منطق گزاره‌ای: فرمول‌ها، مدل‌ها، جداول معنایی

منطق گزاره‌ای یک نظام منطقی ساده است که زیربنای همهٔ نظام‌های دیگر به‌شمار می‌رود. گزاره‌ها ادعاهایی همچون «$1 + 1 = 2$» و «$2 + 2 = 1$» هستند که نمی‌توان آن‌ها را به اجزای کوچکتری شکست و می‌توان به آن‌ها ارزش صدقِ «درست» یا «نادرست» نسبت داد.
از این گزاره‌های اتمی، با کمک عملگرهای بولی فرمول‌های پیچیده‌تری می‌سازیم، برای مثال:

- «$1 + 1 = 2$»
- «زمین از خورشید دورتر از زهره است»

نظام‌های منطقی فرآیند استدلال را رسمی می‌کنند و شباهت بسیاری به زبان‌های برنامه‌نویسی دارند که محاسبات را رسمی می‌کنند. در هر دو حالت لازم است که syntax (نحو) و semantics (معناشناسی) را تعریف کنیم.

نحو (**syntax**) مشخص می‌کند که چه رشته‌هایی از نمادها، فرمول‌های معتبر (یا در مورد زبان‌ها، برنامه‌های معتبر) هستند.

معناشناسی (**semantics**) بیان می‌کند که این فرمول‌های معتبر چه معنایی دارند (یا برنامه‌های معتبر چه محاسباتی انجام می‌دهند).

پس از آنکه نحو و معناشناسی منطق گزاره‌ای تعریف شد، خواهیم آموخت چگونه می‌توان جداول معنایی (semantic tableaux) ساخت که یک روش کارآمد برای تصمیم‌گیری در مورد صدق یا کذب یک فرمول فراهم می‌آورند.

2.1 فرمول‌های گزاره‌ای

در علوم کامپیوتر، عبارت (expression) به محاسبه‌ی یک مقدار از روی مقادیر دیگر اشاره دارد؛ برای مثال، $2 \times 9 + 5$. در منطق گزاره‌ای، به‌جای «عبارت» از واژه‌ی فرمول استفاده می‌شود. تعریف رسمی این واژه بر پایه‌ی درخت‌ها است، چراکه تکنیک اصلی اثبات ما، که استقرا ساختاری نام دارد، زمانی که روی درخت‌ها به‌کار رود آسان‌تر درک می‌شود. زیربخش‌های اختیاری در ادامه، رویکردهای مختلف برای تعریف نحو را بررسی خواهند کرد.

2.1.1 فرمول‌ها به‌صورت درختی

تعریف **2.1**. نمادهایی که برای ساخت فرمول‌های منطق گزاره‌ای استفاده می‌شوند عبارت‌اند از:

- مجموعه‌ای نامحدود از نمادها $\mathscr{F}$ به نام گزاره‌های اتمی (که به اختصار «اتم» نامیده می‌شوند). اتم‌ها با حروف کوچک از مجموعه‌ی $\{p, q, r, ...\}$ نمایش داده می‌شوند، که ممکن است دارای زیرنویس نیز باشند.

- عملگرهای بولی. نام‌ها و نمادهای مربوط به آن‌ها به شرح زیر است:

نام	نماد
نقیض	$\neg$
یای منطقی	$\vee$
هم‌بندی	$\wedge$
شرطی	$\rightarrow$
هم‌ارزی	$\leftrightarrow$
یا_انحصاری	$\oplus$
نور	$\downarrow$
نند	$\uparrow$

عملگر $\neg$ یک‌جمله‌ای (unary) است و تنها یک عملوند می‌گیرد، در حالی که سایر عملگرها دوجمله‌ای (binary) هستند و دو عملوند می‌پذیرند.

تعریف **2.2**. یک فرمول در منطق گزاره‌ای، درختی است که به‌صورت بازگشتی تعریف می‌شود:

- یک برگ با برچسب یک گزاره‌ی اتمی، یک فرمول است.

- گره‌ای با برچسب $\neg$ و تنها یک فرزند که خود یک فرمول باشد، یک فرمول است.

- گره‌ای با برچسب یکی از عملگرهای دوجمله‌ای و دو فرزند که هر دو فرمول باشند، یک فرمول است.

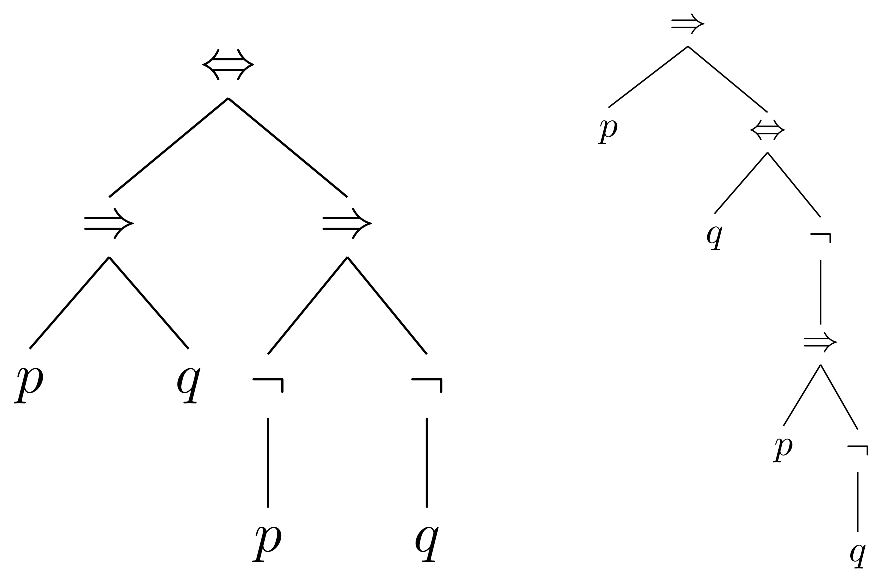
مثال **2.3**. شکل 2.1 دو فرمول مختلف را نمایش می‌دهد.

2.1.2 فرمول‌ها به‌صورت رشته‌ای

همان‌طور که عبارات ریاضی را به‌صورت رشته‌هایی (توالی خطی از نمادها) می‌نویسیم، می‌توانیم فرمول‌ها را نیز به‌شکل رشته‌ای نمایش دهیم. رشته‌ی متناظر با یک فرمول از طریق پیمایش درون‌گرد (preorder traversal) درخت آن به‌دست می‌آید:

الگوریتم **2.4** (نمایش فرمول به‌صورت رشته‌ای)
ورودی: یک فرمول A از منطق گزاره‌ای
خروجی: نمایش رشته‌ای از A رویه‌ی بازگشتی زیر را فراخوانی کن: Inorder(A)

```
Inorder(F):
    if F is a leaf:
        print its label
        return
```



شکل 2.1: دو فرمول

```
Let F1 and F2 be the left and right subtrees of F
Inorder(F1)
print the label of the root of F
Inorder(F2)
```

اگر ریشه‌ی F با نماد $\neg$ برچسب‌گذاری شده باشد، زیردرخت چپ نادیده گرفته می‌شود و مرحله‌ی Inorder(F1) انجام نمی‌گیرد.

تعریف **2.5**. اصطلاح «فرمول» برای رشته نیز به‌کار می‌رود، با این فرض که به درخت زیربنایی آن اشاره دارد.

مثال **2.6**. فرمول سمت چپ در شکل 2.1 را در نظر بگیرید. پیمایش درون‌گرد این فرمول به‌صورت زیر است: ابتدا برگ سمت چپ با برچسب p نوشته می‌شود، سپس ریشه‌ی آن که با $\rightarrow$ برچسب‌گذاری شده، سپس برگ سمت راست آن که q است، و سپس ریشه‌ی کل درخت که با $\leftrightarrow$ برچسب‌گذاری شده، و به همین ترتیب ادامه می‌یابد. نتیجه‌ی پیمایش رشته‌ی زیر است:

$$p \rightarrow q \leftrightarrow \neg p \rightarrow \neg q$$

اکنون فرمول سمت راست در شکل 2.1 را در نظر بگیرید. پیمایش آن نیز دقیقاً همین رشته را تولید می‌کند:

$$p \rightarrow q \leftrightarrow \neg p \rightarrow \neg q$$

2.1.3 رفع ابهام در نمایش رشته‌ای

پرانتزها

ساده‌ترین روش برای جلوگیری از ابهام، استفاده از پرانتزها برای حفظ ساختار درخت هنگام تولید رشته است.

الگوریتم **2.7** (نمایش فرمول به‌صورت رشته با پرانتز)
ورودی: یک فرمول A از منطق گزاره‌ای
خروجی: نمایش رشته‌ای از A با استفاده از پرانتزها
رویه‌ی بازگشتی زیر را فراخوانی کن: Inorder(A)

```
Inorder(F):
    if F is a leaf:
        print its label
        return
    Let F1 and F2 be the left and right subtrees of F
    print '('
    Inorder(F1)
    print the label of the root of F
    Inorder(F2)
    print ')'
```

اگر ریشه‌ی F با نماد $\neg$ برچسب‌گذاری شده باشد، زیردرخت چپ نادیده گرفته می‌شود و مرحله‌ی Inorder(F1) انجام نمی‌گیرد.
در این حالت، دو فرمول موجود در شکل 2.1 به دو رشته‌ی متفاوت نگاشته می‌شوند و دیگر ابهامی وجود ندارد:

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

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

مشکل استفاده از پرانتزها آن است که فرمول‌ها را طولانی کرده و خواندن و نوشتن آن‌ها را دشوار می‌سازد.

اولویت

روش دوم برای رفع ابهام در فرمول‌های مبهم، تعریف اولویت (precedence) و
جهت انجمنی (associativity) بین عملگرها است، همان‌طور که در حساب معمول انجام می‌شود. برای مثال، عبارت
a * b * c + d * e بلافاصله به‌صورت

$$(((a * b) * c) + (d * e))$$

تفسیر می‌شود.
در مورد فرمول‌های منطقی، ترتیب اولویت از بالا به پایین به‌صورت زیر است:

$$\neg$$

$$\wedge, \uparrow$$

$$\vee, \downarrow$$
$$\rightarrow$$
$$\leftrightarrow, \oplus$$

عملگرها به‌طور پیش‌فرض از راست به چپ گروه‌بندی می‌شوند (جهت انجاسی راست‌گرا)، یعنی فرمول $a \vee b \vee c$ به‌صورت $a \vee (b \vee c)$ تفسیر می‌شود.
پرانتزها تنها زمانی استفاده می‌شوند که بخواهیم ترتیبی متفاوت از قواعد اولویت و جهت پیش‌فرض را نشان دهیم؛ مانند حساب معمول که در آن a * (b + c) به معنای انجام جمع پیش از ضرب است.
با حداقل استفاده از پرانتز، دو فرمول مثال در شکل 2.1 را می‌توان به‌صورت زیر نوشت:

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

برای وضوح بیشتر، همیشه می‌توان از پرانتزهای اضافی استفاده کرد؛ مثلاً:

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

عملگرهای بولی $\wedge$، $\vee$، $\leftrightarrow$، $\oplus$ انجاسی هستند، بنابراین در فرمول‌هایی که شامل تکرار این عملگرها هستند، اغلب پرانتزها حذف می‌شوند؛ مثلاً:

$$p \vee q \vee r \vee s$$

اما توجه داشته باشید که عملگرهای $\rightarrow$، $\downarrow$، $\uparrow$ غیرانجاسی هستند، بنابراین استفاده از پرانتز برای جلوگیری از ابهام الزامی است.
اگرچه فرض بر این است که $\rightarrow$ راست‌گرا است و فرمول $p \rightarrow q \rightarrow r$ را به‌صورت $p \rightarrow (q \rightarrow r)$ تفسیر می‌کنیم، اما معمولاً برای وضوح بیشتر، آن را همراه با پرانتز می‌نویسیم:

$$(p \rightarrow q) \rightarrow r$$

نمادگذاری لهستانی (**Polish Notation**)

اگر نمایش رشته‌ای یک فرمول با استفاده از پیمایش پیش‌گرد (preorder traversal) درخت آن انجام شود، دیگر ابهامی نخواهیم داشت.

الگوریتم **2.8** (نمایش فرمول به‌صورت رشته در نماد لهستانی)
ورودی: یک فرمول A از منطق گزاره‌ای
خروجی: نمایش رشته‌ای از A در نمادگذاری لهستانی
رویه‌ی بازگشتی زیر را فراخوانی کن: Preorder(A)

```
Preorder(F):
    print the label of the root of F
    if F is a leaf:
        return
    Let F1 and F2 be the left and right subtrees of F
    Preorder(F1)
    Preorder(F2)
```

اگر ریشه‌ی F با نماد $\neg$ برچسب‌گذاری شده باشد، زیردرخت چپ نادیده گرفته می‌شود و مرحله‌ی Preorder(F1) انجام نمی‌گیرد.

مثال **2.9**. رشته‌های مربوط به دو فرمول موجود در شکل 2.1 در نمادگذاری لهستانی به‌صورت زیر هستند:

$$\leftrightarrow \rightarrow \ p\, q \ \rightarrow \ \neg p\, \neg q$$

$$\rightarrow \ p \ \leftrightarrow \ q\, \neg \ \rightarrow \ p\, \neg q$$

و اکنون دیگر هیچ ابهامی وجود ندارد.

فرمول‌هایی که به این صورت نمایش داده می‌شوند، گفته می‌شود در نمادگذاری لهستانی هستند، که به افتخار گروهی از منطق‌دانان لهستانی به رهبری Jan Łukasiewicz نام‌گذاری شده است.
ما نمایش درون‌گرد (infix) را خواناتر می‌دانیم، چراکه در حساب معمول نیز رایج است. بنابراین نمادگذاری لهستانی معمولاً تنها برای نمایش درونی عبارات حسابی و منطقی در رایانه‌ها استفاده می‌شود. مزیت اصلی آن این است که می‌توان فرمول را در همان ترتیبی که نمادها ظاهر می‌شوند با استفاده از یک stack ارزیابی کرد.
اگر فرمول اول را به‌صورت معکوس بازنویسی کنیم (که به آن نمادگذاری لهستانی معکوس یا RPN نیز گفته می‌شود):

$$q\, \neg\, p\, \neg \ \rightarrow \ q\, p \ \rightarrow \leftrightarrow$$

می‌توان آن را مستقیماً به دنباله‌ای از دستورات اسمبلی زیر ترجمه کرد:

```
Push q
Negate
Push p
Negate
Imply
Push q
Push p
Imply
Equiv
```

در این روش، عملگرها روی عملوندهایی که در بالای پشته قرار دارند اعمال می‌شوند، سپس عملوندها از پشته خارج (pop) شده و نتیجه روی پشته قرار داده می‌شود.

