روش تابلو

نرگس بابااحمدی

۶۱۰۳۹۸۱۰۲

مقدمه:

روش تابلو، روشی برای تعیین ارزش یک گزاره است. در این روش رویکرد ما بر این است که یک گزاره را تا حد امکان ساده کنیم (تا جایی که به گزاره های اتمی و نقیض آن ها برسیم) و در حین انجام این کار، از درختی استفاده می‌کنیم که ریشه این درخت همان گزاره اولیه ما است و با شکستن گزاره اصلی به گزاره‌های کوچکتر به شاخه‌های درخت اضافه می‌شود تا سرانجام به برگ‌های درخت که همان گزاره‌های اتمی و نقیض آن‌ها هستند، می‌رسیم.

در این روش اگر بتوانیم برای گزاره مورد نظر به‌طور روش‌مند مدلی بیابیم آنگاه گزاره، ارضاشدنی است و در غیر این صورت ارضانشدنی است که اگر ارضانشدنی باشد یعنی نقیض آن همانگو است.

در این گزارش روش تابلو را در دو مقوله بررسی می‌کنیم:

- روش تابلو برای منطق گزاره‌ای
- روش تابلو در منطق مرتبه اول

۱. روش تابلو برای منطق گزاره‌ای:

می‌دانیم در روش تابلو با دادن گزاره‌ای به عنوان ورودی، یک درخت بدست می‌آید. این درخت به روش استقرایی ساخته می‌شود که در ادامه با آن آشنا می‌شویم. برای فهم بهتر روش تابلو برای منطق گزاره‌ای، به چند تعریف نیاز داریم:

- **شاخه باز و شاخه بسته :**

برای توصیف این دو مفهوم، آن دو را در درخت مربوط به گزاره $p \wedge (\neg p \vee \neg q)$ بررسی می‌کنیم.

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

$\downarrow$

p و $(\neg p \vee \neg q)$

۱ ۲

$(p$ و $\neg p)$ $(p$ و $\neg q)$

شاخه ۲ شاخه باز و شاخه ۱ که شامل یک فرمول اتمی و نقیض آن است، یک شاخه بسته است.

گزاره بالا ارضاشدنی است اگر و تنها اگر بتوانیم برای یکی از دو مجموعه ۱ یا ۲ مدلی پیدا کنیم (یکی از آن‌ها ارضاشدنی باشد) و ارضانشدنی است اگر برای هیچکدام مدلی نباشد. از آنجایی که برای گزاره ۲ مدلی وجود دارد ($v_{(p)} = 1$ و $v_{(q)} = 0$) پس این گزاره ارضاشدنی است.

در روش تابلو برای منطق گزاره‌ای، گزاره ها به دو بخش تقسیم می‌شوند:

۱. α- قاعده :

گزاره‌هایی هستند که بعد از ساده شدن (اعمال تمامی نقیض‌ها) اپراتور دو موضعی آن‌ها "∧" باشد.
پس برای α- قاعده‌ها، چهار حالت پیش می‌آید:

α	α_1	α_2
$\neg\neg A$	A	
$\neg(A_1 \vee A_2)$	$\neg A_1$	$\neg A_2$
$\neg(A_1 \rightarrow A_2)$	A_1	$\neg A_2$
$A_1 \wedge A_2$	A_1	A_2

یک گزاره از این نوع، ارضاشدنی است اگر و تنها اگر هر دو گزاره α_1 و α_2 ارضاشدنی باشند.

این دسته از گزاره‌ها در درخت گزاره‌ای شاخه‌ای جدید ایجاد نمی‌کنند و تنها شاخه قبلی را بلندتر می‌کنند.

۲. β – قاعده :

گزاره‌هایی هستند که پس از ساده شدن، اپراتور دو موضعی آن‌ها "V" باشد. برای β – قاعده‌ها، سه حالت پیش می‌آید:

β	β_1	β_2
$\neg(B_1 \wedge B_2)$	$\neg B_1$	$\neg B_2$
$B_1 \vee B_2$	B_1	B_2
$B_1 \rightarrow B_2$	$\neg B_1$	B_2

گزاره‌ای از این نوع، ارضاشدنی است اگریکی از دو گزاره β_1 یا β_2 ارضاشدنی باشند.

