3. Tablo algoritms

16.12

Tablo (tableaux) metodi 1955.gadā izgudroja Evert Willem Beth (1908-1964), plašāka biogrāfija un portrets.

Sk. arī Method of analytic tableaux (Wikipedia).

Šī metode risina to pašu uzdevumu, ko **rezolūciju metode** (sk. kursa grāmatas sadaļu 5.5): noskaidrot, vai no dotajām (zināšanu bāzes) formulām $F_1, \dots, F_n$ seko (vaicājuma) formula G.

Šo metodi varēs izmantot arī, lai parādītu, ka kāda formula ir izvedama klasiskajā loģikā. Tagad mums būs 3 metodes, kā to izdarīt: a) izvešana no aksiomām, b) pierādīt, ka formula ir patiesa visās interpretācijās, c) tablo algoritms.

Saskaņā ar Gēdela Pilnības Teorēmu, šis uzdevums ir ekvivalents uzdevumam: noskaidrot, vai formulu G var izvest no hipotēzēm $F_1, \dots, F_n$, izmantojot klasisko loģiku:

$$[L_1 - L_{15}, MP, Gen]: F_1, \dots, F_n \vdash G \quad .$$

Tāpat kā rezolūciju metodē, uzdevumu risināsim **no pretējā**: pievienosim hipotēzēm $F_1, \dots, F_n$ formulas G negāciju, iegūstot kopu $F_1, \dots, F_n, \neg G$.

Teorēma 1. $[L_1 - L_{15}, MP, Gen]: F_1, \dots, F_n \vdash G$ tad un tikai tad, ja formulu kopa $F_1, \dots, F_n, \neg G$ ir **neizpildāma** (t.i. neeksistē interpretācija, kurā visas šīs formulas ir patiesas).

Pierādījums. 1) Aplūkojam (no pretējā) interpretāciju, kurā visas formulas $F_1, \dots, F_n, \neg G$ ir patiesas. Bet tā kā G seko no $F_1, \dots, F_n$, tad šajā interpretācijā patiesa ir arī G. Pretruna, tātad kopa $F_1, \dots, F_n, \neg G$ ir neizpildāma.

2) Ja kopa $F_1, \dots, F_n, \neg G$ ir neizpildāma, un kādā interpretācijā patiesas ir visas formulas $F_1, \dots, F_n$, tad šajā interpretācijā formulai $\neg G$ ir jābūt aplamai, t.i. G jābūt patiesai. Tātad G seko no $F_1, \dots, F_n$. Q.E.D.

Kopas $F_1, \dots, F_n, \neg G$ vietā varam aplūkot vienu formulu $F_1 \wedge \dots \wedge F_n \wedge \neg G$. Līdz ar to mūsu uzdevums (teorētiski) reducējas uz uzdevumu: **dota kādas predikātu valodas formula F, ir jānoskaidro, vai tā ir izpildāma, vai nav.**

Tablo metodes pirmajā solī ir jāiegūst formulas F t.s. **negāciju normālforma**, kurā ir tikai kvantori, konjunkcijas un disjunkcijas (bet nav implikāciju!), bet negācijas atrodas tikai atomāro formulu priekšā.

Piemērs.

$$\forall x (\neg P(x) \vee \exists y (Q(y) \wedge \forall z (\neg R(x, z) \vee S(z)))) \quad .$$

Negāciju normālforma (klasiskajā loģikā) ir ekvivalenta formulai F, un tās iegūšanas algoritms ir aprakstīts kursa grāmatas sadaļas 5.3 sākumā.

Ja apskatām kaut vai tikai izteikumu loģiku, tad "visiem zināms, ka nav zināms" neviens algoritms, kas izteikumu formulas izpildāmību (t.i. vai tās atomiem var piešķirt tādas vērtības, lai formula kļūtu "patiesa") var noteikt laikā, kas mazāks par 2^{cn}, kur n ir formulas garums, un c – konstante (tā atkarīga no algoritma "kvalitātes"). Tas ir t.s. **worst-case estimate** (sliktākā gadījuma novērtējums) – jebkuram no zināmajiem algoritmiem gandrīz katram n atradīsies formula garumā n, kuras izpildāmības noteikšanai algoritms patērēs 2^{cn} laika vienību.[Ievērosim, ka "vairums formulu" garumā n satur Cn dažādu atomu.]

