 \leq k \leq n$ e $v^i(\nu \to \alpha) = V$. Pela hipótese de indução, se $v^i(\gamma'_k) = V$ e $v^i(\nu \to \alpha) = V$, deduz-se que $v^i(\gamma_k) = V$. Logo, tem-se que $v^i(\gamma_k) = V$ para algum $1 \leq k \leq n$ e, portanto, $v^i(\gamma_1 \vee \ldots \vee \gamma_n) = V$, ou seja, $v^i(\gamma) = V$. □

O próximo exemplo mostra como o Teorema 11 pode ser usado para evitar o crescimento exponencial visto no Exemplo 61. Será usada a forma abreviada "sat" para significar "satisfatível".

Exemplo 62 A partir da fórmula $\gamma = (p_1 \wedge q_1) \vee (p_2 \wedge q_2) \vee \cdots \vee (p_n \wedge q_n)$, aplicando-se sucessivamente o Teorema 11, pode-se obter $2n+1$ cláusulas em vez das 2^n apresentadas no Exemplo 61:

$$\begin{aligned}
\gamma \text{ é sat} &\text{ sse } (x_1 \vee (p_2 \wedge q_2) \vee \cdots \vee (p_n \wedge q_n)) \wedge (x_1 \to (p_1 \wedge q_1)) \text{ é sat} \\
&\text{ sse } (x_1 \vee x_2 \vee \cdots \vee (p_n \wedge q_n)) \wedge (x_1 \to (p_1 \wedge q_1)) \wedge (x_2 \to (p_2 \wedge q_2)) \text{ é sat} \\
&\vdots \\
&\text{ sse } (x_1 \vee x_2 \vee \cdots \vee x_n) \wedge (x_1 \to (p_1 \wedge q_1)) \wedge \cdots \wedge (x_n \to (p_n \wedge q_n)) \text{ é sat.}
\end{aligned}$$

Como $x_i \to (p_i \wedge q_i) \equiv (\neg x_i \vee p_i) \wedge (\neg x_i \vee q_i)$, tem-se, finalmente, as $2n + 1$ cláusulas:

[24] Note que esta parte não depende de γ e α estarem na FNN; na verdade, ela vale para fórmulas quaisquer.

$$
\begin{array}{l}
x_1 \vee x_2 \vee \cdots \vee x_n \\
\neg x_1 \vee p_1 \\
\neg x_1 \vee q_1 \\
\vdots \\
\neg x_n \vee p_n \\
\neg x_n \vee q_n
\end{array}
$$

A conjunção destas é satisfatível se, e somente se, γ é satisfatível. □

Segue mais um exemplo, ainda sem definir exatamente como as variáveis novas são criadas para as aplicações sucessivas do Teorema 11. Agora são colocadas diretamente disjunções da forma $\neg x_i \vee \alpha$ em vez de $x_i \rightarrow \alpha$.

Exemplo 63 Seja a fórmula $(p \rightarrow q) \rightarrow ((p \rightarrow r) \wedge q)$. Uma fórmula equivalente na FNN seria $\gamma = (p \wedge \neg q) \vee ((\neg p \vee r) \wedge q)$. Então:

γ é sat sse $x_1 \wedge [\neg x_1 \vee \gamma]$ é sat
sse $x_1 \wedge [\neg x_1 \vee x_2 \vee ((\neg p \vee r) \wedge q)] \wedge (\neg x_2 \vee (p \wedge \neg q))$ é sat
sse $x_1 \wedge [\neg x_1 \vee x_2 \vee x_3] \wedge [\neg x_2 \vee (p \wedge \neg q)] \wedge (\neg x_3 \vee ((\neg p \vee r) \wedge q))$ é sat
sse $x_1 \wedge [\neg x_1 \vee x_2 \vee x_3] \wedge [\neg x_2 \vee (p \wedge \neg q)] \wedge [\neg x_3 \vee (x_4 \wedge q)] \wedge (\neg x_4 \vee (\neg p \vee r))$ é sat.

Como $\neg x_2 \vee (p \wedge \neg q) \equiv (\neg x_2 \vee p) \wedge (\neg x_2 \vee \neg q)$, $\neg x_3 \vee (x_4 \wedge q) \equiv (\neg x_3 \vee x_4) \wedge (\neg x_3 \vee q)$ e $\neg x_4 \vee (\neg p \vee r) \equiv \neg x_4 \vee \neg p \vee r$, segue-se que γ é satisfatível se, e somente se, é satisfatível o conjunto das cláusulas:

$$
\begin{array}{l}
x_1 \\
\neg x_1 \vee x_2 \vee x_3 \\
\neg x_2 \vee p \\
\neg x_2 \vee \neg q \\
\neg x_3 \vee x_4 \\
\neg x_3 \vee q \\
\neg x_4 \vee \neg p \vee r.
\end{array}
$$

Note que a cláusula x_1 pode ser suprimida e a segunda cláusula simplificada para $x_2 \vee x_3$. Ou seja, a criação de x_1 na primeira linha da argumentação acima não precisa ser feita. No algoritmo a ser visto em seguida, por simplicidade, essa primeira variável será sempre criada. Mas ela poderá ser eliminada como neste exemplo. □

O algoritmo da Figura 2.38 obtém o conjunto de cláusulas com base no Teorema 11, supondo que a fórmula esteja na FNN. No algoritmo, para uma (sub)fórmula β, $\eta_\beta = \beta$ se β for um literal, e η_β é uma variável *nova* se β não for um literal. O algoritmo supõe que $subfnn(\alpha)$, para α na FNN, é o conjunto das subfórmulas de α segundo a seguinte definição (adaptada da Definição 4):

proc cláusulas(fórmula na FNN α) **retorna** fórmula na FNC:
  fórmula β;
  para cada $\beta \in$ *subfnn*(α) seja η_β como definido no texto;
  seja γ a conjunção de todas as fórmulas assim obtidas:
    .para cada $\phi = \beta_1 \wedge \cdots \wedge \beta_n \in$ *subfnn*(α):
      $(\neg\eta_\phi \vee \eta_{\beta_1}) \wedge \ldots \wedge (\neg\eta_\phi \vee \eta_{\beta_n})$;
    .para cada $\psi = \beta_1 \vee \cdots \vee \beta_n \in$ *subfnn*(α):
      $\neg\eta_\psi \vee \eta_{\beta_1} \vee \ldots \vee \eta_{\beta_n}$;
  retorne $\eta_\alpha \wedge \gamma$
fim cláusulas

Figura 2.38: FNC a partir de FNN em tempo linear.

Definição 18 *O conjunto das* subfórmulas *de uma fórmula α na FNN, subfnn(α), pode ser assim definido:*

a) se α é literal, $\top$ ou $\bot$, então subfnn$(\alpha) = \{\alpha\}$;

b) se $\alpha = (\alpha_1 \wedge \cdots \wedge \alpha_n)$ ou $\alpha = (\alpha_1 \vee \cdots \vee \alpha_n)$, $n \geq 2$, então subfnn$(\alpha) = \{\alpha\} \cup subfnn(\alpha_1) \cup \cdots \cup subfnn(\alpha_n)$. □

Exemplo 64 Seja novamente a fórmula $\alpha = (p \to q) \to ((p \to r) \wedge q)$ do Exemplo 63. O algoritmo da Figura 2.38, aplicado à fórmula equivalente na FNN $\gamma = (p \wedge \neg q) \vee ((\neg p \vee r) \wedge q)$, usando

- $\eta_\gamma = x_1$,
- $\eta_{p \wedge \neg q} = x_2$,
- $\eta_{(\neg p \vee r) \wedge q} = x_3$ e
- $\eta_{\neg p \vee r} = x_4$,

retorna a conjunção das cláusulas exibidas ao final do Exemplo 63. □

O desempenho do algoritmo apresentado é de ordem polinomial no comprimento das fórmulas na FNN. Uma fórmula qualquer sem conectivos $\leftrightarrow$ aninhados pode ser transformada em tempo linear em uma equivalente na FNN.[25] Assim, o problema da satisfabilidade para tais fórmulas fica reduzido, em tempo polinomial, ao da satisfabilidade para cláusulas, o que aumenta a importância dos métodos que funcionam para fórmulas em FNC, como os baseados em DPLL.

Na verdade, dada uma fórmula α qualquer (mesmo envolvendo o bicondicional), é possível obter uma fórmula na FNC que seja satisfatível se, e somente se, α é satisfatível

[25]Note que na extrema maioria dos contextos reais não existe a possibilidade de se ter fórmulas com muitos $\leftrightarrow$'s aninhados do tipo $(\alpha_1 \leftrightarrow (\alpha_2 \leftrightarrow (\cdots)))$, por exemplo, que possam causar um crescimento exponencial da fórmula na transformação para FNN.

```
proc propague(fórmula na FNC α) retorna fórmula na FNC:
    cláusula C,C′;
    enquanto α contém uma cláusula unitária l faça
        α := resultado de remover de α toda cláusula que contenha l;
        se α = ∅ então retorne ⊤ fimse;
        para cada cláusula C de α que contém l̄ faça
            C′ := C − {l̄};
            se C = ∅ então retorne ⊥ fimse;
            α := resultado de substituir, em α, C por C′
        fimpara
    fimenquanto;
    retorne α
fim propague
```

Figura 2.39: Propagação por cláusulas unitárias.

em tempo polinomial,[26] embora com um algoritmo um pouco mais sofisticado e que produza mais cláusulas para casos típicos do que o algoritmo visto acima.

O algoritmo original do método DPLL tem dois componentes básicos:

- propagação pelas unitárias; e
- escolha de literal a receber valor.

A *propagação pelas unitárias* é inspirada no fato de que uma cláusula unitária (aquela com um único literal), l, força a atribuição de V a l^i, se l for positivo, ou de F a $\overline{l}^i$, se l for negativo. Ao se fazer isso, toda cláusula que contenha o literal pode ser considerada como satisfeita e os literais complementares podem ser retirados das cláusulas restantes (já que $\overline{l}^i = F$). Esta última providência pode fazer com que apareçam novas cláusulas unitárias, o que justifica aplicar repetitivamente o processo enquanto possível. Tal processo, frequentemente denominado BCP (de *boolean constraint propagation*), está especificado no algoritmo da Figura 2.39.

Exemplo 65 Seja α a fórmula na FNC que contém as seguintes cláusulas:

1. p_1

2. $\neg p_1 \vee \neg p_2$

3. $p_1 \vee q_1$

4. $p_1 \vee r$

5. $\neg p_1 \vee \neg q_1 \vee r$

[26] Para isto, basta usar o que se denomina uma transformação definicional, como definida em Plaisted e Greenbaum em A Structure-preserving Clause Transformation, *Journal of Symbolic Computation 2*(1986), pp 293–304.

6. $\neg p_2 \vee \neg r$

7. $p_2 \vee \neg q_1 \vee q_2$

8. $q_1 \vee q_3$

A chamada propague(α) produz o seguinte comportamento: na primeira execução do corpo do **enquanto**, é identificada a cláusula unitária $l = p_1$; removendo de α toda cláusula que contém p_1, ela fica com as cláusulas:

2. $\neg p_1 \vee \neg p_2$

5. $\neg p_1 \vee \neg q_1 \vee r$

6. $\neg p_2 \vee \neg r$

7. $p_2 \vee \neg q_1 \vee q_2$

8. $q_1 \vee q_3$

Em seguida, $\overline{l} = \neg p_1$ é eliminada das cláusulas em que aparece, **2** e **5**, ficando α com:

2′. $\neg p_2$

5′. $\neg q_1 \vee r$

6. $\neg p_2 \vee \neg r$

7. $p_2 \vee \neg q_1 \vee q_2$

8. $q_1 \vee q_3$

Como a cláusula **2′** é unitária, o corpo do comando **enquanto** é executado uma segunda vez, sendo inicialmente identicada $l = \neg p_2$; removendo de α toda cláusula que contém $\neg p_2$, ela fica com as cláusulas:

5′. $\neg q_1 \vee r$

7. $p_2 \vee \neg q_1 \vee q_2$

8. $q_1 \vee q_3$

Em seguida, $\overline{l} = p_2$ é eliminada da cláusula **7**, ficando α com

5′. $\neg q_1 \vee r$

7′. $\neg q_1 \vee q_2$

8. $q_1 \vee q_3$

Como não há mais cláusula unitária, o procedimento retorna a conjunção das cláusulas **5′**, **7′** e **8**: $(\neg q_1 \vee r) \wedge (\neg q_1 \vee q_2) \wedge (q_1 \vee q_3)$. □

```
proc DPLL(fórmula na fnc α) retorna {verdadeiro, falso}:
    literal l;
    α := propague(α);
    se α = ⊤ então
        retorne verdadeiro
    senãose α = ⊥ então
        retorne falso
    fimse;
    l := escolha um literal em α;
    se DPLL(α ∧ l) então
        retorne verdadeiro
    senão
        retorne DPLL(α ∧ l̄)
    fimse
fim DPLL
```

Figura 2.40: O algoritmo DPLL para satisfabilidade.

O outro componente básico do algoritmo é a escolha de literal a receber valor. No algoritmo DPLL original não é especificada nenhuma política específica de escolha; aqui também não será. A escolha será expressa mediante o pseudocomando "escolha um literal em α" para enfatizar que uma escolha apropriada do literal pode depender de características da fórmula α. Por outro lado, o não determinismo daqueles algoritmos relativo à escolha do valor V ou F não existirá aqui. Primeiro, tenta-se atribuir o valor V ao literal, depois F, como mostra a Figura 2.40.

Segue um exemplo bem simples de aplicação do algoritmo.

Exemplo 66 Uma fórmula na FNC equivalente a $(p \vee \neg(q \wedge r)) \wedge \neg((p \leftrightarrow r) \vee q)$ foi obtida no Exemplo 57, qual seja:

$$\alpha = (p \vee \neg q \vee \neg r) \wedge (p \vee r) \wedge (\neg r \vee \neg p) \wedge \neg q.$$

O procedimento DPLL, com α como argumento, começa chamando propague, que retorna a nova fórmula $\alpha = (p \vee r) \wedge (\neg r \vee \neg p)$. Em seguida, escolhe um literal desta última; suponha que seja $l = \neg p$. Ocorre a chamada recursiva com argumento

$$\alpha = (p \vee r) \wedge (\neg r \vee \neg p) \wedge \neg p$$

Ao ser chamado, o procedimento propague retorna $\top$ na segunda iteração de seu comando **enquanto**, o que faz com que a chamada recursiva retorne *verdadeiro*, provocando o retorno de *verdadeiro* para a fórmula α inicial.

Note que, implicitamente, o algoritmo descobriu um modelo para a fórmula:

- $q^i = F$, pois a cláusula unitária $\neg q$ é usada na primeira chamada a propague;
- $p^i = F$, pois o literal $\neg p$ é escolhido, ou, o que dá no mesmo: a cláusula unitária $\neg p$ é usada na segunda chamada a propague/primeira iteração do **enquanto**;

- $r^i = V$, pois a cláusula unitária r é usada na segunda chamada a propague/segunda iteração do **enquanto**. □

Segue um exemplo de aplicação do algoritmo com uma fórmula insatisfatível, também bastante simples.

Exemplo 67 Para provar que $(p \leftrightarrow q) \wedge (p \vee q) \rightarrow (p \wedge q)$ é tautologia, basta provar que $\neg((p \leftrightarrow q) \wedge (p \vee q) \rightarrow (p \wedge q))$ é contraditória, ou seja, é insatisfatível. Obtendo uma fórmula na FNC equivalente:

$$\begin{aligned}
&\neg((p \leftrightarrow q) \wedge (p \vee q) \rightarrow (p \wedge q)) \\
&\quad \equiv \neg((p \rightarrow q) \wedge (q \rightarrow p) \wedge (p \vee q) \rightarrow (p \wedge q)) \\
&\quad \equiv \neg((\neg p \vee q) \wedge (\neg q \vee p) \wedge (p \vee q) \rightarrow (p \wedge q)) \\
&\quad \equiv (\neg p \vee q) \wedge (\neg q \vee p) \wedge (p \vee q) \wedge \neg(p \wedge q)) \\
&\quad \equiv (\neg p \vee q) \wedge (\neg q \vee p) \wedge (p \vee q) \wedge (\neg p \vee \neg q)
\end{aligned}$$

Começando o DPLL com

$$\alpha = (\neg p \vee q) \wedge (\neg q \vee p) \wedge (p \vee q) \wedge (\neg p \vee \neg q),$$

propague(α) retorna a própria α, visto que esta não tem cláusula unitária. Em seguida, DPLL escolhe um literal. Suponha que seja $l = p$. Ocorre a chamada recursiva de DPLL com

$$\alpha = (\neg p \vee q) \wedge (\neg q \vee p) \wedge (p \vee q) \wedge (\neg p \vee \neg q) \wedge p.$$

Agora, o algoritmo propague retorna $\perp$. Com isto, o DPLL aciona a outra chamada recursiva com

$$\alpha = (\neg p \vee q) \wedge (\neg q \vee p) \wedge (p \vee q) \wedge (\neg p \vee \neg q) \wedge \neg p.$$

Neste caso, também, o algoritmo propague retorna $\perp$. E o DPLL, então, retorna *falso*, significando que a fórmula $(\neg p \vee q) \wedge (\neg q \vee p) \wedge (p \vee q) \wedge (\neg p \vee \neg q)$ é insatisfatível. □

Como já ressaltado anteriormente, cláusulas tautológicas podem ser eliminadas sem afetar a equivalência lógica da fórmula obtida com a original. Dessa forma, antes de submeter uma fórmula na FNC ao algoritmo, pode-se simplesmente eliminar todas as cláusulas tautológicas. Um outro tipo de eliminação, aliás presente no DPLL original, é o de cláusulas com *literais puros*. Um literal l que ocorra em uma fórmula na FNC α é dito *puro* se não existe cláusula de α que contenha $\overline{l}$. Uma cláusula com literal puro pode ser eliminada sem afetar a satisfabilidade. A explicação é que, se l ocorre em α e $\overline{l}$ não, ao fazer $v^i(l) = V$, toda cláusula em que ocorre l é satisfeita; e removendo-se tais cláusulas, obtém-se uma fórmula sem o literal $\overline{l}$. Isso pode ser feito, não apenas antes de submeter a fórmula ao algoritmo, como no caso das cláusulas tautológicas, mas também após a eliminação de cláusulas e literais de cláusulas durante a aplicação do algoritmo. A efetividade disso depende das características da fórmula recebida como argumento (indiretamente, da aplicação).

A escolha do literal, não determinística no algoritmo apresentado, é muito importante para se conseguir implementações mais eficientes, tendo sido objeto da proposição de vários tipos heurísticas.

O algoritmo da Figura 2.40 tem uma implementação (recursiva ou não) em que a busca propiciada pelas atribuições tentativas a literais escolhidos é realizada mediante *backtracking*. Uma implementação simples utiliza um backtracking cronológico padrão: se a chamada DPLL($\alpha \wedge l$) retorna falso, recupera-se o estado imediatamente antes da chamada e procede-se à chamada DPLL($\alpha \wedge \overline{l}$). Isso é o que provoca um desempenho exponencial com relação a tempo gasto. Muitos trabalhos têm sido feitos ultimamente para utilizar métodos de backtracking "inteligentes".

Nesta seção foi apresentado o algoritmo DPLL básico. Existem várias versões desse algoritmo que, como dito acima, incorporam estruturas e heurísticas no intuito de melhorar o desempenho do mesmo, pelo menos para classes específicas de aplicações.