بر خلاف α– قاعده‌ها، این نوع گزاره‌ها در درخت شاخه جدید بوجود می‌آوردند.

الگوریتم ساخت تابلو:

در این الگوریتم، ورودی گزاره اصلی (مثلا A) است و خروجی تابلو T برای A است که همه برگ‌های آن نشان‌دار شده است.

تابلو T همان درخت A است که در ابتدا ریشه این درخت {A} است و سپس به هر گره L یک مجموعه $u(L)$ (با به‌کار بردن قواعد) نسبت داده می‌شود (فرمول اصلی که A باشد را به فرمول‌های کوچکتر می‌شکنیم). ساخت وقتی تمام می‌شود که تمام برگ‌ها، گزاره‌های اتمی بوده و بتوان آن‌ها را با X یا O نشان‌دار کرد.

حال ساخت درخت (تابلو T) را به‌صورت استقرایی شرح می‌دهیم:

- اگر برای گزاره‌ای مثل B ، $u(L)$ شامل B و $\neg B$ باشد، برگ L با X نشان‌دار می‌شود و آن شاخه بسته است.
- در غیر این صورت گزاره مانند B را در $u(L)$ انتخاب می‌کنیم که گزاره‌ای اتمی یا نقیض گزاره‌ای اتمی نباشد:

- اگر B یک α- فرمول باشد، همان شاخه را ادامه داده و به آن گره جدید ′L را اضافه می‌کنیم و مجموعه زیر را به آن نسبت می‌دهیم:

$$u(L') = (u(L) - \{B\}) \cup \{\alpha_1, \alpha_2\}$$

- اگر B یک β – فرمول باشد، گره‌های جدید ′L و ″L را به عنوان فرزندان L می‌سازیم و دو مجموعه زیر را به آن‌ها نسبت می‌دهیم:

$$u(L') = (u(L) - \{B\}) \cup \{\beta_1\}$$
$$u(L'') = (u(L) - \{B\}) \cup \{\beta_2\}$$

قضیه:

ساخت تابلو، پایان‌پذیر است.

برهان:

چون گزاره متناهی است پس تعداد اپراتورهای آن هم متناهی است. می‌دانیم در ساخت درخت (با توجه به الگوریتمی که توضیح داده شد) در هر مرحله، اپراتورها یکی یکی کم می‌شوند، پس بدیهی است که در آخر، تعداد اپراتورها صفر شده و به گزاره‌های اتمی می‌رسیم. از آنجایی که با طی کردن تعداد مراحل متناهی به این گزاره‌ها رسیدیم، پس ساخت تابلو پایان‌پذیر است.

- **تابلو کامل:**

  تابلویی است که ساخت آن پایان یافته است.

- **تابلو بسته (باز):**

  اگر تمام برگ‌های یک تابلو کامل، بسته باشد، آن تابلو کامل را بسته و در غیر این صورت تابلو را باز گوییم.

قضیه درستی:

اگر تابلو T برای گزاره A بسته شود آنگاه A ارضاشدنی نیست.

برهان:

می‌دانیم که در تابلو T کل مجموعه مربوط به یک گره به شرطی سازگار است که مجموعه مربوط به تمام فرزندان (یا فرزند) آن گره سازگار باشند (شامل دو گزاره اتمی نقیض نباشند). از آنجایی که در صورت قضیه داریم که تابلو T برای گزاره A بسته است، پس می‌دانیم تمام برگ‌های آن ناسازگارند.

پس با استفاده از استقرا، قضیه اثبات می‌شود:

پایه استقرا:

$$\begin{tikzcd} A \arrow[d] \\ X \end{tikzcd} \quad و \quad \begin{tikzcd} & A \arrow[dl] \arrow[dr] & \\ X & & X \end{tikzcd}$$

در هر دو حالت با توجه به تعاریف α- قاعده و β- قاعده ، A ارضانشدنی است.

فرض استقرا: با فرض این‌که برای هر درخت با عمق n-۱ قضیه برقرار است، ثابت خواهیم کرد که برای درخت با عمق n هم قضیه برقرار می‌شود.