Tablo algoritmam šis worst-case vērtējums ir $P(n)2^{cd}$, kur n ir formulas garums, P – polinoms, c – konstante, bet d – **disjunkciju skaits** formulā. Bet tā kā "vairumam formulu" garumā n, disjunkciju skaits ir Cn, tad, protams, arī tablo algoritmam worst-case vērtējums iznāk tas pats 2^{cn}.

Bet ja nu Jūsu specifiskā praktiskā nozare ir tāda, ka tajā izmantojamās izteikumu formulas satur vidēji nevis Cn disjunkcijas, bet daudz mazāk, piemēram, C log n? Tad, no Jūsu nozares viedokļa, tablo algoritms formulai ar garumu n dos atbildi laikā $P(n)2^{cd}$, kur 2^{cd} vairs nebūs eksponenta no n, bet polinoms (ja d = C log n, tad $2^{cd} <= n^k$, kur k – konstante, t.i. laika vērtējums iznāk polinomiāls).

Tātad labi ir tie algoritmi, kas spēj ievērot praktiski nozīmīgo gadījumu īpatnības. Tablo algoritms tāds esot. Bet labi zināmais algoritms, kas pārbauda formulas patiesumu visām iespējamām tās atomu vērtībām, tāds nav. Jo tā darba laiks ir $P(n)2^{cm}$, kur n ir formulas garums, P – polinoms, c – konstante, bet m – dažādo atomu skaits formulā. Bet, diemžēl, dažādo atomu skaits "praktiskajās" formulās, kas vajadzīgas reālās zināšanu bāzēs, ir ļoti liels.

3.1. Tablo algoritms izteikumu valodai

[Kāpēc valodai? Tāpēc ka pētīsim nevis teorijas aksiomas, bet tikai formulu izpildāmību.]

Tablo algoritma pamatideja vislabāk ir redzama visvienkāršākajā piemērā – izteikumu valodai.

Izteikumu valodas primitīvi ir atomi (mainīgie izteikumi) p, q, ... (katrs no tiem var pieņemt vērtības "patiess" un "aplams") un saikļi $\neg, \wedge, \vee, \rightarrow$ **.**

Algoritma mērķis: noteikt, **vai dotā formula ir izpildāma** (t.i. vai tās atomiem var piešķirt tādas vērtības, lai formula kļūtu "patiesa"). Piemēram, $(p \rightarrow q) \rightarrow q$ ir izpildāma formula, $\neg(p \rightarrow p)$ nav izpildāma.

Algoritma ideja: reducēt dotās formulas izpildāmību uz tās sastāvdaļu kopas vienlaicīgu izpildāmību. Piemēram, formula $A \wedge B$ ir izpildāma tad un tikai tad, ja formulas A, B ir vienlaicīgi izpildāmas (t.i. ar tām pašām atomu vērtībām). Formula $A \vee B$ ir izpildāma tad un tikai tad, ja ir izpildāma vismaz viena no formulām A, vai B.

Ar implikācijām un negācijām tik labi nesanāk, tāpēc doto formulu vispirms reducēsim t.s. **negāciju normāformā**, kas ekvivalenta dotajai formulai (piebilde zinātājiem – ekvivalenta klasiskajā loģikā).

Izmantojam iespējas, ko dod **Substitūcijas teorēma 1**.

Vispirms implikācijas aizstājam ar negācijām un disjunkcijām, izmantojot likumu:

$$(A \to B) \leftrightarrow \neg A \vee B$$

Pēc tam izmantojam de Morgana likumus, "nonesot" negācijas zīmes līdz atomiem:

$$\neg(A \vee B) \leftrightarrow \neg A \wedge \neg B \ ,$$
$$\neg(A \wedge B) \leftrightarrow \neg A \vee \neg B \ .$$

Tādā veidā visas negācijas "sakrājas" atomu priekšā, piemēram, $\neg\neg\neg\neg\neg p \vee \neg\neg q$. Tāpēc beigās izmantojam dubultās negācijas likumu, atstājot katram atomam vienu vai nevienu negāciju:

$$\neg\neg A \leftrightarrow A \ .$$

Šādi apstrādātu formulu sauc par **negāciju normālformu**. Šādas formulas satur tikai konjunkcijas un disjunkcijas un atomus ar vai bez negācijām.