2.1.4 استقرا ساختاری

اگر یک عبارت حسابی مانند a * b + b * c را در نظر بگیریم، به‌راحتی می‌توان دید که این عبارت از دو جمله تشکیل شده که با عملگر جمع ترکیب شده‌اند، و هر جمله‌ی جمعی نیز از دو عامل ضرب تشکیل شده است. به‌طور مشابه، هر فرمول گزاره‌ای را می‌توان بر اساس عملگر سطح بالای آن طبقه‌بندی کرد.

تعریف **2.10**. اگر $A \in \mathscr{F}$ و A یک اتم نباشد، عملگری که ریشه‌ی درخت فرمول A را برچسب‌گذاری می‌کند، عملگر اصلی (principal operator) فرمول A نامیده می‌شود.

مثال **2.11**. عملگر اصلی فرمول سمت چپ در شکل 2.1، عملگر $\leftrightarrow$ و در فرمول سمت راست، عملگر $\rightarrow$ است.

استقرا ساختاری روشی برای اثبات این است که یک خاصیت برای همه‌ی فرمول‌ها برقرار است. این شکل از استقرا، مشابه استقرای عددی آشنایی است که برای اثبات خاصیت برای همه‌ی اعداد طبیعی به‌کار می‌رود (بخش A.6 را ببینید). در استقرای عددی:

- گام پایه: اثبات خاصیت برای صفر است، و
- گام استقرا: فرض می‌کنیم خاصیت برای عدد دلخواه n برقرار است و سپس اثبات می‌کنیم که برای $n+1$ نیز برقرار خواهد بود.

بر اساس تعریف 2.10، یک فرمول یا:

- یک برگ با برچسب اتم است،
- یا یک درخت با عملگر اصلی و یک یا دو زیردرخت.

بنابراین، در استقرا ساختاری:

- گام پایه: اثبات خاصیت برای برگ‌ها (اتم‌ها) است.
- گام استقرا: اثبات خاصیت برای فرمولی است که از به‌کارگیری عملگر اصلی روی زیرفرمول‌ها به‌دست آمده، مشروط بر آنکه خاصیت برای آن زیرفرمول‌ها برقرار باشد.

قضیه **2.12** (استقرا ساختاری). . برای اینکه نشان دهیم خاصیتی برای همه‌ی فرمول‌ها $A \in \mathscr{F}$ برقرار است، کافی است:

۱. اثبات کنیم که خاصیت برای تمام اتم‌ها p برقرار است.

۲. فرض کنیم خاصیت برای فرمولی مانند A برقرار است و اثبات کنیم که خاصیت برای $\neg A$ نیز برقرار است.

۳. فرض کنیم خاصیت برای فرمول‌های A_1 و A_2 برقرار است، و اثبات کنیم که خاصیت برای $A_1 \text{ op } A_2$ نیز برقرار است، برای هر یک از عملگرهای دوجمله‌ای.

اثبات. فرض کنیم A یک فرمول دلخواه باشد و فرض کنیم که بندهای (۱)، (۲)، (۳) برای خاصیت مورد نظر اثبات شده‌اند. ما نشان می‌دهیم که خاصیت برای A برقرار است، با استفاده از استقرای عددی بر حسب n، که ارتفاع درخت مربوط به A است:

- اگر $n = 0$ باشد، درخت یک برگ است، بنابراین A یک اتم p است و خاصیت طبق بند (۱) برقرار است.
- اگر $n > 0$ باشد، زیردرخت‌های A ارتفاعی برابر با $n-1$ دارند، بنابراین طبق فرض استقرای عددی، خاصیت برای آن‌ها برقرار است. از آنجا که عملگر اصلی A یا نقیض ($\neg$) است یا یکی از عملگرهای دوجمله‌ای، بنابراین طبق بند (۲) یا (۳)، خاصیت برای A نیز برقرار است.

□

بعداً نشان خواهیم داد که تمام عملگرهای دوجمله‌ای را می‌توان با ترکیب نقیض و یکی از دو عملگر $\vee$ یا $\wedge$ تعریف کرد؛ بنابراین، برای اثبات خاصیت برای همه‌ی فرمول‌ها، کافی است از استقرا ساختاری با حالت پایه و تنها دو گام استقرا استفاده کنیم.

2.1.5 نمادگذاری

متأسفانه، در کتاب‌های مختلف منطق ریاضی، نمادهای مورد استفاده برای عملگرهای بولی یکسان نیستند؛ افزون بر این، در زبان‌های برنامه‌نویسی نیز این عملگرها معمولاً با نمادهایی متفاوت از آنچه در منابع ریاضی دیده می‌شود نمایش داده می‌شوند. جدول زیر، برخی از این نمادهای جایگزین را نشان می‌دهد:

عملگر	نمادهای جایگزین	زبان **Java**
$\neg$	$\sim$	!
$\wedge$	&	&، &&
$\vee$		$\mid$، $\mid\mid$
$\rightarrow$	$\supset$، $\Rightarrow$	
$\leftrightarrow$	$\equiv$، $\Leftrightarrow$	
$\oplus$	$\not\equiv$	^
$\uparrow$	$\mid$	

2.1.6 دستور زبان صوری برای فرمول ها

(این زیربخش مستلزم آشنایی با دستور زبان‌های صوری است.) به‌جای تعریف فرمول‌ها به‌صورت درختی، می‌توان آن‌ها را به‌شکل رشته‌هایی تعریف کرد که از یک دستور زبان مستقل از متن (context-free grammar) تولید می‌شوند.

تعریف **2.13**. فرمول‌های منطق گزاره‌ای از دستور زبانی مستقل از متن تولید می‌شوند که نمادهای پایانی (terminal symbols) آن به‌صورت زیر تعریف شده‌اند:

- مجموعه‌ای نامتناهی از نمادها به‌نام گزاره‌های اتمی $\mathcal{P}$
- عملگرهای بولی مطابق با تعریف 2.1

قواعد تولید (production rules) این دستور زبان به‌صورت زیر هستند:

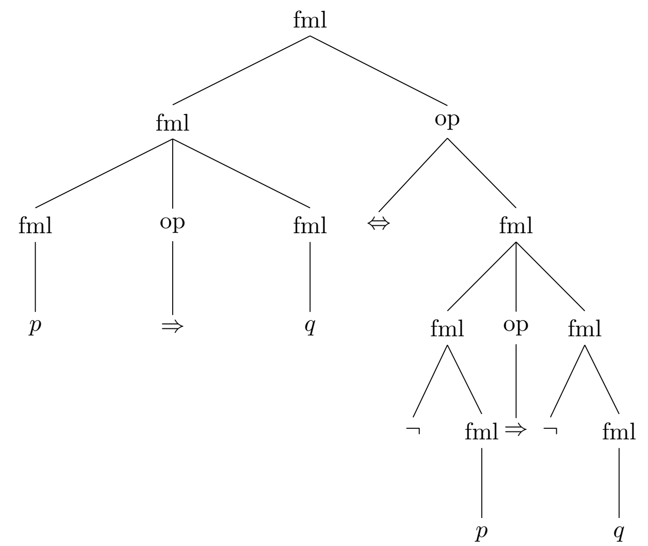
$$\begin{aligned} \mathrm{fml} &::= p \qquad \text{برای هر } p \in \mathcal{P} \\ \mathrm{fml} &::= \neg \mathrm{fml} \\ \mathrm{fml} &::= \mathrm{fml}\ \mathrm{op}\ \mathrm{fml} \\ \mathrm{op} &::= \vee \mid \wedge \mid \rightarrow \mid \leftrightarrow \mid \oplus \mid \uparrow \mid \downarrow \end{aligned}$$

یک فرمول، رشته‌ای است که می‌توان آن را از نماد غیرپایانی fml استخراج کرد. مجموعه‌ی تمام فرمول‌هایی که از این دستور زبان به‌دست می‌آیند با نماد $\mathscr{F}$ نمایش داده می‌شود. فرآیند تولید (اشتقاق) رشته‌ها در یک دستور زبان صوری را می‌توان با درخت‌های اشتقاق (derivation trees) نمایش داد . رشته‌ی نهایی را می‌توان از طریق خواندن برگ‌ها از چپ به راست به‌دست آورد.

مثال **2.14**. در ادامه، فرآیند اشتقاق فرمول زیر در منطق گزاره‌ای نشان داده شده است:

$$p \rightarrow q \leftrightarrow \neg p \rightarrow \neg q$$

درخت مربوطه در شکل 2.2 آمده است. گام‌های اشتقاق به‌صورت زیر هستند:



شکل 2.2: اشتقاق درخت برای $p \to q \leftrightarrow \neg p \to \neg q$

1. fml
2. fml op fml
3. fml $\leftrightarrow$ fml
4. fml $\to$ fml $\leftrightarrow$ fml
5. $p \to$ fml $\leftrightarrow$ fml
6. $p \to q \leftrightarrow$ fml
7. $p \to q \leftrightarrow$ fml $\to$ fml
8. $p \to q \leftrightarrow \neg$fml $\to$ fml
9. $p \to q \leftrightarrow \neg p \to$ fml
10. $p \to q \leftrightarrow \neg p \to \neg$fml
11. $p \to q \leftrightarrow \neg p \to \neg q$

روش‌هایی که در بخش 2.1.2 برای رفع ابهام معرفی شدند، در اینجا نیز قابل‌استفاده هستند. همچنین می‌توان دستور زبان را طوری بازنویسی کرد که پرانتزها را به‌صورت درونی دربر گیرد:

$$\text{fml} ::= (\neg\text{fml})$$
$$\text{fml} ::= (\text{fml op fml})$$

و سپس با تعریف قواعد اولویت، می‌توان تعداد پرانتزها را کاهش داد.

$v_{\mathscr{I}}(A) = \mathscr{I}_A(A)$	اگر A یک اتم باشد
$v_{\mathscr{I}}(\neg A) = T$	اگر $v_{\mathscr{I}}(A) = F$
$v_{\mathscr{I}}(\neg A) = F$	اگر $v_{\mathscr{I}}(A) = T$
$v_{\mathscr{I}}(A_1 \vee A_2) = F$	اگر $v_{\mathscr{I}}(A_1) = F$ و $v_{\mathscr{I}}(A_2) = F$
$v_{\mathscr{I}}(A_1 \vee A_2) = T$	در غیر این صورت
$v_{\mathscr{I}}(A_1 \wedge A_2) = T$	اگر $v_{\mathscr{I}}(A_1) = T$ و $v_{\mathscr{I}}(A_2) = T$
$v_{\mathscr{I}}(A_1 \wedge A_2) = F$	در غیر این صورت
$v_{\mathscr{I}}(A_1 \rightarrow A_2) = F$	اگر $v_{\mathscr{I}}(A_1) = T$ و $v_{\mathscr{I}}(A_2) = F$
$v_{\mathscr{I}}(A_1 \rightarrow A_2) = T$	در غیر این صورت
$v_{\mathscr{I}}(A_1 \uparrow A_2) = F$	اگر $v_{\mathscr{I}}(A_1) = T$ و $v_{\mathscr{I}}(A_2) = T$
$v_{\mathscr{I}}(A_1 \uparrow A_2) = T$	در غیر این صورت
$v_{\mathscr{I}}(A_1 \downarrow A_2) = T$	اگر $v_{\mathscr{I}}(A_1) = F$ و $v_{\mathscr{I}}(A_2) = F$
$v_{\mathscr{I}}(A_1 \downarrow A_2) = F$	در غیر این صورت
$v_{\mathscr{I}}(A_1 \leftrightarrow A_2) = T$	اگر $v_{\mathscr{I}}(A_1) = v_{\mathscr{I}}(A_2)$
$v_{\mathscr{I}}(A_1 \leftrightarrow A_2) = F$	اگر $v_{\mathscr{I}}(A_1) \neq v_{\mathscr{I}}(A_2)$
$v_{\mathscr{I}}(A_1 \oplus A_2) = T$	اگر $v_{\mathscr{I}}(A_1) \neq v_{\mathscr{I}}(A_2)$
$v_{\mathscr{I}}(A_1 \oplus A_2) = F$	اگر $v_{\mathscr{I}}(A_1) = v_{\mathscr{I}}(A_2)$

شکل 2.3: مقادیر منطقی فرمول‌ها

2.2 تعبیرها

اکنون معناشناسی—یعنی معنای فرمول‌ها—را تعریف می‌کنیم. دوباره به عبارت‌های حسابی فکر کنید. فرض کنید عبارت

$$E \ = \ a * b + 2$$

باشد؛ می‌توانیم برای a و b مقادیری اختصاص دهیم و سپس مقدار عبارت را محاسبه کنیم. مثلاً اگر $a = 2$ و $b = 3$ باشد، آنگاه مقدار E برابر 8 خواهد بود. در منطق گزاره‌ای، مقادیر صدق به اتم‌های یک فرمول نسبت داده می‌شوند تا مقدار صدق کل فرمول تعیین شود.

2.2.1 تعریف یک تعبیر

تعریف **2.15**. فرض کنید $A \in \mathscr{F}$ یک فرمول باشد و $\mathscr{P}_A$ مجموعهٔ اتم‌های ظاهرشده در A باشد. یک تعبیر برای A، تابع کلی

$$\mathscr{I}_A : \mathscr{P}_A \ \longrightarrow \ \{T, F\}$$

است که به هر اتم در $\mathscr{P}_A$ یکی از مقادیر صدق T (درست) یا F (نادرست) اختصاص می‌دهد.

تعریف **2.16**. اگر $\mathscr{I}_A$ یک تعبیر برای $A \in \mathscr{F}$ باشد، مقدار صدق A تحت $\mathscr{I}_A$، که با

$$v_{\mathscr{I}_A}(A)$$

نشان داده می‌شود، به صورت بازگشتی بر اساس ساختار A و مطابق شکل ۳.۲ تعریف می‌شود. در عمل، وقتی زمینه مشخص باشد، از خلاصهٔ $\mathscr{I}$ به جای $\mathscr{I}_A$ استفاده می‌کنیم و مقدار صدق را به صورت $v_{\mathscr{I}}(A)$ می‌نویسیم.