A seguir, será apresentada a noção de consequência lógica, que modela as situações em que se pode dizer que uma proposição segue (logicamente) de um conjunto de proposições assumidas como verdadeiras. É baseado nessa noção que se pode definir o que segue de uma base de conhecimento de forma que se possa depois, no nível formal, trabalhar com base em um sistema dedutivo para realizar inferências. Este último, de qualquer forma, só pode ser avaliado, quanto a correção e completude, a partir da definição de consequência lógica.

Exercícios

1. Projete um algoritmo similar a cláusulas da Figura 2.38 que receba como argumento uma fórmula qualquer, não necessariamente na FNN.

2. Modifique o algoritmo DPLL para que seja determinado um modelo da fórmula, caso exista algum. *Dica*: Observe o final do Exemplo 66.

3. Modifique o algoritmo DPLL para que faça a eliminação de cláusulas com literais puros.

4. Apresente alguma heurística, acompanhada de justificativa, para a escolha do próximo literal no algoritmo DPLL.

2.7 Consequência lógica

A seguir, será abordado o conceito de consequência lógica, que explicita o que significa dizer que uma certa proposição segue necessariamente de um conjunto de outras proposições.

Definição 19 *Uma fórmula α é uma* consequência lógica *de um conjunto de fórmulas H (ou H* implica logicamente *α), $H \models \alpha$, se, e somente se, para qualquer i, se i satisfaz H, então i satisfaz α.*

Em outras palavras, $H \models \alpha$ se, e somente se, todo modelo para H é modelo para α. Em particular, observe que $H \models \tau$ para todo $H \subseteq \mathcal{F}$ e toda tautologia τ; assim, por exemplo, $H \models \top$ para todo $H \subseteq \mathcal{F}$ e $\emptyset \models \tau$ para toda tautologia τ. Além disso, se não existir modelo para H, qualquer fórmula será consequência lógica de H.

Exemplo 68 As únicas interpretações i que satisfazem $p \wedge q$ (ou seja, os únicos modelos para $p \wedge q$) são aquelas em que $p^i = q^i = V$. E, neste caso, $v^i(p) = V$, $v^i(q) = V$, $v^i(p \vee q) = V$, $v^i(p \to q) = V$, o que permite concluir, respectivamente que:

- $\{p \wedge q\} \models p$;
- $\{p \wedge q\} \models q$;
- $\{p \wedge q\} \models p \vee q$;
- $\{p \wedge q\} \models p \to q$.

Analisando-se as tabelas da verdade, vê-se que sempre que $v^i(p) = V$, segue-se que $v^i(p \vee q) = V$, $v^i(q \vee p) = V$ e $v^i(q \to p) = V$. Logo:

- $\{p\} \models p \vee q$;
- $\{p\} \models q \vee p$;
- $\{p\} \models q \to p$.

□

A seguir apresenta-se dois exemplos de padrões de inferência clássicos da lógica, mostrando-se, assim, que eles são válidos.

Exemplo 69 Um modelo i para $\{p \to q, p\}$ é tal que $v^i(p) = V$ e $v^i(p \to q) = V$. Disso, segue-se que $v^i(q) = V$. Portanto, $\{p \to q, p\} \models q$. Assim, dado que $p \to q$ e p sejam verdadeiras, q é necessariamente verdadeira! Na verdade, raciocínio análogo mostra um resultado mais geral: $\{\alpha \to \beta, \alpha\} \models \beta$ *para quaisquer fórmulas* α e β. Isto corrobora uma das regras de inferência mais utilizadas em demonstrações de teoremas: a regra *modus ponens*. Esta regra diz: se forem deduzidas as fórmulas $\alpha \to \beta$ e α, então pode-se deduzir a fórmula β.

De maneira similar, tem-se a regra *modus tollens*, que permite concluir $\neg\alpha$ a partir de $\alpha \to \beta$ e $\neg\beta$. Esta regra é corroborada pelo fato de que $\{\alpha \to \beta, \neg\beta\} \models \neg\alpha$. □

O conectivo lógico $\to$ tem uma estreita relação com consequência lógica, relação esta expressa pelo teorema a seguir.

Teorema 12 *Sejam $H \subseteq \mathcal{F}$ e $\alpha \in \mathcal{F}$. Então $H \cup \{\alpha\} \models \beta$ se, e somente se, $H \models \alpha \rightarrow \beta$.*

Prova

($\rightarrow$) Suponha que $H \cup \{\alpha\} \models \beta$. Seja um modelo i para H. Basta mostrar que $v^i(\alpha \rightarrow \beta) = V$. Dois casos:

Caso 1. $v^i(\alpha) = V$. Então i é também modelo para $H \cup \{\alpha\}$ e, como $H \cup \{\alpha\} \models \beta$, $v^i(\beta) = V$. Como se $v^i(\alpha) = V$, então $v^i(\beta) = V$, segue-se que $v^i(\alpha \rightarrow \beta) = V$.

Caso 2. $v^i(\alpha) = F$. Neste caso, $v^i(\alpha) \bigoplus v^i(\beta) = V$ e, portanto, $v^i(\alpha \rightarrow \beta) = V$.

($\leftarrow$) Suponha que $H \models \alpha \rightarrow \beta$. Seja um modelo i para $H \cup \{\alpha\}$. Basta mostrar que $v^i(\beta) = V$. Sendo i um modelo para $H \cup \{\alpha\}$, i é modelo para H. Assim, como $H \models \alpha \rightarrow \beta$, $v^i(\alpha \rightarrow \beta) = V$. Como $v^i(\alpha) = V$, segue-se destas duas últimas que $v^i(\beta) = V$. □

Observe, em particular, que se $H = \emptyset$, então $\{\alpha\} \models \beta$ se, e somente se, $\emptyset \models \alpha \rightarrow \beta$, ou seja $\{\alpha\} \models \beta$ se, e somente se, $\alpha \rightarrow \beta$ é uma tautologia.

Exemplo 70 Para mostrar que $\{q\} \models p \rightarrow q$, basta mostrar que $q \rightarrow (p \rightarrow q)$ é uma tautologia, o que pode ser feito de muitas formas. Por exemplo, pode-se usar o método da tabela da verdade. Pode-se usar um dos algoritmos de verificação de satisfabilidade, já que $q \rightarrow (p \rightarrow q)$ é uma tautologia se, e somente se, $\neg(q \rightarrow (p \rightarrow q))$ é insatisfatível. Pode-se também usar o sistema formal $\mathcal{S}_T$ (encontrando a seguinte derivação, por exemplo, através do algoritmo taut):

1. $\neg q, \neg p, q$ (Ax)
2. $\neg q, p \rightarrow q$ (condp 1)
3. $q \rightarrow (p \rightarrow q)$ (condp 2)

Ou pode-se ainda usar o sistema formal $\mathcal{S}_I$, provando que é $\neg(q \rightarrow (p \rightarrow q))$ insatisfatível:

1. $q, p, \neg q$ (Ax)
2. $q, \neg(p \rightarrow q)$ (condn 1)
3. $\neg(q \rightarrow (p \rightarrow q))$ (condn 2)

□

Exemplo 71 Qualquer fórmula da forma $(\alpha \rightarrow \beta) \wedge (\beta \rightarrow \gamma) \rightarrow (\alpha \rightarrow \gamma)$ é tautológica. Segue-se, pelo teorema anterior, que $\{(\alpha \rightarrow \beta) \wedge (\beta \rightarrow \gamma)\} \models \alpha \rightarrow \gamma$. Disto, segue-se também que $\{\alpha \rightarrow \beta, \beta \rightarrow \gamma\} \models \alpha \rightarrow \gamma$. Isto corrobora a chamada regra do *silogismo hipotético*: de $\alpha \rightarrow \beta$ e $\beta \rightarrow \gamma$, pode-se deduzir $\alpha \rightarrow \gamma$. □

Dadas duas fórmulas α e β, se $\{\alpha\} \models \beta$, nem sempre $\{\beta\} \models \alpha$. A relação entre consequência e equivalência lógicas é dada por:

$$\alpha \equiv \beta \textit{ se, e somente se, } \{\alpha\} \models \beta \textit{ e } \{\beta\} \models \alpha.$$

Fórmulas da forma $\bot \rightarrow \beta$ são tautologias. Logo, pelo Teorema 12, segue-se que $H \cup \{\bot\} \models \beta$, para qualquer fórmula β. Notando-se que $\bot \equiv \gamma$, para qualquer contradição γ, vê-se que qualquer fórmula é consequência lógica de um conjunto de fórmulas que contenha uma contradição. Mais do que isso, se $H \models \gamma$, sendo γ uma contradição, então $H \models \alpha$ para toda $\alpha \in \mathcal{F}$. [27]

O teorema a seguir faz uma ligação entre os conceitos de consequência lógica e satisfabilidade, mostrando que eles são redutíveis um ao outro.

Teorema 13 *Sejam $H \subseteq \mathcal{F}$ e $\alpha \in \mathcal{F}$. Então $H \models \alpha$ se e somente se $H \cup \{\neg\alpha\}$ é insatisfatível.*

Prova

($\rightarrow$) Suponha que $H \models \alpha$. Seja i uma interpretação arbitrária. Se i é modelo para H, então como $H \models \alpha$, i é modelo para α e, neste caso, $v^i(\neg\alpha) = F$; portanto, i não é modelo para $H \cup \{\neg\alpha\}$. E se i não é modelo para H, então não é modelo para alguma fórmula de H e, portanto, não é modelo também para $H \cup \{\neg\alpha\}$. Como i é arbitrária, não existe modelo para $H \cup \{\neg\alpha\}$.

($\leftarrow$) Suponha que $H \cup \{\neg\alpha\}$ é insatisfatível. Seja i um modelo para H. Como $H \cup \{\neg\alpha\}$ é insatisfatível, segue-se que $v^i(\neg\alpha) = F$. Logo, $v^i(\alpha) = V$. Como i é modelo arbitrário de H, conclui-se que $H \models \alpha$. □

O problema de, dados um conjunto de fórmulas H e uma fórmula α, determinar se $H \models \alpha$, pode ser atacado de múltiplas formas. Por exemplo:

- por meio do algoritmo insat, da Figura 2.30, recebendo $H \cup \{\neg\alpha\}$ como argumento: $H \models \alpha$ se e somente se insat$(H \cup \{\neg\alpha\}) =$ *verdadeiro*; pode-se apresentar, por exemplo, uma derivação no sistema $\mathcal{S}_I$;
- por meio de uma versão determinística de satB-nd, satB, em que a variável A é inicializada com $H \cup \{\neg\alpha\}$, que retorne a resposta afirmativa se, e somente se, $H \cup \{\neg\alpha\}$ é insatisfatível; neste caso, pode-se também apresentar uma derivação no sistema $\mathcal{S}_I$.
- por meio de um algoritmo do tipo DPLL que receba como entrada $H \cup \{\neg\alpha\}$ na FNC.
- por meio de um algoritmo baseado no sistema $\mathcal{S}_T$, similar a insat, que receba como entrada $\{\neg\gamma \mid \gamma \in H\} \cup \{\alpha\}$ (veja Exercício 10 da Seção 2.4.3); neste caso, pode-se também apresentar uma derivação no sistema $\mathcal{S}_T$.

Na Seção 2.8 serão vistos sistemas formais especificamente construídos para determinação de consequência lógica, e que não dependem, portanto, de um resultado como o do Teorema 13.

[27] Tal propriedade, embora corroborada pela lógica clássica, pode não ser desejável em ampla gama de situações. Assim, por exemplo, em um sistema baseado em conhecimento, pode ser mais adequada uma lógica subjacente para expressão e manipulação de conhecimento em que isto não aconteça, ou seja, em que a presença de contradições não contamine o sistema de forma que qualquer fórmula passe a ser consequência lógica da base de conhecimento. Uma tal lógica (existem várias) é dita ser *paraconsistente*.

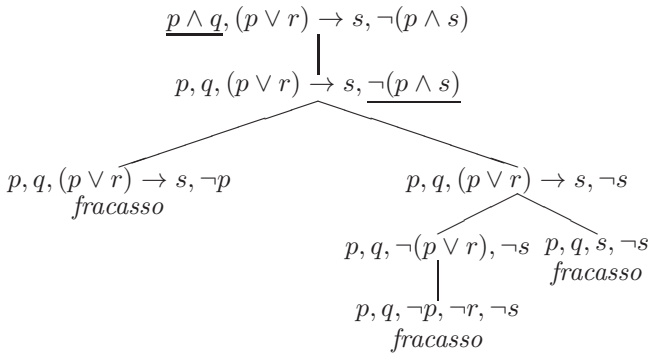


Figura 2.41: Exemplificando uso de satB-nd para consequência lógica.

Exemplo 72 Usando-se o algoritmo satB-nd, inicializando-se A como dito acima, tem-se uma árvore de computação para $\{p \wedge q, (p \vee r) \rightarrow s\} \cup \{\neg(p \wedge s)\}$ cujos ramos são todos de fracasso, mostrando que tal conjunto é insatisfatível e que, portanto, $\{p \wedge q, (p \vee r) \rightarrow s\} \models p \wedge s$. A árvore de computação, em sua versão simplificada, está mostrada na Figura 2.41. Usando-se o sistema formal $\mathcal{S}_I$:

1. $p, q, \neg p, \neg r, \neg s$ (Ax)
2. $p, q, \neg(p \vee r), \neg s$ (disjn 1)
3. $p, q, s, \neg s$ (Ax)
4. $p, q, (p \vee r) \rightarrow s, \neg s$ (condp 2,3)
5. $p, q, (p \vee r) \rightarrow s, \neg p$ (Ax)
6. $p, q, (p \vee r) \rightarrow s, \neg(p \wedge s)$ (conjn 5,4)
7. $p \wedge q, (p \vee r) \rightarrow s, \neg(p \wedge s)$ (conjp 6)

Usando-se o sistema formal $\mathcal{S}_T$:

1. $\neg p, \neg q, p, r, s$ (Ax)
2. $\neg p, \neg q, p \vee r, s$ (disjp 1)
3. $\neg p, \neg q, \neg s, s$ (Ax)
4. $\neg p, \neg q, \neg((p \vee r) \rightarrow s), s$ (condn 2,3)
5. $\neg p, \neg q, \neg((p \vee r) \rightarrow s), p$ (Ax)
6. $\neg p, \neg q, \neg((p \vee r) \rightarrow s), p \wedge s$ (conjp 5,4)
7. $\neg(p \wedge q), \neg((p \vee r) \rightarrow s), p \wedge s$ (conjn 6)

□

Propriedades bastante intuitivas da consequência lógica, utilizadas no dia a dia quando se demonstra teoremas são ($H \subseteq \mathcal{F}$ e $\alpha, \beta \in \mathcal{F}$):

- $H \cup \{\alpha\} \models \alpha$ para toda α. Ou seja, qualquer fórmula é consequência de um conjunto de fórmulas que a contém.
- Se $H \models \alpha$ e $H \cup \{\alpha\} \models \beta$, então $H \models \beta$. Ou seja, se uma fórmula β é consequência de um conjunto de fórmulas H acrescido de uma fórmula α, sendo α consequência do conjunto H, segue-se que β é consequência de H.

- Se $H \models \alpha$, então $H \cup \{\beta\} \models \alpha$. Ou seja, ao se acrescentar novas fórmulas a um conjunto, toda afirmativa que era consequência antes do acréscimo continua sendo consequência após o acréscimo. Uma lógica para a qual esta propriedade se verifica é dita ser *monotônica*.[28]

Por meio do conceito de consequência lógica pode-se definir o que é uma *teoria*.

Definição 20 *Uma* teoria T *é um conjunto de fórmulas fechado sob consequência lógica. Ou seja,* $T \subseteq \mathcal{F}$ *é uma teoria se, e somente se, se* $T \models \alpha$ *então* $\alpha \in T$.

Dado um conjunto $A \subseteq \mathcal{F}$, usa-se a notação $Cn(A)$ para designar o conjunto de todas as consequências de A, ou seja, $Cn(A) = \{\alpha \mid A \models \alpha\}$. Assim, dizer que $A \models \alpha$ é o mesmo que dizer que $\alpha \in Cn(A)$. Evidentemente, $Cn(A) = Cn(Cn(A))$ para todo $A \subseteq \mathcal{F}$. Pode-se também afirmar que T é uma teoria se, e somente se, $T = Cn(T)$. A *menor* teoria é $T = Cn(\emptyset) = \{\alpha \mid \emptyset \models \alpha\}$, ou seja, o conjunto de todas as tautologias. Pode-se mostrar ainda que toda tautologia pertence a toda teoria. Por outro lado, a *maior* teoria é $\mathcal{F}$. Esta é a única teoria insatisfatível (já que toda fórmula é consequência lógica de um conjunto insatisfatível): uma teoria T é insatisfatível se, e somente se, $T = \mathcal{F}$. Segue-se que uma teoria T é satisfatível se, e somente se, existe $\alpha \in \mathcal{F}$ tal que $\alpha \notin T$.

Muitas vezes, se especifica uma "teoria" exibindo-se um conjunto A, de axiomas (nível formal), que expressam os chamados *postulados* (nível conceitual) da teoria. A teoria propriamente é o conjunto $Cn(A)$.

Seja i uma interpretação qualquer, e $\mathcal{T}(i) = \{\alpha \in \mathcal{F} \mid v^i(\alpha) = V\}$. Como, $\mathcal{T}(i) \subseteq \mathcal{F}$ e se $\mathcal{T}(i) \models \alpha$ então $v^i(\alpha) = V$ e, assim, $\alpha \in \mathcal{T}(i)$, conclui-se, pela definição 20, que $\mathcal{T}(i)$ é uma teoria. De forma análoga, mostra-se também que se I é um conjunto de interpretações, então $\mathcal{T}(I) = \{\alpha \in \mathcal{F} \mid v^i(\alpha) = V \text{ para toda } i \in I\}$ também é uma teoria (veja Exercício 8). Assim, se I é o conjunto de todos os modelos correspondentes a uma certa aplicação, o problema da formalização usando lógica proposicional é o de encontrar um conjunto de fórmulas A (axiomas da teoria) tal que $\mathcal{T}(I) = Cn(A)$.

Uma teoria T é dita *completa* se, e somente se, para qualquer $\alpha \in \mathcal{F}$, $\alpha \in T$ ou $\neg\alpha \in T$. Assim, por exemplo, dada uma interpretação i qualquer, $\mathcal{T}(i)$ é sempre uma teoria completa, já que se $\alpha \notin \mathcal{T}(i)$, então $v^i(\alpha) = F$ e, neste caso, $v^i(\neg\alpha) = V$ e portanto $\neg\alpha \in \mathcal{T}(i)$. Por outro lado, $\mathcal{T}(I)$ não é completa se I tem mais de um membro (Veja Exercício 10).

Exercícios

1. Mostre que:

a) $\emptyset \models (\neg\alpha \to \alpha) \to \alpha$ para toda $\alpha \in \mathcal{F}$.

b) $\{\neg p\} \models p \to q$

[28] Em algumas aplicações, a introdução de novas afirmativas pode invalidar conclusões anteriores. Neste caso, a lógica subjacente não é a lógica clássica e sim uma lógica dita *não monotônica*.

c) $\{p \to (\top \to q)\} \models p \to q$

d) $\{p \to \neg q, q\} \models \neg p$

e) $\{p \to \neg q, p, q\} \models r$

2. Para os pares de fórmulas a seguir, descubra se são logicamente equivalentes, ou se uma é consequência lógica da outra.

a) $(p \to r) \wedge (q \to r)$ e $(p \vee q) \to r$

b) $(p \to r) \wedge (q \to s)$ e $(p \wedge q) \to (r \vee s)$

c) $(p \to r) \wedge (q \to s)$ e $(p \vee q) \to (r \wedge s)$

d) $p \leftrightarrow q$ e $(p \wedge q) \vee (\neg p \wedge \neg q)$

3. Seguem alguns argumentos expressos em português. Expresse-os em lógica proposicional. Depois verifique se os mesmos são corretos usando o conceito de consequência lógica.

a) Se eu ganhei na megasena, então estou rico. Eu não ganhei na megasena. Logo, não estou rico.

b) Se não chover, a roça não produz e o pasto seca. Se o pasto seca, o gado não resiste. Portanto, se não chover, o gado não resiste.

c) João será absolvido ou condenado. Se for absolvido, será porque a testemunha mentiu. Mas a testemunha não mentiu. Portanto, João será condenado.

d) Se Deus é todo-poderoso e extremamente bom, então não existe o diabo. Deus é todo-poderoso e existe o diabo. Portanto, Deus não é extremamente bom.

e) Se você come demais, acaba ficando obeso. Se faz exercício físico todo dia, fica bem condicionado. E se não é obeso e é bem condicionado, não adoece com facilidade. Mas você adoece com facilidade. Portanto, você come demais e não faz exercício físico todo dia.

