«روش تابلو»

مهیار محمدی متین - 610398166

تابستان 1399

مقدمه

گزارهء دلخواه A را در نظر بگیرید. می خواهیم همانگو بودن A را بررسی کنیم. یکی از روش های مرسوم، استفاده از جدول ارزش گزاره است. این روش برای بررسی یک گزاره، هر چند معتبر است، اما قابلیت ماشینی شدن به صورت الگوریتم را ندارد. در واقع کد جدول ارزش گزاره از اوردر $O(2^n)$ خواهد بود.(n تعداد انواع گزاره های اتمی است). پس به روش مناسب تری با اوردر پایین تر نیاز داریم. در اینجا روش تابلو استفاده می شود.

بخش اول : روش تابلو برای منطق گزاره ای

روش تابلو قرار است یک گزاره به عنوان ورودی بگیرد و به ما یک درخت تحویل دهد که گزاره را تجزیه کرده است. این درخت به صورت استقرایی و با استفاده از قوانین زیر ساخته می شود. (دقت کنید که رسم الخط و نحوه نمایش این درخت در منابع علمی مختلف، متفاوت است و نحوه نمایش در این متن کاملا طبق قرارداد شخصی نویسنده (م.محمدی) خواهد بود که البته تغییری در اصل الگوریتم ایجاد نخواهد کرد) :

در این الگوریتم، گزاره ها به دو دسته تقسیم می شوند.

- α – فرمول :

شامل گزاره هایی که بیرونی ترین عملگر دو موضعی (ادات منطقی) آن" ∧ " باشد یا قابل تبدیل به " ∧ " باشد. مثل $p \wedge q$, $\sim(p \vee q)$, $(p \to q) \wedge t$

این دسته از گزاره ها در درخت، شاخه جدید ایجاد نمی کنند و شاخه جاری را صرفا بلند تر می کنند. بر این أساس **α – فرمول‌ها** شامل این چهار فرمول خواهند بود:

1. $\sim\sim \phi$

$$\{\phi\}$$

2. $\phi_1 \wedge \phi_2$

$$\{\phi_1 , \phi_2\}$$

3. $\sim(\phi_1 \vee \phi_2)$

$$\{\sim\phi_1 , \sim\phi_2\}$$

4. $\sim(\phi_1 \rightarrow \phi_2)$

$\{\phi_1, \sim\phi_2\}$

- **$\boldsymbol{\beta}$ – فرمول :**

شامل گزاره هایی که بیرونی ترین عملگر دو موضعی (ادات منطقی) آن " ∨ " باشد یا قابل تبدیل به "∨ " باشد. مثل $(p \rightarrow q) \vee t$, $(p \vee q)$

این دسته از گزاره ها برخلاف $\boldsymbol{\alpha}$ – فرمول**ها** در درختان شاخه جدیدی به وجود می آورند.

بر این اساس $\boldsymbol{\beta}$ – فرمول**ها** به سه فرمول زیر تقسیم میشوند.

1. $\phi_1 \vee \phi_2$

$\{\phi_1\}$ $\{\phi_2\}$

2. $\sim(\phi_1 \wedge \phi_2)$

$\{\sim\phi_1\}$ $\{\sim\phi_2\}$

3. $\phi_1 \rightarrow \phi_2$

$\{\sim\phi_1\}$ $\{\phi_2\}$

حال به شرح الگوریتم ساخت تابلو می پردازیم:

الگوریتم ساخت تابلو:

فرض کنید گزاره ϕ به ما داده شده است. ابتدا بررسی می کنیم که این گزاره، α – فرمول است یا β – فرمول و بر اساس آن طبق هفت قاعدهء ذکر شده، یک مرحله جلو می رویم. در اینجا ما مجموعه ای داریم مثلا $\{\phi_1 , \phi_2\}$. کافی است گزاره ای انتخاب کنیم که اتمی نباشد و مراحل قبل را برایش تکرار کنیم. الگوریتم زمانی پایان می یابد که به مجموعه ای از لیترال ها برسیم. در اینجا اگر دو گزاره اتمی متناقض در مجموعه باشد، شاخه بسته و در غیر این صورت، شاخه باز است. تعریف باز و بسته بودن شاخه را به طور مفصل تر در ادامه خواهید دید.