مثال **2.17**. فرض کنید

$$A \;=\; (p \to q) \;\leftrightarrow\; (\neg q \to \neg p)$$

و تعبیر زیر را برای آن در نظر بگیریم:

$$\mathscr{I}_A(p) = F, \quad \mathscr{I}_A(q) = T.$$

آنگاه مقدار صدق A تحت $\mathscr{I}$ با استفاده از قواعد ارزیابی (شکل ۳.۲) چنین خواهد بود:

$$\begin{aligned}
v_{\mathscr{I}}(p) &= \mathscr{I}_A(p) = F,\\
v_{\mathscr{I}}(q) &= \mathscr{I}_A(q) = T,\\
v_{\mathscr{I}}(p \to q) &= T,\\
v_{\mathscr{I}}(\neg q) &= F,\\
v_{\mathscr{I}}(\neg p) &= T,\\
v_{\mathscr{I}}(\neg q \to \neg p) &= T,\\
v_{\mathscr{I}}\big((p \to q) \leftrightarrow (\neg q \to \neg p)\big) &= T.
\end{aligned}$$

تعبیرهای جزئی

تعریف **2.18**. فرض کنید $A \in \mathscr{F}$ یک فرمول در منطق گزاره‌ای باشد و $\mathscr{P}_A$ مجموعهٔ اتم‌های ظاهرشده در A باشد. یک «تعبیر جزئی» برای A، تابع جزئی

$$\mathscr{I}_A : \mathscr{P}_A \rightharpoonup \{T, F\}$$

است که به برخی (نه لزوماً همه) اتم‌های $\mathscr{P}_A$ ارزش T یا F تخصیص می‌دهد. ممکن است در یک تعبیر جزئی، بدون این‌که به همهٔ اتم‌ها مقدار داده باشیم، بتوان ارزش صدق فرمول A را مشخص کرد.

مثال **2.19**. فرمول

$$A = p \wedge q$$

را در نظر بگیرید. یک تعبیر جزئی را در نظر بگیرید که تنها به p مقدار F می‌دهد. روشن است که در این تعبیر جزئی، ارزش صدق A برابر F خواهد بود، زیرا $p \wedge q$ تنها زمانی T است که هر دو p و q برابر T باشند. اگر همین تعبیر جزئی به p مقدار T می‌داد ولی به q مقداری تخصیص نمی‌داد، آنگاه ارزش صدق A قابل تعیین نبود.

2.2.2 جداول ارزش

تعریف **2.20**. فرض کنید $A \in \mathscr{F}$ و مجموعهٔ اتم‌های ظاهرشده در A، $\mathscr{P}_A$، شامل n اتم باشد. یک «جدول ارزش» برای A جدولی است با $n+1$ ستون و 2^n سطر. ستون‌های اول (تعداد n) هر یک یک اتم از $\mathscr{P}_A$ را نشان می‌دهند و تمام ترکیب‌های ممکنِ انتساب T یا F به آن‌ها را فهرست می‌کنند (هر سطر یک تعبیر $\mathscr{I}$ را مشخص می‌کند). ستون آخر، ارزش صدق A، یعنی $v_{\mathscr{I}}(A)$، را در هر تعبیر $\mathscr{I}$ نمایش می‌دهد.

مثال **2.21**. جدول ارزش فرمول

$$p \;\to\; q$$

به صورت زیر است:

p	q	$p \to q$
T	T	T
T	F	F
F	T	T
F	F	T

مثال **2.22**. برای فرمول

$$(p \to q) \ \leftrightarrow \ (\neg q \to \neg p)$$

ستون‌های میانی ارزش زیرفرمول‌ها را نیز نمایش می‌دهیم:

p	q	$p \to q$	$\neg p$	$\neg q$	$\neg q \to \neg p$	$(p \to q) \leftrightarrow (\neg q \to \neg p)$
T	T	T	F	F	T	T
T	F	F	F	T	F	T
F	T	T	T	F	T	T
F	F	T	T	T	T	T

مثال **2.23**. برای تعبیر $\mathscr{I}$ با $\mathscr{I}(p) = T$ و $\mathscr{I}(q) = F$، مراحل گام‌به‌گام محاسبهٔ ارزش صدق

$$(p \to q) \ \leftrightarrow \ (\neg q \to \neg p)$$

را می‌توان به صورت زیر در یک جدول نمایش داد. در هر سطر یک مقدار جدید از زیرفرمول اضافه می‌شود:

$(p$	$\to$	$q)$	$\leftrightarrow$	$(\neg$	q	$\to$	$\neg$	$p)$
T		F			F		T	
T		F			F		T	
T		F		F	F	F	T	
T		F		F	F	F	T	
T	F	F		F	F	F	T	
T	F	F	T	F	F	F	T	

اگر همهٔ زیرفرمول‌ها یک‌جا در یک سطر قرار گیرند، همان جدول کامل مثال 2.22 حاصل می‌شود:

p	q	$p \to q$	$\neg p$	$\neg q$	$\neg q \to \neg p$	$(p \to q) \leftrightarrow (\neg q \to \neg p)$
T	T	T	F	F	T	T
T	F	F	F	T	F	T
F	T	T	T	F	T	T
F	F	T	T	T	T	T

2.2.3 فهم عملگرهای بولی

خوانش طبیعی از عملگرهای بولی «نقیض» ($\neg$) و «و» ($\wedge$) با معناهای رسمی‌ای که در شکل . تعریف شدند، منطبق است. عملگرهای «ناند» ($\uparrow$) و «نُر» ($\downarrow$) صرفاً نقیض‌های $\wedge$ و $\vee$ هستند. در این‌جا دربارهٔ عملگرهای «یا» ($\vee$)، «یاِحصری» ($\oplus$) و «التزام» ($\to$) که معناهای رسمی‌شان می‌تواند موجب سردرگمی شود، توضیح می‌دهیم.

«یا»ی شمولی در برابر «یاِحصری»

عملگر «یا» ($\vee$) همان یاِ شمولی است و با «یاِحصری» ($\oplus$) متفاوت است. برای مثال، فرض کنید می‌گوییم:

ساعت هشت «به سینما می‌روم» یا «به تئاتر می‌روم».

منظور از این گزاره عبارت است از «سینما» $\oplus$ «تئاتر»، چرا که در یک لحظه نمی‌توان هم‌زمان در هر دو مکان بود. این در حالی است که عملگر $\vee$ زمانی مقدار درستی (T) می‌گیرد که دست‌کم یکی از جملات صادق باشد:

از پاپ‌کورن یا از آب‌نبات می‌خواهید؟

در این حالت می‌توان گفت: «پاپ‌کورن» $\vee$ «آب‌نبات»، زیرا ممکن است هم پاپ‌کورن و هم آب‌نبات را بخواهیم. برای $\vee$ کافی است یکی از زیرجملات صادق باشد تا کلِ عبارت صادق شود؛ بنابراین عبارت نامأنوس زیر نیز صادق است، تنها به این دلیل که جملهٔ اول به تنهایی کافی است:

«زمین دورتر از خورشید است تا ونوس» $\vee$ «$1 + 1 = 3$»

تفاوت وقتی هر دو زیرجمله صادق باشند

- با $\vee$: اگر دو زیرجمله صادق باشند، یاِ شمولی همچنان صادق است.
- با $\oplus$: اگر دو زیرجمله صادق باشند، یاِحصری نادرست است (چرا که حصری بودن ایجاب می‌کند دقیقاً یکی صادق باشد).

اینگونه می‌توان تفکیک روشنی میان $\vee$ و $\oplus$ قائل شد.

یا شمولی در برابر یا اختصاصی در زبان‌های برنامه‌نویسی

وقتی or در زمینهٔ زبان‌های برنامه‌نویسی به کار می‌رود، معمولاً منظور همان یا شمولی است:

```
if (index < min || index > max) /* There is an error */
```

در این مثال، درستی یکی از دو زیرعبارت باعث اجرای دستورات بعدی می‌شود. عملگر `||` در اصل یک عملگر بولی واقعی نیست، زیرا از «ارزیابی کوتاه‌مدت» (short-circuit evaluation) استفاده می‌کند: اگر زیرعبارت اول درست باشد، زیرعبارت دوم اصلاً ارزیابی نمی‌شود، چرا که نتیجهٔ آن نمی‌تواند تصمیم به اجرای ادامهٔ دستورات را تغییر دهد. برای ارزیابی بولی واقعی می‌توان از عملگر `|` استفاده کرد؛ این عملگر معمولاً هنگام کار با بردارهای بیتی به کار می‌رود:

```
mask1 = 0xA0;
mask2 = 0x0A;
mask  = mask1 | mask2;
```

یاِحصری ($\oplus$ در منطق) که در زبان‌های برنامه‌نویسی با نماد `^` نمایش داده می‌شود، برای رمزگذاری و رمزگشایی در سیستم‌های تصحیح خطا و رمزنگاری کاربرد دارد. دلیل این کار این است که با استفادهٔ دوباره از آن می‌توان مقدار اصلی را بازیافت. فرض کنید داده‌ای را با یک کلید مخفی رمزگذاری کنیم:

```
codedMessage = data ^ key;
```

گیرندهٔ پیام نیز می‌تواند به این صورت آن را رمزگشایی نماید:

```
clearMessage = codedMessage ^ key;
```

با دقت در محاسبهٔ زیر می‌بینیم که مقدار اولیه بازیابی می‌شود:

```
clearMessage == codedMessage ^ key
             == (data ^ key) ^ key
             == data ^ (key ^ key)
             == data ^ false
             == data
```

التزام (Implication)

عملگر $p \to q$ «التزام مادی» نامیده می‌شود؛ p مقدم و q مؤخر نامیده می‌شود. التزام مادی ادعای علیّت نمی‌کند؛ یعنی نمی‌گوید که مقدم، باعث مؤخر شده یا حتی با آن مرتبط است. یک التزام مادی تنها بیان می‌کند که اگر مقدم درست باشد، مؤخر نیز باید درست باشد؛ بنابراین تنها زمانی که مقدم درست و مؤخر نادرست باشد می‌توان آن را نادرست یافت. برای مثال، به دو گزارهٔ زیر توجه کنید:

«$1+1=3$» $\to$ «زمین دورتر از خورشید است تا ونوس»

این گزاره نادرست است، چرا که مقدم درست و مؤخر نادرست است. اما:

«$1+1=3$» $\to$ «زمین دورتر از خورشید است تا مریخ»

این گزاره صادق است، زیرا نادرستی مقدم به تنهایی برای صادق بودن کل التزام کافی است.

2.2.4 تعبیری برای یک مجموعه از فرمول‌ها

تعریف **2.24**. فرض کنید

$$S = \{A_1, A_2, \dots\}$$

مجموعه‌ای از فرمول‌ها باشد و بگذارید

$$\mathscr{P}_S = \bigcup_i \mathscr{P}_{A_i}$$

که $\mathscr{P}_S$ مجموعهٔ تمام اتم‌هایی است که در فرمول‌های S ظاهر می‌شوند.
یک تعبیر برای S تابعی است به صورت:

$$\mathscr{I}_S : \mathscr{P}_S \to \{T, F\}.$$

برای هر $A_i \in S$، مقدار صدق $v_{\mathscr{I}_S}(A_i)$ (یعنی ارزش صدق A_i تحت تعبیر $\mathscr{I}_S$) دقیقاً همانند تعریف 2.16 تعیین می‌شود.
تعریف $\mathscr{P}_S$ به‌صورت اجتماع مجموعه‌های اتم‌ها در فرمول‌های S تضمین می‌کند که به هر اتم دقیقاً یک مقدار «درست» یا «غلط» اختصاص یابد.

مثال **2.25**. فرض کنید

$$S = \{\, p \to q,\ p,\ q \wedge r,\ p \vee s \leftrightarrow s \wedge q \,\}$$

و تعبیر $\mathscr{I}_S$ چنان است که

$$\mathscr{I}_S(p) = T, \quad \mathscr{I}_S(q) = F, \quad \mathscr{I}_S(r) = T, \quad \mathscr{I}_S(s) = T.$$

آنگاه مقادیر صدق اعضای S به‌صورت زیر محاسبه می‌شوند:

$$\begin{aligned} v_{\mathscr{I}_S}(p \to q) &= F,\\ v_{\mathscr{I}_S}(p) &= \mathscr{I}_S(p) = T,\\ v_{\mathscr{I}_S}(q \wedge r) &= F,\\ v_{\mathscr{I}_S}(p \vee s) &= T,\\ v_{\mathscr{I}_S}(s \wedge q) &= F,\\ v_{\mathscr{I}_S}(p \vee s \leftrightarrow s \wedge q) &= F. \end{aligned}$$

2.3 معادل‌های منطقی

تعریف **2.26**. فرض کنید $A_1, A_2 \in \mathscr{F}$. اگر برای همهٔ تعبیرها $\mathscr{I}$ داشته باشیم

$$v_{\mathscr{I}}(A_1) = v_{\mathscr{I}}(A_2),$$

آنگاه می‌گوییم A_1 معادل منطقی A_2 است و با

$$A_1 \equiv A_2$$

نشان می‌دهیم.

مثال **2.27**. آیا فرمول

$$p \vee q$$

معادل منطقی

$$q \vee p$$

است؟ چهار تعبیر متمایز برای اتم‌های p و q وجود دارد:

$v_{\mathscr{I}}(q \vee p)$	$v_{\mathscr{I}}(p \vee q)$	$\mathscr{I}(q)$	$\mathscr{I}(p)$
T	T	T	T
T	T	F	T
T	T	T	F
F	F	F	F

چون در همهٔ این تعبیرها صدق دو فرمول یکسان است، داریم:

$$p \vee q \;\equiv\; q \vee p.$$

قضیه **2.28**. برای هر $A_1, A_2 \in \mathscr{F}$،

$$A_1 \vee A_2 \;\equiv\; A_2 \vee A_1.$$

اثبات. بگذارید $\mathscr{I}$ یک تعبیر دلخواه برای $A_1 \vee A_2$ باشد. واضح است که $\mathscr{I}$ تعبیر معتبری برای $A_2 \vee A_1$ نیز هست، زیرا:

$$\mathscr{P}_{A_1} \cup \mathscr{P}_{A_2} \;=\; \mathscr{P}_{A_2} \cup \mathscr{P}_{A_1}.$$

از آنجا که $\mathscr{P}_{A_1} \subseteq \mathscr{P}_{A_1} \cup \mathscr{P}_{A_2}$، تعبیر $\mathscr{I}$ به همهٔ اتم‌های A_1 مقدار می‌دهد و بنابراین تعبیر معتبری برای A_1 است؛ به‌طور مشابه برای A_2.
اکنون داریم:

$$v_{\mathscr{I}}(A_1 \vee A_2) = T \iff v_{\mathscr{I}}(A_1) = T \;\vee\; v_{\mathscr{I}}(A_2) = T,$$

و

$$v_{\mathscr{I}}(A_2 \vee A_1) = T \iff v_{\mathscr{I}}(A_2) = T \;\vee\; v_{\mathscr{I}}(A_1) = T.$$

اگر $v_{\mathscr{I}}(A_1) = T$، آنگاه

$$v_{\mathscr{I}}(A_1 \vee A_2) = T = v_{\mathscr{I}}(A_2 \vee A_1),$$

و همین‌طور اگر $v_{\mathscr{I}}(A_2) = T$.
چون $\mathscr{I}$ دلخواه بود، نتیجه می‌گیریم:

$$A_1 \vee A_2 \equiv A_2 \vee A_1.$$

□

2.3.1 (رابطهٔ بین $\leftrightarrow$ و $\equiv$)

عملگر «معادل» ($\leftrightarrow$) یک عملگر بولی در منطق گزاره‌ای است و می‌تواند در فرمول‌های این منطق ظاهر شود. اما «معادل منطقی» ($\equiv$) یک عملگر بولی نیست؛ بلکه نشانه‌ای برای ویژگی یک جفت فرمول در منطق گزاره‌ای است. این دو مفهوم می‌توانند سردرگمی ایجاد کنند، زیرا ما از واژگان مشابه هم برای زبان مَحْتوایی (در اینجا زبان منطق گزاره‌ای) و هم برای زبان مَدرِکی—آنچه برای استدلال دربارهٔ زبان محتوا استفاده می‌کنیم—بهره می‌بریم.
با این حال، معادل بودن ($\leftrightarrow$) و معادل منطقی ($\equiv$) ارتباط نزدیکی دارند، چنان‌که در قضیهٔ زیر نشان داده شده است:

قضیه **2.29**. $A_1 \equiv A_2$ اگر و تنها اگر $A_1 \leftrightarrow A_2$ در هر تعبیر صدق کند.