4. Demonstre as quatro propriedades da consequência lógica ressaltadas no final da seção:

a) se $H \models \gamma$, sendo γ uma contradição, então $H \models \alpha$ para toda $\alpha \in \mathcal{F}$;

b) $H \cup \{\alpha\} \models \alpha$ para toda α.

c) Se $H \models \alpha$ e $H \cup \{\alpha\} \models \beta$, então $H \models \beta$.

d) Se $H \models \alpha$, então $H \cup \{\beta\} \models \alpha$.

5. Suponha que você tenha algoritmos para determinar, para uma fórmula qualquer, se ela é:

a) tautologia;

b) contradição;

c) satisfatível;

d) falseável.

Explique como você poderia utilizar cada um desses quatro algoritmos para determinar, dados um conjunto finito de fórmulas H e uma fórmula α, se $H \models \alpha$.

6. Uma abordagem para demonstração automática, alternativa ao método dedutivo a ser apresentado na próxima seção, é baseada em verificar se um conjunto de fórmulas tem ou não um modelo. A abordagem é justificada pelo fato de que:

$$H \models \alpha \text{ se, e somente se } H \cup \{\neg\alpha\} \models \bot$$

ou melhor:

$$H \models \alpha \text{ se, e somente se } H \cup \{\neg\alpha\} \text{ não tem um modelo.}$$

Suponha que um algoritmo determine um modelo para $H \cup \{\neg\alpha\}$. Como tal modelo pode ser usado para justificar que $H \not\models \alpha$?

7. Mostre que diferentes interpretações i_1 e i_2 levam a diferentes teorias $\mathcal{T}(i_1)$ e $\mathcal{T}(i_2)$.

8. Mostre que se I é um conjunto de interpretações, então $\mathcal{T}(I) = \{\alpha \in \mathcal{F} \mid v^i(\alpha) = V \text{ para toda } i \in I\}$ é uma teoria.

9. Dado um conjunto de fórmulas A, seja $\mathcal{M}(A)$ o conjunto de todos os modelos de A, isto é, $\mathcal{M}(A) = \{i : \mathcal{V} \to \{V, F\} \mid v^i(\alpha) = V \text{ para toda } \alpha \in A\}$. Mostre que $\mathcal{T}(\mathcal{M}(A)) = Cn(A)$.

10. Mostre que a teoria $\mathcal{T}(I)$ não é completa, se o conjunto I de interpretações tem mais de um membro.

2.8 Dedução

Como dito na Seção 2.1, um argumento envolve afirmar que uma certa proposição é verdadeira caso todas as proposições de certo conjunto sejam (consideradas) verdadeiras. Um argumento é modelado pela afirmação de que certa fórmula (a conclusão) é consequência lógica de um conjunto de fórmulas (as premissas). Assim, considere a seguir um argumento como sendo a afirmação de que certa fórmula é consequência lógica de um conjunto de fórmulas.

Um argumento que diz que α é consequência lógica de $\{\beta_1, \beta_2, \ldots, \beta_n\}$ é *correto* se, e somente se, $\{\beta_1, \beta_2, \ldots, \beta_n\} \models \alpha$ e, portanto, se $(\beta_1 \wedge \beta_2 \wedge \ldots \wedge \beta_n) \to \alpha$ é uma tautologia. Um argumento correto é dito também ser válido, legítimo. Um argumento incorreto é dito também ser inválido, ilegítimo, um sofisma.

Como já foi dito, a noção correspondente a consequência lógica no nível formal é a de dedução. Assim, um argumento "formal" seria a afirmação de que certa fórmula (a

conclusão) é dedutível a partir de um conjunto de fórmulas (as premissas). Em especial, um *argumento formal* em que a conclusão siga *diretamente* do conjunto de premissas é uma instância do que se denomina uma *regra de inferência*. Na verdade, são comuns dois tipos de sistemas formais para capturar a noção de consequência lógica:

- um em que a linguagem é $\mathcal{F}$ e em que as premissas e conclusões de regras de inferência são fórmulas;

- outro em que a linguagem contém um símbolo para representar consequência lógica e em que as premissas e conclusões são expressões dessa linguagem "enriquecida".

De qualquer forma, uma regra de inferência deve obedecer ao que prescreve a noção de consequência lógica. Para afirmar que uma fórmula (ou expressão envolvendo um símbolo para $\models$) α, a *conclusão*, segue diretamente de outras fórmulas (ou expressões envolvendo o símbolo para $\models$) com formatos β_1, β_2,..., β_n, as *premissas*, será usada a notação (a ordem dos formatos das premissas é irrelevante):

$$\frac{\beta_1 \ \beta_2 \ \cdots \ \beta_n}{\alpha}.$$

Em particular, em sistemas formais em que a linguagem é $\mathcal{F}$, para se qualificar como uma regra de inferência, os argumentos respectivos, que dizem que uma conclusão no formato α é consequência lógica das premissas de formatos β_1, β_2,..., β_n, devem ser corretos.

Definição 21 *Um* sistema dedutivo *para uma lógica é um sistema formal que prescreve em que condições uma fórmula é consequência lógica de um conjunto de fórmulas. As regras são denominadas* regras de inferência *e os axiomas são denominados* axiomas lógicos. *As derivações são denominadas* deduções.

A seguir, serão apresentados exemplos de regras de inferência passíveis de serem utilizadas em sistemas dedutivos e em sistemas computacionais subjacentes, assim como exemplos de deduções usando tais regras. Em seguida, nas Seções 2.8.2 e 2.8.3, serão apresentados sistemas dedutivos completos clássicos, um do tipo Hilbert e outro do tipo Gentzen. Depois disso, nas duas seções seguintes a essas, serão apresentados os dois sistemas dedutivos que têm sido mais utilizados em raciocínio automatizado, o de tableaux semânticos e o de resolução, com menos ênfase em aspectos formais do que os anteriores e um pouco mais de ênfase em processamento computacional. Para terminar, ainda nessa linha, será apresentado um sistema que pode ser visto como tableaux especificamente para cláusulas.

2.8.1 Exemplos de regras de inferência e deduções

Seguem vários exemplos de regras de inferência, passíveis de serem usadas em sistemas dedutivos, em sistemas computacionais, e também em demonstrações informais:

Adição:

ad1: $\dfrac{\alpha}{\alpha \vee \beta}$ ad2: $\dfrac{\beta}{\alpha \vee \beta}$

Simplificação

simp1: $\dfrac{\alpha \wedge \beta}{\alpha}$ simp2: $\dfrac{\alpha \wedge \beta}{\beta}$

Conjunção

conj: $\dfrac{\alpha \quad \beta}{\alpha \wedge \beta}$

Modus ponens

mp: $\dfrac{\alpha \rightarrow \beta \quad \alpha}{\beta}$

Modus tollens

mt: $\dfrac{\alpha \rightarrow \beta \quad \neg \beta}{\neg \alpha}$

Silogismo disjuntivo

sd1: $\dfrac{\alpha \vee \beta \quad \neg \alpha}{\beta}$ sd2: $\dfrac{\alpha \vee \beta \quad \neg \beta}{\alpha}$

Silogismo hipotético

sh: $\dfrac{\alpha \rightarrow \beta \quad \beta \rightarrow \gamma}{\alpha \rightarrow \gamma}$

Dilema construtivo

dc: $\dfrac{\alpha \rightarrow \beta \quad \gamma \rightarrow \delta \quad \alpha \vee \gamma}{\beta \vee \delta}$

Algumas dessas regras são idênticas às do sistema formal $\mathcal{S}_B$. Por exemplo, a regra ad1 é idêntica à regra disjp1 de $\mathcal{S}_B$. No entanto, os *propósitos* no uso das regras de $\mathcal{S}_B$ e no uso das regras acima são diferentes: em $\mathcal{S}_B$, a regra disjp1 expressa o fato de que para uma interpretação i (obtida de Σ_i), se $v^i(\alpha) = V$, então $v^i(\alpha \vee \beta) = V$, já que a intenção é mostrar que i é modelo para a fórmula; aqui, ad1 expressa o fato de que para *toda* interpretação i, se $v^i(\alpha) = V$, então $v^i(\alpha \vee \beta) = V$, já que a intenção no uso das regras acima é encontrar uma *dedução* de uma fórmula a partir de um conjunto de hipóteses, provando-se, assim, que a fórmula é consequência lógica de H.

Seja α uma fórmula e H um conjunto de fórmulas. Rememorando notação introduzida no Capítulo 1, para indicar que α *é dedutível a partir de* H, ou seja que existe uma derivação de α a partir de H ($H \Rightarrow \alpha$ na notação introduzida no final do Capítulo 1), será usada a notação $H \vdash \alpha$. Como já ressaltado anteriormente, uma derivação de α a partir de H será denominada *dedução* de α (a partir de H).

Sendo as regras de inferência argumentos válidos, pode-se provar por indução no comprimento das derivações que *se* $H \vdash \alpha$, *então* $H \models \alpha$. Com isto, uma maneira de verificar que $H \models \alpha$ é encontrar uma dedução de α a partir de H.

Exemplo 73 A dedução a seguir mostra que $\{p \wedge q, (p \vee r) \rightarrow s\} \vdash p \wedge s$ e, portanto, $\{p \wedge q, (p \vee r) \rightarrow s\} \models p \wedge s$.

1.	$p \wedge q$	(H)
2.	$(p \vee r) \rightarrow s$	(H)
3.	p	(simp1 1)
4.	$p \vee r$	(ad1 3)
5.	s	(mp 2,4)
6.	$p \wedge s$	(conj 3,5)

□

Segue mais um exemplo.

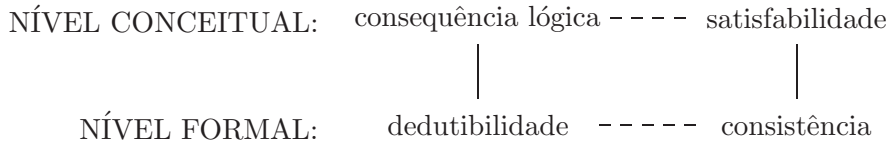


Figura 2.42: Relacionamentos nos níveis formal e conceitual.

Exemplo 74 A dedução a seguir mostra que

$$\{q \vee (r_1 \to r_2), q \to s, \neg s \to (r_2 \to p), \neg s\} \vdash r_1 \to p$$

e, portanto, $\{q \vee (r_1 \to r_2), q \to s, \neg s \to (r_2 \to p), \neg s\} \models r_1 \to p$.

1. $q \vee (r_1 \to r_2)$ (H)
2. $q \to s$ (H)
3. $\neg s \to (r_2 \to p)$ (H)
4. $\neg s$ (H)
5. $\neg q$ (mt 2,4)
6. $r_1 \to r_2$ (sd1 1,5)
7. $r_2 \to p$ (mp 3,4)
8. $r_1 \to p$ (sh 6,7)

□

Uma noção importante relativamente a deduções é a de *consistência*.

Definição 22 *Um conjunto de fórmulas $H \subseteq \mathcal{F}$ é dito* consistente *se e somente se não existe $\alpha \in \mathcal{F}$ tal que $H \vdash \alpha$ e $H \vdash \neg\alpha$.*

Dado um sistema dedutivo correto e completo, o conceito de consistência (no nível formal) corresponde ao de satisfabilidade (no nível conceitual): H é consistente e, e somente se, H é satisfatível (veja Exercício 3). A Figura 2.42 ilustra o relacionamento entre os conceitos de consequência lógica, dedutibilidade, satisfabilidade e consistência enfatizando os níveis respectivos. As linhas pontilhadas apontam relacionamentos no mesmo nível; no nível conceitual, aquele dado por:

$$H \models \alpha \text{ se somente se } H \cup \{\neg\alpha\} \text{ é insatisfatível (Teorema13),}$$

e no nível formal, aquele dado por:

$$H \vdash \alpha \text{ se somente se } H \cup \{\neg\alpha\} \text{ é inconsistente.}$$

Um resultado importante, já aludido na Seção 2.7, é que qualquer fórmula é consequência lógica de um conjunto de fórmulas insatisfatível. Considerando um sistema dedutivo completo, um resultado similar é que qualquer fórmula pode ser deduzida de

um conjunto inconsistente. Com isto, se H é inconsistente, então $H \vdash \alpha$ para toda $\alpha \in \mathcal{F}$. Ou ainda, o conjunto de todas as fórmulas dedutíveis a partir de um conjunto inconsistente é o conjunto $\mathcal{F}$. Esta característica da lógica clássica nem sempre é desejável, como já foi mencionado na Seção 2.7. Por exemplo, se uma base de conhecimentos H se tornar inconsistente com o acréscimo de uma fórmula α, então tudo passa a ser dedutível, o que é claramente indesejável. Lógicas que evitam essa "trivialização" causada por inconsistências são denominadas *paraconsistentes*. O tratamento desse tipo de lógica foge um pouco ao escopo deste texto, mas observações sobre "comportamento paraconsistente" serão emitidas de vez em quando.

Exercícios

1. Encontre deduções que demonstrem as consequências a seguir. Para isto, use as regras de inferência vistas nesta seção.

a) $\{p \vee q, \neg p, q \rightarrow r\} \models r$.
b) $\{p \rightarrow q, (q \vee \neg r_1) \rightarrow (r_2 \wedge \neg s), p\} \models \neg s$.
c) $\{p \vee q, \neg r \rightarrow \neg p, r \rightarrow s, \neg s\} \models q$.
d) $\{p \rightarrow (q \rightarrow r), p \vee r, \neg r\} \models \neg q$.
e) $\{(p \vee s_1) \rightarrow (q \rightarrow s_2), (p \vee r_1) \rightarrow (s_2 \rightarrow s_3), p \wedge \neg r_3\} \models q \rightarrow s_3$.
f) $\{\neg p \rightarrow (q \rightarrow \neg r), \neg s \rightarrow (\neg r \rightarrow p), \neg p \vee s, \neg s\} \models \neg q$.

2. Supondo um sistema dedutivo correto (ou seja, tal que se $H \vdash \alpha$ então $H \models \alpha$), mostre que:

a) se H é inconsistente, então $H \models \alpha$ para toda $\alpha \in \mathcal{F}$.
b) se $H \cup \{\alpha\}$ é inconsistente, então $H \models \neg\alpha$.

3. Considerando um sistema dedutivo correto e completo, mostre que H é consistente se, e somente se, H é satisfatível.

2.8.2 Sistemas dedutivos do tipo Hilbert

Observe que se o sistema dedutivo só contém regras de inferência baseadas em argumentos corretos, ele é correto (não deduz fórmulas que não sejam consequências lógicas). Daí, as conclusões expostas nos exemplos da seção anterior. Agora o outro lado: existe um sistema dedutivo tal que[29]

$$\text{se } H \models \alpha, \text{ então } H \vdash \alpha?$$

Ou seja, existe um sistema dedutivo completo? Se sim, quais seriam então suas regras de inferência e seus axiomas?

[29]Quando se diz $H \vdash \alpha$ pressupõe-se, evidentemente, um sistema dedutivo específico. Para explicitá-lo, poderia ser usada, por exemplo, a notação $H \vdash_S \alpha$, em que S é o sistema dedutivo. Isto não será feito para não sobrecarregar a notação; o sistema dedutivo considerado deverá ficar claro pelo contexto.

Como já foi dito, existem múltiplos sistemas dedutivos completos. Alguns podem se basear em mais regras de inferência e menos axiomas e outros em menos regras e mais axiomas (ou regras sem premissas). De qualquer forma, deve-se notar que um axioma lógico, α, deve ser uma tautologia, já que se $\emptyset \vdash \alpha$, então deve-se ter que $\emptyset \models \alpha$. Além disso, para que o sistema dedutivo seja completo toda tautologia τ deve ser dedutível, já que $\emptyset \models \tau$.

Antes de prosseguir, ressalte-se que um *esquema de axiomas*, com variáveis sintáticas para fórmulas, normalmente representa um conjunto *infinito* de axiomas. Por exemplo, um esquema de axiomas presente em alguns sistemas dedutivos é:

$$\alpha \rightarrow (\beta \rightarrow \alpha).$$

Este esquema representa um conjunto infinito de axiomas, um axioma para cada par de fórmulas α e β.

Apenas para dar um exemplo de sistema dedutivo correto e completo para a lógica proposicional, será apresentado o seguinte sistema com três esquemas de axiomas e uma regra de inferência:[30]

- Se α, β e γ são fórmulas, então são axiomas:

  Ax1: $\alpha \rightarrow (\beta \rightarrow \alpha)$
  Ax2: $(\alpha \rightarrow (\beta \rightarrow \gamma)) \rightarrow ((\alpha \rightarrow \beta) \rightarrow (\alpha \rightarrow \gamma))$
  Ax3: $(\neg\beta \rightarrow \neg\alpha) \rightarrow ((\neg\beta \rightarrow \alpha) \rightarrow \beta)$

- Regra de inferência: *modus ponens*.

Evidentemente este sistema dedutivo aproveita o fato de que o conjunto de conectivos $\{\rightarrow, \neg\}$ é completo, dadas as seguintes equivalências:

- $\alpha \vee \beta \equiv \neg\alpha \rightarrow \beta$
- $\alpha \wedge \beta \equiv \neg(\alpha \rightarrow \neg\beta)$
- $\alpha \leftrightarrow \beta \equiv (\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha) \equiv \neg((\alpha \rightarrow \beta) \rightarrow \neg(\beta \rightarrow \alpha))$

Uma leitura intuitiva do esquema de axiomas Ax1, dado que as deduções se processam via a regra *modus ponens*, seria: se α for dedutível, então se qualquer β for dedutível, ainda assim α será dedutível. Para o esquema Ax2 a leitura é um pouco mais complexa: dado que se α e β forem dedutíveis então γ também será, segue-se que se a dedução de α implicar na dedução de β, então implicará também na dedução de γ. Fica para o leitor imaginar como seria a leitura para Ax3.

Esse sistema dedutivo que acaba de ser apresentado é o que se denomina um *sistema do tipo Hilbert*, sendo de interesse principalmente teórico, não sendo adequado para servir de base para processamento em computadores. Na verdade, obter uma dedução utilizando um sistema dedutivo do tipo Hilbert pode ser uma tarefa muito penosa, e a dedução resultante pode não ser nada legível (embora correta). Segue um exemplo.

[30] Este sistema dedutivo foi tomado emprestado de Mendelson, E. *Introduction to Mathematical Logic*, 3rd edition, Wadsworth & Brooks/Cole, 1987, página 29.