گزاره A با عمق n را در نظر بگیرید.

۱. A یک α- قاعده باشد، که با توجه به الگوریتم ساخت تابلو می‌دانیم که فرزندی به نام A' با عمق n-۱ و تمام برگ‌های بسته دارد، پس A' ارضاشدنی نیست. پس طبق پایه استقرا، A هم ارضاشدنی نیست.

۲. A یک β – قاعده که با توجه به الگوریتم ساخت تابلو می‌دانیم دو فرزند A' و A'' دارد که تمام برگ‌هایشان بسته است و عمقشان هم بسته است پس هم A' و هم A'' ارضانشدنی بوده، پس طبق پایه استقرا A هم ارضاشدنی نیست.

پس قضیه اثبات شد.

تعریف مجموعه هینتیکا :

یک مجموعه هینتیکا است اگر و تنها اگر سه شرط زیر را دارا باشد:

(اگر مجموعه هینتیکا را u بگیریم)

۱. برای هر گزاره اتمی p یا $p \in u$ باشد یا $\neg p \in u$

۲. اگر $A \in u$ یک α - فرمول باشد آنگاه $\alpha_1 \in u$ و $\alpha_2 \in u$

۳. اگر $A \in u$ یک β - فرمول باشد آنگاه $\beta_1 \in u$ یا $\beta_2 \in u$

لم ۱-۱:

مجموعه هینتیکا مدل دارد.

برهان:

اثبات را با کمک یک تابع ارزش شروع می‌کنیم.

$$p \in u \Rightarrow v(p) = 1$$
$$\neg p \in u \Rightarrow v(p) = 0$$
$$p, \neg p \notin u \Rightarrow v(p) = 1$$

با استقرا ثابت می‌کنیم تعبیر I تولید شده توسط v ، ریشه A را درست تعبیر می‌کند. $(I(A) = 1)$

(استقرا روی پیچیدگی فرمول ریشه)

پایه استقرا: اگر $A = p$ یا $A = \neg p$ $\longleftarrow I(A) = 1$

فرض استقرا: فرض می‌کنیم برای گزاره‌هایی با n-۱ ادات دو موضعی یا کمتر حکم برقرار باشد، باید ثابت کنیم برای درخت با عمق n هم برقرار است.

حال A' و A'' را فرض کنید که دو زیرمجموعه A هستند و با هم A را می‌سازند. عمق A، n است. پس عمق هر کدام از A' و A'' کوچکتر یا مساوی n-۱ بوده و مجموع ادات دو موضعی‌شان n-۱ می‌شود. چون هر کدام از این دو عنصر مجموعه هینتیکا هستند و کمتر از n ادات دو موضعی دارند، طبق فرض استقرا داریم:

$I(A') = I(A'') = 1$. حال برای A با n ادات موضعی، درکل چهار حالت ممکن است رخ بدهد:

$A = A'' \to A'$ و $A = A' \wedge A''$ و $A = A' \vee A''$ و $A = A' \to A''$

که در همه حالات با توجه به اینکه I هم A' و هم A'' را درست تعبیر می‌کند، به این نتیجه می‌رسیم که I ، A را هم درست تعبیر می‌کند.

لم۲-۱:

فرض می‌کنم T یک تابلوی کامل و L مجموعه همه گره‌های یک شاخه باز در T باشد آنگاه $u=\bigcup_{i\in L} u(i)$ یک مجموعه هینتیکا است.

برهان:

فرض کنیم A یکی از اعضای u و یک گره از شاخه L باشد. اگر A یک β – فرمول باشد آنگاه طبق تعریف تابلو β_1 یا β_2 عضو u هستند و اگر A یک α – فرمول باشد، α_1 و α_2 عضو u هستند. پس تا اینجا، دو تا از شرط‌ها را داریم. حال فقط کافی است ثابت کنیم برای هر گزاره اتمی یا $p\in u$ یا $\neg p\in u$. برای اثبات از برهان خلف استفاده می‌کنیم. پس فرض می‌کنیم یک p وجود دارد به طوری که $p,\neg p\in u$. پس آنگاه p و $\neg p$ در یک شاخه باز درخت و در نتیجه (درختمان شاخه فرعی ندارد) در برگ ها هستند که این با باز بودن این شاخه در تناقض است. پس به این نتیجه می‌رسیم که u ، هینتیکا است.