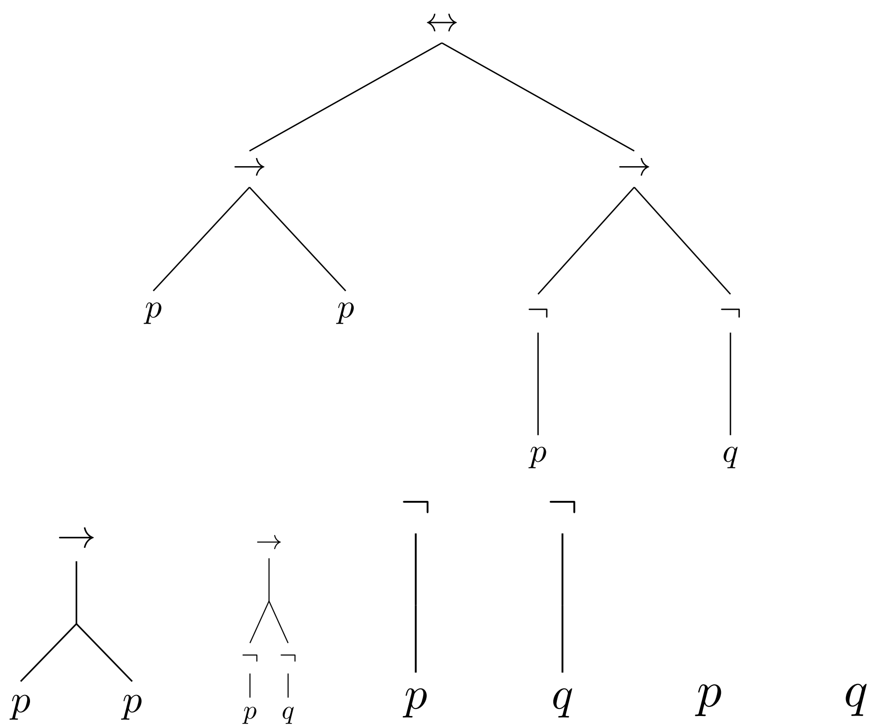
اثبات. فرض کنید $A_1 \equiv A_2$ و $\mathscr{I}$ یک تعبیر دلخواه باشد. از تعریف معادل منطقی می‌دانیم:

$$v_{\mathscr{I}}(A_1) = v_{\mathscr{I}}(A_2).$$

طبق جدول ارزش صدق (شکل 2.3)، در این صورت داریم:

$$v_{\mathscr{I}}(A_1 \leftrightarrow A_2) = T.$$

از آنجا که $\mathscr{I}$ دلخواه بود، نتیجه می‌شود $A_1 \leftrightarrow A_2$ در همهٔ تعبیرها درست است.
اثبات جهت معکوس (یعنی اگر $A_1 \leftrightarrow A_2$ در همهٔ تعبیرها درست باشد، آنگاه $A_1 \equiv A_2$) به‌صورتی مشابه انجام می‌پذیرد. □



شکل 2.4: زیر فرمول ها

2.3.2 جایگزینی

معادل منطقی توجیه‌کنندهٔ جایگزینی یک فرمول به‌جای فرمول دیگری است.

تعریف **2.30**. اگر A زیردرختی از فرمول B باشد، آنگاه می‌گوییم A یک زیرفرمول از B است. اگر A دقیقاً با B یکسان نباشد، آن را زیرفرمول درست B می‌نامیم.

مثال **2.31**. شکل 2.4 فرمول

$$(p \to q) \;\leftrightarrow\; (\neg p \to \neg q)$$

(فرمول سمت چپ شکل 2.1) و زیرفرمول‌های درست آن را نشان می‌دهد. اگر به‌صورت رشته نمایش داده شود، زیرفرمول‌های درست عبارتند از:

$$p \to q, \quad \neg p \to \neg q, \quad \neg p, \quad \neg q, \quad p, \quad q.$$

تعریف **2.32**. فرض کنید A یک زیرفرمول از B باشد و A' هر فرمول دلخواهی باشد. $B\{A \leftarrow A'\}$ یعنی جایگزینی A' به‌جای A در B، فرمولی است که از B به‌دست می‌آید وقتی همهٔ زیردرخت‌های متناظر با A در B با A' جایگزین شوند.

مثال **2.33**. بگذارید

$$B = (p \to q) \leftrightarrow (\neg p \to \neg q), \quad A = p \to q, \quad A' = \neg p \vee q.$$

آنگاه

$$B\{A \leftarrow A'\} \;=\; (\neg p \vee q) \;\leftrightarrow\; (\neg q \rightarrow \neg p).$$

اگر در فرمول B، یک زیرفرمول A با فرمولی که معادل منطقی A است جایگزین شود، ارزش‌گذاری B در هیچ تعبیر تغییر نمی‌کند.

قضیه **2.34**. فرض کنید A زیرفرمولی از B باشد و A' فرمولی باشد که $A \equiv A'$. آنگاه

$$B \;\equiv\; B\{A \leftarrow A'\}.$$

اثبات. بگذارید $\mathscr{I}$ یک تعبیر دلخواه باشد. چون $A \equiv A'$، پس:

$$v_{\mathscr{I}}(A) = v_{\mathscr{I}}(A').$$

باید نشان دهیم:

$$v_{\mathscr{I}}(B) = v_{\mathscr{I}}\big(B\{A \leftarrow A'\}\big).$$

اثبات با استقرا بر عمق d بالاترین وقوع زیردرخت A در B انجام می‌شود:

- حالت پایه ($d = 0$): در این حالت، تنها یک وقوع از A وجود دارد که همان خود B است. پس:

$$v_{\mathscr{I}}(B) = v_{\mathscr{I}}(A) = v_{\mathscr{I}}(A') = v_{\mathscr{I}}\big(B\{A \leftarrow A'\}\big).$$

- گام استقرا ($d > 0$): در این حالت، B یکی از دو صورت زیر است:

  ۱. $B = \neg B_1$،
  ۲. $B = B_1 \text{ op } B_2$ برای برخی فرمول‌های B_1, B_2 و یک عملگر بولی op.

  در هر دو صورت، عمق A در زیردرخت‌های B_1 و B_2 کمتر از d است. بنابراین، بر اساس فرض استقرا:

$$v_{\mathscr{I}}(B_1) = v_{\mathscr{I}}\big(B_1\{A \leftarrow A'\}\big), \quad v_{\mathscr{I}}(B_2) = v_{\mathscr{I}}\big(B_2\{A \leftarrow A'\}\big).$$

  پس بر اساس تعریف ارزش‌گذاری برای عملگرهای بولی:

$$v_{\mathscr{I}}(B) = v_{\mathscr{I}}\big(B\{A \leftarrow A'\}\big).$$

در نتیجه، از آنجا که $\mathscr{I}$ دلخواه بود داریم:

$$B \equiv B\{A \leftarrow A'\}.$$

□

2.3.3 فرمول‌های معادل منطقی

جایگزینی فرمول‌های معادل منطقی اغلب انجام می‌شود، برای مثال در ساده‌سازی فرمول‌ها، و آشنایی با معادل‌های پرکاربرد فهرست‌شده در این زیربخش ضروری است. اثبات آن‌ها از تعاریف ابتدایی به‌دست می‌آید و به‌عنوان تمرین باقی گذاشته شده است.

جذب ثابت‌ها (**Absorption of Constants**)

اجازه دهید نحو فرمول‌های بولی را طوری گسترش دهیم که دو گزارهٔ اتمی ثابت true و false را نیز شامل شود. (نمادهای دیگر: $\top$ برای true و $\bot$ برای false.) معنای آن‌ها به صورت زیر تعریف می‌شود:

$$\mathscr{I}(\text{true}) = T \quad \text{و} \quad \mathscr{I}(\text{false}) = F$$

برای هر تعبیر $\mathscr{I}$. این نمادها را نباید با مقادیر صدق T و F که در تعریف ارزش‌گذاری به‌کار می‌روند، اشتباه گرفت. همچنین می‌توان true و false را به‌ترتیب به‌عنوان اختصار برای فرمول‌های زیر در نظر گرفت:

$$p \vee \neg p \quad \text{و} \quad p \wedge \neg p.$$

وقوع یک ثابت در فرمول ممکن است آن را چنان فرو بکاهد که عملگر دودویی بی‌نیاز شود یا حتی فرمول به یک ثابت تبدیل شود:

$$\begin{array}{cc}
A \vee \text{true} \equiv \text{true} & A \wedge \text{true} \equiv A \\
A \vee \text{false} \equiv A & A \wedge \text{false} \equiv \text{false} \\
A \rightarrow \text{true} \equiv \text{true} & \text{true} \rightarrow A \equiv A \\
A \rightarrow \text{false} \equiv \neg A & \text{false} \rightarrow A \equiv \text{true} \\
A \leftrightarrow \text{true} \equiv A & A \oplus \text{true} \equiv \neg A \\
A \leftrightarrow \text{false} \equiv \neg A & A \oplus \text{false} \equiv A
\end{array}$$

عملوندهای همسان (**Identical Operands**)

فروپاشی (collapse) همچنین وقتی رخ می‌دهد که هر دو عملوند یکسان باشند یا یکی نقیض دیگری:

$$\begin{array}{rr}
A \equiv \neg\neg A, & \\
A \equiv A \wedge A & A \equiv A \vee A, \\
A \vee \neg A \equiv \text{true} & A \wedge \neg A \equiv \text{false}, \\
A \rightarrow A \equiv \text{true}, & \\
A \leftrightarrow A \equiv \text{true} & A \oplus A \equiv \text{false}, \\
\neg A \equiv A \uparrow A & \neg A \equiv A \downarrow A.
\end{array}$$

جابجایی، پیمایش‌پذیری، و توزیع‌پذیری

اپراتورهای دودویی بولی—به‌جز «تضمین» ($\rightarrow$)—هم جابجایی‌پذیرند و هم پیمایش‌پذیر:

$$\begin{array}{cc}
A \vee B \equiv B \vee A & A \wedge B \equiv B \wedge A, \\
A \leftrightarrow B \equiv B \leftrightarrow A & A \oplus B \equiv B \oplus A, \\
A \uparrow B \equiv B \uparrow A & A \downarrow B \equiv B \downarrow A.
\end{array}$$

با ورود نقیض، جهت یک تضمین می‌تواند وارونه شود:

$$A \rightarrow B \;\equiv\; \neg B \rightarrow \neg A.$$

فرمول $\neg B \rightarrow \neg A$ را مخالف‌مقدم (contrapositive) $A \rightarrow B$ می‌نامند.

جمع‌گزاره ($\vee$)، ضرب‌گزاره ($\wedge$)، معادل ($\leftrightarrow$) و نامعادل ($\oplus$) پیمایش‌پذیرند:

$$A \vee (B \vee C) \equiv (A \vee B) \vee C \qquad A \wedge (B \wedge C) \equiv (A \wedge B) \wedge C,$$
$$A \leftrightarrow (B \leftrightarrow C) \equiv (A \leftrightarrow B) \leftrightarrow C \qquad A \oplus (B \oplus C) \equiv (A \oplus B) \oplus C.$$

ولی تضمین ($\rightarrow$)، nor ($\downarrow$) و nand ($\uparrow$) پیمایش‌پذیر نیستند.
همچنین، جمع‌گزاره و ضرب‌گزاره بر یکدیگر توزیع‌پذیرند:

$$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C),$$
$$A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C).$$

تعریف یک عملگر به‌واسطهٔ عملگر دیگر

در اثبات قضایا با استقرا ساختاری، گام استقرایی باید برای هر عملگر دودویی جداگانه انجام شود. این گام‌ها ساده‌تر می‌شوند اگر بتوان عملگرهای خاص را با جایگزینی زیرفرمول‌هایی حذف کرد و فقط از مجموعه‌ای از عملگرهای پایه بهره گرفت. مثلاً معادل ($\leftrightarrow$) را می‌توان با ترکیب تضمین و ضرب‌گزاره تعریف کرد.
همچنین، برخی الگوریتم‌ها برای تبدیل فرمول به شکل نرمال، نیازمند حذف برخی عملگرها هستند. فهرست معادل‌هایی که برای این کار کاربرد دارند:

$$A \leftrightarrow B \equiv (A \rightarrow B) \wedge (B \rightarrow A) \qquad A \oplus B \equiv \neg(A \rightarrow B) \vee \neg(B \rightarrow A),$$
$$A \rightarrow B \equiv \neg A \vee B \qquad A \rightarrow B \equiv \neg(A \wedge \neg B),$$
$$A \vee B \equiv \neg(\neg A \wedge \neg B) \qquad A \wedge B \equiv \neg(\neg A \vee \neg B),$$
$$A \vee B \equiv \neg A \rightarrow B \qquad A \wedge B \equiv \neg(A \rightarrow \neg B).$$

تعریف ضرب‌گزاره برحسب جمع‌گزاره و نقیض و بالعکس را قضایای دمورگان (De Morgan's laws) می‌نامند.

2.4 مجموعه‌ای از عملگرهای بولی

از دوران ابتدایی مدرسه به ما آموخته‌اند که چهار عملگر پایه در حساب عبارت‌اند از: جمع، تفریق، ضرب و تقسیم. بعدها با عملگرهای اضافی‌تری مانند مدولو (مانده) و قدر مطلق آشنا می‌شویم. از سوی دیگر، از دید نظری می‌دانیم ضرب و تقسیم در حقیقت زائد هستند، زیرا می‌توان آن‌ها را برحسب جمع و تفریق تعریف کرد.
در این بخش، به دو پرسش می‌پردازیم:

۱. چه عملگرهای بولی وجود دارند؟
۲. چه مجموعه‌ای از این عملگرها «کافی» است، به این معنا که بتوان همهٔ عملگرهای دیگر را تنها با استفاده از عملگرهای آن مجموعه تعریف کرد؟

2.4.1 عملگرهای بولی یگانی و دودویی

از آنجایی که تنها دو مقدار بولی T و F وجود دارد، تعداد عملگرهای با n ورودی برابر است با 2^{2^n}، زیرا برای هر یک از n گزارهٔ ورودی می‌توان یکی از دو مقدار T یا F را انتخاب کرد (در مجموع 2^n ترکیب ممکن) و برای هر یک از این ترکیب‌ها می‌توان مقدار خروجی را T یا F قرار داد. در اینجا خود را به عملگرهای تک‌جایی (یک‌جایی) و دوجایی (دوجایی) محدود می‌کنیم.