قضیه: ساخت تابلو پایان پذیر است. (برای گزاره متناهی)

برهان: چون گزاره متناهی است، تعداد ادات منطقی آن نیز، متناهی است. از طرفی در ساخت درخت، چون تعداد ادات در هر مرحله کاهش می یابد (در α – فرمول ادات " $\wedge$ " و در β – فرمول ادات " $\vee$ " ، حذف می شوند) پس با پیشروی بر روی درخت، در نهایت تعداد ادات به صفر می رسد که در اینجا اصطلاحاً به **برگ** درخت رسیده ایم که مجموعه ای متشکل از لیترال هاست. چون با تعداد مراحل متناهی به برگ رسیده ایم، پس تابلو پایان پذیر است.

نشان دار کردن درخت:

حال که اثبات کردیم ساخت تابلو ها پایان پذیر است، به بررسی و تحلیل درخت می پردازیم.

در برگ های درخت کامل شده، مجموعه ای از لیترال ها وجود دارد. اگر این مجموعه سازگار باشد (شامل دو گزاره اتمی متناقض نباشد) آن شاخه از درخت شاخه باز و اگر ناسازگار باشد، آن را شاخه بسته گوییم.

شرط سازگاری برای برگ ψ : $\forall_{p \in Atom} \ ((p \in \psi \to \sim p \notin \psi) \wedge (\sim p \in \psi \to p \notin \psi))$

قرارداد میکنیم که برگ باز را با O و برگ بسته را با X نشان دهیم. در درخت زیر که طبق الگوریتم تابلو ساخته شده است، یک برگ بسته و یک برگ باز وجود دارد.

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

$\{p, \sim p \vee q\}$

$\{p, q\}$ $\{\sim p, q\}$

O X

نکته: فرض کنید α_1 و α_2 فرزندان (شاخه زیرین) α و β_1 و β_2 فرزندان گزاره β باشند.

آنگاه α ارضا شدنی است اگر و تنها اگر α_1 و α_2 ارضا شدنی باشند و β ارضاشدنی است اگر و تنها اگر β_1 یا β_2 ارضا شدنی باشند.

قضیه (قضیه درستی): اگر تابلو T برای گزاره ϕ (ریشه درخت) بسته باشد، ϕ ارضا نشدنی است.

برهان: روی عمق درخت گزاره ϕ استقرا می زنیم:

پایه استقرا: ϕ ϕ در هر دوی این حالات، طبق نکته فوق : ϕ ارضا ناپذیر

است. X X X (چون فرزندانش ارضا نشدنی اند)

فرض استقرا: فرض کنید در تمام درختان با برگ های بسته به عمق حداکثر n – 1 ، ریشه ارضا نشدنی باشد. ثابت خواهیم کرد برای درخت با عمق n هم این قضیه برقرار است.

1) اگر گزاره ϕ یک α – فرمول باشد : در این صورت فرزندی به نام ϕ' به عمق n-1 دارد که برگ های مشترک دارند. طبق فرض استقرا، چون طول ϕ' ، n – 1 است و برگ هایش بسته است (برگ هایش با ϕ مشترک است) پس طبق فرض استقرا ϕ' ارضا نشدنی است. طبق نکته گفته شده چون فرزند ϕ ارضا نشدنی است، خود ϕ هم ارضا نشدنی است.

2) اگر گزاره ϕ یک β – فرمول باشد : در این صورت دو فرزند ϕ' و ϕ'' دارد که ماکسیمم عمق هایشان n – 1 است. چون برگ های ϕ' و ϕ'' زیر مجموعه برگ ϕ است و تمام برگ های ϕ بسته است، پس برگ های ϕ' و ϕ'' نیز بسته است. پس طبق فرض استقرا، ϕ' و ϕ'' ارضا نشدنی هستند و چون فرزندان ϕ اند، پس طبق نکته گفته شده، ϕ نیز ارضا نشدنی است.