Piemērs. Dota formula $(p \to q) \to q$. Atbrīvojamies no implikācijas: $\neg(\neg p \vee q) \vee q$. Lietojam de Morgana likumu: $(\neg\neg p \wedge \neg q) \vee q$. Atbrīvojamies no dubultās negācijas: $(p \wedge \neg q) \vee q$. Esam ieguvuši ekvivalentu formulu, kas ir negāciju normālformā – formula satur tikai konjunkcijas un disjunkcijas, bet negācijas (ja tādas ir) atrodas tikai pie atomiem.

Teorēma. Dotās formulas redukcijai negāciju normālformā vajadzīgs **lineārs laiks** (no formulas garuma).

Uzdevums. Pārliecinieties detalizēti, ka tas tā tiešām ir. Kā redukcijas laikā mainās formulas garums? (Pirmkārt, pirmie trīs minētie likumi būs jāizmanto pavisam ne vairāk kā 2i+k+d reizes, kur i, k, d ir attiecīgi implikāciju, konjunkciju un disjunkciju skaits formulā. Otrkārt, katrs šo triju likumu lietojums palielina saikļu skaitu formulā tikai par 1.)

Nedeterminētais tablo algoritms

Tālāk sekojošo algoritma daļu visvieglāk ir aprakstīt kā nedeterminētu procesu, kas pārveido formulu sarakstu:

1) Sākumā sarakstā ir tikai dotā formula (negāciju normālformā).

2) Ja saraksta pirmā "neatomārā" formula ir A&B, tad to no saraksta izmetam, bet saraksta beigās pievienojam divas formulas A, B. (Vārdu "neatomārs" šeit liekam pēdiņās, jo par "atomārām" formulām šeit jāuzskata gan atomi, gan atomi ar negācijām.)

3) Ja saraksta pirmā "neatomārā formula" ir AvB, tad to no saraksta izmetam, bet saraksta beigās pievienojam tikai vienu formulu – formulu A vai formulu B (te mums ir **nedeterminēta izvēle)**.

4) Iterējam soļus 2) un 3), bet ja tas nav iespējams, t.i. ja sarakstā ir tikai "atomāras" formulas (t.i. tikai atomi ar vai bez negācijām), tad darbu beidzam.

Algoritma izpildāmo soļu skaits (līdz tas apstāsies) nepārsniegs konjunkciju un disjunkciju kopskaitu dotajā formulā (precīzāk, tās negāciju normālformā). Varam ievērot arī, ka solis 2) samazina saraksta formulu summāro garumu par 1, bet solis 3) – vismaz par 2.

Pēc apstāšanās iespējamas divas situācijas:

a) Formulu sarakstā kāds atoms p sastopams divreiz - ar un bez negācijas: p, ~p. Tad darba rezultātu sauksim par **negatīvu**.

b) Formulu sarakstā katrs atoms p sastopams tikai vienreiz - vai nu kā p, vai kā ~p. Tad darba rezultāts sauksim par **pozitīvu**.

Parādīsim, ka ja esam ieguvuši pozitīvu rezultātu, tad sākumā dotā formula ir izpildāma. Šim nolūkam mums jāpiešķir formulas atomiem vērtības "patiess" vai "aplams" tādā veidā, lai formula kļūtu patiesa. Rīkojamies tā: ja beigu sarakstā atoms p parādās bez negācijas, tad tam piešķirsim vērtību "patiess", ja ar negāciju - tad "aplams". Ar tādām atomu vērtībām sākumā dotā formula tiešām kļūst patiesa. Pierādījums: Pēc vērtību piešķiršanas, beigu stāvoklī visas saraksta formulas (tās visas ir atomāras - ar vai bez negācijām) ir patiesas. Izdarot soļus 2) un 3) "uz atpakaļ", redzam, ka visas saraksta formulas ir patiesas visos stāvokļos, t.i. arī sākuma stāvoklī, kad mums ir tikai dotā formula negāciju normālformā.

Tātad mūsu nedeterminētais algoritms ir korekts (sound) - ja tas izdod pozitīvu rezultātu, tad dotā formula ir izpildāma.