عملگرهای تک‌جایی جدول زیر چهار عملگر تک‌جایی ممکن $\circ_1, \ldots, \circ_4$ را نشان می‌دهد. ستون اول مقدار ورودی x را و ستون‌های بعدی مقدار $\circ_n(x)$ را می‌دهند:

x	$\circ_1$	$\circ_2$	$\circ_3$	$\circ_4$
T	T	T	F	F
F	T	F	T	F

عملگرهای دوجایی شکل 2.5 عملگرهای دوجایی $\circ_1, \ldots, \circ_{16}$ را بر اساس مقادیر ورودی (x_1, x_2) فهرست می‌کند: از چهار عملگر تک‌جایی، سه تای آن‌ها بدیهی هستند: $\circ_1$ و $\circ_4$ عملگرهای ثابتند (همیشه

x_1	x_2	$\circ_1$	$\circ_2$	$\circ_3$	$\circ_4$	$\circ_5$	$\circ_6$	$\circ_7$	$\circ_8$
T	T	T	T	T	T	T	T	T	T
T	F	T	T	T	T	F	F	F	F
F	T	T	T	F	F	T	T	F	F
F	F	T	F	T	F	T	F	T	F

x_1	x_2	$\circ_9$	$\circ_{10}$	$\circ_{11}$	$\circ_{12}$	$\circ_{13}$	$\circ_{14}$	$\circ_{15}$	$\circ_{16}$
T	T	F	F	F	F	F	F	F	F
T	F	T	T	T	T	F	F	F	F
F	T	T	T	F	F	T	T	F	F
F	F	T	F	T	F	T	F	T	F

شکل 2.5: عملگر های بولی دو جایی

T یا همیشه F) و $\circ_2$ عملگر هویت است که ورودی را بدون تغییر برمی‌گرداند. تنها عملگر غیربدیهی تک‌جایی $\circ_3$ است که نقیض (negation) را پیاده می‌کند. برای عملگرهای دوجایی ($2^{2^2} = 16$ حالت) نیز چند عملگر بدیهی وجود دارد:

- $\circ_1$ و $\circ_{16}$ عملگرهای ثابت،
- $\circ_4$ و $\circ_6$ عملگرهای projection (مقدار خروجی تنها به یکی از ورودی‌ها بستگی دارد)،
- $\circ_{11}$ و $\circ_{13}$ منفیِ همین عملگرهای projection هستند.

جدول زیر تطابق نمادهای $\circ_n$ را با عملگرهای شناخته‌شده در تعریف 2.1 نشان می‌دهد. نمادهای ستون راست منفیِ نمادهای هم‌ان‌ردیف در ستون چپ هستند.

$\circ_n$	نام عملگر	نماد	$\circ_m$	نام عملگر	نماد
$\circ_2$	جمع‌گزاره	$\vee$	$\circ_{15}$	nor	$\downarrow$
$\circ_8$	ضرب‌گزاره	$\wedge$	$\circ_9$	nand	$\uparrow$
$\circ_5$	تضمین (implication)	$\rightarrow$	$\circ_{10}$	xor (نامعادل)	$\oplus$
$\circ_7$	معادل	$\leftrightarrow$	—	—	—

عملگر $\circ_{12}$ منفیِ تضمین است و معمولاً استفاده نمی‌شود. «تضمین معکوس» $\circ_3$ در زبان‌های منطق برنامه‌نویسی (فصل ۱۱) کاربرد دارد؛ منفیِ آن ($\circ_{14}$) نیز معمولاً به‌کار نمی‌رود.

2.4.2 مجموعه‌های کافی از عملگرها

تعریف **2.35**. یک عملگر دودویی $\circ$ از مجموعهٔ عملگرها $\{\circ_1, \ldots, \circ_n\}$ تعریف می‌شود اگر و تنها اگر معادل منطقی

$$A_1 \ \circ \ A_2 \ \equiv \ A$$

وجود داشته باشد، جایی که A فرمولی است ساخته‌شده از وقوع‌های A_1 و A_2 با استفاده از عملگرهای $\{\circ_1, \ldots, \circ_n\}$. عملگر یک‌جایی $\neg$ نیز زمانی تعریف‌شده محسوب می‌شود که معادل منطقی

$$\neg A_1 \ \equiv \ A$$

وجود داشته باشد، با این توضیح که A از وقوع‌های A_1 و عملگرهای مجموعه ساخته شده است.

قضیه **2.36**. می‌توان همهٔ عملگرهای بولی

$$\vee, \ \wedge, \ \rightarrow, \ \leftrightarrow, \ \oplus, \ \uparrow, \ \downarrow$$

را تنها از طریق $\neg$ و یکی از عملگرهای $\vee$، $\wedge$، یا $\rightarrow$ تعریف کرد.

اثبات. اثبات با استفاده از معادل‌های منطقیِ فهرست‌شده در زیربخش 2.3.3 به‌دست می‌آید. دو عملگر nand ($\uparrow$) و nor ($\downarrow$) به‌ترتیب منفیِ ضرب‌گزاره و جمع‌گزاره هستند. معادل ($\leftrightarrow$) را می‌توان از تضمین ($\rightarrow$) و ضرب‌گزاره ($\wedge$) تعریف کرد و نامعادل ($\oplus$) را نیز از همین عملگرها و نقیض تعریف نمود. بنابراین تنها نیاز به $\rightarrow$، $\vee$، $\wedge$ داریم، ولی هر یک از این سه عملگر را نیز می‌توان با استفاده از دیگری‌ها و نقیض تعریف کرد (مطابق معادل‌های صفحهٔ ۲۶).
جالب آن است که می‌توان همهٔ عملگرهای بولی را تنها از طریق nand یا تنها از طریق nor تعریف کرد.
برای مثال، از معادل

$$\neg A \ \equiv \ A \uparrow A$$

برای تعریف نقیض از nand استفاده می‌کنیم. آنگاه برای ضرب‌گزاره داریم:

$$\begin{aligned}
(A \uparrow B) \uparrow (A \uparrow B) \ &\equiv \ \neg\big((A \uparrow B) \wedge (A \uparrow B)\big) && (\text{تعریف } \uparrow) \\
&\equiv \ \neg\big(A \uparrow B\big) && (\text{ایدمپوتنسی } X \wedge X \equiv X) \\
&\equiv \ \neg\neg(A \wedge B) && (\text{تعریف } \uparrow) \\
&\equiv \ A \wedge B && (\text{دوگانه‌نقیض}).
\end{aligned}$$

پس از تعریف نقیض و ضرب‌گزاره از nand، می‌توان همهٔ عملگرهای دیگر را تعریف کرد. به‌طور مشابه، nor نیز مجموعهٔ کافی‌ای از عملگرها را تشکیل می‌دهد. در واقع، می‌توان ثابت کرد که فقط nand و nor این خاصیت را دارند. $\square$

قضیه **2.37**. فرض کنید $\circ$ یک عملگر دودویی باشد که بتوان با آن نقیض و همهٔ عملگرهای دودویی دیگر را تعریف کرد. آنگاه $\circ$ یا nand است یا nor.

طرح اثبات (خلاصه).

۱. از آنجا که $\circ$ باید نقیض را تعریف کند، باید معادلی از شکل

$$\neg A \ \equiv \ A \circ \cdots \circ A$$

وجود داشته باشد.

۲. هر عملگر دودویی op باید معادلی از صورت

$$A_1 \text{ op } A_2 \equiv B_1 \circ \cdots \circ B_n$$

داشته باشد، جایی که هر B_i یا A_1 یا A_2 است (با پرانتزگذاری مناسب).

۳. با در نظر گرفتن یک تعبیر $\mathscr{I}$ که $v_{\mathscr{I}}(A) = T$ و استفاده از استقراء روی تعداد وقوع‌های $\circ$، نشان می‌دهیم که

$$v_{\mathscr{I}}(A_1 \circ A_2) = F \quad \text{وقتی} \quad v_{\mathscr{I}}(A_1) = T,\ v_{\mathscr{I}}(A_2) = T.$$

به‌طور مشابه برای $v_{\mathscr{I}}(A) = F$ نتیجه می‌شود $v_{\mathscr{I}}(A_1 \circ A_2) = T$.

۴. بنابراین، آزادی عمل $\circ$ فقط در حالتی است که دو عملوند مقادیر متفاوت داشته باشند:

A_1	A_2	$A_1 \circ A_2$
T	T	F
T	F	T یا F
F	T	T یا F
F	F	T

اگر $\circ$ برای دو سطر وسط مقدار T را انتخاب کند، $\circ$ همان nand است، و اگر F را انتخاب کند، همان nor خواهد بود.

۵. تنها حالت باقیمانده این است که $\circ$ برای این دو سطر مقادیر متفاوت بدهد. با استقراء نشان دهید که در این صورت تنها می‌توان projection یا negated projection ساخت:

$$B_1 \circ \cdots \circ B_n \equiv \neg \cdots \neg B_i$$

برای بعضی i و صفر یا چند نقیض.

2.5 Satisfiability, Validity and Consequence

ما اکنون مفاهیم بنیادین معناشناسی (سمانتیک) فرمول‌ها را تعریف می‌کنیم:

تعریف **2.38**. فرض کنید $A \in \mathscr{F}$.

- A قابل ارضا (satisfiable) است اگر و تنها اگر برای برخی تعبیر $\mathscr{I}$ داشته باشیم

$$v_{\mathscr{I}}(A) = T.$$

تعبیر $\mathscr{I}$ که فرمول A را ارضا می‌کند، مدل A نامیده می‌شود.

- A صادق همگانی (valid)، که با $\models A$ نشان داده می‌شود، است اگر و تنها اگر برای همهٔ تعبیرها $\mathscr{I}$ داشته باشیم

$$v_{\mathscr{I}}(A) = T.$$

یک فرمول صادق همگانی در منطق گزاره‌ای را تناقض‌ناپذیر یا تاتولوژی نیز می‌نامند.

- A ناقابل ارضا (unsatisfiable) است اگر قابل ارضا نباشد، یعنی اگر برای همهٔ تعبیرها $\mathscr{I}$ داشته باشیم
$$v_{\mathscr{I}}(A) = F.$$

- A قابل تکذیب (falsifiable)، که با $\not\models A$ نشان داده می‌شود، است اگر ناصادق همگانی باشد، یعنی اگر برای برخی تعبیر $\mathscr{I}$ داشته باشیم
$$v_{\mathscr{I}}(A) = F.$$

این چهار مفهوم معناشناختی در شکل 2.6 نشان داده شده‌اند.

این مفاهیم به‌طور نزدیکی با یکدیگر مرتبط‌اند:

قضیه **2.39**. برای هر $A \in \mathscr{F}$:

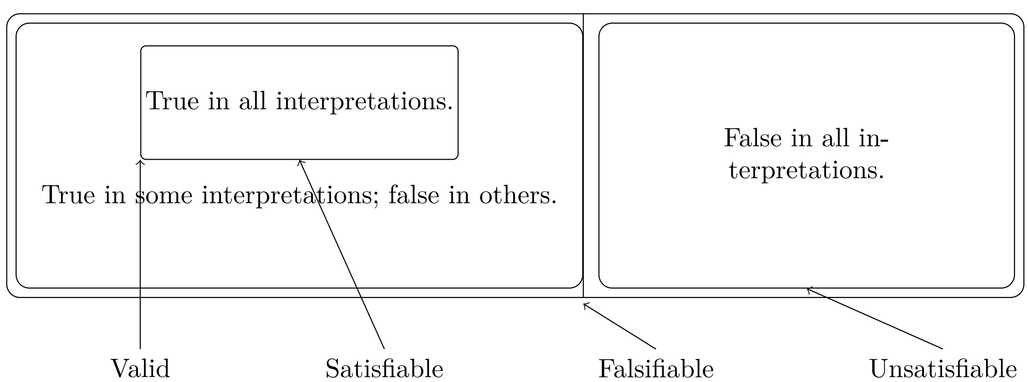
۱. A صادق همگانی است اگر و تنها اگر $\neg A$ ناقابل ارضا باشد.

۲. A قابل ارضا است اگر و تنها اگر $\neg A$ قابل تکذیب باشد.

اثبات. بگذارید $\mathscr{I}$ یک تعبیر دلخواه باشد. از تعریف ارزش صدق نقیض، داریم

$$v_{\mathscr{I}}(A) = T \iff v_{\mathscr{I}}(\neg A) = F.$$

از آنجا که $\mathscr{I}$ دلخواه بود، نتیجه می‌شود A در همهٔ تعبیرها درست است اگر و تنها اگر $\neg A$ در همهٔ تعبیرها نادرست باشد، یعنی $\neg A$ ناقابل ارضا باشد.
اگر A قابل ارضا باشد، آنگاه برای برخی تعبیر $\mathscr{I}$ داریم $v_{\mathscr{I}}(A) = T$. با تعریف نقیض، $v_{\mathscr{I}}(\neg A) = F$، پس $\neg A$ قابل تکذیب است. بالعکس اگر $v_{\mathscr{I}}(\neg A) = F$ برای برخی $\mathscr{I}$، آنگاه $v_{\mathscr{I}}(A) = T$، یعنی A قابل ارضا است. □



شکل 2.6: formulas of validity and Satisfiability

2.5.1 Logic Propositional in Procedures Decision

تعریف **2.40**. فرض کنید $\mathscr{U} \subseteq \mathscr{F}$ مجموعه‌ای از فرمول‌ها باشد. الگوریتمی یک رویهٔ تصمیم برای $\mathscr{U}$ است اگر برای هر فرمول دلخواه $A \in \mathscr{F}$ پایان یابد و پاسخ «بله» را بازگرداند اگر $A \in \mathscr{U}$ و پاسخ «خیر» را اگر $A \notin \mathscr{U}$.

اگر $\mathscr{U}$ مجموعهٔ فرمول‌های قابل ارضا باشد، رویهٔ تصمیم برای $\mathscr{U}$ را رویهٔ تصمیم برای ارضاپذیری (decision procedure for satisfiability) می‌نامیم

و به‌طور مشابه برای همگانی‌صادق‌بودن.