قضیه (قضیه تمامیت): اگر گزاره دلخواه A ارضا شدنی نباشد، آنگار هر تابلو کامل T آن، بسته است.

برای اثبات به یک تعریف و دو لم نیاز داریم:

تعریف: مجموعه از گزاره ها مانند U را هینتیکا گوییم، هر گاه:

- برای هر گزاره اتمی p که در گزاره ای V وجود داشته باشد، $p \notin U$ یا $\sim p \notin U$
- اگر $\phi \in U$ ، یک $\alpha -$ فرمول باشد، آنگاه $\alpha_2 \in U$ و $\alpha_1 \in U$
- اگر $\phi \in U$ ، یک $\beta -$ فرمول باشد، آنگاه $\beta_2 \in U$ یا $\beta_1 \in U$

لم اول: هر مجموعه هینتیکا مدل دارد.

برهان: تابع ارزش v را برای مجموعه هینتیکا U به صورت زیر تعریف می کنیم:

اگر $p \in U$ آنگاه $v(p) = 1$ ، اگر $\sim p \in U$ آنگاه $v(p) = 0$ و اگر $\sim p \notin U$ و $p \notin U$ آنگاه $v(p) = 1$

با استقرا روی پیچیدگی فرمول ریشه ثابت میکنیم تعببیر I تولید شده توسط v ، ریشه ϕ را درست تعبیر میکند. $I(\phi) = 1$

پایه استقرا: اگر $\phi = p$ یا $\phi =\sim p$ آنگاه $I(\phi) = 1$

فرض استقرا: فرض کنید برای گزاره هایی با n – 1 ادات دو موضعی حکم برقرار باشد، ثابت می کنیم برای گزاره های با n ادات دو موضعی هم برقرار است.

گزاره های ϕ' و ϕ'' را در نظر بگیرید به طوری که هر کدام حداکثر n – 1 ادات دو موضعی داشته باشند و جمع ادات دو موضعی آن ها نیز n – 1 باشد. چون هر کدام عضو مجموعه هینتیکا اند و کمتر از n ادات موضعی دارند. طبق فرض استقرا $I(\phi') =$

1 و $I(\phi'') = 1$. حال برای ϕ با n ادات دو موضعی این چهار حالت را داریم: $\phi = \phi' \wedge \phi''$ و $\phi = \phi' \vee \phi''$ و $\phi = \phi' \to \phi''$ و $\phi = \phi'' \to \phi'$

که در هر چهار حالت، $I(\phi) = 1$

لم دوم: در تابلوی T ، L را مجموعهء تمام گره های یک شاخه باز فرض کنید. آنگاه مجموعهء $U = \bigcup_{l \in U} v(l)$ یک مجموعه هینتیکا است.

برهان: فرض کنید $\phi \in U$ یکی از اعضای دلخواه U و یکی از گره های شاخه L در تابلو T است. اگر ϕ یک α – فرمول باشد، آنگاه α_2 و α_1 هر دو در زیر شاخه ϕ قرار دارند، پس عضو U هم هستند. (⁕ $\alpha_2 \in U$ و $\alpha_1 \in U$). اگر ϕ یک β – فرمول باشد، β_1 و β_2 در زیر شاخه L هستند (طبق تعریف تابلو) پس یکی از آن ها در U هست. (⁕⁕ $\beta_2 \in U$ یا $\beta_1 \in U$)

حال طبق ⁕ و ⁕⁕، مجموعه U دو شرط لازم هینتیکا بودن را دارد. کافی است ثابت کنیم برای هر گزاره اتمی p ، $p \notin U$ یا $\sim p \notin U$ تا مجموعه هینتیکا باشد.