قضیه تمامیت:

اگر A ارضانشدنی باشد آنگاه هر تابلوی کامل T برای آن بسته می‌شود.

برهان:

از برهان خلف استفاده می‌کنیم. یعنی نشان می‌دهیم اگر تابلو کامل T باز باشد آنگاه A ارضاشدنی است که این با فرض قضیه در تناقض است. پس از این تناقض نتیجه می‌گیریم که تابلوی T برای مجموعه ارضانشدنی A، بسته است.

فرض می‌کنیم که تابلوی کامل T، باز است. پس حتما یک شاخه باز مانند L در این درخت وجود دارد.

از لم ۲-۱ داریم $u=\bigcup_{I\in L} u(I)$ یک مجموعه هینتیکا است و از لم ۱-۱ داریم که هر مجموعه هینتیکا مدل دارد. پس چون $A\in u$ ، A ارضاشدنی است و این همان تناقض است. پس تابلو T، بسته است.

قضیه درستی و تمامیت:

گزاره A همانگو است اگر و تنها اگر تابلو $\neg A$ بسته باشد.

۲. روش تابلو برای منطق مرتبه اول:

برای ارائه الگوریتم تابلو برای منطق مرتبه اول ابتدا قواعدی برای سورها عمومی و وجودی را ارائه می‌دهیم و بعد از اثبات درستی روش تابلو، قضیه تمامیت را اثبات می‌کنیم.

در منطق مرتبه اول دو قاعده جدید اضافه می‌شود؛ γ- قاعده و δ- قاعده:

γ- قاعده برای فرمول‌های دارای سور عمومی و δ- قاعده برای فرمول‌های دارای سور وجودی است.

γ	$\gamma(a)$
$\forall x A(x)$	$A(a)$
$\neg\exists x A(x)$	$\neg A(a)$

δ	$\delta(a)$
$\exists x A(x)$	$A(a)$
$\neg\forall x A(x)$	$\neg A(a)$

الگوریتم ساخت تابلو در منطق مرتبه اول:

الگوریتم ساخت مانند منطق گزاره‌ها است، فقط در اینجا باید قواعد برای γ- فرمول ها و δ- فرمول ها را بیان کنیم.

- اگر B یک فرمول اتمی یا نقیض فرمول اتمی نباشد و یک γ- فرمول باشد، گره L' را به این صورت می‌سازیم:

$$u(L') = u(L) \cup \{\gamma(a)\}$$
$$\forall x A x \rightarrow \{\forall x A(x), A(a)\}$$
$$\neg\exists x A(x) \rightarrow \{\neg\exists x A(x), \neg A(a)\}$$

که a نماد ثابتی است که در $u(L)$ ظاهر شده است. (ثابتی در شاخه پدر)

اگر $u(L)$ شامل جملات اتمی یا نقیض آن‌ها باشد و γ- فرمول هم باشد آنگاه برای هر انتخاب a، $u(L') = u(L)$ و برگ L با O نشان‌دار می‌شود.

- اگر B یک فرمول اتمی یا نقیض آن نباشد و یک δ - فرمول باشد، گره L' را به این صورت می‌سازیم:

$$u(L') = (u(L) - \{\delta\}) \cup \{\delta(a)\}$$
$$\exists x A x \rightarrow A(x)$$
$$\neg\forall x A(x) \rightarrow \neg A(x)$$

که a نماد ثابتی است که در $u(L)$ و شاخه پدر، ظاهر نشده است.

حال مرحله به مرحله با استفاده از این قواعد و یک B که اتمی یا نقیض اتمی نیست، گزاره را ساده و ساده‌تر می‌کنیم تا جایی که $u(L)$ فقط شامل گزاره‌های اتمی یا نقیض آن‌ها باشد و بعد سازگاری را بررسی می‌کنیم.