با توجه به قضیهٔ 2.39، می‌توان از رویهٔ تصمیم برای ارضاپذیری به‌عنوان رویهٔ تصمیم برای همگانی‌صادق‌بودن استفاده کرد. برای تصمیم‌گیری دربارهٔ اینکه آیا فرمول A همگانی‌صادق است یا نه، کافی است رویهٔ تصمیم برای ارضاپذیری را روی $\neg A$ اجرا کنیم. اگر گزارش دهد $\neg A$ قابل ارضا است، آنگاه A همگانی‌صادق نیست؛ و اگر گزارش دهد $\neg A$ قابل ارضا نیست، آنگاه A همگانی‌صادق است. چنین رویه‌ای را رویهٔ ابطال (refutation procedure) می‌نامند، زیرا به‌جای اثبات مستقیم اینکه فرمول همیشه درست است، تنها به جستجوی مثال نقض می‌پردازد که کارآمدی بیشتری دارد.

وجود رویهٔ تصمیم برای ارضاپذیری در منطق گزاره‌ای بدیهی است، زیرا می‌توانیم برای هر فرمول یک جدول ارزش صدق بسازیم. جدول ارزش صدق در مثال 2.21 نشان می‌دهد که $p \to q$ قابل ارضا ولی ناخودهمگانی‌صادق است؛ در مثال 2.22 نشان داده شد که

$$(p \to q) \;\leftrightarrow\; (\neg q \to \neg p)$$

همگانی‌صادق است. مثال زیر یک فرمول ناتوان از ارضا را نمایش می‌دهد.

مثال **2.41**. فرمول

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

ناتوان از ارضا است، زیرا در همهٔ سطرهای جدول ارزش صدق آن، مقدار F به‌دست می‌آید:

p	q	$p \vee q$	$\neg p$	$\neg q$	$(p \vee q) \wedge \neg p \wedge \neg q$
T	T	T	F	F	F
T	F	T	F	T	F
F	T	T	T	F	F
F	F	F	T	T	F

روش جدول ارزش صدق یک رویهٔ تصمیم بسیار ناکارآمد است؛ زیرا برای فرمولی با n اتم متمایز، باید آن را برای هر یک از 2^n تعبیر ممکن ارزیابی کنیم. در فصول بعدی، رویه‌های تصمیم کارآمدتری برای ارضاپذیری ارائه خواهد شد، اگرچه بسیار بعید است رویهٔ تصمیمی وجود داشته باشد که برای همهٔ فرمول‌ها به‌طور کارآمد اجرا شود (به بخش 6.7 مراجعه کنید).

2.5.2 ارضاپذیری یک مجموعه از فرمول‌ها

مفهوم ارضاپذیری را می‌توان به مجموعه‌ای از فرمول‌ها نیز تعمیم داد.

تعریف **2.42**. یک مجموعهٔ فرمول‌ها

$$U = \{A_1, A_2, \dots\}$$

هنگامی (به‌طور هم‌زمان) ارضاپذیر است که تعبیری $\mathscr{I}$ وجود داشته باشد به‌گونه‌ای که برای همهٔ اندیس‌ها i داشته باشیم:

$$v_{\mathscr{I}}(A_i) = T.$$

چنین تعبیری $\mathscr{I}$ را مدل U می‌نامیم.

مجموعهٔ U را ناتوان از ارضا (unsatisfiable) می‌نامیم هرگاه برای هر تعبیر $\mathscr{I}$، دست‌کم یک i وجود داشته باشد که:

$$v_{\mathscr{I}}(A_i) = F.$$

مثال **2.43**. مجموعهٔ

$$U_1 = \{p,\ \neg p \vee q,\ q \wedge r\}$$

ارضاپذیر است، زیرا تعبیری که به هر سه اتم مقدار T اختصاص دهد، همهٔ فرمول‌ها را ارضا می‌کند. اما مجموعهٔ

$$U_2 = \{p,\ \neg p \vee q,\ \neg p\}$$

ناتوان از ارضا است. هر فرمول از این مجموعه به‌تنهایی قابل ارضا است، ولی هم‌زمان با یکدیگر ارضاپذیر نیستند.

اثبات قضایای ابتدایی زیر به‌عنوان تمرین باقی گذاشته شده‌اند:

- قضیه **2.44**: اگر U ارضاپذیر باشد، آنگاه برای هر i، مجموعهٔ

  $$U - \{A_i\}$$

  نیز ارضاپذیر است.

- قضیه **2.45**: اگر U ارضاپذیر باشد و B فرمولی صادق همگانی (valid) باشد، آنگاه

  $$U \cup \{B\}$$

  نیز ارضاپذیر است.

- قضیه **2.46**: اگر U ناتوان از ارضا باشد، آنگاه برای هر فرمول B، مجموعهٔ

  $$U \cup \{B\}$$

  نیز ناتوان از ارضا است.

- قضیه **2.47**: اگر U ناتوان از ارضا باشد و برای بعضی i، فرمول A_i صادق همگانی باشد، آنگاه

  $$U - \{A_i\}$$

  نیز ناتوان از ارضا خواهد بود.

2.5.3 نتیجهٔ منطقی (Logical Consequence)

تعریف **2.48**. فرض کنید U مجموعه‌ای از فرمول‌ها و A یک فرمول باشد. فرمول A نتیجهٔ منطقی U است، اگر و تنها اگر هر مدل از U، مدل A نیز باشد. این مفهوم با نماد

$$U \models A$$

نشان داده می‌شود.

نیازی نیست که A در همهٔ تعبیرهای ممکن صادق باشد، بلکه کافی است تنها در تمام تعبیرهایی که U را ارضا می‌کنند (یعنی همهٔ فرمول‌های U را درست می‌سازند) صادق باشد. اگر U تهی باشد، نتیجهٔ منطقی همان همگانی‌صادق بودن (validity) است.

مثال **2.49**. بگذارید

$$A = (p \vee r) \wedge (\neg q \vee \neg r).$$

آنگاه

$$\{p, \neg q\} \models A.$$

زیرا A در همهٔ تعبیرهایی که در آن‌ها $p = T$ و $q = F$ باشد، درست است. اما A همگانی‌صادق نیست، زیرا در تعبیری مانند

$$\mathscr{I}' : \quad \mathscr{I}'(p) = F, \quad \mathscr{I}'(q) = T, \quad \mathscr{I}'(r) = T$$

نادرست است.

نکته‌ای که پیش‌تر دربارهٔ تفاوت بین $\leftrightarrow$ و $\equiv$ گفته شد، دربارهٔ $\rightarrow$ و $\models$ نیز صدق می‌کند:

- $\rightarrow$ یک عملگر در زبان محتوایی منطق گزاره‌ای است،
- $\models$ یک نماد مفهومی در زبان مدرکی (ابردستگاه) است.

اما، همچون معادل منطقی، این دو مفهوم به‌هم مرتبط‌اند:

قضیه **2.50**. اگر $U = \{A_1, A_2, \dots\}$، آنگاه:

$$U \models A \quad \text{اگر و تنها اگر} \quad \models \left(\bigwedge_i A_i\right) \rightarrow A.$$

تعریف **2.51**. عبارت

$$\bigwedge_{i=1}^{n} A_i$$

یعنی $A_1 \wedge A_2 \wedge \cdots \wedge A_n$. نماد $\bigwedge_i A_i$ هنگامی استفاده می‌شود که بازهٔ اندیس‌ها از متن قابل فهم باشد، یا اگر مجموعه فرمول‌ها نامتناهی باشد. برای جمع‌گزاره نیز از نماد مشابه $\bigvee_i A_i$ استفاده می‌شود.

مثال **2.52**. از مثال 2.49 داریم:

$$\{p, \neg q\} \models (p \vee r) \wedge (\neg q \vee \neg r).$$

پس طبق قضیه 2.50:

$$\models (p \wedge \neg q) \rightarrow \big((p \vee r) \wedge (\neg q \vee \neg r)\big).$$

اثبات قضیهٔ 2.50 و همچنین دو قضیهٔ زیر به‌عنوان تمرین باقی گذاشته شده‌اند:

- قضیه **2.53**. اگر $U \models A$، آنگاه برای هر فرمول B،

$$U \cup \{B\} \models A.$$

- قضیه **2.54**. اگر $U \models A$ و فرمول B همگانی‌صادق باشد، آنگاه

$$U - \{B\} \models A.$$

2.5.4 نظریه‌ها

نتیجهٔ منطقی، مفهوم مرکزی در بنیان‌های ریاضیات است. فرمول‌های منطقی همواره‌صادق مانند

$$p \vee q \leftrightarrow q \vee p$$

اهمیت چندانی در ریاضیات ندارند. آنچه جالب‌تر است، فرض گرفتن مجموعه‌ای از فرمول‌ها به‌عنوان صادق و سپس بررسی پیامدهای منطقی آن‌ها است.
برای مثال، اقلیدس پنج اصل دربارهٔ هندسه را فرض گرفت و مجموعهٔ گسترده‌ای از نتایج منطقی را از آن‌ها نتیجه گرفت. تعریف صوری یک «نظریهٔ ریاضی» به‌صورت زیر است:

تعریف **2.55**. بگذارید $\mathscr{T}$ یک مجموعه از فرمول‌ها باشد. T تحت نتیجهٔ منطقی بسته است هرگاه برای هر فرمول A، اگر

$$\mathscr{T} \models A,$$

آنگاه

$$A \in \mathscr{T}.$$

مجموعه‌ای از فرمول‌ها که تحت نتیجهٔ منطقی بسته باشد، یک نظریه (theory) نامیده می‌شود. فرمول‌های عضو $\mathscr{T}$، قضیه‌های (theorems) نظریه نامیده می‌شوند.

نظریه‌ها با انتخاب مجموعه‌ای از فرمول‌ها به‌عنوان اصول موضوع (axioms) ساخته می‌شوند و سپس نتایج منطقی آن اصول به نظریه افزوده می‌شوند.

تعریف **2.56**. اگر $\mathscr{T}$ یک نظریه باشد، آنگاه گفته می‌شود $\mathscr{T}$ قابل اصل‌گذاری (axiomatizable) است هرگاه مجموعه‌ای از فرمول‌ها مانند U وجود داشته باشد به‌طوری‌که

$$\mathscr{T} = \{A \mid U \models A\}.$$

مجموعهٔ U اصول موضوع نظریهٔ $\mathscr{T}$ هستند. اگر U متناهی باشد، آنگاه گفته می‌شود که $\mathscr{T}$ قابل اصل‌گذاری به‌صورت متناهی است.

برای مثال، نظریهٔ حساب (Arithmetic) قابل اصل‌گذاری است: مجموعه‌ای از اصول توسط پئانو (Peano) ارائه شده که نتایج منطقی آن‌ها، قضیه‌های حساب هستند. اما نظریهٔ حساب قابل اصل‌گذاری به‌صورت متناهی نیست، چرا که اصل استقرا (induction axiom) به‌صورت یک طرح اصل است که برای هر خاصیت در حساب، نمونه‌ای دارد و نمی‌توان آن را با تنها یک اصل بیان کرد.

2.6 جدول معنایی

روش جدول معنایی یک رویهٔ تصمیم‌گیری کارآمد برای بررسی ارضاپذیری (و به‌طور دوگانه، همگانی‌صدقی) در منطق گزاره‌ای است. در فصل بعد، از جدول معنایی به‌طور گسترده برای اثبات قضایای مهم در مورد سیستم‌های استنتاجی استفاده خواهیم کرد.
اصل بنیادین جدول معنایی بسیار ساده است: با تجزیهٔ فرمول به مجموعه‌هایی از اتم‌ها و نقیض آن‌ها، در پی یافتن مدلی (تعبیری ارضاکننده) برای فرمول هستیم.
بررسی امکان ارضا برای هر یک از این مجموعه‌ها آسان است: یک مجموعه از اتم‌ها و نقیض اتم‌ها ارضاپذیر است اگر و تنها اگر شامل اتمی و نقیض همان اتم نباشد. در نتیجه، فرمول اولیه ارضاپذیر است اگر حداقل یکی از این مجموعه‌ها ارضاپذیر باشد.
اکنون با چند تعریف آغاز می‌کنیم و سپس ارضاپذیری دو فرمول را تحلیل می‌کنیم تا انگیزه‌ای برای ساختار جدول معنایی فراهم شود.

2.6.1 تجزیهٔ فرمول‌ها به مجموعه‌ای از لفظ‌ها

تعریف **2.57**. یک لفظ (literal)، یا اتم است یا نفی یک اتم.

- یک اتم، لفظِ مثبت نامیده می‌شود.
- نفی یک اتم، لفظِ منفی نام دارد.

برای هر اتم p، مجموعهٔ $\{p,\ \neg p\}$ یک زوج متمم از لفظ‌ها تشکیل می‌دهد. برای هر فرمول A، مجموعهٔ $\{A,\ \neg A\}$ نیز یک زوج متمم از فرمول‌ها است. فرمول A متمم $\neg A$ و بالعکس است.

مثال **2.58**. در مجموعهٔ لفظ‌ها

$$\{\neg p,\ q,\ r,\ \neg r\}$$

- q و r لفظِ مثبت هستند،
- $\neg p$ و $\neg r$ لفظِ منفی هستند.

این مجموعه شامل زوج متمم $\{r, \neg r\}$ است.

مثال **2.59**. فرمول زیر را تحلیل می‌کنیم تا ارضاپذیری آن را در تعبیر دلخواه $\mathscr{I}$ بررسی کنیم:

$$A = p \wedge (\neg q \vee \neg p).$$

با استفاده از قواعد بازگشتی ارزیابی صدق:

$$\begin{aligned} v_{\mathscr{I}}(A) = T &\iff v_{\mathscr{I}}(p) = T \ \wedge\ v_{\mathscr{I}}(\neg q \vee \neg p) = T, \\ v_{\mathscr{I}}(\neg q \vee \neg p) = T &\iff v_{\mathscr{I}}(\neg q) = T \ \vee\ v_{\mathscr{I}}(\neg p) = T. \end{aligned}$$

در نتیجه، $v_{\mathscr{I}}(A) = T$ اگر و تنها اگر یکی از دو حالت زیر برقرار باشد:

۱. $v_{\mathscr{I}}(p) = T$ و $v_{\mathscr{I}}(\neg q) = T$

۲. $v_{\mathscr{I}}(p) = T$ و $v_{\mathscr{I}}(\neg p) = T$

پس مسئلهٔ ارضاپذیری A به بررسی ارضاپذیری دو مجموعهٔ لفظ‌ها کاهش می‌یابد.

قضیه **2.60**. مجموعه‌ای از لفظها ارضاپذیر است اگر و تنها اگر شامل هیچ زوج متممی از لفظها نباشد.