برهان خلف: فرض کنید $p \in U$ یا $\sim p \in U$ آنگاه p و $\sim p$ در شاخه باز درختمان قرار دارد. چون شاخه L، بدون شاخه فرعی است پس p و $\sim p$ در برگ هم وجود دارند پس برگ بسته است. در حالی که فرض کردیم برگ شاخه L باز است و این موضوع تناقض است. پس U تمام شرایط هینتیکا بودن را داراست.

حال به اثبات قضیه تمامیت می پردازیم:

برهان(برهان خلف) : تابلوی T را یک تابلوی باز برای ریشه ϕ در نظر بگیرید. لزوماً شاخهء با نام L در تابلو وجود دارد که باز باشد. طبق لم دوم، اجتماع تمام گره های این شاخه $\mathrm{U} = \mathrm{U}_{l \in L} U(l)$ یک مجموعه هینتیکا است. بنابر لم اول، U مدل دارد. چون $\phi \in U$ پس ϕ ارضا شدنی است و این تناقض است. پس حکم ثابت میشود.

قضیه درستی و تمامیت: گزاره ϕ همانگو است، اگر و تنها اگر تابلو $\sim \phi$ بسته باشد.

بخش دوم: روش تابلو برای منطق مرتبه اول

مانند منطق گزاره ای باید به صورت استقرایی و با استفاده از قواعدی، مراحل ساخت تابلو را شرح دهیم. با این تفاوت که دو گروه جدید از فرمول ها اضافه شده اند.

- γ _ فرمول‌**ها:**

فرمول هایی که احکام کلی ارائه می کنند. " همه " یا " هیچ ".

در این فرمول ها، گره جدید همان قبلی می ماند اما فرمول جدیدی که در آن متغیر x با a جایگزین شده است، به گره اضافه می شود. به صورت کلی به این دو حالت تقسیم میشوند:

1. $\forall_x \phi(x)$

$$\{ \forall_x \phi(x) , \phi(a) \}$$

2. $\sim \exists_x \phi(x)$

$$\{ \sim \exists_x \phi(x) , \sim \phi(a) \}$$

نکته مهم : می بایست a ثابتی باشد که در شاخهء پدر ظاهر شده است.

- **δ _ فرمول‌ها:**

فرمول هایی که احکام جزئی ارائه می کنند."وجود دارد" یا "چنین نیست که به ازای هر" در این فرمول ها کافی است سور را حذف و متغیر x را با ثابت a جایگزین شود.

$$1.\ \exists_x\ \phi(x) \quad \Big| \quad \{\ \phi(a)\ \}$$

$$2.\ \sim \forall_x\ \phi(x) \quad \Big| \quad \{\sim\ \phi(a)\ \}$$

نکته مهم : میبایست a ثابتی باشد که در شاخهء پدر ظاهر نشده است.

حال مثل روش تابلو برای منطق گزاره ای، در هر مرحله، از مجموعه جاری ϕ را انتخاب می کنیم به طوری که ϕ اتمی نباشد. اگر تمامی اعضای $\mathrm{U}(l)$ اتمی یا نقیض اتمی (لیترال) بود، سازگاری مجموعه را بررسی میکنیم. اگر $\sim p$ و p در گزاره بود، بسته می شود و اگر γ _ فرمول داشته باشیم، $\mathrm{U}(l) = \mathrm{U}(l')$ آنگاه باز است.

قضیه درستی: اگر تابلو T برای ϕ (در منطق مرتبه اول) در تمام شاخه ها بسته باشد، آنگاه ϕ ارضا ناپذیر است.

برهان:

1- اگر ϕ یک α – فرمول یا β – فرمول باشد، اثبات مانند قبل است.