Bet kā ar algoritma pilnību (completeness), t.i. **ja dotā formula ir izpildāma, vai tad kāds no mūsu nedeterminētā algoritma izvēles "ceļiem" dos pozitīvu rezultātu**? [Kā "strādā" nedeterminēts algoritms?] Pieņemsim, ka dotā formula ir patiesa kādam atomu vērtību komplektam X, un palūkosim, kā varētu strādāt mūsu nedeterminētais algoritms. Ja kādā brīdī formula A&B (un visas pārējās tā brīža saraksta formulas) ir patiesa komplektam X, tad šim pašam komplektam patiesa būs gan A, gan B, t.i. arī nākošajā solī visas saraksta formulas būs patiesas. Ja formula AvB (un visas pārējās saraksta formulas) ir patiesa komplektam X, tad vienai no abām formulām arī ir jābūt patiesai šim pašam komplektam: formulai A vai formulai B. **Šo formulu tad**

arī izvēlēsimies solī 3). Tad arī nākošajā solī visas saraksta formulas būs patiesas. Tātad ar šādām izvēlēm algoritma "ceļā" formulu sarakstā visas formulas vienmēr būs patiesas komplektam X. Un tātad beigās mums būs komplektam X patiesu atomāru (ar vai bez negācijām) formulu saraksts. Tādā saraksta katrs atoms var ieiet tikai vienā veidā - bez negācijas, vai ar negāciju. Tātad šai gadījumā algoritms dos pozitīvu rezultātu. Tas nozīmē, ka mūsu algoritms ir **pilnīgs (complete)**.

Algoritma ceļa garums un izmantotā atmiņa jebkurā gadījumā ir lineāri pēc dotās formulas garuma, bet rēķināšanas laiks - ne vairāk kā kvadrātisks.

Uzdevums. Pārliecinieties detalizēti, ka tas tā tiešām ir.

[Esam "lieku reizi" pierādījuši, ka izteikumu valodas formulu izpildāmības problēma (t.s. problēma SAT) pieder klasei NP.]

Prof. Gunta Bārzdiņa pirmais eksperiments, programmējot valodā Prolog tablo algoritmu izteikumu valodai.

Determinētais tablo algoritms

Protams, reālai izmantošanai nedeterminēts algoritms neder - to vajag "determinizēt".

Determinētais algoritms, realizē to pašu ideju (ko tikko aplūkotais nedeterminētais algoritms), bet tas "**būvē koku**", kura virsotnēs ir formulas. Šis algoritms kaut kādā secībā apstrādā koka virsotnes, "audzējot" klāt arvien jaunas virsotnes.

1) Koka saknes virsotnē liekam doto formulu (negāciju normālformā). Ja formulas vietā ir dota formulu kopa, tad koka saknē liekam kopas formulu konjunkciju.

2) Ja virsotnē ir formula $A \wedge B$, tad **katram zaram, kas iet caur šo virsotni**, "pieaudzējam" turpinājumu ar 2 virsotnēm A un B:

$A \wedge B$ ----- ...-----$-A - B$.

Un virsotni $A \wedge B$ atzīmējam kā "apstrādātu".

3) Ja virsotnē ir formula $A \vee B$, tad **katram zaram, kas iet caur šo virsotni**, "pieaudzējam" sazarojumu – attiecīgi ar virsotni A, un ar B:

$A \vee B$ -----...-----
$|- A$
$|- B$

Un virsotni AvB atzīmējam kā "apstrādātu".

4) Iterējam soļus 2) un 3), kā nākošo apstrādājamo virsotni izvēloties jebkuru no tām "neapstrādātajām" virsotnēm, kas satur "neatomāras" formulas un kas

koka saknei ir vistuvāk. Kad vairs nav "neapstrādātu" virsotņu ar "neatomārām" formulām, darbu beidzam. (Vārdu "neatomārs" liekam pēdiņās, jo par "atomārām" formulām šeit jāuzskata gan atomi, gan atomi ar negācijām.)

Piemērs.

Formula $(p \vee \neg q) \wedge (q \vee \neg r)$ jau ir negāciju normālformā. Tablo algoritms tai izveidos šādu koku:

Vispirms apstrādājam konjunkciju un atzīmējam koka sakni kā apstrādātu:

$$(p \vee \neg q) \wedge (q \vee \neg r) \text{---- } p \vee \neg q \text{ ---- } q \vee \neg r \ .$$

Pēc tam apstrādājam $p \vee \neg q$, koku aiz $q \vee \neg r$ sazarojot uz p un uz $\neg q$.

Atliek apstrādāt $q \vee \neg r$, zaru aiz p sazarojot uz q un $\neg r$, un arī zaru aiz $\neg q$ tāpat sazarojot uz q un $\neg r$.