Exemplo 75 A seguinte dedução mostra que $\emptyset \vdash p \to p$:

1. $(p \to ((p \to p) \to p)) \to ((p \to (p \to p)) \to (p \to p))$ (Ax2)
2. $p \to ((p \to p) \to p)$ (Ax1)
3. $(p \to (p \to p)) \to (p \to p)$ (mp 1,2)
4. $p \to (p \to p)$ (Ax1)
5. $p \to p$ (mp 3,4)

Portanto, $\emptyset \models p \to p$, ou seja $p \to p$ é uma tautologia. Compare com a demonstração no sistema $\mathcal{S}_T$, Exemplo 52. □

A partir do sistema dedutivo anterior, é possível derivar novas regras de inferência que podem facilitar enormemente o trabalho de obter uma dedução. As regras mostradas até agora, incluindo a de *modus ponens*, só permitem inferências *diretas*, ou seja, a partir de certas fórmulas já deduzidas. Um outro tipo de regra de inferência é aquela que permite deduzir fórmulas com base também em fórmulas supostas temporariamente em trechos de (sub)deduções. Por exemplo:

$$\mathsf{i}\to: \quad \frac{\boxed{\begin{array}{c}\alpha \\ \vdots \\ \beta\end{array}}}{\alpha \to \beta}$$

Esta regra diz que para deduzir uma fórmula da forma $\alpha \to \beta$, basta *supor* α como uma *hipótese adicional*, e provar β; após a dedução de $\alpha \to \beta$, a suposição α deixa de existir (é descartada); isso pode ser ressaltado em uma dedução delimitando-se de alguma forma as fórmulas desde a suposição α até a fórmula β (por exemplo, colocando-se tais fórmulas dentro de um retângulo, como sugerido na regra). A delimitação referida constitui o *escopo* em que a suposição pode ser usada como premissa.

O teorema a seguir, apresentado sem prova, justifica essa regra. Ele é denominado *teorema da dedução*.

Teorema 14 *Sejam $\alpha, \beta \in \mathcal{F}$ e $H \subseteq \mathcal{F}$. Tem-se: se $H \cup \{\alpha\} \vdash \beta$, então $H \vdash \alpha \to \beta$.*

Este teorema segue do teorema 12 e do teorema da completude da lógica proposicional (considerando-se qualquer um dos inúmeros sistemas dedutivos para a mesma), enunciado a seguir, mas pode também ser demonstrado diretamente a partir do sistema dedutivo apresentado anteriormente.[31]

O teorema da correção segue trivialmente do fato de que os axiomas para qualquer um dos esquemas Ax1, Ax2 e Ax3 são tautologias e a regra *modus ponens* é válida.

Teorema 15 *(Teorema da Correção) Sejam $H \subseteq \mathcal{F}$ e $\alpha \in \mathcal{F}$. Se $H \vdash \alpha$ então $H \models \alpha$.*

[31] Veja em Mendelson, E. *Introduction to Mathematical Logic*, 3rd edition, Wadsworth & Brooks/Cole, 1987, página 30.

A demonstração do teorema da completude é bem mais elaborada e será omitida aqui.

Teorema 16 *(Teorema da Completude) Sejam $H \subseteq \mathcal{F}$ e $\alpha \in \mathcal{F}$. Se $H \models \alpha$ então $H \vdash \alpha$.*

O exemplo a seguir mostra como o teorema da dedução pode ser usado para simplificar a obtenção de uma dedução.

Exemplo 76 Segue uma dedução que mostra que $\{p \to (q \to r), q\} \vdash p \to r$ sem usar a regra de inferência $i \to$:

1. $p \to (q \to r)$	(H)
2. q	(H)
3. $(p \to (q \to r)) \to ((p \to q) \to (p \to r))$	(Ax2)
4. $(p \to q) \to (p \to r)$	(mp 1,3)
5. $q \to (p \to q)$	(Ax1)
6. $p \to q$	(mp 2,5)
8. $p \to r$	(mp 4,6)

Agora, outra usando a nova regra:

1. $p \to (q \to r)$	(H)
2. q	(H)
3. p	(Suposição)
4. $q \to r$	(mp 1,3)
5. r	(mp 2,4)
6. $p \to r$	(i$\to$ 3—5)

□

Um expediente que pode facilitar bastante a obtenção de deduções no sistema dedutivo apresentado é o uso de lemas. Por exemplo, como a dedução de $p \to p$ a partir de $\emptyset$ do Exemplo 75 pode ser adaptada para gerar uma dedução de $\alpha \to \alpha$ para qualquer $\alpha \in \mathcal{F}$, qualquer fórmula desse formato pode ser usada (como lema) numa dedução. Ainda para facilitar, podem ser usadas regras de inferência adicionais, como uma obtida a partir do Exemplo 76: já que $\{p \to (q \to r), q\} \vdash p \to r$, a dedução lá apresentada pode ser adaptada para mostrar que $\{\alpha \to (\beta \to \gamma), \beta\} \vdash \alpha \to \gamma$ para quaisquer $\alpha, \beta, \gamma \in \mathcal{F}$, justificando o uso da regra:

$$\frac{\alpha \to (\beta \to \gamma) \quad \beta}{\alpha \to \gamma}.$$

Como o sistema dedutivo é completo, na verdade todas as regras vistas no início da Seção 2.8.1 e inúmeras outras podem ser usadas como regras adicionais (existem deduções que as corroboram).

Existe uma classe de sistemas dedutivos que usa regras de inferência "naturais" associadas a cada conectivo lógico. A cada um é associada uma regra de introdução do conectivo, para inferir uma fórmula em que o conectivo figura como principal, e uma regra de eliminação do mesmo, para inferir fórmulas a partir de premissa em que o conectivo figura como principal. Por exemplo, associadas ao conectivo $\rightarrow$ existe a regra $I\rightarrow$ acima, para inferir fórmulas da forma $\alpha \rightarrow \beta$, e existe a regra *modus ponens* para inferir fórmulas a partir de premissa da forma $\alpha \rightarrow \beta$. Tais regras são intuitivas para o ser humano e, muitas vezes, elas redundam em deduções que seriam as correspondentes formais do que se usa normalmente em demonstrações (informais) de teoremas. No entanto, nem toda dedução é tão intuitiva (natural) como seria de se desejar. De qualquer forma, neste texto não veremos um exemplo específico de sistema de dedução natural.

Nas próximas seções serão abordados sistemas dedutivos mais adequados para servir de base para processamento, por meio de computadores, de conhecimento expresso em lógica.

Exercícios

1. Prove, pelo sistema dedutivo visto na seção, sem usar e também usando a regra I$\rightarrow$ justificada pelo teorema da dedução: $\{p \rightarrow q, q \rightarrow r\} \vdash p \rightarrow r$.

2. Em algumas demonstrações por absurdo, supõe-se α e, em seguida, prova-se $\neg\alpha$. Concui-se, por absurdo, α. Como se expressa esse padrão de inferência no estilo usado para a regra i$\rightarrow$?

3. Na técnica de demonstração por casos, se $\alpha_1 \vee \alpha_2$ for verdadeira (hipótese ou provada anteriormente), então se:

- supondo α_1, for possível provar β, e
- supondo α_2, for possível provar β,

pode-se concluir β. Como se expressa esse padrão de inferência no estilo usado para a regra i$\rightarrow$?

2.8.3 Sistemas dedutivos do tipo Gentzen

Estritamente falando, a regra i$\rightarrow$ apresentada na Seção 2.8.2 não satisfaz a definição de regra de derivação (ou de inferência, no caso de sistema dedutivo) apresentada no Capítulo 1, pois tal regra não apresenta uma conclusão como dependendo de um conjunto de *premissas* simplesmente, mas como dependendo da existência de uma *subdedução* em que haja a ocorrência de uma fórmula que deve ser descartada (a suposição). Existe um tipo de sistema, muitas vezes denominado de sistema de *dedução natural*, em que ocorrem esta e mais algumas outras regras no mesmo estilo. Aqui se evita chamar tal tipo de "sistema" de sistema dedutivo, apenas pelo fato dele não se enquadrar em nossa definição, que classifica sistema dedutivo como um sistema formal (como definido

no Capítulo 1). Sendo assim, como obter um sistema formal em que haja uma regra correspondente a i→? Seria a correspondente formal da "regra":

$$\frac{\Gamma \cup \{\alpha\} \models \beta}{\Gamma \models \alpha \rightarrow \beta}$$

que expressa o intuito de dizer que dado que $\Gamma \cup \{\alpha\} \models \beta$, pode-se concluir que $\Gamma \models \alpha \rightarrow \beta$ (como diz o Teorema 12). Evidentemente isso deve ser possível, pois, do contrário, o conceito de sistema formal não seria assim tão adequado...

A idéia é simples: "enriquecer" a linguagem do sistema formal com um símbolo que promova $\models$ da metalinguagem para a linguagem básica; aqui será usado o símbolo $\vdash$ para tal propósito, consistente com o propósito de que o conceito $\models$ seja denotado (formalmente) por $\vdash$.[32] A linguagem de um sistema dedutivo do tipo Gentzen é definida a seguir.

Definição 23 *O conjunto de* sequentes*, $\mathcal{S}$, que constitui a linguagem de um sistema do tipo Gentzen, é constituído de palavras da forma $\Psi_1 \vdash \Psi_2$ em que Ψ_1 e Ψ_2 são sequências de fórmulas (elementos de $\mathcal{F}$) separadas por vírgulas.*

Denotando-se por $c(\Psi)$ o conjunto das fórmulas presentes na sequência de fórmulas Ψ, um sequente $\Psi_1 \vdash \Psi_2$ é interpretado *no presente contexto*,[33] como dizendo que toda interpretação que satisfaz todas as fórmulas de $c(\Psi_1)$ satisfaz também alguma fórmula de $c(\Psi_2)$, ou ainda, que para toda interpretação i, i falseia alguma fórmula de $c(\Psi_1)$ ou i satisfaz alguma fórmula de $c(\Psi_2)$. Sendo Ψ_1 e Ψ_2 sequências finitas, isto é o mesmo que dizer que a *disjunção das fórmulas de* Ψ_2 é consequência lógica da *conjunção das fórmulas de* Ψ_1. Assim, ter a sequência vazia no lado esquerdo equivale a ter $\top$ e tê-la no lado direito equivale a ter $\bot$. É muito comum denotar a sequência vazia em sequentes pela ausência de símbolos mesmo. Aqui será adotada também essa prática. Assim, por exemplo, em vez de escrever $\lambda \vdash p, \neg p$ escreve-se simplesmente $\vdash p, \neg p$; em vez de $\lambda \vdash \lambda$, escreve-se $\vdash$ simplesmente. A seguir, apresenta-se alguns exemplos de sequentes e o que se intenciona que eles denotem no presente contexto.

Exemplo 77 Alguns sequentes e seus significados:

- $\vdash$: diz que $\emptyset \models \bot$, ou seja, que $\bot$ é tautológica (um absurdo);
- $\vdash p$: diz que p é tautológica (o que não é verdade, evidentemente);
- $\vdash p \rightarrow p, \neg p$: diz que $(p \rightarrow p) \vee \neg p$ é tautológica;
- $p \vdash$: diz que p é contraditória;
- $p \rightarrow p, \neg p \vdash$: diz que $(p \rightarrow p) \wedge \neg p$ é contraditória;

[32] É comum também o uso de $\Rightarrow$, que, aqui, está sendo usado para derivação no sistema formal.

[33] A formulação por sequentes pode ser usada para outros tipos de lógica em que sequências devem ser vistas como sequências mesmo (repetição e ordem das fórmulas faz diferença).

- $p \to q, p \vdash q$: diz que $\{p \to q, p\} \models q$;
- $p \to q, p, \neg q \vdash q, r$: diz que $\{p \to q, p, \neg q\} \models q \vee r$. □

Na proposição do sistema formal, a intenção é que todo sequente derivado seja *válido* (correção) e que todos os sequentes válidos sejam deriváveis (completude). Segue a noção de validade para sequentes.

Definição 24 *Um sequente $\Psi_1 \vdash \Psi_2$ é dito* válido *se, e somente se, toda interpretação que satisfaz $c(\Psi_1)$, também satisfaz alguma fórmula de $c(\Psi_2)$.* □

Exemplo 78 Do Exemplo 77, apenas o terceiro e os dois últimos sequentes são válidos:

- $\vdash$ não é válido: $\emptyset \not\models \bot$;
- $\vdash p$ não é válido: $\emptyset \not\models p$;
- $\vdash p \to p, \neg p$ é válido: $\emptyset \models (p \to p) \vee \neg p$;
- $p \vdash$: não é válido: $\{p\} \not\models \bot$;
- $p \to p, \neg p \vdash$ não é válido: $\{(p \to p), \neg p\} \not\models \bot$;
- $p \to q, p \vdash q$ é válido: $\{p \to q, p\} \models q$;
- $p \to q, p, \neg q \vdash q, r$ é válido: $\{p \to q, p, \neg q\} \models q \vee r$. □

O sistema $\mathcal{S}_I$ da Seção 2.4.3 pode ser visto como um sistema do tipo Gentzen em que o lado direito de cada sequente é sempre λ, enquanto que $\mathcal{S}_T$ pode ser visto como um sistema do tipo Gentzen em que o lado esquerdo é sempre λ: $c(\Psi)$ é insatisfatível se e somente se $\Psi \vdash$ é válido; toda interpretação satisfaz alguma fórmula de $c(\Psi)$ se e somente se $\vdash \Psi$ é válido. O seguinte lema serve para fazer a ligação entre os três sistemas formais. Uma sequência Ψ' é o resultado de substituir na sequência Ψ cada fórmula por uma complementar.

Lema 2 *$\Psi_1 \vdash \Psi_2$ é válido se e somente se $\Psi_1, \Psi_2' \vdash$ é válido se e somente se $\vdash \Psi_1', \Psi_2$ é válido.*

Prova

A demonstração é simples, deixada como exercício. □

A partir dos sistemas formais $\mathcal{S}_I$ e $\mathcal{S}_T$, tendo em vista o Lema 2, pode-se reformular o esquema de axioma e as regras de modo a obter o esquema de axioma e as regras de inferência para o sistema formal do tipo Gentzen $\mathcal{S}_G$ a ser apresentado a seguir. No sistema $\mathcal{S}_G$, em vez de usar $\vdash \Psi, \nu, \neg\nu$ ou $\Psi, \nu, \neg\nu \vdash$ como esquema de axioma, será usado:

$$\mathsf{Ax}: \quad \Psi_1, \nu \vdash \Psi_2, \nu$$

em que $\nu \in \mathcal{V}$. (Note que $\vdash \Psi, \nu, \neg\nu$ é válido se e somente se $\Psi', \nu, \neg\nu \vdash$ é válido se e somente se $\Psi_1, \nu \vdash \Psi_2, \nu$ é válido, sendo $c(\Psi'_1) \cup c(\Psi_2) = c(\Psi)$ e $c(\Psi_1) \cup c(\Psi'_2) = c(\Psi')$.)

Ao contrário dos sistemas do tipo Hilbert, os do tipo Gentzen se concentram na descrição do comportamento dos conectivos via regras de inferência, de forma análoga ao que se faz nos sistemas $\mathcal{S}_C$, $\mathcal{S}_I$ e $\mathcal{S}_T$ vistos nas Seções 2.4.2 e 2.4.3. Enquanto lá havia regras dos tipos p e n para cada conectivo, aqui há regras dos tipos d e e, sendo que as regras do tipo d lidam com o lado direito (sequência após $\vdash$) de um sequente conclusão e as do tipo e lidam com o lado esquerdo (sequência antes de $\vdash$). Enquanto lá há apenas uma regra estrutural, a regra tr, aqui há duas, uma para cada lado do sequente. Como lá, as regras estruturais lidam com a estrutura dos sequentes, ignorando o conteúdo das fórmulas (ou seja, os conectivos) que o compõem. São elas:

$$\mathsf{tre}: \ \frac{\Psi_1, \alpha, \beta, \Psi'_1 \vdash \Psi_2}{\Psi_1, \beta, \alpha, \Psi'_1 \vdash \Psi_2}$$

$$\mathsf{trd}: \ \frac{\Psi_1 \vdash \Psi_2, \alpha, \beta, \Psi'_2}{\Psi_1 \vdash \Psi_2, \beta, \alpha, \Psi'_2}$$

Aplicações sucessivas destas regras podem ser usadas para mover fórmulas para posições arbitrárias em sequentes, fazendo com que os lados esquerdo e direito tenham a funcionalidade de multiconjuntos. Assim, é comum, no lugar de assumir a existência destas regras, supor que os lados direito e esquerdo são multiconjuntos, como na Seção 2.4.3, o que será feito daqui para frente.

Como dito acima, haverá duas regras lógicas para cada conectivo, uma para introdução do conectivo à esquerda e outra para introdução à direita, as quais podem ser obtidas a partir das regras de $\mathcal{S}_I$ ou de $\mathcal{S}_T$. No caso de $\mathcal{S}_I$, as regras de sufixo p levam a regras de sufixo e e as de sufixo n levam a regras de sufixo d (ou seja, fórmulas negativas "trocam de lado"). E no caso de $\mathcal{S}_T$, ocorre o inverso. Por exemplo, para *falsum* a regra de $\mathcal{S}_I$ é:

$$\mathsf{falp}: \ \frac{}{\Psi, \bot}$$

o que leva à regra de introdução de $\bot$ à esquerda:

$$\mathsf{fale}: \ \frac{}{\Psi_1, \bot \vdash \Psi_2}$$

Note que o que seria Ψ na regra de $\mathcal{S}_I$ agora pode estar espalhado em ambos os lados do sequente em $\mathcal{S}_G$. Para *verum* a regra de $\mathcal{S}_I$ é:

$$\mathsf{vern}: \ \frac{}{\Psi, \neg\top}$$

o que leva à regra de introdução de $\top$ à direita:

$$\mathsf{verp}: \ \frac{}{\Psi_1 \vdash \top, \Psi_2}$$

Verum

$$\mathsf{vere}: \frac{\Psi_1 \vdash \Psi_2}{\Psi_1, \top \vdash \Psi_2} \qquad \mathsf{verd}: \frac{}{\Psi_1 \vdash \top, \Psi_2}$$

Falsum

$$\mathsf{fale}: \frac{}{\Psi_1, \bot \vdash \Psi_2} \qquad \mathsf{fald}: \frac{\Psi_1 \vdash \Psi_2}{\Psi_1 \vdash \bot, \Psi_2}$$

Negação

$$\mathsf{nege}: \frac{\Psi_1 \vdash \alpha, \Psi_2}{\Psi_1, \neg\alpha \vdash \Psi_2} \qquad \mathsf{negd}: \frac{\Psi_1, \alpha \vdash \Psi_2}{\Psi_1 \vdash \neg\alpha, \Psi_2}$$

Conjunção

$$\mathsf{conje}: \frac{\Psi_1, \alpha, \beta \vdash \Psi_2}{\Psi_1, \alpha \wedge \beta \vdash \Psi_2} \qquad \mathsf{conjd}: \frac{\Psi_1 \vdash \alpha, \Psi_2 \quad \Psi_1 \vdash \beta, \Psi_2}{\Psi_1 \vdash \alpha \wedge \beta, \Psi_2}$$

Disjunção

$$\mathsf{disje}: \frac{\Psi_1, \alpha \vdash \Psi_2 \quad \Psi_1, \beta \vdash \Psi_2}{\Psi_1, \alpha \vee \beta \vdash \Psi_2} \qquad \mathsf{disjd}: \frac{\Psi_1 \vdash \alpha, \beta, \Psi_2}{\Psi_1 \vdash \alpha \vee \beta, \Psi_2}$$

Condicional

$$\mathsf{conde}: \frac{\Psi_1 \vdash \alpha, \Psi_2 \quad \Psi_1, \beta \vdash \Psi_2}{\Psi_1, \alpha \to \beta \vdash \Psi_2} \qquad \mathsf{condd}: \frac{\Psi_1, \alpha \vdash \beta, \Psi_2}{\Psi_1 \vdash \alpha \to \beta, \Psi_2}$$

Bicondicional

$$\mathsf{bice}: \frac{\Psi_1, \alpha, \beta \vdash \Psi_2 \quad \Psi_1 \vdash \alpha, \beta, \Psi_2}{\Psi_1, \alpha \leftrightarrow \beta \vdash \Psi_2} \qquad \mathsf{bicd}: \frac{\Psi_1, \alpha \vdash \beta, \Psi_2 \quad \Psi_1, \beta \vdash \alpha, \Psi_2}{\Psi_1 \vdash \alpha \leftrightarrow \beta, \Psi_2}$$

Figura 2.43: Regras lógicas do sistema formal $\mathcal{S}_G$.