اگر p و $\neg p$ در گزاره بود بسته می‌شود و اگر γ- فرمول داشتیم آنگاه باز است.

قضیه درستی:

اگر تابلو T برای فرمول A در منطق مرتبه اول بسته شود، آنگاه A ارضاشدنی نیست.

برهان:

برای اثبات قضیه، درست مانند بخش قبل برای انواع مختلف فرمول‌ها باید اثبات کنیم.

برای α – فرمول‌ها و β – فرمول‌ها مانند قبل است. در اینجا برای حالتی که γ- قاعده و حالتی که δ- قاعده باشد، بررسی می‌کنیم:

γ- قاعده:

اگر فرض کنیم $u(n)=u_0 \cup \{\forall x A(x)\}$ پس داریم:

$$u(n')=u_0 \cup \{\forall x A(x), A(a)\}$$

اگر فرض کنیم $u(n)$ ارضاپذیر باشد پس m وجود دارد که مدل $\forall x A(x)$ است و از آن جایی که a ثابتی است که در $u(n)$ ظاهر شده (با توجه به تعریف) پس بدیهی است که m مدل $u(n')$ باشد. پس $u(n')$ هم ارضاپذیر است، در نتیجه شاخه باز است که تناقض است.

δ- قاعده:

اگر فرض کنیم $u(n)$ ارضاشدنی و m یک مدل آن باشد آنگاه $d \in m$ وجود دارد که m مدل $A(d)$ باشد. با تعبیر d برای نماد ثابت a مدل m' بدست می‌آید که مدلی برای $u(m')$ است.

برای اثبات قضیه تمامیت نیاز داریم که یک نسخه بهبود یافته و به‌گونه‌ای بهینه سازی شده از الگوریتم قبلی را ارائه بدهیم:

الگوریتم ساخت:

در این روش به هر گره $w(L)=\{u(L),c(L)\}$ نسبت داده می‌شود که $u(L)$ همان مجموعه فرمول‌های هر گره و $c(L)$ مجموعه ثابت هاست. به ریشه هم $\{A,\{a_1,...,a_k\}\}$ نسبت داده می‌شود که $\{a_1,...,a_k\}$مجموعه ثابت‌ها است. اگر هیچ ثابتی هم نداشت $\{a\}$ را به دلخواه انتخاب می‌کنیم. برای بررسی

سازگاری، $u(L)$ را بررسی می‌کنیم. اگر فقط شامل گزاره‌های اتمی یا نقیض آن‌ها بود، سازگاری آن را بررسی می‌کنیم و اگر نبود فرمول غیر اتمی مثل B را انتخاب کرده و ساخت درخت را ادامه می‌دهیم.

ساخت:

- اگر B، α – فرمول باشد برای گره جدید L' داریم:

$$w(L') = (u(L'), c(L')) = ((u(L) - \{B\}) \cup \{\alpha_1, \alpha_2\}, c(L))$$

- اگر B، β – فرمول باشد برای گره‌های جدید L' و L'' داریم:

$$w(L') = (u(L'), c(L')) = ((u(L) - \{B\}) \cup \{\beta_1\}, c(L))$$
$$w(L'') = (u(L''), c(L'')) = ((u(L) - \{B\}) \cup \{\beta_2\}, c(L))$$

- اگر B، δ– فرمول باشد برای گره جدید L' داریم:

$$w(L') = (u(L'), c(L')) = ((u(L) - \{B\}) \cup \delta(a), c(L) \cup \{a\})$$

- اگر B، γ – فرمول باشد داریم:

$$w(L) = (u(L), c(L))$$
$$\{\gamma_1, ..., \gamma_m\} \subseteq u(L)$$
$$c(L) = \{a_1, ..., a_k\}$$
$$w(L') = (u(L'), c(L')) = (u(L) \cup \bigcup_{i=1, j=1}^{i=m, j=k} \{\gamma_i(a_j)\}, c(L))$$

اگر $u(L)$ فقط شامل جمله‌های اتمی و یا نقیض آن‌ها و γ – فرمول‌ها باشد و $u(L') = u(L)$ آنگاه برگ L با o نشان‌دار می‌شود.