2- اگر γ ـ فرمول باشد، آنگاه فرض کنید $\mathrm{U}(n) = U_0 \cup \{ \forall_x A(x)\}$ و فرزندش

$U(n') = U_0 \cup \{\forall_x A(x) , A(a)\}$:

برهان خلف: فرض کنید $\mathrm{U}(n)$ ارضا پذیر باشد. پس مدلی مثلا مانند m دارد. پس $m \vDash \forall_x A(x)$ چون $\mathrm{U}(n')$ یک A(a) اضافه تر دارد و a ثابتی است که در U(n) ظاهر شده بود(تعریف گسترش γ ـ فرمول‌ها) پس m مدل $\mathrm{U}(n')$ نیز هست. پس شاخه باز است.

3- اگر ϕ یک δ ـ فرمول باشد، آنگاه $\mathrm{U}(n) = U_0 \cup \{ \exists_x A(x)\}$ و فرزندش $U(n') = U_0 \cup \{A(a)\}$ را فرض کنید.

برهان خلف: فرض کنید U(x) ارضا شدنی و m یک مدل آن باشد. پس $d \in M$ موجود است که $m \vDash A(d)$. با تعبیر d برای نماد ثابت a، مدل m' بدست می آید که مدلی برای $\mathrm{U}(m')$ است. برای اثبات قضیه تمامیت، باید ابتدا ساخت روش مند تابلو را شرح دهیم.

ساخت روش مند تابلو : در این روش در هر گره، علاوه بر خود فرمول، ثابت های آن فرمول را هم داریم. به این صورت که هر گره دوتایی $(\mathrm{U}(l) , \mathrm{C}(l))$ برای گره L را داراست که $U(l)$ مجموعه فرمول ها و $\mathrm{C}(l)$ مجموعه ثابت ها می باشد. برای ریشه به طور مثال داریم: $(\{A\},\{a_1 , a_2 , \dots , a_n\})$ که $\{a_1 , a_2 , \dots , a_n\}$ مجموعهٔ

ثابت ها ظاهر شده در A است. اگر A دارای ثابت نباشد، به طور دلخواه a را انتخاب و نسبت می دهیم. حال در هر مرحله $\mathrm{U}(l)$ را بررسی می کنیم. اگر شامل لیترال ها بود، سازگاری آن را بررسی می کنیم و اگر نبود، فرمولی مثل ϕ را انتخاب می کنیم که اتمی نباشد.

α – فرمول و β – فرمول: اگر ϕ یک α – فرمول یا β – فرمول باشد، برای گره l'، $U(l')$ مانند قبل تعریف می شود و $\mathrm{C}(l')$ همان $\mathrm{C}(l)$ مرحله قبل است.

قاعده δ : اگر ϕ یک δ ـ فرمول باشد، در گره l' به جای δ، $\delta(a)$ قرار داده و قرار دهید: $C(l') = C(l) \cup \{a\}$

قاعده γ : فرض کنید $\{\gamma_1 , \gamma_2 , \ldots , \gamma_n\}$ مجموعه γ ـ فرمول‌های عضو $\mathrm{U}(l)$ باشند و $\mathrm{C}(l) = \{ \alpha_1 , \alpha_2 , \ldots , \alpha_K\}$ در گره جدید، $C(l) = C(l')$ اما برای $\mathrm{U}(l')$ باید مجموعه $\bigcup_{i=1}^{n} \bigcup_{j=1}^{k} \{\gamma_i(a_j)\}$ را به $\mathrm{U}(l)$ اضافه کنیم.

با توجه به الگوریتم فوق، روش تابلو برای منطق مرتبه اول، یک روش جستجو موثر برای یافتن مدل نیست زیرا ممکن است شاخه نامتناهی ایجاد کند.

هدف ما از تعریف روش هدفمند اثبات قضیه تمامیت در حالاتی است که شاخهء نامتناهی نداشته باشیم. اما همچنان با اثبات قضیه درستی با الگوریتم قبلی (غیر هدفمند) می توانیم اعتبار یک فرمول را با نشان دادن اینکه تابلو برای نقیض آن بسته می شود، بسنجیم.