De maneira similar, a partir das regras de $\mathcal{S}_I$ para a conjunção:

$$\mathsf{conjp}: \frac{\Psi, \alpha, \beta}{\Psi, \alpha \wedge \beta}$$

$$\mathsf{conjn}: \frac{\Psi, \neg\alpha \quad \Psi, \neg\beta}{\Psi, \neg(\alpha \wedge \beta)}$$

chega-se às regras da conjunção para $\mathcal{S}_G$

$$\mathsf{conje}: \frac{\Psi_1, \alpha, \beta \vdash \Psi_2}{\Psi_1, \alpha \wedge \beta \vdash \Psi_2}$$

$$\mathsf{conjd}: \frac{\Psi_1 \vdash \alpha, \Psi_2 \quad \Psi_1 \vdash \beta, \Psi_2}{\Psi_1 \vdash \alpha \wedge \beta, \Psi_2}$$

E assim por diante, exceto para a negação, chega-se às regras da figura 2.43. Para a negação, como se vê na Figura 2.43, a negação de uma fórmula é introduzida à esquerda se ela ocorre à direita, e vice-versa. Já as regras vere e fald não são necessárias, mas estão presentes apenas para facilitar a demonstração de completude.

O sistema formal é então $\mathcal{S}_G = (\mathcal{S}, R_G, A)$, em que R_G é o conjunto das regras apresentadas e A contém apenas os sequentes que satisfazem o esquema Ax.

Cada regra, se aplicada de trás para frente (da conclusão para as premissas), elimina um conectivo. Assim, para determinar se $H \models \alpha$, pode-se construir uma dedução de trás para frente começando com o sequente $\Psi \vdash \alpha$, em que Ψ é uma sequência com as fórmulas de H, produzir premissas menores e a partir destas (se não forem axiomas), repetir o processo continuadamente até produzir apenas axiomas; caso seja produzido um sequente sem conectivos que não seja axioma, é porque não há dedução do sequente $H \vdash \alpha$. O algoritmo da Figura 2.44 constrói uma dedução de trás para frente e a retorna, caso exista uma; se não existir, ele retorna uma dedução "vazia". A variável D, inicializada com vazia, vai sendo atualizada colocando-se o próximo sequente da dedução. O pseudocomando D := D+($\Psi_1 \vdash \Psi_2$, regra) adiciona o sequente $\Psi_1 \vdash \Psi_2$ e a anotação da regra utilizada (regra) à esquerda em D. O problema de anotar, além da regra, identificações das premissas, é proposto como exercício (Exercício 5).

É importante salientar que o algoritmo só apresenta o não determinismo provocado pelas escolhas do pseudocomando *escolhanl*; embora tais escolhas possam influir no desempenho, elas não levam a computações de fracasso. Ou seja, as escolhas não são importantes do ponto de vista da possibilidade de encontrar ou não uma dedução. Observe que são duas escolhas efetuadas em uma chamada de *escolhanl*: a do lado do sequente (esquerdo ou direto) e a da fórmula. Uma possível heurística na direção de tentar evitar explosão combinatória é procurar escolher sempre uma fórmula que, para ser concluída, seja usada uma regra com o menor número possível de premissas. Assim, as regras seriam divididas nas três classes (em ordem de preferência de escolha):

1. fale e verd;
2. nege, negd, conje, disjd, condd;
3. conjd, disje, conde, bice, bicd.

As regras vere e fald, na verdade, são supérfluas. Note que, assim como para os sistemas $\mathcal{S}_I$ e $\mathcal{S}_T$, um comportamento exponencial do algoritmo é provocado por fórmulas que propiciem o uso de regras com duas premissas, no presente caso, as regras do grupo 3 acima.

Nos exemplos a seguir, serão usadas as regras do sistema dedutivo sem menção explícita ao algoritmo apresentado, mas será usada a mesma estratégia usada pelo algoritmo: a dedução será construída de trás para frente.

Compare a dedução do seguinte exemplo com a do Exemplo 75.

Exemplo 79 Para mostrar que $\emptyset \vdash p \to p$, deve-se encontrar uma dedução do sequente $\vdash p \to p$. Existe uma única regra que permite concluir tal sequente, a regra condd com $\Psi_1 = \Psi_2 =$ vazio, o que leva à dedução:

1. $p \vdash p$ (Ax)
2. $\vdash p \to p$ (condd 1)

□

```
proc Gentzen(sequente Ψ1 ⊢ Ψ2): retorna dedução
    conjunto de sequentes CS;
    nome de regra regra;
    dedução D;
    fórmula α, β, γ;
    {e, d} lado;
    CS := {Ψ1 ⊢ Ψ2}; D := vazia;
    repita
        Ψ1 ⊢ Ψ2 := escolha(CS); { escolhe próximo sequente } CS := CS − {Ψ1 ⊢ Ψ2};
        se Ψ1 ∩ Ψ2 ≠ ∅ então
            regra := Ax
        senãose Ψ1 ⊢ Ψ2 contém fórmula que não é literal então
            (γ, lado) := escolhanl(Ψ1 ⊢ Ψ2); { escolhe não literal };
            se lado = e então Ψ1 := Ψ1 − {γ} senão Ψ2 := Ψ2 − {γ} fimse;
            caso (lado, γ) seja
                (e, ⊤):      CS := CS ∪ {Ψ1 ⊢ Ψ2};                          regra := vere
                (d, ⊤):      { continue }                                   regra := verd
                (e, ⊥):      { continue }                                   regra := fale
                (d, ⊥):      CS := CS ∪ {Ψ1 ⊢ Ψ2};                          regra := fald
                (e, ¬α):     CS := CS ∪ {Ψ1 ⊢ α, Ψ2};                       regra := nege
                (d, ¬α):     CS := CS ∪ {Ψ1, α ⊢ Ψ2};                       regra := negd
                (e, α ∧ β):  CS := CS{Ψ1, α, β ⊢ Ψ2};                       regra := conje
                (d, α ∧ β):  CS := CS ∪ {Ψ1 ⊢ α, Ψ2, Ψ1 ⊢ β, Ψ2};           regra := conjd
                (e, α ∨ β):  CS := CS ∪ {Ψ1, α ⊢ Ψ2, Ψ1, β ⊢ Ψ2};           regra := disje
                (d, α ∨ β):  CS := CS ∪ {Ψ1 ⊢ α, β, Ψ2};                    regra := disjd
                (e, α → β):  CS := CS ∪ {Ψ1 ⊢ α, Ψ2, Ψ1, β ⊢ Ψ2};           regra := conde
                (d, α → β):  CS := CS ∪ {Ψ1, α ⊢ β, Ψ2};                    regra := condd
                (e, α ↔ β):  CS := LS ∪ {Ψ1, α, β ⊢ Ψ2, Ψ1 ⊢ α, β, Ψ2};     regra := bice
                (d, α ↔ β):  CS := CS ∪ {Ψ1, α ⊢ β, Ψ2, Ψ1, β ⊢ α, Ψ2};     regra := bicd
            fimcaso
        senão { Ψ1 contém só literais e/ou ⊥ e Ψ2 contém só literais e/ou ⊤ e Ψ1 ⊢ Ψ2 não é axioma}
            retorne vazia
        fimse;
        D := D+(Ψ1 ⊢ Ψ2, regra)
    até LS = ∅;
    retorne D
fim Gentzen
```

Figura 2.44: Um algoritmo baseado em $\mathcal{S}_G$.

Agora compare a dedução a seguir com as do Exemplo 76.

Exemplo 80 Para mostrar que $\{p \to (q \to r), q\} \models p \to r$, deve ser encontrada uma dedução de $p \to (q \to r), q \vdash p \to r$. Existem duas regras que se aplicadas levam a este sequente: as regras condd e conde. Para aplicar a primeira (com $\Psi_1 = p \to (q \to r), q$ e $\Psi_2 =$ vazio) a premissa é $p \to (q \to r), q, p \vdash r$. Para deduzir esta, a única regra aplicável é conde com $\Psi_1 = q, p$ e $\Psi_2 = r$ com as premissas $q \to r, q, p \vdash r$ e $q, p \vdash p, r$. O último sequente é uma instância de Ax e o outro pode ser deduzido por aplicação de conde a partir duas instâncias de Ax, como mostra a dedução:

1. $q, p \vdash q, r$ (Ax)
2. $q, p, r \vdash r$ (Ax)
3. $q \to r, q, p \vdash r$ (conde 1,2)
4. $q, p \vdash p, r$ (Ax)
5. $p \to (q \to r), q, p \vdash r$ (conde 3,4)
6. $p \to (q \to r), q \vdash p \to r$ (condd 5)

□

Embora a dedução encontrada no exemplo anterior não seja muito clara, sua obtenção pelo uso do sistema formal $\mathcal{S}_G$ é trivial, muito mais fácil do que utilizando um sistema do tipo Hilbert como o da Seção 2.8.2.

Uma característica importante das regras de inferência do sistema dedutivo $\mathcal{S}_G$, é que, como as regras de $\mathcal{S}_I$ e $\mathcal{S}_T$, as regras são *reversíveis*: além do sequente conclusão ser válido se os sequentes premissa forem válidos, se o sequente conclusão for válido, então os sequentes premissa também são válidos. Isto faz com que seja possível uma demonstração simples do teorema da completude via indução.

Lema 3 *Para cada regra de $\mathcal{S}_G$, suas premissas são válidas (todas) se, e somente se, sua conclusão é válida.*

Prova

Segue do Lema 2 e dos resultados mostrados nos Exercícios 2 e 6 da Seção 2.4.3. □

O teorema da correção mostra que todo sequente dedutível é válido.

Teorema 17 *(Teorema da correção) se $\emptyset \Rightarrow \Psi_1 \vdash \Psi_2$ então $\Psi_1 \vdash \Psi_2$ é válido.*

Prova

Por indução no comprimento da dedução.[34] Suponha que o resultado valha para deduções com menos de n sequentes. Seja $n \geq 0$ e suponha que exista uma dedução de comprimento n de $\Psi_1 \vdash \Psi_2$ a partir de $\emptyset$. Deve-se mostrar que $\Psi_1 \vdash \Psi_2$ é válido. Se $n = 0$, se $\emptyset \Rightarrow \Psi_1 \vdash \Psi_2$, então $\Psi_1 \vdash \Psi_2$ é uma instância de Ax ou a conclusão de uma das regras verd ou fale. Em qualquer caso, $\Psi_1 \vdash \Psi_2$ é válido. Por outro lado, se $n > 0$, o último passo da dedução terá sido dado aplicando-se uma das regras:

[34] Em conformidade com o dito do Capítulo 1, o comprimento de uma dedução é o número de sequentes da mesma menos um.

- vere. Então a premissa utilizada no último passo é da forma $\Psi_1' \vdash \Psi_2$, sendo $\Psi_1 = \Psi_1', \top$. Como a dedução de $\Psi_1' \vdash \Psi_2$ tem menos de n sequentes, pela hipótese de indução $\Psi_1' \vdash \Psi_2$ é válido. De onde se conclui, pelo Lema 3, que $\Psi_1', \top \vdash \Psi_2$ é válido.

- fald. Então a premissa utilizada no último passo é da forma $\Psi_1 \vdash \Psi_2'$, sendo $\Psi_2 = \bot, \Psi_2'$. Como a dedução de $\Psi_1 \vdash \Psi_2'$ tem menos de n sequentes, pela hipótese de indução $\Psi_1 \vdash \Psi_2'$ é válido. De onde se conclui, pelo Lema 3, que $\Psi_1 \vdash \bot, \Psi_2'$ é válido.

- nege. Então a premissa utilizada no último passo é da forma $\Psi_1' \vdash \alpha\Psi_2$, sendo $\Psi_1 = \Psi_1', \neg\alpha$. Como a dedução de $\Psi_1' \vdash \alpha\Psi_2$ tem menos de n sequentes, pela hipótese de indução $\Psi_1' \vdash \alpha, \Psi_2$ é válido. De onde se conclui que $\Psi_1', \neg\alpha \vdash \Psi_2$ é válido, pelo Lema 3.

- negd. Então a premissa utilizada no último passo é da forma $\Psi_1, \alpha \vdash \Psi_2'$, sendo $\Psi_2 = \alpha\Psi_2'$. Como a dedução de $\Psi_1, \alpha \vdash \Psi_2'$ tem menos de n sequentes, pela hipótese de indução $\Psi_1, \alpha \vdash \Psi_2'$ é válido. De onde se conclui que $\Psi_1 \vdash \neg\Psi_2'$ é válido, pelo Lema 3.

- conje. Então a premissa utilizada no último passo é da forma $\Psi_1', \alpha, \beta \vdash \Psi_2$, sendo $\Psi_1 = \Psi_1', \alpha \wedge \beta$. Como a dedução de $\Psi_1', \alpha, \beta \vdash \Psi_2$ tem menos de n sequentes, pela hipótese de indução $\Psi_1', \alpha, \beta \vdash \Psi_2$ é válido. De onde se conclui que $\Psi_1', \alpha \wedge \beta \vdash \Psi_2$ é válido, pelo Lema 3.

- conjd. Então as duas premissas utilizadas no último passo são das formas $\Psi_1 \vdash \alpha, \Psi_2'$ e $\Psi_1 \vdash \beta, \Psi_2'$, sendo $\Psi_2 = \alpha \wedge \beta, \Psi_2'$. Como as deduções de $\Psi_1 \vdash \alpha, \Psi_2'$ e de $\Psi_1 \vdash \beta, \Psi_2'$ têm menos de n sequentes, pela hipótese de indução tais sequentes são válidos. De onde se conclui que $\Psi_1 \vdash \alpha \wedge \beta, \Psi_2'$ é válido, pelo Lema 3.

E assim por diante, até esgotar as regras. □

O teorema da completude vem a seguir.

Teorema 18 *(Teorema da completude) se* $\Psi_1 \vdash \Psi_2$ *é válido então* $\emptyset \Rightarrow \Psi_1 \vdash \Psi_2$.

Prova

Por indução no número de ocorrência de conectivos em $\Psi_1 \vdash \Psi_2$. Suponha que o resultado valha para $\Psi_1 \vdash \Psi_2$ com menos de n ocorrências de conectivos. Seja $\Psi_1 \vdash \Psi_2$ um sequente com $n \geq 0$ conectivos e suponha que tal sequente seja válido. Deve-se mostrar que $\emptyset \Rightarrow \Psi_1 \vdash \Psi_2$. Se $n = 0$, então, como $\Psi_1 \vdash \Psi_2$ só contém variáveis proposicionais, o sequente é válido se, e somente se, $c(\Psi_1) \cap c(\Psi_2) \neq \emptyset$. Neste caso, $\Psi_1 \vdash \Psi_2$ é uma instância de Ax. Logo, $\emptyset \Rightarrow \Psi_1 \vdash \Psi_2$. Por outro lado, se $n > 0$, seja $\gamma \in c(\Psi_1) \cup c(\Psi_2)$ uma fórmula com conectivo. Se $\gamma \in c(\Psi_2)$, tem-se os casos em que γ é:

- $\top$. Então $\Psi_2 = \top, \Psi_2'$ e, pela regra verd, $\emptyset \Rightarrow \Psi_1 \vdash \top, \Psi_2'$.

- $\bot$. Então $\Psi_2 = \bot, \Psi_2'$. Como $\Psi_1 \vdash \Psi_2$ é válido, pelo lema 3 $\Psi_1 \vdash \Psi_2'$ é válido. Pela hipótese de indução, segue-se que $\emptyset \Rightarrow \Psi_1 \vdash \Psi_2'$. Usando-se a regra regra fald, uma dedução de $\Psi_1 \vdash \Psi_2'$ pode ser estendida para se deduzir $\Psi_1 \vdash \bot, \Psi_2'$.

- $\neg$. Então $\Psi_2 = \neg\alpha, \Psi_2'$. Como $\Psi_1 \vdash \Psi_2$ é válido, pelo lema 3 $\Psi_1, \alpha \vdash \Psi_2'$ é válido. Pela hipótese de indução, segue-se que $\emptyset \Rightarrow \Psi_1, \alpha \vdash \Psi_2'$. Usando-se a regra regra negd, uma dedução de $\Psi_1, \alpha \vdash \Psi_2'$ pode ser estendida para se deduzir $\Psi_1 \vdash \neg\alpha, \Psi_2'$.

- $\wedge$. Então $\Psi_2 = \alpha \wedge \beta, \Psi_2'$. Como $\Psi_1 \vdash \Psi_2$ é válido, $\Psi_1 \vdash \alpha, \Psi_2'$ e $\Psi_1 \vdash \beta, \Psi_2'$ são válidos pelo Lema 3. Como eles têm menos de n conectivos, pela hipótese de indução, $\emptyset \Rightarrow \Psi_1 \vdash \alpha, \Psi_2'$ e $\emptyset \Rightarrow \Psi_1 \vdash \beta, \Psi_2'$. A partir de deduções deste dois sequentes, usando-se a regra conjd, obtém-se uma dedução de $\Psi_1 \vdash \alpha \wedge \beta, \Psi_2'$.

E assim por diante, até esgotar todas as opções para o lado direito. Depois repete-se raciocínio análogo para o lado esquerdo ($\gamma \in c(\Psi_1)$). □

Exercícios

1. Encontre deduções no sistema formal $\mathcal{S}_G$ para os seguintes sequentes:

a) $\vdash p \vee \neg p$
b) $\vdash \neg(p \wedge \neg p)$
c) $\vdash \neg\neg p \rightarrow p$
d) $p \rightarrow q, p \vdash q$
e) $p \rightarrow q, p, \neg q \vdash r$
f) $p \wedge \neg p \vdash \bot$
g) $\bot \vdash p \wedge \neg p$
h) $p \wedge q \vdash q \wedge p$
i) $\vdash (((p \rightarrow q) \rightarrow q) \rightarrow q) \rightarrow (p \rightarrow q)$
j) $\vdash (p \wedge \neg p) \leftrightarrow \bot$
k) $\vdash ((p \rightarrow \bot) \rightarrow \bot) \leftrightarrow p$

2. Conclua a demonstração do Teorema 17 (teorema da correção) construindo os casos faltantes.

3. Conclua a demonstração do Teorema 18 (teorema da completude) construindo os casos faltantes.

4. Descubra um(a forma de) sequente que provoque no algoritmo Gentzen um comportamento exponencial.

5. Mostre como alterar o algoritmo Gentzen para anotar que premissas foram usadas para gerar cada conclusão.

6. Seja BC uma base de conhecimento expressa em lógica proposicional. Suponha que se deseje responder perguntas do tipo

$$\mathsf{BC} \models \alpha?$$

Sugira uma heurística que possa ser usada pelo algoritmo Gentzen na escolha da próxima fórmula pelo comando *escolhanl*, de forma que o mesmo não fique "perdido" devido ao potencial número de fórmulas em BC que nada tenham a ver com a pergunta.