در این روش ممکن است شاخه نامتناهی ایجاد بشود (برای منطق مرتبه اول) پس یک روش مناسب برای جستجوی مدل نیست اما هدف ما اثبات قضیه تمامیت برای حالاتی است که شاخه نامتناهی نداشته باشیم. این روش همچنین برای اثبات درستی یک فرمول یا نقیض آن به کار می‌رود.

برای اثبات قضیه تمامیت به یک تعریف و دو لم نیاز داریم.

مجموعه هینتیکا:

تمام قواعد منطق گزاره‌ها را داریم و فقط دو قاعده برای γ – فرمول‌ها و δ– فرمول‌ها اضافه می‌شود.

- اگر $A \in u$ یک γ – فرمول باشد آنگاه برای هر ثابت a که در u ظاهر شده باشد، داریم: $\gamma(a) \in u$
- اگر $A \in u$ یک δ– فرمول باشد آنگاه ثابت a وجود دارد که: $\delta(a) \in u$

لم ۱–۲:

اگر L یک شاخه باز باشد و $u = \bigcup_{n \in L} u(n)$ آنگاه u یک مجموعه هینتیکا است.

برهان:

برای اثبات این لم، به یک لم دیگر نیاز داریم:

- لم*: اگر n یک گره از شاخه L بوده و $A \in u(n)$ آنگاه حتما در ادامه این شاخه بعد از n، در یکی از نواده‌های n مثل m یک قاعده بر A اعمال شده. حال اگر A یک γ – فرمول باشد و $a \in c(n)$ آنگاه $\gamma(n) \in u(m)$.

  برهان: وقتی به انجام الگوریتم ساخت تابلو ادامه می‌دهیم که به برگ‌ها و یعنی گره‌هایی که فقط شامل گزاره‌های اتمی یا نقیض آن‌ها باشند، برسیم. پس اگر یک A وجود داشته باشد که اتمی نباشد، حتما در یکی از مراحل در یکی از مراحل، قاعده‌ای بر آن اعمال شده است.

حال برای اثبات لم ۱–۲ فرض کنید $A \in u$:

اگر A اتمی یا α – فرمول یا β – فرمول باشد که اثبات مثل قبل است.

اگر A یک γ – فرمول و a یک ثابت دلخواه ظاهر شده در u باشد آنگاه گره m عضو L وجود دارد که $a \in c(m)$ و از آنجایی که مجموعه γ – فرمول‌ها و ثابت‌ها در طول یک شاخه غیرکاهشی است، برای $k = \max(n, m)$ و $A \in u(k)$ و $a \in c(k)$ با توجه به لم* داریم، وجود دارد $k' > k$ به طوری که $\gamma(a) \in u(k') \subseteq u$.

لم ۲-۲:

هر مجموعه هینتیکا U مدل دارد.

برهان:

قضیه تمامیت:

اگر A معتبر باشد آنگاه تابلو T برای $\neg A$ بسته می‌شود.

برهان:

(برهان خلف) فرض می‌کنیم تابلو T برای $\neg A$ باز باشد. پس شاخه L وجود دارد که باز است. با توجه به لم ۱-۲، $u=\bigcup_{n\in L} u(n)$ یک مجموعه هینتیکا است. پس با توجه به لم ۲-۲، این u یک مدل دارد و چون $\neg A\in u$ پس A نامعتبر است و این با فرض قضیه در تناقض است. پس تابلو T برای $\neg A$ بسته می‌شود.

توضیح الگوریتم کد:

این کد به عنوان ورودی، یک گزاره اتمی از ما دریافت می‌کند و در آخر به عنوان خروجی، درخت این گزاره را به ما نمایش می‌دهد. در اینجا رویکرد ما بر آن است که درخت این گزاره در واقع یک گراف است و تمام زیر گزاره‌ها، راس‌های این گراف هستند. راس‌هایی که در خروجی مشاهده می‌شود، با روش DFS شماره‌گذاری شده‌اند. ساخت این شماره‌ها به این صورت است که یک متغیر global به نام branchNum تعریف شده که در شروع، مقدارش صفر است و با اضافه شدن هر راس، این متغیر را با یک، جمع می‌کنیم. همچنین یک آرایه global به نام fatherNumArr برای نگهداری پدر هر راس، در شروع تعریف کرده‌ایم.