برای اثبات تمامیت به یک تعریف و دو لم نیاز داریم:

تعمیم تعریف مجموعه هینتیکا: همان قواعد منطق گزاره ای می باشد اما با دو قانون جدید:

1) اگر $\phi \in U$ یک γ ـ فرمول باشد، برای هر ثابت a که در فرمولی در U ظاهر شده، $\gamma(a) \in U$

2) اگر $\phi \in U$ یک δ ـ فرمول باشد، آنگاه ثابت a وجود دارد که $\delta(a) \in U$.

لم اول: فرض کنید l یک شاخه باز از تابلو باشد و $U = \bigcup_{n\in l} U(n)$ آنگاه U هینتیکا است.

برهان: اگر $\phi \in U$ اتمی، α – فرمول یا β – فرمول باشد مانند قبل اثبات میشود.

اگر ϕ یک γ ـ فرمول باشد و a یک ثابت دلخواه ظاهر شده در فرمولی در U باشد آنگاه گره ای مانند m عضو L داریم که $a \in C(m)$ مجموعه γ ـ فرمول ها و ثابت ها در طول یک شاخه غیر کاهشی هستند پس برای $k = \max(n, m)$ داریم $\phi \in U(k)$ و $a \in C(k)$ و $k' > k$ وجود دارد که $\gamma(a) \in U(k') \subseteq U$

لم دوم: هر مجموعه هینتیکا U مدل دارد.

قضیه تمامیت: اگر A معتبر باشد، آنگاه تابلو کامل T برای نقیض A بسته می شود.

برهان: فرض کنید تابلوی A~ باز است (برهان خلف) بنابر لم اول،

$U = \cup_{n\in l}\ U(n)$ یک مجموعه هینتیکا است و بنابر لم دوم، مدل دارد چون $\sim A\ \in$ U است. پس A نامعتبر است و این موضوع تناقض است. بنابراین فرض خلف باطل و حکم ثابت است.

بخش سوم: توضیح الگوریتم پروژه

ورودی و خروجی:

این برنامه یک گزاره عضو prop(از منطق گزاره‌ای) و همچنین گزاره های اتمی موجود در آن را ورودی می‌گیرد و با دو روش متفاوت، تعیین می‌کند که این گزاره همانگو هست یا نه.

روش جدول ارزش: در این روش تمام حالت های ممکن برای تابع ارزش v برای هر گزاره اتمی در نظر گرفته شده و در خروجی جدول ارزش گزاره چاپ می‌شود. این روش با وجود سادگی از اوردر زمانی نمایی O(2n) است(که در آن n تعداد گزاره های اتمی متفاوت است). همچنین زمان اجرای این روش برحسب میکروثانیه نیز محاسبه خواهد شد.

روش الگوریتم تابلو: در این روش درخت مربوط به نقیض گزاره چاپ شده و برگ هایش نشان‌دار می‌شود. اگر تمام برگ ها بسته بودند، خود گزاره همانگو است. این برنامه از اوردر خطی O(n) است (که در آن n تعداد اپراتور های منطقی است) زمان اجرای این برنامه هم برحسب میکروثانیه محاسبه می‌شود.

الگوریتم روش جدول ارزش:

- تابع remove_negation:

این تابع را قبل از به کاربردن گزاره استفاده می‌کنیم که برای حذف نقیض های پشت پرانتز است. در واقع هدفش پخش کردن نقیض های پشت پرانتز است به طوری که نقیض صرفا پشت گزاره های اتمی باشد نه جای دیگر.

- تابع validate:

کافی است یک شمارنده باینری با تعداد ارقام n(که در آن n تعداد گزاره های اتمی متفاوت است) داشته باشیم که در آن ۱ نماینده گزاره درست و ۰ نماینده گزاره نادرست است. برای مثال برای گزاره های p , q , r شمارنده باینری ۳ رقمی داریم که از ۰۰۰(هر سه گزاره نادرست) شروع و به ۱۱۱(هر سه گزاره درست) ختم می‌شود.(مثلا ۰۱۱ یعنی q , r درست و p نادرست است.)