2.8.4 Tableaux semânticos

Os tableaux semânticos, também chamados de tableaux analíticos, costumam ser utilizados para implementação em computadores. Eles são baseados nos sistemas formais $\mathcal{S}_A$ e $\mathcal{S}_B$, definidos na Seção 2.4.1 e utilizados nos algoritmos satA-nd e satB-nd, embora costumem ser apresentados de uma forma diferente da que será feita aqui.

Embora os sistemas formais $\mathcal{S}_A$ e $\mathcal{S}_B$ sirvam para derivar uma fórmula se, e somente se, ela é satisfatível, eles são usados no sistema de tableaux semânticos, em processos semelhantes aos algoritmos satA-nd e satB-nd, para a construção de tableaux. Os tableaux podem ser vistos como sendo as árvores de computação introduzidas na Seção 2.4.2 para explicar o funcionamento de satA-nd e satB-nd. Nos tableaux, as folhas que contenham fórmulas complementares são ditas *fechadas* e as que contenham apenas literais em que nenhum seja complementar a outro (portanto, indicando uma interpretação que satisfaz a fórmula da raiz) são ditas *abertas*. [35] Um *tableau fechado* é uma árvore em que as folhas são todas fechadas. Com isto, pode-se dizer que uma fórmula α é uma tautologia se, e somente se, existe um tableau fechado com $F\alpha$ ou $\neg\alpha$ na raiz. Ou seja, na verdade usa-se o fato de que α é tautológica se, e somente se, $\neg\alpha$ é insatisfatível (havendo, assim, um tableau fechado com $\neg\alpha$ na raiz).

A Figura 2.45 apresenta uma algoritmo baseado nessa ideia, que nada mais é do que uma adapatação do algoritmo satA, Figura 2.13, devidamente adaptada para verificar insatisfatibilidade no lugar de satisfabilidade. A variável LA guarda as folhas abertas do que seria a árvore de computação. Cada elemento de LA é um par (Σ, A), sendo Σ e A variáveis do algoritmo. Por exemplo, LA pode conter, em determinado momento, os pares $(\{Fp, Fq\}, \{Fr\})$, $(\{Vp\}, \{Fq \vee r\})$ e $(\{Vq\}, \{Vp\})$, estando o primeiro localizado no nível 6 da árvore da Figura 2.21, e os outros dois no nível 4. Veja que o par $(\{Fp, \}, \{Vp\})$, do nível 4 da árvore da Figura 2.21, já não aparece, por sido processado chegando-se a uma folha fechada: $\{Fp, Vp\}$. No próximo passo, se for escolhido $\mathsf{A} = (\{Fp, Fq\}, \{Fr\})$, será descoberta uma folha aberta, $\{Fp, Fq, Fr\}$, e será retornado *falso*. Apenas quando se verificar que todas as folhas foram fechadas ($\mathsf{LA} = \emptyset$), o algoritmo retorna *verdadeiro*. As escolhas feitas em $\mathsf{A} :=$ *escolha*(LA) (próximo ramo a

[35] Na terminologia padrão, o que aqui se denomina folha aberta corresponde a *ramo aberto* em um tableau, e o que aqui se denomina folha fechada corresponde a *ramo fechado*.

estender) e $s\gamma := \mathit{escolha}(\mathsf{A})$ são não determinísticas, mas são irrelevantes do ponto de vista de completude, embora não o sejam sob o ponto de vista de desempenho. Uma pequena providência que pode economizar em termos de gastos de espaço e tempo é priorizar a escolha de $s\gamma$, em A, em que γ seja uma fórmula não disjuntiva, que, em consequência, propicie $\mathsf{A2} = \emptyset$ (no algoritmo satA-nd isso corresponderia a escolha γ que não levasse a não determinismo).

Um tableau é dito *terminado* se cada folha é fechada ou então é aberta mas só contém literais. Assim, uma fórmula é insatisfatível se e somente se um tableau terminado para a mesma é fechado.

Para determinar se $H \models \alpha$ basta verificar se $H \cup \{\neg\alpha\}$ é insatisfatível. A adaptação do algoritmo da Figura 2.45 para isto é trivial (Exercício 1), podendo ser considerado uma algoritmo baseado no sistema $\mathcal{S}_I$ apresentado na Seção 2.4.3.

De maneira similar ao que foi feito na Seção 2.4.2 para apresentação de árvores de computação, aqui também o comportamento do algoritmo tableau será apreciado apresentando-se a evolução dos pares (A, Σ) de LA por meio da exibição de $\mathsf{A} \cup \Sigma$ apenas. Com isto não se tem com exatidão a evolução dos pares, mas obtém-se uma árvore menos carregada e que expressa derivações no sistema $\mathcal{S}_I$, quando lida das folhas em direção à raiz.

Exemplo 81 A fórmula $(p \leftrightarrow (\neg q \vee r)) \rightarrow (\neg p \rightarrow q)$ é uma tautologia, como mostra o tableau com fórmulas marcadas da Figura 2.46. Abaixo das folhas fechadas (folhas com literais complementares) coloca-se um X. Veja que tal tableau corresponde ao comportamento do algoritmo tableau recebendo $(p \vee \neg(q \wedge r)) \wedge \neg((p \leftrightarrow r) \vee q)$ como argumento. □

A construção de um algoritmo para sistema de tableaux com fórmulas não marcadas é proposta como um exercício (Exercício 2). Um tableau com fórmulas não marcadas para mostrar que $(p \leftrightarrow (\neg q \vee r)) \rightarrow (\neg p \rightarrow q)$ é uma tautologia seria a própria árvore da Figura 2.46, apagando-se as marcas V e substituindo-se F por $\neg$, como mostrado na Figura 2.47.

Exercícios

1. Adapte o algoritmo da Figura 2.45 para determinar se $H \models \alpha$, sendo H um conjunto finito de fórmulas.

2. Escreva um algoritmo para sistema de tableaux com fórmulas não marcadas. Como seria sua adaptação para determinar se $H \models \alpha$?

3. Mostre por meio de tableaux marcados e também não marcados:

a) $\emptyset \models p \rightarrow (q \rightarrow p)$

b) $\emptyset \models p \wedge q \rightarrow p \vee q$

c) $\{p \rightarrow q, q \rightarrow r\} \models p \rightarrow r$

proc tableau(fórmula γ) **retorna** {*verdadeiro*, *falso*}:
  conjunto de pares (conjunto de variáveis marcadas, conjunto de fórmulas marcadas) LA;
  conjunto de fórmulas marcadas A;
  conjunto de variáveis marcadas Σ;
  elemento de $\{V, F\}$ s;
  LA := $\{(\emptyset, \{F\gamma\})\}$;
  repita
    { escolhe próximo ramo a estender }
    (Σ, A) := escolha(LA); LA := LA $-$ (Σ, A);
    { escolhe fórmula no ramo }
    $s\gamma$:= escolha(A); A := A $-$ $\{s\gamma\}$;
    caso sγ **seja**
      ******var*: **se** $\overline{s}\gamma \notin \Sigma$ **então**
        $\Sigma := \Sigma \cup \{s\gamma\}$;
        se A $= \emptyset$ **então** { ramo aberto } **retorne** *falso* **fimse**
        LA := LA $\cup$ $\{(\Sigma, \mathsf{A})\}$
      fimse
      { caso contrário, ramo fechado; continue }
      $V\top$,$F\bot$: **se** A $= \emptyset$ **então retorne** *verdadeiro* **fimse**;
      LA := LA $\cup$ $\{(\Sigma, \mathsf{A})\}$ { regras verp e faln }
      $V\bot$,$F\top$: { ramo fechado; continue }
      $V\neg\alpha$: LA := LA $\cup$ $\{(\Sigma, \mathsf{A} \cup \{F\alpha\})\}$ { regra negp }
      $F\neg\alpha$: LA := LA $\cup$ $\{(\Sigma, \mathsf{A} \cup \{V\alpha\})\}$ { regra negn }
      $V\alpha \wedge \beta$: LA := LA $\cup$ $\{(\Sigma, \mathsf{A} \cup \{V\alpha, V\beta\})\}$ { regra conjp }
      $F\alpha \wedge \beta$: LA := LA $\cup$ $\{(\Sigma, \mathsf{A} \cup \{F\alpha\}), (\Sigma, \mathsf{A} \cup \{F\beta\})\}$ { regras conjn }
      $V\alpha \vee \beta$: LA := LA $\cup$ $\{(\Sigma, \mathsf{A} \cup \{V\alpha\}), (\Sigma, \mathsf{A} \cup \{V\beta\})\}$ { regras disjp }
      $F\alpha \vee \beta$: LA := LA $\cup$ $\{(\Sigma, \mathsf{A} \cup \{F\alpha, F\beta\})\}$ { regra disjn }
      $V\alpha \rightarrow \beta$: LA := LA $\cup$ $\{(\Sigma, \mathsf{A} \cup \{F\alpha\}), (\Sigma, \mathsf{A} \cup \{V\beta\})\}$ { regras condp }
      $F\alpha \rightarrow \beta$: LA := LA $\cup$ $\{(\Sigma, \mathsf{A} \cup \{V\alpha, F\beta\})\}$ { regra condn }
      $V\alpha \leftrightarrow \beta$: LA := LA $\cup$ $\{(\Sigma, \mathsf{A} \cup \{V\alpha, V\beta\}), (\Sigma, \mathsf{A} \cup \{F\alpha, F\beta\})\}$ { regras bicp }
      $F\alpha \leftrightarrow \beta$: LA := LA $\cup$ $\{(\Sigma, \mathsf{A} \cup \{F\alpha, V\beta\}), (\Sigma, \mathsf{A} \cup \{V\alpha, F\beta\})\}$ { regras bicn }
    fimcaso;
  até LA $= \emptyset$;
  retorne *verdadeiro* { todos os ramos fechados }
fim tableau

Figura 2.45: Um algoritmo para sistema de tableaux com fórmulas marcadas.

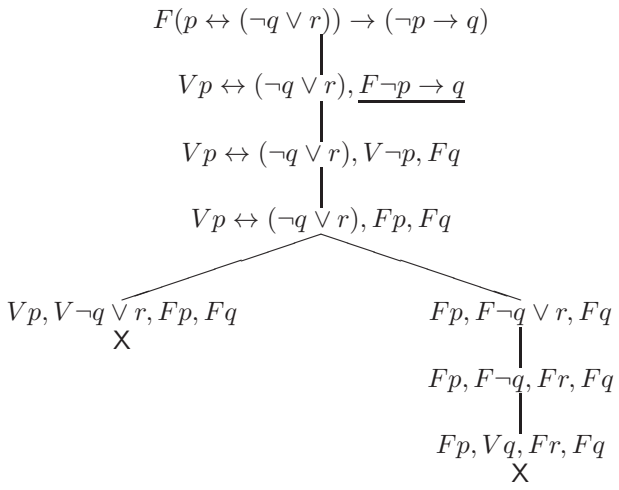


Figura 2.46: Tableau de fórmulas marcadas para $F(p \leftrightarrow (\neg q \vee r)) \rightarrow (\neg p \rightarrow q)$.

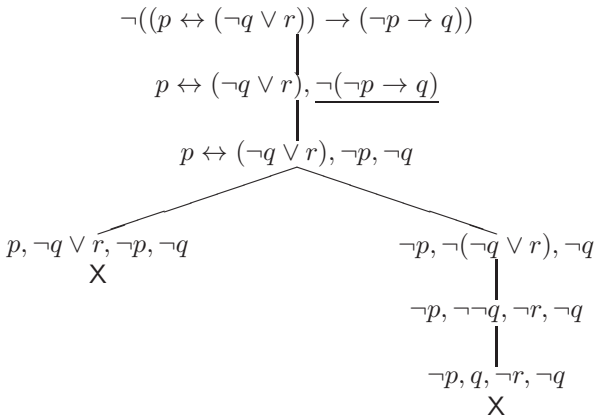


Figura 2.47: Tableau de fórmulas não marcadas para $\neg((p \leftrightarrow (\neg q \vee r)) \rightarrow (\neg p \rightarrow q))$.

d) $\{p \to (q \to r)\} \models p \wedge q \to r$

4. Resolva o Exercício 1 da Seção 2.8.1 usando tableaux não marcados.

5. Desenhe um tableau fechado com $\neg((p \vee r) \wedge (q \to r))$ na raiz e recupere, a partir dele, uma fórmula na FND logicamente equivalente a essa fórmula.

2.8.5 Resolução

Um outro sistema dedutivo para lógica proposicional é o de resolução. A maioria dos sistemas para processamento de conhecimento expresso em lógica proposicional usa o DPLL que, na verdade, pode ser visto como um refinamento de resolução. Por outro lado, a maioria dos sistemas para processamento de conhecimento expresso em lógica de predicados de primeira ordem é baseada em resolução. A linguagem de programação em Lógica Prolog também pode ser vista como uma das aplicações do sistema dedutivo de resolução no contexto de lógica de primeira ordem. Aqui será apresentada inicialmente a versão mais simples para lógica proposicional.

O sistema dedutivo de resolução tem como liguagem o conjunto das cláusulas. Assim como na Seção 2.6, é conveniente encarar cada cláusula como um conjunto de literais. O sistema dedutivo não tem axiomas lógicos e tem uma única regra de inferência: a regra de resolução. A *regra de resolução*, dadas duas cláusulas C_1 e C_2 como premissas, tais que há um literal l tal que $l \in C_1$ e $\overline{l} \in C_2$, produz como conclusão a cláusula $C_1 - \{l\} \cup C_2 - \{\overline{l}\}$. Tal cláusula é denominada *resolvente*.

Observe que $\{C_1, C_2\} \models C_1 - \{l\} \cup C_2 - \{\overline{l}\}$, visto que:

- se $v^i(l) = V$, para que $v^i(C_2)$ seja V, deve-se ter $v^i(C_2 - \{\overline{l}\}) = V$; e
- se $v^i(l) = F$, para que $v^i(C_1)$ seja V, deve-se ter $v^i(C_1 - \{l\}) = V$.

Assim, em qualquer caso, $v^i(C_1 - \{l\} \cup C_2 - \{\overline{l}\}) = V$ se $v^i(C_1) = v^i(C_2) = V$. Portanto, todo resolvente obtido direta ou indiretamente a partir de um conjunto de cláusulas B é consequência lógica de B.

Exemplo 82 A seguinte derivação mostra que $B \models q$, sendo $B = \{\neg p \vee q, p\}$:

1. $\neg p \vee q$ (B)
2. p (B)
3. q (res 1,2)

□

Um caso particular ocorre quando C_1 contém apenas l e C_2 apenas $\overline{l}$. Neste caso, o resolvente é a cláusula vazia, $\perp$ (evidentemente, dados l e $\overline{l}$, obtém-se uma contradição!). Uma dedução de $\perp$ a partir de um conjunto de cláusulas B é dita ser uma *refutação de* B. Evidentemente, se há uma refutação de B, então B é insatisfatível.

Segue um exemplo de refutação de um conjunto de cláusulas. Já que a única regra de inferência é a de resolução, daqui para frente será anotado à direita de cada resolvente apenas os números das premissas.

Exemplo 83 Seja $B = \{p \vee \neg q, \neg p \vee \neg q, q\}$. Segue uma dedução de $\bot$ a partir de B, ou seja, uma refutação de B:

1. $p \vee \neg q$ (B)
2. $\neg p \vee \neg q$ (B)
3. q (B)
4. $\neg q$ (1,2)
5. $\bot$ (3,4)

Logo, B é insatisfatível. Note como considerar uma cláusula um conjunto produz $\neg q$ como cláusula 4, em vez de $\neg q \vee \neg q$. □

Mesmo considerando apenas a linguagem das cláusulas, o sistema dedutivo de resolução não é completo com relação a consequência lógica; em particular, note que não há como deduzir $p \vee \neg p$: $\emptyset \nvdash p \vee \neg p$. No entanto, o sistema dedutivo é *refutacionalmente completo*: se um conjunto de cláusulas B é insatisfatível, então há uma dedução de $\bot$ a partir de B. O seguinte teorema, apresentado sem demonstração, enuncia a correção e completude de resolução com respeito a mostrar insatisfabilidade.

Teorema 19 *Um conjunto de cláusulas B é insatisfatível se, se somente se há uma dedução de $\bot$ a partir de B aplicando-se resolução.*

Para se provar que $H \models \alpha$ utilizando-se resolução, deve-se provar que $H \cup \{\neg\alpha\}$ é insatisfatível, o que é feito obtendo-se uma refutação de um conjunto de cláusulas equivalente a $H \cup \{\neg\alpha\}$. Veja o próximo exemplo.

Exemplo 84 Seja H o conjunto das fórmulas:

1. $inoc \vee culp$ (o mordomo é inocente ou culpado);
2. $inoc \rightarrow tmen$ (se o mordomo é inocente, a testemunha mentiu);
3. $cump \rightarrow \neg tmen$ (se o mordomo tem um cúmplice, a testemunha não mentiu); e
4. $cump$ (o mordomo tem um cúmplice).

Para mostrar que $H \models culp$, obtém-se primeiro o conjunto de cláusulas B a partir de H:

1. $inoc \vee culp$ (de 1)
2. $\neg inoc \vee tmen$ (de 2)
3. $\neg cump \vee \neg tmen$ (de 3)
4. $cump$ (de 4)

e também o conjunto de cláusulas nc a partir da negação da consulta:

```
proc res-satura(conjunto de cláusulas B) retorna {verdadeiro, falso}:
    conjunto de cláusulas N, R;
    N := B;
    enquanto N ≠ ∅  e  ⊥ ∉ N faça
        R := {resolventes de C1 e C2 | C1 ∈ N e C2 ∈ B};
        N := R − B; B := B ∪ N
    fimenquanto;
    retorne ⊥ ∈ N
fim res-satura
```

Figura 2.48: Resolução por saturação.

5. $\neg culp$ (da negação da consulta)

Segue uma dedução de $\bot$ a partir de $B \cup \mathsf{nc}$:

1. $inoc \vee culp$ (B)
2. $\neg inoc \vee tmen$ (B)
3. $\neg cump \vee \neg tmen$ (B)
4. $cump$ (B)
5. $\neg culp$ (nc)
6. $\neg tmen$ $(4,3)$
7. $\neg inoc$ $(6,2)$
8. $culp$ $(7,1)$
9. $\bot$ $(8,5)$

Logo, $B \cup \mathsf{nc}$ é insatisfatível e, portanto, $H \models culp$. □

É importante ressaltar que todas as fórmulas, tanto as da base de conhecimento original, quanto a referente à negação da consulta, devem ser transformadas para a FNC antes do uso de um procedimento de prova baseado em resolução. Para isto, pode-se usar os algoritmos vistos nas Seções 2.5 e 2.6. Note que tal transformação, por si só, pode ser exponencial no pior caso, mas em muitas situações isso não constitui empecilho, visto que (1) cada fórmula (da base de conhecimento e da negação da consulta) pode ser transformada isoladamente e (2) a transformação de cada fórmula inserida na base de conhecimento é feita uma única vez.

O algoritmo mais simples para tentar encontrar uma refutação a partir de um conjunto de cláusulas B (ou determinar que não existe uma) é o de *saturação*, apresentado na Figura 2.48. Esse algoritmo é extremamente ineficiente em geral, como mostra o próximo exemplo. No exemplo, o valor inicial de N será denotado N_0; o valor seguinte, no final do **enquanto**, será denotado N_1; o próximo valor, na segunda iteração do **enquanto**, será denotado N_2, e assim por diante.