تابع pr-erase:

این تابع یک رشته به عنوان ورودی دریافت می‌کند و پرانتزهای اضافی آن را پاک می‌کند.

تابع negation-find:

این تابع یک رشته به عنوان ورودی دریافت می‌کند و ابتدا با استفاده از تابع pr-erase پرانتزهای اضافی را حذف کرده و سپس به دنبال اولین اپراتور آزاد (اپراتوری که داخل هیچ پرانتزی نباشد) می‌گردد، آن اپراتور را نقیض کرده و برای نمادهای قبل و بعد از آن اپراتور، دوباره خودش را به صورت بازگشتی صدا می‌زند.

تابع negation-match:

این تابع یک رشته به عنوان ورودی دریافت می‌کند و در آن رشته دنبال علامت‌های نقیض که پشت هم تکرار شده‌اند، می‌گردد و آن ها را می‌شمارد. اگر تعدادشان فرد بود، همه‌ی آن‌ها را حذف کرده و یک علامت نقیض می‌گذارد و اگر تعدادشان زوج بود، فقط کل نقیض‌ها را حذف می‌کند.

تابع leaf-check:

این تابع یک رشته به عنوان ورودی دریافت می‌کند و یک Boolean بر می‌گرداند که به ما نشان می‌دهد این رشته برگ هست یا خیر.

تابع successor:

این تابع یک رشته و یک عدد به عنوان پدر آن رشته در درخت (که در شروع برنامه، این عدد صفر است) به عنوان ورودی دریافت می‌کند، پرانتزهای اضافی آن را حذف کرده و شروع به حرکت روی رشته می‌کند و به دنبال اپراتورهای آزاد می‌گردد:

۱. اگر به ∧ رسید، آن را تبدیل به "," می‌کند، سپس رشته را چاپ کرده و branchNum را به اضافه یک می‌کند.

۲. اگر به ∨ رسید، تابع or-string-help با ورودی رشته x و father که همان branchNum-۱ است، صدا می‌زند و در آخر branchNum را با یک جمع می‌کند.

۳. اگر به > رسید، تابع ifmaker را با دو متغیر رشته x و جایگاه > در x صدا می‌زند.

وقتی به آخر رشته رسید، دوباره به اول برمیگردد و تک‌تک رشته‌هایی که با "," جدا شده‌اند را به pr-erase داده. این تابع تا جایی اجرا می‌شود که تابع leaf-check به ما بگوید که به برگ رسیده‌ایم.

تابع or-string-help:

یک رشته x و پدرش را از ورودی دریافت می‌کند و می‌گردد و اپراتور ∨ که آزاد باشد را پیدا می‌کند. سپس چک می‌کند که این اپراتور برای کدام دو گزاره اتمی است. سپس آن دو را از رشته x جدا کرده و هر کدام را جدا جدا به این رشته اضافه می‌کند. در نتیجه در آخر، دو رشته به به عنوان خروجی می‌دهد.

و اول تابع successor برای یکی از آن‌ها فراخوانی می‌کند و سپس به branchNum یکی اضافه کرده و تابع successor را برای بعدی صدا می‌کند. دقت کنید که در هر دو حالت، پدر آن راس همان عددی است که در ورودی تابع or-string-help به آن دادیم. توجه کنید که ما هر پدر را در آرایه‌ی fatherNumArr، اضافه می‌کنیم و بعد از اینکه در تابع or-string-help برای بار دوم تابع successor را صدا زدیم، آخرین عضو این آرایه را حذف می‌کنیم.

تابع ifmaker:

این تابع یک استرینگ که شامل ">" است و جایگاه ">" در آن را به عنوان ورودی دریافت می‌کند و سپس فرم شامل ∨ آن را بر می‌گرداند.

اجرای برنامه:

یک رشته را به عنون ورودی از کاربر دریافت می‌کنیم و سپس با استفاده از توابعی که برای نقیض کردن نوشته شده، نقیض‌ها را اعمال کرده و سپس تابع successor را صدا می‌زنیم.