در هر مرحله ابتدا با تابع put ارزش‌دهی می‌کنیم و به جای گزاره ها ۰ و ۱ قرار می‌دهیم. سپس با تابع check ارزش گزاره را تعیین می‌کنیم.

- تابع put:

صـــرفا روی اســـترینگ حرکت می‌کند و به جای گزاره های اتمی‌اش ارزش دهی که در تابع قبل شده بود را جایگزین گزاره اتمی می‌کند.

- تابع check:

به صورت بازگشتی با تعاریف ساده خود اپراتور ها چک می‌کنیم اگر ۱ بود true و اگر ۰ بود false برمی‌گردانیم. اگر ^ داشتیم، ارزش گزاره چپ و راست ^ باید هر دو درست باشـــد(برای اینکه ببینیم چه ارزشـــی دارند دوباره به خود تابع می‌دهیم) تا true برگردانیم. اگر v داشـتیم، گزاره چپ یا راسـت باید درسـت باشـد و اگر شـرط داشـتیم، ارزش گزاره چپ باید غلط یا ارزش گزاره راسـت درسـت باشـد. اگر این حالت ها نبود و true برگردانده نشد false بر می‌گرداند.

-توضیحات نهایی:

درآخر هرمرحله جدول به صـــورت ســـطربه‌ســـطر چاپ شـــده و اگرهمه جدول درســـت بود،گزاره همانگو است.

الگوریتم روش تابلو:

-توضیحات کلی:

یک متغییر گلوبال به نام branch_num داریم که در هر مرحله که شـــاخه جدیدی ایجاد می‌شـود به آن اضـافه می‌شـود و مسـئول شـماره‌دهی به شـاخه ها اسـت. از طرفی الگوریتم به این شـــکل اســـت که یک وکتور دو بعدی داریم که در آن جواب ها ذخیره می‌شـود. هر خط خروجی برنامه نماینده یک شـاخه از درخت است که مرحله به مرحله اســـترینگ مربوط به هر گره در آن ذخیره شـــده اســـت. همچنین ابتدای هر شـــاخه شـناسـنامه دار شـده اسـت(شـماره خودش و پدرش). در نهایت جواب ها از این وکتور برداشته و چاپ می‌شوند.

- تابع negation:

این تابع به صـورت بازگشـتی قوانین اعمال نقیض را انجام می‌دهد. اگر گزاره اتمی باشـد نقیضـش، و اگر نقیض گزاره اتمی باشـد خود گزاره را برمی‌گرداند. اگر اپراتور داشـتیم، کافی اســـت خارجی ترین اپراتور را عوض کنیم(اگر ^بود v بذاریم و برعکس) و چپ و راســت آن را هم دوباره به خود تابع negation بدهیم. اگر شـــرط بود نیز به صـــورت p^~q بنویسیم.

-تابع expand_branch:

این تابع یک گزاره و شـــماره پدر آن گزاره را ورودی می‌گیرد و شـــاخه را تا جایی که شاخه جدید ایجاد نشود(به v برخورد نکند) و یا به برگ نرسد گسترش می‌دهد و در هر مرحله به جای ^ کاما قرار می‌دهد. اگر به v برخورد کردیم کافی اســت گزاره های این گره را همراه با چپ و راســـت v دوباره به خود تابع expand_branch بدهیم تا شـاخه مربوطه را گســترش دهند. با این تفاوت که قبل از ورودی دادن شــماره شــاخه جاری(branch_num) را یک واحد افزایش داده و شـــاخه فعلی را به عنوان پدر به تابعمان ورودی دهیم.

-تابع display:

وظیفه نمایش درخت بدســت آمده در تابع expand_branch را دارد و علاوه بر آن به کمک تابع isopen بررسی می‌کند که برگ باز است یا بسته و بر اساس آن برگ هارا نشان دار شده چاپ می‌کند.