Exemplo 85 Uma refutação de $B = \{p \vee q, p \vee \neg q, \neg p \vee q, \neg p \vee \neg q\}$ é encontrada pelo algoritmo de saturação da seguinte maneira. Inicialmente, N recebe todas as cláusulas de B:

N_0: 1. $p \vee q$ (B)
2. $p \vee \neg q$ (B)
3. $\neg p \vee q$ (B)
4. $\neg p \vee \neg q$ (B)

Ao final da primeira iteração do corpo do **enquanto**, tem-se as seguintes novas cláusulas em N, sendo que duas cláusulas, 7 e 8, são geradas de duas formas:

N_1: 5. p (1,2)
6. q (1,3)
7. $q \vee \neg q$ (1,4)(2,3)
8. $p \vee \neg p$ (1,4)(2,3)
9. $\neg q$ (2,4)
10. $\neg p$ (3,4)

(A ordem das cláusulas não é importante aqui.) Em seguida, o algoritmo gera 22 resolventes, só que 20 deles idênticos a cláusulas de $\mathsf{B} = N_0 \cup N_1$; e a cláusula vazia, $\perp$, é gerada duas vezes, resolvendo as cláusulas 5 e 10 e resolvendo 6 e 9. Como $N_2 = \{\perp\}$, o algoritmo retorna *verdadeiro*. □

Um primeiro tipo de refinamento que se pode fazer no algoritmo de saturação (e outros) é a eliminação de cláusulas de uma forma em que a completude não seja afetada. Exemplos são:

- Eliminação de tautologias. Pode-se mostrar que para qualquer conjunto de cláusulas B e cláusula tautológica C, $B \cup \{C\}$ é insatisfatível se e somente se B é insatisfatível.
- Eliminação de cláusulas puras. Tem-se: para qualquer conjunto de cláusulas B e cláusula C tais que $l \in C$ e $\overline{l}$ não ocorre em B, $B \cup \{C\}$ é insatisfatível se e somente se B é insatisfatível.
- Eliminação de cláusulas subjugadas. A cláusula C_2 *subjuga* C_1 se, e somente se, $C_2 \subset C_1$. Isso é corroborado por: para qualquer conjunto de cláusulas B e cláusulas C_1 e C_2 tais que $C_1 \in B$, $C_2 \subset C_1$ e $B \models C_2$, B é insatisfatível se e somente se $(B - \{C_1\}) \cup \{C_2\}$ é insatisfatível.

O exemplo anterior é reconsiderado aplicando-se eliminação de tautologias e subjugação: dentre todas as cláusulas, resolventes ou não, uma cláusula só é utilizada posteriormente se não for tautologia nem subjugada por alguma outra cláusula.

Exemplo 86 Uma refutação de $\{p \vee q, p \vee \neg q, \neg p \vee q, \neg p \vee \neg q\}$ encontrada pelo algoritmo de saturação com eliminação de tautologias e cláusulas subjugadas:

N_0: 1. $p \vee q$ (B)
2. $p \vee \neg q$ (B)

	3. $\neg p \vee q$	(B)	
	4. $\neg p \vee \neg q$	(B)	
N_1:	5. p	(1,2)	[subjuga 1 e 2]
	6. $\neg p$	(3,4)	[subjuga 3 e 4]
N_2:	7. $\perp$	(5,6)	

Após a geração de N_1, a variável B do algoritmo contém apenas as cláusulas 5 e 6, que subjugam todas as cláusulas de N_0. O algoritmo pode gerar algumas tautologias, dependendo da ordem de geração dos resolventes (como $q \vee \neg q$ (1,4)), que são imediatamente descartadas. □

Existem inúmeras estratégias de busca associadas ao sistema de resolução. A seguir serão brevemente descritas duas delas de especial importância: conjunto de suporte e linear.

Na estratégia *conjunto de suporte*, é exigido que uma das premissas seja escolhida do conjunto de suporte. Inicialmente, ele é um subconjunto S do conjunto inicial B, normalmente o conjunto de cláusulas relativas à negação da consulta. Durante o processo, cada novo resolvente produzido é adicionado ao conjunto de suporte (assim, todo resolvente pode ser usado depois como premissa). Essa estratégia é completa se, e somente se, $B - S$ é satisfatível. (A insatisfabilidade tem que ter relação com S.)

Ao ser usado o conjunto das cláusulas obtidas a partir da negação da consulta como conjunto de suporte, não há garantia de completude. Há certeza de preservação de completude apenas no caso em o conjunto original seja consistente. No entanto, isso pode ser positivo (uma situação em que perda de completude é desejável): pode ser uma forma de "isolar" uma eventual inconsistência da base de conhecimento, não permitindo uma resposta afirmativa fundamentada na inconsistência da base de conhecimento. Tem-se, neste caso, um certo "comportamento paraconsistente", visto que uma inconsistência nunca é usada como justificativa para uma resposta afirmativa, a menos que ela já esteja presente no próprio conjunto de cláusulas correspondente à negação da consulta. E, neste último caso, a fórmula correspondente à consulta seria tautológica. Segue um pequeno exemplo para esclarecer esse ponto.

Exemplo 87 Seja uma base de conhecimento $H = \{p \rightarrow q, p, \neg q\}$, inconsistente, que leva a um conjunto B constituído das cláusulas:

1. $\neg p \vee q$
2. p
3. $\neg q$

À consulta $H \models p$, corresponde a dedução imediata de $\perp$: resolução de $\neg p$ com a cláusula 2: p segue de H, não porque H é inconsistente, mas porque está afirmada em B (cláusula 2). Analogamente, $H \models \neg p$, pois há a refutação:

1. $\neg p \vee q$ (B)
2. p (B)
3. $\neg q$ (B)
4. p (nc)
5. q $(4,1)$
6. $\bot$ $(5,3)$

Veja que, novamente, a resposta afirmativa nada tem com a inconsistência: $\neg p$ segue das cláusulas 1 e 3. Ainda de forma similar, pode-se determinar que $H \models p \rightarrow q$, $H \models p \rightarrow \neg q$ etc., mas nunca apelando para a inconsistência de H. Agora, a consulta $H \models r$ recebe a resposta negativa, visto que $B \cup \{\neg r\}$ não é refutável utilizando-se $\neg r$ como conjunto de suporte. Por outro lado, tem-se que $H \models r \rightarrow r$, visto que utilizando-se $\{r, \neg r\}$ como conjunto de suporte obtém-se $\bot$ imediatamente. □

Já uma estatégia *linear* é aquela que só produz deduções lineares. Uma *dedução linear* de uma cláusula C_n, a partir do conjunto de cláusulas B, começando na cláusula C_0, é tal que:

- $C_0 \in B$;
- uma das premissas para obtenção do resolvente C_i é C_{i-1}.

Observe que a segunda premissa para obtenção de C_i é, ou alguma cláusula de B ou algum resolvente prévio. Caso se requeira que a segunda premissa seja sempre uma cláusula de B, tem-se o que se denomina uma *dedução linear de entrada*. Uma estratégia que produza apenas deduções lineares é completa, mas uma que explore apenas deduções lineares de entrada não é completa. Segue um exemplo.

Exemplo 88 Uma dedução linear de $\bot$, refutação de $\{p \vee q, p \vee \neg q, \neg p \vee q, \neg p \vee \neg q\}$, seria (escolhendo a cláusula 4 como inicial):

1. $p \vee q$ (B)
2. $p \vee \neg q$ (B)
3. $\neg p \vee q$ (B)
4. $\neg p \vee \neg q$ (B)
5. $\neg q$ $(4,2)$
6. p $(5,1)$
7. q $(6,3)$
8. $\bot$ $(7,5)$

Note que a partir de 5, a cláusula anterior é sempre uma premissa. □

É interessante notar que para o conjunto do exemplo anterior, não existe refutação linear de entrada, visto que todas as cláusulas de entrada têm dois literais; resolvendo-se uma delas com uma outra de um ou mais literais, obtém-se um resolvente de um ou mais literais; este, portanto, não pode ser a cláusula vazia.

Em uma estratégia linear, qualquer cláusula pode ser escolhida para iniciar uma dedução (no Exemplo 88 foi escolhida a cláusula 4). Não havendo uma refutação linear para a escolha efetuada, pode haver (ou não) para uma outra. Ou seja, a escolha de quais cláusulas devem ser consideradas para a referida escolha é importante, sob pena de se perder em completude. Evidentemente, pode-se considerar todas as cláusulas do conjunto original como podendo iniciar uma dedução. No entanto, uma opção mais interessante é combinar a estratégia linear com a de conjunto de suporte; neste caso, apenas o conjunto de suporte inicial supre cláusulas para iniciar uma dedução.

A principal virtude de uma estratégia linear, particularmente a linear de entrada, é que admite uma implementação eficiente, como bem ilustra a linguagem de programação em lógica Prolog. O mecanismo de inferência por trás de Prolog pode ser visto como usando uma estratégia linear de entrada. No caso, o sistema de inferência é completo porque lida apenas com cláusulas de Horn e a estratégia linear de entrada é completa se o conjunto de partida tem apenas cláusulas de Horn.

Uma *cláusula de Horn* é uma cláusula que contém no máximo um literal positivo. Um programa em Prolog, desprezando a sintaxe usual da linguagem, pode ser visto como constando de cláusulas, cada uma com exatamente um literal positivo (os outros literais, se houver, são negativos). Como cada cláusula tem um literal positivo, é impossível que o conjunto seja inconsistente (pois ele é satisfatível para a interpretação que satisfaz todo literal positivo). Para o programa ser executado, deve ser feita uma consulta que é uma conjunção de literais positivos. Ao se negar a consulta, obtém-se uma cláusula com apenas literais negativos. Tal cláusula é usada como conjunto (unitário) de suporte. Como resulatado, todos os resolventes obtidos têm apenas literais negativos (ou é $\bot$). Veja um pequeno exemplo a seguir.

Exemplo 89 Seja o seguinte conjunto de cláusulas, B, com exatamente um literal positivo cada uma:

1. $p \vee \neg q \vee \neg r$
2. $q \vee \neg s$
3. r
4. s

As cláusulas não unitárias podem ser lidas como condicionais: $q \wedge r \rightarrow p$ e $s \rightarrow q$. Negando-se a consulta $p \wedge s$ (para saber se $B \models p \wedge s$), obtém-se a cláusula com literais negativos:

5. $\neg p \vee \neg s$ (nc)

Segue uma dedução linear de entrada de $\bot$:

1. $p \vee \neg q \vee \neg r$ (B)
2. $q \vee \neg s$ (B)
3. r (B)

4. s (B)
5. $\neg p \vee \neg s$ (nc)
6. $\neg q \vee \neg r \vee \neg s$ (5,1)
7. $\neg r \vee \neg s$ (6,2)
8. $\neg s$ (7,4)
9. $\bot$ (8,3)

Exercícios

1. Demonstre o resultado que justifica eliminação de tautologias: para qualquer conjunto de cláusulas B e cláusula tautológica C, $B \cup \{C\}$ é insatisfatível se e somente se B é insatisfatível.

2. Demonstre o resultado que justifica eliminação de cláusulas puras: para qualquer conjunto de cláusulas B e cláusula C tais que $l \in C$ e $\overline{l}$ não ocorre em B, $B \cup \{C\}$ é insatisfatível se e somente se B é insatisfatível.

3. Demonstre o resultado que justifica eliminação de cláusulas subjugadas: para qualquer conjunto de cláusulas B e cláusulas C_1 e C_2 tais que $C_1 \in B$, $C_2 \subset C_1$ e $B \models C_2$, B é insatisfatível se e somente se $(B - \{C_1\}) \cup \{C_2\}$ é insatisfatível.

4. Modifique o algoritmo de resolução por saturação para que ele faça eliminação de tautologias, de cláusulas puras e de cláusulas subjugadas.

5. Mostre usando resolução:

a) $\emptyset \models p \rightarrow (q \rightarrow p)$
b) $\emptyset \models p \wedge q \rightarrow p \vee q$
c) $\{p \rightarrow q, q \rightarrow r\} \models p \rightarrow r$
d) $\{p \rightarrow (q \rightarrow r)\} \models p \wedge q \rightarrow r$

6. Resolva o Exercício 1 da Seção 2.8.1 usando resolução. Encontre deduções lineares.

7. Faça um algoritmo baseado em uma estratégia linear com conjunto de suporte. Ele deve receber como entrada dois conjuntos de cláusulas, B e S, sendo S o conjunto de suporte inicial. E deve procurar por uma refutação linear.

2.8.6 Sistema de tableaux conectados

A seguir será apresentado um sistema de tableaux para cláusulas denominado de *sistema de tableaux conectados*[36] que traz embutida uma estratégia linear de entrada. Informalmente, um tableau conectado fechado apresenta um subconjunto de cláusulas

[36] *Connection tableaux* em inglês.

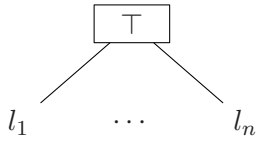
conectadas de tal forma que garante sua insatisfabilidade. Além disso, um tableau conectado fechado pode ser transformado facilmente no que se denominará aqui uma *árvore de prova*. E a partir de árvores prova pode-se extrair, também facilmente, deduções diretas no lugar da refutação original, como será visto adiante, o que pode facilitar a comunicação de um sistema dedutivo com um usuário relativamente leigo.

O sistema de tableaux conectados não contém axiomas lógicos e contém três regras de inferência, sendo que uma delas é utilizada apenas para começar uma dedução. O sistema lida com dois tipos de expressões: cláusulas (de entrada) e tableaux conectados (derivados). Embora os tableaux conectados possam ser expressos por meio de expressões de uma linguagem de um sistema formal, aqui elas serão apresentadas de maneira informal, como usual na literatura sobre raciocínio automatizado. Com isto, em particular, dispensa-se o uso de regras estruturais explícitas.

As regras de inferência serão apresentadas como operações sobre um tableau conectado e/ou uma cláusula de entrada (cláusula do conjunto que se quer mostrar ser insatisfatível). Um tableau conectado é uma árvore, cuja construção é propiciada pelas três regras de inferência definidas a seguir usando a terminologia associada a árvores. Daqui para frente será usado apenas o termo tableau para denotar "tableau conectado para cláusulas".

- *Regra de codificação*:

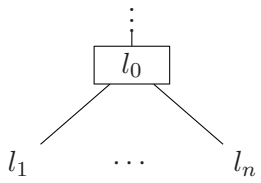
  Recebendo como premissa uma cláusula de entrada $l_1 \vee \cdots \vee l_n$, $n \geq 1$, a codificação produz um tableau de raiz com rótulo $\top$ e folhas com rótulos l_1, $\ldots$, l_n, todas filhas da raiz:



  O "literal" $\top$ é o literal raiz e os literais $l_1, \ldots, l_n$ são chamados *literais candidatos*.

- *Regra de expansão*:

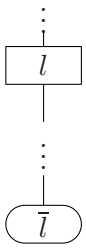
  Recebendo como premissa um tableau T com uma folha de rótulo literal candidato l_0 e uma cláusula de entrada $C = \overline{l_0} \vee l_1 \vee \cdots \vee l_n$, $n \geq 0$, a expansão de T via C produz um tableau em que o vértice de rótulo l_0 em T recebe como filhos folhas com rótulos $l_1, \ldots, l_n$:



O literal l_0, que era candidato, torna-se um *literal expandido* e os literais $l_1, \ldots, l_n$ são *literais candidatos*.

- *Regra de redução*:

  Recebendo como premissa um tableau com uma folha de rótulo literal candidato $\overline{l}$ que tenha como ancestral um vértice com rótulo l, uma redução de T torna o literal l *reduzido*:



Um tableau sem literais candidatos é dito ser um tableau *fechado*.

Note como um tableau construído com as regras acima é realmente um tableau para *fórmulas na FNC*:

- No início, tem-se um conjunto de cláusulas, ou seja, é como se os $\wedge$s tenham sido substituídos por ","s.
- Usando-se a regra de codificação, escolhe-se uma cláusula e estende-se o tableau colocando-se uma alternativa para cada literal. É realmente assim que se leva em conta a primeira fórmula disjuntiva em um tableau.
- A regra de expansão limita a escolha de uma cláusula a estender um ramo a uma que tenha literal complementar ao último literal do ramo, implicitamente provocando o fechamento do ramo obtido com o literal complementar. É esta característica que justifica o qualificativo *conectado*. Os outros literais estendem o ramo em questão como deve ser.
- A regra de redução corresponde ao fechamento de ramo com literais complementares (exceto aqueles implicitamente fechados na operação de expansão).

O seguinte teorema, apresentado sem demonstração, enuncia a correção e completude do sistema de tableaux conectados.

Teorema 20 *Um conjunto de cláusulas B é insatisfatível se, e somente se, existe um tableau fechado construído a partir de B.*

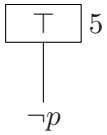
Com isto, um sistema de tableaux desse tipo pode ser usado para determinar se $H \models \alpha$ de forma similar a resolução: obtém-se um conjunto de cláusulas B correspondente a H, um conjunto de cláusulas nc correspondente a $\neg\alpha$ e o problema torna-se, então, determinar um tableau fechado a partir de $B \cup \mathsf{nc}$: existe tal tableau se e somente se $H \models \alpha$.

Segue um exemplo. Nele, no lado direito de cada vértice correspondente a literal expandido é anotado o número que identifica a cláusula de entrada usada na expansão; e no lado direito da raiz é anotada a cláusula codificada.

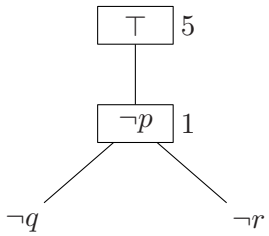
Exemplo 90 Para mostrar $\{p \vee \neg q \vee \neg r, q \vee \neg r \vee s, r, \neg s\} \models p$, primeiramente, como em resolução, obtém-se o conjunto:

1. $p \vee \neg q \vee \neg r$ (B)
2. $q \vee \neg r \vee s$ (B)
3. r (B)
4. $\neg s$ (B)
5. $\neg p$ (nc)

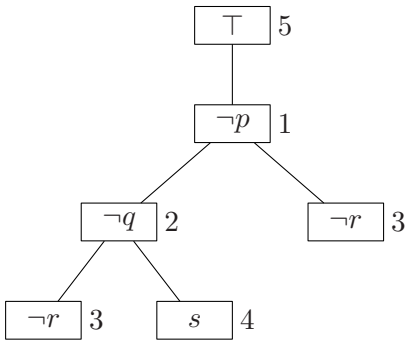
Uma boa política é, normalmente, escolher uma cláusula da negação da consulta para codificar. Aplicando-se a regra de codificação à cláusula 5, obtém-se:



Fazendo-se a expansão desta via a cláusula 1, obtém-se:



Fazendo a expansão desta via as cláusulas 2, 3 (duas vezes) e 4, obtém-se o tableau conectado fechado:



Observe como fica clara a insatisfabilidade de $B \cup \mathsf{nc}$, lendo-se a árvore das folhas para a raiz: de r (cláusula 3) e $\neg s$ (cláusula 4) conclui-se, usando-se a cláusula 2, q. Desta e de r (cláusula 3) conclui-se, usando-se a cláusula 1, p. Mas p contradiz $\neg p$ (cláusula 5). □

O sistema de tableaux conectados com apenas as regras de codificação e expansão é completo para cláusulas de Horn, constituindo um sistema bastante aderente ao mecanismo de inferência da linguagem Prolog. No entanto, para alguns conjuntos é necessário o uso da regra de redução. Veja o exemplo a seguir.