Nu mums ir izveidojies koks ar 4 zariem:

$$(p \vee \neg q) \wedge (q \vee \neg r) \text{---- } p \vee \neg q \text{ ---- } q \vee \neg r \text{ ---- } p \text{ ---- } q$$
$$(p \vee \neg q) \wedge (q \vee \neg r) \text{---- } p \vee \neg q \text{ ---- } q \vee \neg r \text{ ---- } p \text{ ---- } \neg r$$
$$(p \vee \neg q) \wedge (q \vee \neg r) \text{---- } p \vee \neg q \text{ ---- } q \vee \neg r \text{ ---- } \neg q \text{ ---- } q$$
$$(p \vee \neg q) \wedge (q \vee \neg r) \text{---- } p \vee \neg q \text{ ---- } q \vee \neg r \text{ ---- } \neg q \text{ ---- } \neg r$$

Piemēra beigas.

Šāda algoritma darba rezultātā vienmēr izveidosies **koks ar galīgu skaitu virsotņu**. Kāpēc ar galīgu skaitu? Pirmkārt, visi koka zari ir galīgi. Tiešām, aplūkojam vienu no zariem un spriežam tāpat kā nedeterminētā algoritma gadījumā: katrs soļu 2), 3) lietojums samazina zarā esošo "neapstrādāto" formulu kopgarumu par 1 vai par 2. Bezgalīgi tā turpināties nevar, tāpēc visi koka zari ir galīgi. Otrkārt, koks katrā virsotnē sazarojas ne vairāk kā divos zaros, tātad n-jā līmenī tam būs ne vairāk kā 2^n virsotņu. Ja tādā kokā būtu bezgalīgi daudz virsotņu, tad ejot pa līmeņiem uz priekšu, mēs varētu izveidot bezgalīgu koka zaru, bet tas, kā tikko noskaidrojām, nav iespējams.

Piezīme. Šeit mēs būtībā esam pierādījuši arī t.s. Deneša **Kēniga** (Kēniga jaunākā) **lemmu**: ja koks katrā virsotnē sazarojas tikai galīgā skaitā zaru, un kokā ir bezgalīgi daudz virsotņu, tad tajā eksistē vismaz viens bezgalīgs zars.

Tātad pienāks brīdis, kad neviens no soļiem 2), 3) nebūs izpildāms, t.i. mūsu algoritms darbu beigs. Ko tad mēs varētu iesākt ar algoritma darba rezultātu - minēto koku ar galīgu skaitu virsotņu? Dažos koka zaros viens atoms var parādīties abos veidos – ar negāciju un bez tās. Tādus zarus pieņemts saukt par **aizvērtiem zariem** (tie atbilst nedeterminētā algoritma negatīvajiem rezultātiem). Pārējie zari tad jāsauc par **atvērtiem zariem** - tajos katrs atoms ir sastopams vai nu tikai ar negāciju, vai tikai bez tās (šie zari atbilst

nedeterminētā algoritma pozitīvajiem rezultātiem).

Piemērs. Augšminētājā piemērā mums ir iznākuši 3 atvērti zari (pirmais, otrais un ceturtais) un viens aizvērts (trešais).

Katrs no atvērtājiem zariem dod **interpretāciju, kas dara patiesas visas šajā zarā esošās formulas** – tātad arī koka sakne esošo formulu, kurās dēļ koku būvējām:

$(p \vee \neg q) \wedge (q \vee \neg r)$ ---- $p \vee \neg q$ ---- $q \vee \neg r$ ---- p ---- q ; $\boldsymbol{p=1; q=1}$
$(p \vee \neg q) \wedge (q \vee \neg r)$ ---- $p \vee \neg q$ ---- $q \vee \neg r$ ---- p ---- $\neg r$; $\boldsymbol{p=1; r=0}$
$(p \vee \neg q) \wedge (q \vee \neg r)$ ---- $p \vee \neg q$ ---- $q \vee \neg r$ ---- $\neg q$ ---- q
$(p \vee \neg q) \wedge (q \vee \neg r)$ ---- $p \vee \neg q$ ---- $q \vee \neg r$ ---- $\neg q$ ---- $\neg r$; $\boldsymbol{q=0; r=0}$

Tātad mūsu sākotnējā formula $(p \vee \neg q) \wedge (q \vee \neg r)$ ir izpildāma (piemēram, paņemam no pirmā zara: p=1 un q=1).

Teorēma 2. Ja tablo algoritma uzbūvētajā kokā ir kaut viens atvērts zars, tad dotā formula ir izpildāma. Un otrādi, ja kokā visi zari ir aizvērti, tad dotā formula ir neizpildāma.