اثبات. فرض کنید L مجموعه‌ای از لفظها باشد که هیچ زوج متممی در آن وجود ندارد. تعبیر $\mathscr{I}$ را به‌صورت زیر تعریف می‌کنیم:

$$\mathscr{I}(p) = \begin{cases} T, & \text{اگر}\, p \in L, \\ F, & \text{اگر}\, \neg p \in L. \end{cases}$$

این تعریف سازگار و کامل است، زیرا در L هیچ اتمی همراه نقیضش حضور ندارد. هر لفظِ موجود در L در این تعبیر مقدار T می‌گیرد؛ بنابراین L ارضاپذیر است.
حال اگر $\{p, \neg p\} \subseteq L$، در هر تعبیر ممکن یا $v_{\mathscr{I}}(p) = F$ یا $v_{\mathscr{I}}(\neg p) = F$ است، پس L ارضاپذیر نخواهد بود. $\square$

مثال **2.61**. ادامهٔ مثال 2.59: برای $A = p \wedge (\neg q \vee \neg p)$، دو مجموعه‌ای که باید بررسی شوند:

$$\{p, \neg p\} \quad \longrightarrow \quad \text{دارای زوج متمم} \;\rightarrow\; \text{ناتوان از ارضا},$$

$$\{p, \neg q\} \quad \longrightarrow \quad \text{فاقد زوج متمم} \;\rightarrow\; \text{ارضاپذیر}.$$

بنابراین فقط مجموعهٔ دوم ارضاپذیر است و از قضیهٔ 2.60 تعبیری به‌دست می‌آید که در آن

$$\mathscr{I}(p) = T, \quad \mathscr{I}(q) = F.$$

مثال **2.62**. اگر فرمول ناتوان از ارضا باشد، مثلاً:

$$B = (p \vee q) \wedge (\neg p \wedge \neg q),$$

آنگاه:

۱. برای ارضا شدن B باید $v_{\mathscr{I}}(p \vee q) = T$ و $v_{\mathscr{I}}(\neg p \wedge \neg q) = T$.

۲. تجزیهٔ ترکیب عطفی و ترکیب فصلی نشان می‌دهد که دو حالت ممکن به مجموعه‌های لفظها می‌انجامد:
$$\{p, \neg p, \neg q\}, \quad \{q, \neg p, \neg q\}.$$

۳. هر دو شامل زوج متمم‌اند؛ بنابراین ناتوان از ارضا هستند (قضیهٔ 2.60).

نتیجه می‌گیریم که هیچ مدلی برای B وجود ندارد، یعنی

$$B \text{ناتوان از ارضا است.}$$

2.6.2 ساخت جدول معنایی (Construction of Semantic Tableaux)

تجزیهٔ فرمول به مجموعه‌ای از لفظها در قالب متنی دشوار است. در روش جدول معنایی، مجموعه‌های فرمول برچسبِ گره‌های یک درخت را تشکیل می‌دهند، به‌طوری که هر مسیر در درخت نمایانگر فرمول‌هایی است که باید در یک تعبیر ممکن صدق‌پذیر شوند.

- فرمول اولیه برچسبِ ریشهٔ درخت است.

$$A$$

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

$$p, \neg q \vee \neg p$$

$$\begin{array}{cc} p, \neg q & p, \neg p \\ \odot & \times \end{array}$$

$$B$$

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

$$p \vee q, \neg p \wedge \neg q$$

$$p \vee q, \neg p, \neg q$$

$$\begin{array}{cc} p, \neg p, \neg q & q, \neg p, \neg q \\ \times & \times \end{array}$$

شکل 2.7: درخت صدق

- هر گره، بسته به نوع فرمول برچسب‌خورده، یک یا دو فرزند دارد.
- برگ‌ها با مجموعه‌ای از لفظها برچسب می‌خورند.
- برگ‌هایی که شامل یک زوج متمم از لفظها باشند با «$\times$» (بسته) و برگ‌هایی که فاقد زوج متمم هستند با $\odot$ (باز) علامت‌گذاری می‌شوند.

شکل 2.7 جدول‌های معنایی مثال‌های پیشین را نشان می‌دهد. جدول زیر نمونهٔ دیگری از جدول معنایی برای فرمول

$$B = (p \vee q) \; \wedge \; (\neg p \wedge \neg q)$$

را نمایش می‌دهد که ابتدا برای $p \vee q$ منشعب شده و سپس $\neg p \wedge \neg q$ را پردازش می‌کند. واضح است که اگر همگرایی ($\wedge$) را پیش از جمع‌گزاره ($\vee$) بگشاییم، تعداد گره‌ها کمتر خواهد بود و در نتیجه کارآمدتر است.

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

$$p \vee q, \neg q \wedge \neg p$$

$$\begin{array}{cc} p, \neg p \wedge \neg q & q, \neg p \wedge \neg q \\ | & | \\ p, \neg p, \neg q & q, \neg p, \neg q \\ \times & \times \end{array}$$

برای ساده‌سازی و ایجاز، فرمول‌ها را بر اساس اپراتور اصلی شان به دو دستهٔ αـ فرمول و βـ فرمول تقسیم می‌کنیم (شکل 2.8):
برای مثال، $p \wedge q$ یک αـ فرمول است (زیرا هر دو p و q باید صدق کنند)، و $\neg(p \wedge q)$ یک βـ فرمول است (معادل $\neg p \vee \neg q$).

زیرفرمول‌ها	شکل عام	نوع
$\alpha_1 = A_1,\ \alpha_2 = A_2$	$A_1 \wedge A_2$	α
$\alpha_1 = A_1$	$\neg(\neg A_1)$	α
$\alpha_1 = \neg A_1,\ \alpha_2 = \neg A_2$	$\neg(A_1 \vee A_2)$	α
$\alpha_1 = A_1,\ \alpha_2 = \neg A_2$	$\neg(A_1 \to A_2)$	α
$\beta_1 = A_1,\ \beta_2 = A_2$	$A_1 \vee A_2$	β
$\beta_1 = \neg A_1,\ \beta_2 = A_2$	$A_1 \to A_2$	β
$\beta_1 = \neg A_1,\ \beta_2 = \neg A_2$	$\neg(A_1 \wedge A_2)$	β

شکل 2.8: طبقه‌بندی فرمول‌های آلفا و بتا

2.64 الگوریتم ساخت جدول معنایی

۱. ابتدایی‌سازی:
درختی با یک گرهٔ ریشه برچسب‌خورده ϕ بسازید. این گره هنوز علامت ندارد.

۲. گسترش:
تا زمانی که برگ علامت‌نگرفته‌ای باقی بماند، مراحل زیر را تکرار کنید:

آ) برگ l را انتخاب کنید با مجموعهٔ برچسب $U(l)$ که هنوز علامت ندارد.

ب) اگر $U(l)$ مجموعه‌ای از لفظ‌ها باشد:

- اگر شامل زوج متمم باشد، برگ را بسته ($\times$) علامت بزنید.
- وگرنه، برگ را باز ($\odot$) علامت بزنید.

ج) در غیر این صورت (یعنی $U(l)$ شامل فرمول غیرلفظی است):

i) فرمولی $A \in U(l)$ را انتخاب کنید که لفظ نباشد.

ii) بر اساس α یا β بودن A عمل کنید:

- αــ فرمول:
یک فرزند l' بسازید با

$$U(l') = \big(U(l) - \{A\}\big) \cup \{A_1, A_2\}.$$

(اگر $A = \neg\neg A_1$ بود، تنها α_1 را اضافه کنید.)

- βــ فرمول:
دو فرزند l' و l'' بسازید با

$$U(l') = \big(U(l) - \{B\}\big) \cup \{B_1\}, \quad U(l'') = \big(U(l) - \{B\}\big) \cup \{B_2\}.$$

۳. پایان:
وقتی هیچ برگ علامت‌نگرفته‌ای باقی نماند، الگوریتم تمام می‌شود.

تعریف **2.65**.

- جدولی که ساخت آن تکمیل شده، جدول تکمیل‌شده نامیده می‌شود.
- یک جدول تکمیل‌شده بسته است اگر همهٔ برگ‌ها بسته ($\times$) باشند.
- در غیر این صورت (اگر دست‌کم یک برگ باز $\odot$ باشد)، جدول باز است.

2.6.3 پایان‌پذیری ساخت جدول معنایی

از آنجا که هر گام از الگوریتم یک فرمول را به یک یا دو فرمول ساده‌تر فرو می‌شکند، بدیهی است که ساخت جدول معنایی برای هر فرمول خاتمه می‌یابد؛ با این حال، اثبات این ادعا ارزشمند است.

قضیه **2.66**. ساخت جدول معنایی برای هر فرمول ϕ پایان‌پذیر است. هنگامی که ساخت خاتمه می‌یابد، تمام برگ‌ها با «$\times$» یا «$\odot$» علامت‌گذاری شده‌اند.

اثبات. فرض کنیم در فرمول ϕ، عملگرهای $\leftrightarrow$ و $\oplus$ ظاهر نشوند (تعمیم به این دو مورد به‌صورت تمرین باقی گذاشته می‌شود). برای هر برگ علامت‌نگرفته l که در مرحله‌ای از گسترش انتخاب می‌شود، بگذارید

$b(l)$ تعداد کل عملگرهای دودویی موجود در همهٔ فرمول‌های $U(l)$،

$n(l)$ تعداد کل نقیض‌ها در $U(l)$

سپس وزن

$$W(l) \;=\; 3\,b(l) \;+\; n(l)$$

را تعریف می‌کنیم. برای مثال اگر

$$U(l) = \{\, p \vee q,\ \neg p \wedge \neg q \},$$

آنگاه $b(l) = 2$ و $n(l) = 2$ و بنابراین

$$W(l) = 3 \cdot 2 + 2 = 8.$$

هر گام از الگوریتم یا یک گرهٔ جدید l' یا دو گرهٔ جدید l', l'' را به‌عنوان فرزند l می‌افزاید. ادعا می‌کنیم که در هر حالت:

$W(l') < W(l)$ و اگر گرهٔ دوم وجود داشته باشد، $W(l'') < W(l).$

برای نمونه، فرض کنید فرمولی از نوع α داشته باشیم:

$$A = \neg\,(A_1 \vee A_2),$$

و قاعدهٔ α را روی برگ l اعمال کنیم تا برگ جدید l' برچسب‌خورد:

$$U(l') = \big(U(l) \setminus \{\neg(A_1 \vee A_2)\}\big) \;\cup\; \{\neg A_1,\ \neg A_2\}.$$

در این صورت یکی از عملگرهای دودویی (یعنی $\vee$) و یک نقیض (علامت نفی بیرونی) حذف می‌شوند و دو نقیض جدید (برای A_1 و A_2) افزوده می‌شود. بنابراین:

$$W(l') = W(l) - \big(3 \cdot 1 + 1\big) + 2 = W(l) - 2 < W(l).$$

به همین ترتیب برای هر قاعدهٔ α یا β، وزن کاهش می‌یابد. از آنجا که $W(l)$ عددی طبیعی است و در هر گام کاهش پیدا می‌کند، الگوریتم نمی‌تواند بی‌پایان ادامه یابد و نهایتاً به برگ‌هایی منتهی می‌شود که همهٔ آن‌ها علامت‌گذاری شده‌اند. $\square$

2.6.4 بهبود کارایی الگوریتم

الگوریتم ساخت جدول معنایی «قطعی» نیست: در اکثر مراحل، انتخاب برگ برای گسترش و در صورتی که برگ بیش از یک فرمول غیرلفظی داشته باشد، انتخاب فرمول برای تجزیه آزاد است. این امکان را فراهم می‌کند که از هدایت‌کننده‌ها (heuristics) استفاده کنیم تا جدول سریع‌تر تکمیل شود. همان‌طور که در بخش 2.6.2 دیدیم، بهتر است ابتدا αـ فرمول‌ها را باز کنیم و سپس βـ فرمول‌ها تا از تکرار بیهوده جلوگیری گردد.

کوتاه‌سازی جدول با بستن زودهنگام شاخه: می‌توان یک شاخه را به‌محض آنکه شامل یک فرمول و متمم آن شود (نه صرفاً یک زوج متمم از لفظ‌ها) بست. واضح است که ادامهٔ گسترش گره‌ای که برچسب

$$\{\, p \wedge (q \vee r),\ \neg\big(p \wedge (q \vee r)\big)\}$$

را دارد بی‌معنی است. اثبات اینکه این تغییر در صحت الگوریتم خللی ایجاد نمی‌کند به‌عنوان تمرین باقی گذاشته شده است.

کاهش تکرارِ بازنویسی فرمول‌ها: انتقال مکرر فرمول‌ها از یک گره به گرهٔ فرزند:

$$U(l') = \big(U(l) \setminus \{A\}\big) \ \cup\ \{A_1, A_2\}$$

منجر به تکرارهای بیهوده می‌شود. در گونه‌ای از جدول معنایی به نام جدول‌های تحلیلی، هنگامی که گرهٔ جدیدی ایجاد می‌شود، تنها با فرمول‌های تازه برچسب می‌خورد:

$$U(l') = \{A_1, A_2\}.$$

الگوریتم چنان تغییر می‌کند که فرمولی برای تجزیه انتخاب شود که در مسیر از ریشه تا برگ وجود دارد (به شرطی که تاکنون انتخاب نشده باشد).

- برگ «بسته» می‌شود اگر دو لفظ متمم (یا دو فرمول متمم) در برچسب‌های یک یا دو گره در همان شاخه ظاهر شوند.
- برگ «باز» علامت می‌خورد اگر شاخه بسته نباشد و دیگر فرمولی برای تجزیه نمانده باشد.

. برای فرمول

$$B = (p \vee q) \wedge (\neg p \wedge \neg q)$$

جدول تحلیلی به‌صورت زیر است؛ دقت کنید که پس از باز کردن $p \wedge q$، $p \vee q$ از گرهٔ دوم به گرهٔ سوم منتقل نمی‌شود:

$$(p \vee q) \wedge (\neg p \wedge \neg q)$$
$$|$$
$$p \vee q, \neg p \wedge \neg q$$
$$|$$
$$\neg p, \neg q$$
$$\begin{array}{cc} p & q \\ \times & \times \end{array}$$

با این حال، ما جدول معنایی کلاسیک را ترجیح می‌دهیم زیرا در آن به‌وضوح دیده می‌شود که کدام فرمول‌ها نامزد تجزیه هستند و چگونه برگ‌ها علامت‌گذاری می‌گردند.

2.7 صداپذیری و کاملیت (Soundness and Completeness)

ساخت جدول معنایی یک فرایند کاملاً رسمی است. تجزیهٔ فرمول تنها مبتنی بر خواص نحوّی آن است: اپراتور اصلی فرمول و—در صورت منفی بودن—اپراتور اصلی فرمول منفی‌شده. تا کنون چند مثال برای انگیزهٔ جدول معنایی ارائه کردیم، اما صحت الگوریتم (ارتباط خروجی نحوی جدول با مفهوم معناشناختی ارزش صدق) را هنوز اثبات نکرده‌ایم. در این بخش نشان می‌دهیم که الگوریتم دقیقاً زمانی گزارش می‌دهد که فرمول ارضاپذیر یا ناتوان از ارضا است که مدل وجود دارد یا ندارد.
روش‌های اثبات این بخش بسیار اهمیت دارد، زیرا در سیستم‌های منطقی دیگر بارها به‌کار گرفته می‌شوند.