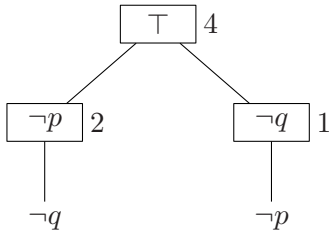
Exemplo 91 Seja $H = \{p \leftrightarrow q, p \vee q\}$. Em cláusulas (note que B contém as cláusulas relativas a H):

1. $\neg p \vee q \quad (B)$
2. $\neg q \vee p \quad (B)$
3. $p \vee q \quad (B)$

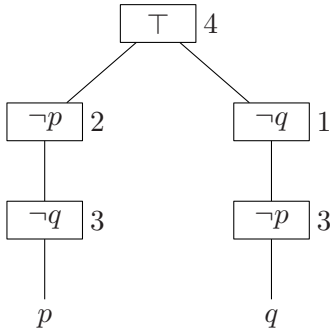
Para provar que $H \models p \wedge q$, obtém-se, primeiro, a cláusula correspondente à negação da consulta:

4. $\neg p \vee \neg q \quad (\mathsf{nc})$

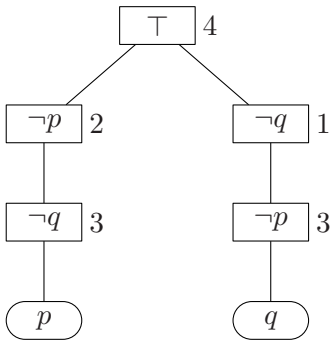
Aplicando-se a regra de codificação à cláusula 4, e depois as regras de expansão via 1 e 2, obtém-se:



Usando-se a cláusula 3 para expandir o tableau em ambos os literais candidatos:



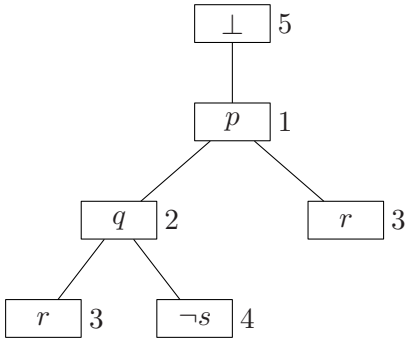
Agora ambos os literais candidatos podem ser reduzidos, obtendo-se um tableau fechado:



□

Um problema quanto ao uso de resolução ou de tableaux conectados para cláusulas é a dificuldade com relação ao *entendimento* de uma dedução (na verdade, uma refutação). A facilidade de entendimento pode depender de muita prática, o que torna o uso de ambos restrito a especialistas. No entanto, um tableau conectado em que todos os literais são negados leva a uma "leitura" do tableau como um conjunto de implicações conectadas. Tal tableau com todos os literais complementados (e em que $\top$ vira $\bot$) será denominado aqui de *árvore de prova*. Veja o exemplo a seguir.

Exemplo 92 Considere o tableau fechado obtido no Exemplo 90. Complementando-se todos os seu literais obtém-se a seguinte árvore de prova:



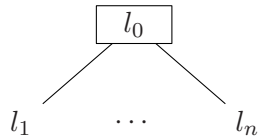
Tal árvore mostra que por meio de *implicações conectadas*, navegando-se *das folhas em direção à raiz, da esquerda para a direita*, chega-se à dedução de $\bot$, como mostrado a seguir ($B : n$ indica a cláusula n de B):

1. r $(B : 3)$
2. $\neg s$ $(B : 4)$
3. $r \wedge \neg q$ (conj **1**,**2**)

4. $r \wedge \neg s \rightarrow q$ $(B:2)$
5. q (mp **4**,**3**)
6. $q \wedge r$ (conj **5**,**1**)
7. $q \wedge r \rightarrow p$ $(B:1)$
8. p (mp **7**,**6**)
9. $p \rightarrow \bot$ $(B:5)$
10. $\bot$ (mp **9**,**8**)

Observe como são usados apenas a ocorrência de cláusulas no conjunto B e as regras conj e mp (vistas na seção 2.8.1). Logo, $B \models \bot$ e, portanto, B é insatisfatível. E levando-se em conta que a cláusula 5, $\neg p$, corresponde à negação da *consulta*, o prefixo **1** a **8** apresenta uma dedução de p a partir de B! □

Assim, como exemplificado, existe uma leitura bastante natural de um trecho de árvore da forma



que é: $l_1 \wedge \cdots \wedge l_n \rightarrow l_0$, pois esta é equivalente à cláusula usada na expansão para obtenção desse trecho: $l_0 \vee \overline{l_1} \vee \cdots \vee \overline{l_n}$. É a partir de tal leitura que é possível apresentar *explicações* bastante claras de respostas, em vez das deduções ilegíveis obtidas no sistema de resolução.

Tendo em vista a estrutura da demonstração do exemplo anterior, vê-se que a regra conj só é usada para obter o antecedente de uma condicional equivalente a uma cláusula de B. Assim, para encurtar e aumentar a legibilidade de uma dedução, pode-se combinar as duas regras e usar a seguinte regra, que será considerada aqui como uma "versão" de mp:

$$\frac{l_1 \ \ldots \ l_n \ \ l_1 \wedge \ldots \wedge l_n \rightarrow l}{l}$$

O exemplo a seguir reformula a dedução do Exemplo 92.

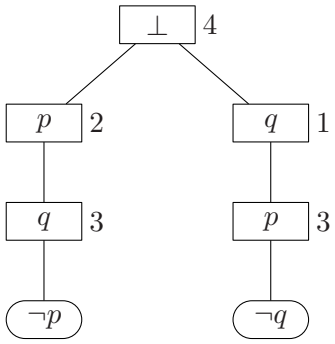
Exemplo 93 Reformulação da dedução do Exemplo 92 usando-se a nova versão da regra mp:

1. r $(B:3)$
2. $\neg s$ $(B:4)$
3. $r \wedge \neg s \rightarrow q$ $(B:2)$
4. q (mp **1**,**2**,**3**)
5. $q \wedge r \rightarrow p$ $(B:1)$
6. p (mp **4**,**1**,**5**)
7. $p \rightarrow \bot$ $(B:5)$
8. $\bot$ (mp **6**,**7**)

□

Uma pequena complicação aparece quando é aplicada a regra de redução. Uma maneira de adaptar o esquema dedutivo do Exemplo 93 para levar em conta reduções é encarar os literais reduzidos como antecedentes de condicionais (repesentando *suposições*), como exemplificado a seguir.

Exemplo 94 Seja a seguinte árvore de prova obtida a partir do tableau fechado do Exemplo 91:



Uma forma de levar em conta as reduções, é "carregar" o literal reduzido como uma suposição, enquanto se navega em direção à raiz, até encontrar seu complemento (novamente a árvore é utilizada como guia, navegando-se das folhas em direção à raiz, da esquerda para a direita):

1. $\neg p \rightarrow q$ $(B:3)$
2. $q \rightarrow p$ $(B:2)$
3. $\neg p \rightarrow p$ (sh **1**,**2**)
4. p (ra **3**)
5. $\neg q \rightarrow p$ $(B:3)$
6. $p \rightarrow q$ $(B:1)$
7. $\neg q \rightarrow q$ (sh **5**,**6**)
8. q (ra **7**)
9. $p \wedge q \rightarrow \bot$ $(B:4)$
10. $\bot$ (mp **4**,**8**,**9**)

Veja que além de *modus ponens* (em **10**), usou-se a regra sh (silogismo hipotético), vista na Seção 2.8.1, e a regra ra, que permite concluir α a partir da fórmula logicamente equivalente $\overline{\alpha} \rightarrow \alpha$. Considerando que a cláusula codificada é a 4, negação da consulta $p \wedge q$, vê-se que a dedução de **1** a **8**, acrescida de

9'. $p \wedge q$ (conj **4**,**8**)

seria uma dedução de $p \wedge q$ a partir de B. □

Os exemplos anteriores apresentam a ideia de que se uma cláusula advinda da negação da consulta for a codificada, o conjunto de implicações sugerido por uma árvore de prova fornece a base para a obtenção de uma dedução direta (não por refutação), senda esta feita por meio de regras de inferência mais acessíveis a usuários leigos. Mas, mesmo que a cláusula codificada não seja advinda da negação da consulta, se existir tal cláusula no tableau existe como obter um tableau (e consequentemente árvore de prova) em que esta figure como a codificada (veja Exercício 5).

Quando o conjunto B é diferente do conjunto original H, para a dedução ficar completa basta adicionar os passos usados na obtenção da FNC (cada passo justificado por uma equivalência lógica).

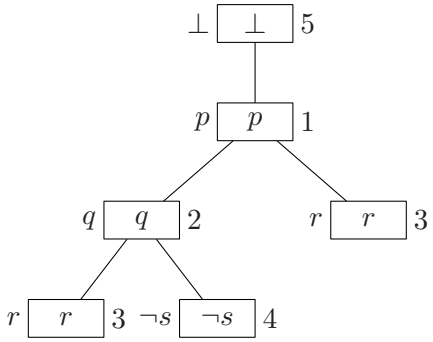
Uma outra coisa que ressalta nos exemplos acima é que a cada literal expandido corresponde um pequeno trecho da dedução. A fórmula concluída no trecho é justamente um *lema* que se pode associar àquele literal expandido. No caso em que a regra de redução não é usada na construção da árvore, o lema é unitário e consta exatamente do literal expandido. No caso mais geral, o lema associado a um literal expandido l é a cláusula constituída dos literais;

- l; e
- todo literal $\overline{r}$ tal que:
  - l é ancestral de literal reduzido r, e
  - l é descendente do literal $\overline{r}$ usado na redução.

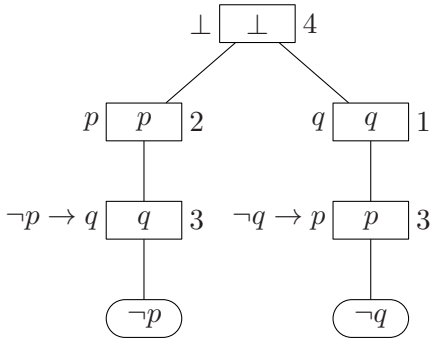
No Exemplo 94 um lema $\overline{r_1} \vee \ldots \vee \overline{r_k} \vee l$ foi apresentado como uma condicional $r_1 \wedge \ldots \wedge r_k \rightarrow l$, ressaltando que l é verdadeira, *se* $r_1, \ldots, r_k$ forem.

O exemplo a seguir mostra os lemas para as árvores de prova dos dois últimos exemplos no formato de condicionais.

Exemplo 95 Os lemas para a árvore de prova do Exemplo 92, todos unitários, estão anotados no lado esquerdo de cada literal expandido:



Agora os lemas para a árvore de prova do Exemplo 94 apresentados como condicionais em que o antecedente é formado por literais reduzidos e o consequente por pelo literal expandido:



Note como há lemas não unitários por causa dos literais reduzidos. □

O estilo de dedução apresentado no Exemplo 93 será generalizado de tal forma que o lema associado a cada literal expandido seja derivável por meio da aplicação de uma regra de inferência, que será a única a ser utilizada. Com isto, o número de aplicações de regras será o número de vértices com literais expandidos, excetuados aqueles que sejam folhas ou cujos filhos sejam todos rotulados com literais reduzidos. Seja v um vértice, cujo rótulo é um literal expandido, com filhos cujos rótulos sejam literais expandidos $u_1,\ldots u_n$ $(n \geq 0)$ e filhos cujos rótulos sejam reduzidos $r_1,\ldots r_k$ $(k \geq 0)$. O lema associado a um vértice v cujo rótulo é um literal expandido será denotado por $lema(v)$ e o literal rótulo de v por $lit(v)$. Se $n > 0$, a partir das premissas

$lema(u_1)$ (previamente deduzido),
$\vdots$
$lema(u_n)$ (previamente deduzido),
$lit(r_1) \wedge \ldots \wedge lit(r_k) \wedge lit(u_1) \wedge \ldots \wedge lit(u_n) \rightarrow lit(v)$ (cláusula de v)

conclui-se:

$lema(v)$.

sendo que $lema(v)$ é uma condicional cujo consequente é $lit(v)$ e cujo antecendente contém como literais: $lit(r_1), \ldots, lit(r_k)$ e os literais nos antecedentes de $lema(u_1),\ldots,$ $lema(u_n)$, com exceção de $\overline{lit(v)}$. Se $n = 0$, $lema(v) = lit(r_1) \wedge \ldots \wedge lit(r_k) \rightarrow lit(v)$ (ou $lema(v) = lit(v)$ se $k = 0$) é uma cláusula de entrada. Antes de apresentar a regra de inferência que corrobora o primeiro esquema acima, denominada mpg (*modus ponens generalizada*), será visto um exemplo.

Exemplo 96 Uma dedução correspondente à segunda árvore do Exemplo 95, usando mpg, seria:

1. $\neg p \rightarrow q$ $(B:3)$
2. $q \rightarrow p$ $(B:2)$
3. p (mpg **1**,**2**)
4. $\neg q \rightarrow p$ $(B:3)$

5. $p \to q$ $(B:1)$
6. q (mpg **4**,**5**)
7. $p \wedge q \to \bot$ $(B:4)$
8. $\bot$ (mpg **3**,**6**,**7**)

Veja que agora há apenas 3 aplicações de regras. □

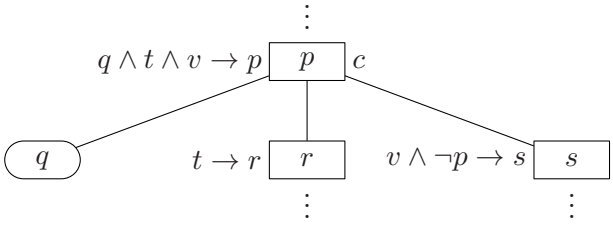
A seguir, dado um conjunto de literais $C = \{l_1, \ldots, l_n\}$, $n \geq 0$, e um literal l, será usada a notação $C \rightsquigarrow l$ para significar $l_1 \wedge \ldots \wedge l_n \to l$ (a ordem dos literais sendo irrelevante), se $n > 0$, ou l, se $n = 0$.

A regra de inferência mpg pode ser assim especificada, sendo R um conjunto de literais (que pode ser vazio), l_1, ..., l_n e l literais, e cada S_i, para $1 \leq i \leq n$, conjuntos de literais (cada um podendo ser vazio) e $n \geq 1$:

$$\textsf{mpg}: \quad \frac{\begin{array}{l} R \cup \{l_1, \ldots, l_n\} \rightsquigarrow l \\ S_1 \rightsquigarrow l_1 \\ \vdots \\ S_n \rightsquigarrow l_n \end{array}}{(\bigcup_{i=1}^{n} S_i \cup R) - \{l\} \rightsquigarrow l}$$

A seguir, um exemplo.

Exemplo 97 Seja o seguinte trecho de uma árvore de prova, em que o lema associado a cada literal expandido está mostrado à sua esquerda:



Note como a regra mpg permite concluir

$q \wedge t \wedge v \to p$

a partir das premissas:

$q \wedge r \wedge s \to p$ (cláusula c),
$t \to r$ (previamente deduzida), e
$v \wedge \neg p \to s$ (previamente deduzida). □

Segue um exemplo completo.

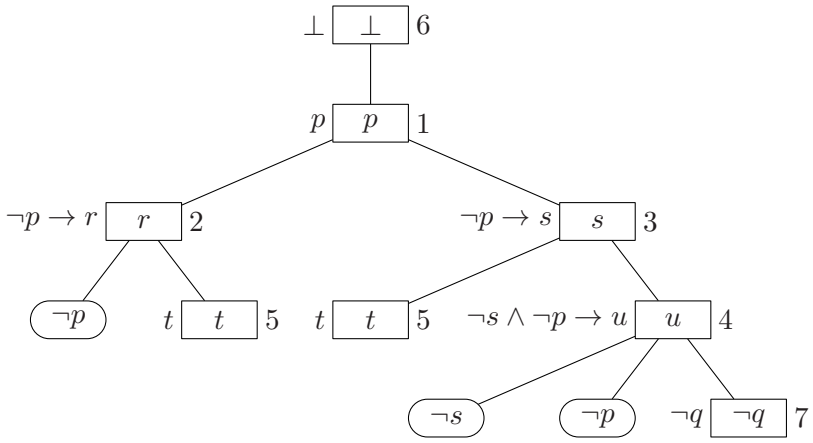
Exemplo 98 Seja o seguinte conjunto de cláusulas relativa a uma base de conhecimentos:

1. $\neg r \vee \neg s \vee p \quad (B)$
2. $p \vee \neg t \vee r \quad (B)$
3. $\neg t \vee \neg u \vee s \quad (B)$
4. $s \vee p \vee q \vee u \quad (B)$
5. $t \quad (B)$

e seja a consulta $p \vee q$. Negando-se e obtendo-se cláusulas:

6. $\neg p \quad (\mathsf{nc1})$
7. $\neg q \quad (\mathsf{nc2})$

A seguinte árvore de prova mostra a insatisfabilidade do conjunto 1 a 7:



Caminhando-se das folhas para a raiz, da esquerda para a direita, obtém-se a seguinte dedução de $\bot$:

1. $t \quad (B:5)$
2. $\neg p \wedge t \to r \quad (B:2)$
3. $\neg p \to r \quad (\mathsf{mpg}\ \mathbf{1},\mathbf{2})$
4. $\neg q \quad (B:7)$
5. $\neg s \wedge \neg p \wedge \neg q \to u \quad (B:4)$
6. $\neg s \wedge \neg p \to u \quad (\mathsf{mpg}\ \mathbf{4},\mathbf{5})$
7. $t \wedge u \to s \quad (B:3)$
8. $\neg p \to s \quad (\mathsf{mpg}\ \mathbf{1},\mathbf{6},\mathbf{7})$
9. $r \wedge s \to p \quad (B:1)$
10. $p \quad (\mathsf{mpg}\ \mathbf{3},\mathbf{8},\mathbf{9})$
11. $p \to \bot \quad (B:6)$
12. $\bot \quad (\mathsf{mpg}\ \mathbf{10},\mathbf{11})$

Assim, dado B e supondo-se $\neg q$, obtém-se uma dedução de p (subdedução de **1** a **10**). Segue-se que $p \vee q$ é consequência lógica de B. □

A afirmação ao final do exemplo anterior procede, pois: a subdedução de **1** a **10** demonstra que $B \cup \{\neg q\} \models p$, pois a regra mpg é válida (Exercício 3); disto, segue-se que $B \models p \vee q$.

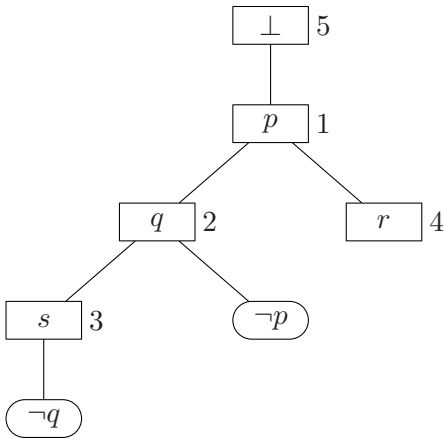
Exercícios

1. Resolva o Exercício 1 da Seção 2.8.1, usando tableaux conectados, codificando sempre a cláusula derivada da negação da consulta.

2. Determine os lemas para cada literal expandido de cada uma das árvores de prova correspondentes aos quatro últimos tableaux conectados do exercício anterior e apresente deduções de $\perp$ usando a regra mpg. Mostre como seria uma dedução direta da fórmula da consulta obtida a partir da dedução respectiva de $\perp$. No caso do item e, como há duas cláusulas referentes à negação da consulta, sendo apenas uma delas codificada, faça como no Exemplo 98.

3. Demonstre que a regra de inferência mpg é válida.

4. Apresente os lemas associados a literais expandidos da árvore a seguir, e uma dedução de p usando a regra mpg.



Cláusulas:

1. $\neg q \vee \neg r \vee p$
2. $\neg s \vee p \vee q$
3. $q \vee s$
4. r
5. $\neg p$ (nc)

5. Dado um tableau conectado fechado, como produzir um tableau conectado fechado com um mínimo de modificações em que a cláusula codificada seja uma outra (cuja instância figure no tableau)?