Otrs piemērs. Gribam pierādīt, ka $p \to q; q \to r \mid\!- p \to r$. Pievienojam hipotēzēm $p \to q; q \to r$ izvedamās formulas negāciju: $\neg(p \to r)$:

$p \to q; q \to r; \neg(p \to r)$.

Visas 3 formulas pārveidojam negāciju normālformā:

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

$\neg p \vee q; \neg q \vee r; \neg\neg p \wedge \neg r$;

$\neg p \vee q; \neg q \vee r; p \wedge \neg r$.

Būvējam koku. Izrādās, ka **visi zari ir aizvērti**. Tātad formulu kopa

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

ir neizpildāma, un saskaņā ar Teorēmu 1: $p \to q; q \to r \mid\!- p \to r$ (klasiskajā izteikumu loģikā, protams).

Domas par šo situāciju vispārīgajā gadījumā palīdz sakārtot Jākko Hintikka lemma. Tās pierādījums ir triviāls, bet netriviāla ir pati ideja šādu lemmu formulēt.

J. Hintikkas lemma izteikumu valodai. Ja negāciju normālformā esošu izteikumu formulu kopa:

a) līdz ar katru savu formulu $A \wedge B$ satur arī abas formulas A, B;
b) līdz ar katru savu formulu $A \vee B$ satur vismaz vienu no formulām A, B;
c) nevienam atomam p nesatur vienlaicīgi p un $\neg p$,
tad šīs kopas visas formulas ir vienlaicīgi izpildāmas (t.i. eksistē tāds formulās

ieejošo atomu vērtību komplekts, kuram visas kopas formulas ir patiesas).

Pierādījums. Vajadzīgo atomu vērtību komplektu būvēsim tā: ja kopā ieiet pats atoms p, tad piešķirsim p vērtību "patiess", bet ja kopā ieiet $\neg p$ – tad piešķirsim p vērtību "aplams". Visas atomārās formulas (ar vai bez negācijas), kas ieiet kopā, tad būs patiesas. Parādīsim, ka arī visas pārējās kopas formulas šim atomu vērtību komplektam būs patiesas. No pretējā: ja kopā kādas formulas ir aplamas, tad paņemsim vienu no tām aplamajām, kurās ir vismazākais konjunkciju un disjunkciju kopskaits. Ja šī formula ir $A \wedge B$, tad formulās A, B konjunkciju un disjunkciju kopskaits ir mazāks, t.i. tās abas (piederēdamas kopai) ir patiesas. Bet tad arī $A \wedge B$ ir jābūt patiesai. Pretruna. Tāpat otrajā gadījumā: ja aplamā formula ir $A \vee B$, tad arī viena no formulām A, B (piederēdama kopai) ir patiesa. Bet tad arī $A \vee B$ ir jābūt patiesai. Pretruna. Līdz ar to esam pierādījuši, ka mūsu uzbūvētajam atomu vērtību komplektam visas kopas formulas ir patiesas. Q.E.D.

Kopsavilkums

Teorēma par tablo algoritma konverģenci, korektību un pilnību.

1) Jebkurai izteikumu valodas formulai tablo algoritms galīgā laikā uzģenerē galīgu analītisko koku.

2) Ja tablo algoritma uzģenerētajā kokā eksistē atvērts zars, tad koka saknē dotā formula ir izpildāma.

3) Ja tablo algoritma ģenerētajā kokā visi zari ir aizvērti, tad koka saknē dotā formula nav izpildāma.

Pierādījums. 2) Ja tablo algoritma uzģenerētajā kokā eksistē atvērts zars, tad šī zara formulu kopa apmierina Hintikkas lemmas nosacījumus, t.i. koka saknē dotā formula ir izpildāma.

3) Pierādām no pretējā: ja formula ir izpildāma, tad tablo algoritma kokā eksistē atvērts zars.

Algoritma uzlabojumi. Ja domājam par sava algoritma realizāciju datorprogrammās, tad savlaicīgi ir jāpadomā par tā optimizāciju.

Pirmkārt, par koka zaru aizvēršanu "pirms laika". Tiklīdz mēs pamanām, ka kādā zarā vienlaicīgi parādās atoms p un $\neg p$, tad šo zaru var aizvērt un tālāk "neaudzēt". Liekas, ka šo optimizāciju var realizēt, daudz nepalielinot ne algoritma darba laiku, ne tā izmantoto atmiņu.