قضیه **2.67** صداپذیری و کاملیت. بگذارید $\mathscr{T}$ یک جدول تکمیل‌شده برای فرمول A باشد. آنگاه:

$$\mathscr{T} \text{ بسته است.} \iff A \text{ ناتوان از ارضا است.}$$

از این قضیه چند نتیجهٔ مهم به‌دست می‌آید:

نتیجه‌گیری **2.67**. فرمول A ارضاپذیر است اگر و تنها اگر $\mathscr{T}$ باز باشد.

اثبات. A ارضاپذیر است $\iff$ (به‌تعریف) A ناتوان از ارضا نیست $\iff$ (با قضیهٔ 2.67) $\mathscr{T}$ بسته نیست $\iff$ (به‌تعریف) $\mathscr{T}$ باز است. $\square$

نتیجه‌گیری **2.68**. فرمول A همگانی‌صادق (valid) است اگر و تنها اگر جدول معنایی برای $\neg A$ بسته شود.

اثبات. A همگانی‌صادق است $\iff$ $\neg A$ ناتوان از ارضا است $\iff$ جدول $\neg A$ بسته می‌شود. $\square$

نتیجه‌گیری **2.70**. روش جدول معنایی یک رویهٔ تصمیم (decision procedure) برای همگانی‌صدقی در منطق گزاره‌ای است.

اثبات. برای هر فرمول گزاره‌ای A، به کمک قضیهٔ 2.66 ساخت جدول $\neg A$ پایان‌پذیر است و جدول تکمیل‌شده به‌دست می‌آید. از نتیجه‌گیری 2.69 نتیجه می‌شود که A همگانی‌صادق $\iff$ جدول $\neg A$ بسته است. $\square$

سمت مستقیم نتیجه‌گیری 2.69 را کاملیت (completeness) می‌نامند: اگر A همگانی‌صادق باشد، با ساخت جدول برای $\neg A$ حتماً آن جدول بسته می‌شود. جهت عکس را صداپذیری (soundness) می‌گویند: هر فرمولی که جدول‌سازی آن را همگانی‌صادق اعلام کند (چون جدول $\neg A$ بسته است) واقعاً همگانی‌صادق است. در منطق، معمولاً اثبات صداپذیری آسان‌تر از اثبات کاملیت است؛ زیرا قواعد سیستم را طوری انتخاب می‌کنیم که آشکارا صداپذیر باشند، ولی سخت بتوانیم اطمینان یابیم که هیچ قاعدهٔ مفقودی نداریم که برای کاملیت ضروری باشد.
به‌عنوان مثال، الگوریتم ضعیف زیر صداپذیر اما کاملاً ناقص است:

الگوریتم **2.71** **(Incomplete decision procedure for validity)**
ورودی: فرمول A در منطق گزاره‌ای
خروجی: گزارش «A همگانی‌صادق نیست.»

مثال **2.72**. اگر قاعدهٔ مربوط به $\neg(A_1 \vee A_2)$ را حذف کنیم، ساخت جدول هنوز صداپذیر خواهد بود، اما کاملیت را از دست می‌دهد. برای مثال فرمول واضحاً همگانی‌صادق

$$A = \neg p \vee p$$

را در نظر بگیرید. ریشهٔ جدول با $\neg A = \neg(\neg p \vee p)$ برچسب می‌خورد، اما هیچ قاعده‌ای برای تجزیهٔ این فرمول موجود نیست و بنابراین جدول بسته نمی‌شود، درحالی‌که A به‌راستی همگانی‌صادق است.

2.7.1 اثبات درستی

. اگر جدول (tableau) $\mathscr{T}$ برای فرمول A بسته شود، آنگاه A نارضایت‌پذیر unsatisfiable است.

ما قضیه‌ای کلی‌تر را اثبات می‌کنیم:

. اگر زیردرخت $\mathscr{T}_n$، که در گرهٔ n از $\mathscr{T}$ ریشه دارد، بسته باشد، آنگاه مجموعهٔ فرمول‌های $U(n)$ که برچسب گرهٔ n است، نارضایت‌پذیر است.

درستیِ جدول (soundness) حالت خاصی از این قضیه است هنگامی که n همان گرهٔ ریشه باشد.
برای سادگی اثبات، فرض می‌کنیم که فقط دو شکل از فرمول‌های α و β داریم:

$$\alpha\colon\ A_1 \wedge A_2, \qquad \beta\colon\ B_1 \vee B_2.$$

اثبات. با استقرا بر ارتفاع h_n گرهٔ n در زیردرخت $\mathscr{T}_n$.
قضیهٔ پایه ($h_n = 0$). اگر $h_n = 0$، آنگاه n یک برگه (leaf) است و چون $\mathscr{T}_n$ بسته است، در برچسب $U(n)$ حتماً یک جفت متضاد از لیترال‌ها وجود دارد. بنابراین $U(n)$ نارضایت‌پذیر است.
فرض استقرا. فرض کنید برای هر گرهٔ m با ارتفاع کمتر از h_n داریم:

$$\forall m,\ h_m < h_n,\ \left[\mathscr{T}_m \text{ بسته}\right] \implies U(m) \text{ نارضایت‌پذیر}.$$

گام استقرایی. حال گرهٔ n را در نظر بگیرید که $h_n > 0$. از آنجا که $\mathscr{T}_n$ بسته است، حتماً روی n یکی از دو قاعدهٔ α یا β اعمال شده است:

$$n : A_1 \wedge A_2 \cup U_0$$
$$|$$
$$n' : A_1, A_2 \cup U_0$$

$$n : B_1 \vee B_2 \cup U_0$$
$$n' : B_1 \cup U_0 \qquad n'' : B_2 \cup U_0$$

۱) α اگر قاعدهٔ α (برای $A_1 \wedge A_2$) روی n اعمال شده باشد، آنگاه

$$U(n) = \{\, A_1 \wedge A_2 \} \cup U_0, \quad U(n') = \{\, A_1,\ A_2 \} \cup U_0,$$

و $\mathscr{T}_{n'}$ نیز بسته است. از آنجا که ارتفاع n' برابر $h_n - 1$ است، به‌موجب فرض استقرا $U(n')$ نارضایت‌پذیر است. برای اثبات نارضایت‌پذیری $U(n)$، هر تفسیر دلخواه $\mathscr{I}$ را در نظر بگیرید:

- اگر برای بعضی $A_0 \in U_0$، $v_{\mathscr{I}}(A_0) = F$، چون $U_0 \subset U(n)$، نتیجه می‌شود $U(n)$ نارضایت‌پذیر است.
- در غیر این صورت $v_{\mathscr{I}}(A_0) = T$ برای همهٔ $A_0 \in U_0$. چون $U(n')$ نارضایت‌پذیر است، باید $v_{\mathscr{I}}(A_1) = F$ یا $v_{\mathscr{I}}(A_2) = F$. اگر $v_{\mathscr{I}}(A_1) = F$، آنگاه

$$v_{\mathscr{I}}(A_1 \wedge A_2) = F,$$

و چون $A_1 \wedge A_2 \in U(n)$، $U(n)$ نارضایت‌پذیر است. (استدلال مشابه برای $v_{\mathscr{I}}(A_2) = F$.)

۲) β اگر قاعدهٔ β (برای $B_1 \vee B_2$) روی n اعمال شده باشد، آنگاه

$$U(n) = \{\, B_1 \vee B_2\} \cup U_0, \quad U(n') = \{\, B_1\} \cup U_0, \quad U(n'') = \{\, B_2\} \cup U_0,$$

که هر دو $\mathscr{T}_{n'}$ و $\mathscr{T}_{n''}$ بسته‌اند و ارتفاع‌شان کمتر از h_n است. بنابراین $U(n')$ و $U(n'')$ نارضایت‌پذیرند. برای هر تفسیر $\mathscr{I}$:

- اگر برای بعضی $B_0 \in U_0$، $v_{\mathscr{I}}(B_0) = F$، چون $U_0 \subset U(n)$، آنگاه $U(n)$ نارضایت‌پذیر است.
- وگرنه $v_{\mathscr{I}}(B_0) = T$ برای همهٔ $B_0 \in U_0$. از نارضایت‌پذیری $U(n')$ و $U(n'')$ داریم $v_{\mathscr{I}}(B_1) = F$ و $v_{\mathscr{I}}(B_2) = F$. بنابراین

$$v_{\mathscr{I}}(B_1 \vee B_2) = F,$$

و چون $B_1 \vee B_2 \in U(n)$، $U(n)$ نارضایت‌پذیر است.

در هر دو حالت، نشان دادیم که اگر زیردرختهای فرزندان n بسته باشند، آنگاه $U(n)$ نیز نارضایت‌پذیر است. این کامل‌کنندهٔ گام استقرایی است.
بنابراین، به‌موجب اصل استقرا، برای هر گره n، اگر زیردرخت $\mathscr{T}_n$ بسته باشد، آنگاه $U(n)$ نارضایت‌پذیر است. حالت ویژه برای گرهٔ ریشه ($n = \mathrm{root}$) اثبات می‌کند که اگر کل جدول $\mathscr{T}$ بسته شود، آنگاه فرمول اولیه A نارضایت‌پذیر است. $\square$

2.7.2 اثبات کامل بودن (Completeness)

. اگر فرمول A نارضایت‌پذیر باشد، آنگاه هر جدولی (tableau) که برای A ساخته شود، بسته می‌شود.

برای اثبات، به‌جای مستقیم نشان می‌دهیم:

نتیجه گیری **2.68**. اگر جدولی برای A باز باشد (یعنی دارای حداقل یک شاخهٔ باز)، آنگاه A برآورده‌پذیر (satisfiable) است.

روش کار در چهار گام اصلی است:

۱) تعریف مجموعهٔ هینتیکا (Hintikka set).

۲) اثبات اینکه اجتماع برچسب‌های گره‌های یک شاخهٔ باز، یک مجموعهٔ هینتیکا است.

۳) اثبات اینکه هر مجموعهٔ هینتیکا برآورده‌پذیر است.

۴) نشان دادن اینکه خود فرمول A (برچسب ریشه) در آن مجموعه حضور دارد.

مجموعهٔ هینتیکا. مجموعه‌ای از فرمول‌ها را مجموعهٔ هینتیکا می‌نامیم اگر:

۱) برای هر اتم p که در مجموعه هست، دقیقاً یکی از $p \in U$ یا $\neg p \in U$ برقرار باشد.

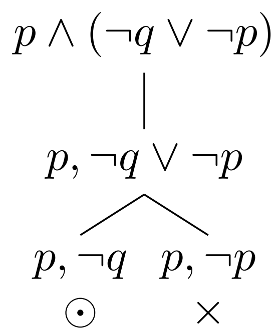
۲) اگر $A \in U$ یک فرمول α باشد (یعنی $A = A_1 \wedge A_2$)، آنگاه $A_1, A_2 \in U$.

۳) اگر $B \in U$ یک فرمول β باشد (یعنی $B = B_1 \vee B_2$)، آنگاه $B_1 \in U$ یا $B_2 \in U$.

مثال **2.73**. فرض کنید

$$A = p \wedge (\neg q \vee \neg p).$$

یک جدول ممکن:

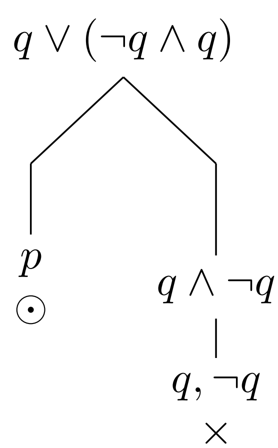


شاخهٔ $p, \neg q$ باز است با $p = T,\ q = F$ که مدلی برای A می‌سازد.

مثال **2.74**. فرض کنید

$$A = p \vee (q \wedge \neg q).$$

یک جدول ممکن:



شاخهٔ p باز است، پس هر مدلی باید $p = T$ باشد.

قضیهٔ **2.77**. اگر l یک برگ باز در جدول تکمیل‌شده T باشد، آنگاه

$$U = \bigcup_{i \in \text{شاخه از ریشه تا } l} U(i)$$

یک مجموعهٔ هینتیکا است.

اثبات. ۱) لیترال‌ها در هیچ قاعده‌ای شکسته نمی‌شوند و از ریشه تا برگ منتقل می‌گردند. چون برگ باز است، جفت متضاد در $U(l)$ نیست، پس شرط (۱) برقرار است.

۲) اگر $A \in U$ یک αـ فرمول باشد، در مسیر تجزیه شده و زیرفرمول‌های A_1, A_2 در U قرار می‌گیرند.

۳) اگر $B \in U$ یک βـ فرمول باشد، در مسیر تجزیه شده و یکی از زیرفرمول‌های B_1 یا B_2 در U قرار می‌گیرد.

بنابراین U هینتیکا است. $\square$

لم هینتیکا **2.78**. اگر U یک مجموعهٔ هینتیکا باشد، آنگاه U برآورده‌پذیر است.

اثبات. اتم‌های P_U را مجموعهٔ اتم‌های ظاهرشده در U در نظر بگیرید. تفسیر $\mathscr{I}$ را تعریف می‌کنیم:

$$\mathscr{I}(p) = \begin{cases} \mathsf{T}, & p \in U, \\ \mathsf{F}, & \neg p \in U, \\ \mathsf{T}, & \text{اگر} p, \neg p \notin U. \end{cases}$$

شرط (۱) تناقض را نفی می‌کند. سپس با استقرا بر ساختار فرمول‌ها نشان می‌دهیم برای هر $A \in U$، $v_{\mathscr{I}}(A) = \mathsf{T}$:

- اگر $A = p$ یا $A = \neg p$، واضح است.
- اگر $A = A_1 \wedge A_2$، شرط (۲) هر دو $A_1, A_2 \in U$ را تضمین می‌کند.
- اگر $A = B_1 \vee B_2$، شرط (۳) یکی از آنها را تضمین می‌کند.

پس $\mathscr{I}$ مدلی از U است. $\square$

نتیجه‌گیری

اگر T جدولی باز و تکمیل‌شده برای A باشد، اجتماع برچسب‌های شاخهٔ باز مجموعه‌ای هینتیکا می‌سازد (قضیهٔ 2.77) و از لم هینتیکا (قضیهٔ 2.78) این مجموعه مدل دارد. چون A در برچسب ریشه هست، $\mathscr{I}$ مدلی برای A می‌شود و بنابراین اثبات کامل بودن پایان می‌یابد.