Otrs priekšlikums (sk. [1], 337.lpp.): mainām koka virsotņu apstrādes secību - visu laiku vispirms cenšamies apstrādāt virsotnes ar konjunkcijām, bet virsotnes ar disjunkcijām atliekam, cik iespējams, uz vēlāku laiku. [Kāpēc tā ir labāk? Tāpēc, ka disjunkcijas apstrāde palielina koka zaru skaitu, bet konjunkcijas apstrāde to nepalielina.]

[Atgriežamies pie jautājuma par tablo algoritma darba laiku formulai, kuras

garums ir n? To var novērtēt kā $P(n)2^{cd}$, kur d ir disjunkciju skaits formulas negāciju normālformā.]

3.2. Tablo algoritms predikātu valodām

Predikātu valodām tablo algoritms ir sarežģītāks. Vispirms tiksim galā ar t.s. tīrajām predikātu valodām.

Tīrā predikātu valodā nav funkciju konstantu (t.i. ir atļautas tikai objektu konstantes un predikātu konstantes). Vispirms tiksim galā ar šīm tīrajām valodām, pēc tam rezultātu vispārināsim.

Arī šeit tablo algoritma mērķis ir noteikt, **vai dotā formula ir izpildāma** (t.i. noteikt, vai eksistē tāda dotās valodas interpretācija, kurā šī formula ir patiesa vismaz vienai brīvo mainīgo vērtību kombinācijai).

Pirmais solis tad būtu – ja dotā formula F satur brīvu mainīgo x, tad tās vietā aplūkot formulu $\exists x F$. Abas formulas ir vai nu izpildāmas, vai neizpildāmas. Tātad turpmāk varam uzskatīt, ka **dotā formula ir slēgta** (t.i. tai nav brīvu mainīgo).

Tāpat kā izteikumu formulas, arī predikātu formulas vispirms reducēsim ekvivalentā **negāciju normālformā** (zinātājiem: ekvivalence – atkal tikai klasiskajā loģikā). Tā saturēs tikai konjunkcijas, disjunkcijas un kvantorus, bet negācijas – tikai pie atomārām formulām (piemēram, $\neg p(x, y)$, kur p – predikātu konstante, bet x , y vietā var būt mainīgie vai objektu konstantes). Lai šo redukciju izpildītu, papildus augstāk minētajām loģikas kārtulām jāizmanto vēl divas:

$\neg \forall x\, B(x) \leftrightarrow \exists x \neg B(x)$,

$\neg \exists x\, B(x) \leftrightarrow \forall x \neg B(x)$.

Piemērs. Pārveidosim negāciju normālformā formulu $\forall x\, p(x) \rightarrow \neg \exists y (q(y) \vee r(y))$:

$\neg \forall x\, p(x) \vee \neg \exists y (q(y) \vee r(y))$ (implikāciju aizstājam ar disjunkciju un negāciju),

$\exists x \neg p(x) \vee \forall Ay \neg (q(y) \vee r(y))$ (negācijas "ienesam" zem kvantoriem),

$\exists x \neg p(x) \vee \forall\, y (\neg q(y) \wedge \neg r(y))$ (de Morgana likums).

Esam ieguvuši ekvivalentu formulu, kurā nav implikāciju un visas negācijas ir tikai pie atomārajām formulām.

Arī šeit dotās formulas redukcijai negāciju normāflormā vajadzīgs **lineārs laiks** (no formulas garuma).

Uzdevums. Pārliecinieties detalizēti, ka tas tā tiešām ir. Kā redukcijas laikā

mainās formulas garums?

Nedeterminētais tablo algoritms

Tablo algoritmu arī predikātu valodām visvieglāk ir aprakstīt kā nedeterminētu procesu, kas pārveido formulu sarakstu (tāpat kā dotā formula, šajā sarakstā visas formulas būs slēgtas):

1) Sākumā sarakstā ir dotā formula (negāciju normālformā). Ja formulā nav sastopama neviena objektu konstante, tad ievedam jaunu objektu konstanti c_0 (to darām tapēc, ka interpretācijas ar tukšu apgabalu mūs neinteresē).

2) Ja saraksta pirmā "neatomārā" formula ir $A \wedge B$, tad to no saraksta izmetam, bet saraksta beigās pievienojam divas formulas A, B.

3) Ja saraksta pirmā "neatomārā" formula ir $A \vee B$, tad to no saraksta izmetam, bet saraksta beigās pievienojam formulu A vai formulu B (**nedeterminēta izvēle)**.

4) Ja saraksta pirmā "neatomārā" formula ir $\exists x B$, tad to no saraksta izmetam, bet saraksta beigās pievienojam formulu B(c), kur c – pilnīgi jauna, tikko izveidota objektu konstante (t.i. konstante, kas saraksta formulās līdz šim nebija sastopama).

5) Ja saraksta pirmā "neatomārā" formula ir $\forall x B$, tad to no saraksta izmetam, bet saraksta beigās pievienojam formulas B(c), kur c - visas saraksta formulās jau sastopamās objektu konstantes, kam B(c) vēl nav sarakstā, **un aiz tām pievienojam atkal formulu** $\forall x B$. (Kāpēc mēs tā darām? Tāpēc, ka solis 4 arī turpmāk var radīt jaunas konstantes.)

6) Iterējam soļus 2) – 5) .

Tā kā mūsu saraksts visu laiku sastāv tikai no slēgtām formulām, tad atomārās formulas tajā (ar vai bez negācijas) var būt tikai izskatā $p(c_1, ..., c_n)$, kur p – predikātu konstante, bet $c_1, ..., c_n$ - objektu konstantes.

Algoritma izpildāmo soļu skaits te jau **var būt bezgalīgs** (protams, "vainīgs" ir solis 5, kas, ja sākumformulā ir universālkvantors, var tikt izpildīts bezgalīgi daudz reižu). Attiecīgi arī algoritma darba rezultātā var rasties kā galīgs, tā arī bezgalīgs skaits formulu. Bet ikviena atomāra formula $p(c_1, ..., c_n)$ (ar vai bez negācijas), ja tā ir nonākusi sarakstā, tad tur arī paliek. Un ikviena "neatomāra" formula agrāk vai vēlāk tiks apstrādāta (un izmesta no saraksta).

Ir iespējamas divas situācijas:

a) Formulu sarakstā kāds atoms $p(c_1, ..., c_n)$ sastopams divreiz - ar un bez negācijas: $p(c_1, \ldots, c_n), \neg p(c_1, \ldots, c_n)$. Tad algoritmas darba rezultāts ir **negatīvs**. Tiklīdz šāda situācija ir radusies, algoritma darbību var uzreiz apturēt, jo turpmāk nekas labāks nav sagaidāms (ir konstatēta pretruna un tā tālak nekur nepazudīs).

b) (Galīgajā vai bezgalīgajā) formulu sarakstā katrs atoms $p(c_1, ..., c_n)$ ir sastopams tikai vienreiz - vai nu kā $p(c_1, ..., c_n)$, vai kā $\sim p(c_1, ..., c_n)$. Tad darba rezultāts ir **pozitīvs** (pat tad, ja algoritms strādā, izpildot bezgalīgi daudz soļu!).

Piemērs formulai, kuru apstrādajot, tablo algoritms strādas bezgalīgi. Mums jāpanāk, lai formula nevarētu būt patiesa interpretācijā, kuras apgabals ir galīgs, piemēram:

$$\forall x \forall y \forall z (x<y \wedge y<z \rightarrow x<z) \wedge \forall x \neg(x<x) \wedge \forall x \exists y (x<y) \quad .$$

Pirmā daļa prasa, lai < būtu tranzitīva asociācija, otrā - lai tajā nebūtu ciklu, trešā - lai tajā nebūtu maksimālo elementu. Formula ir izpildāma - ņemam naturālos skaitļus un predikātu "mazāks".

[Kā mēs pozitīvajā gadījumā iegūstam interpretāciju, kurā sākumformula ir patiesa? Sk. zemāk pierādījuma sākumu.]

J. Hintikkas lemma tīrajām predikātu valodām. Ja negāciju normālformā esošu slēgtu formulu kopa:

a) līdz ar katru savu formulu $B \wedge C$ satur arī abas formulas B, C;
b) līdz ar katru savu formulu $B \vee C$ satur vismaz vienu no formulām B, C;

c) līdz ar katru savu formulu $\exists x B$ satur vismaz vienu formulu B(c), kur c – objektu konstante;

d) līdz ar katru savu formulu $\forall x B$ satur visas formulas B(c) visām objektu konstantēm c, kas parādās kopā;

e) nevienam atomam $p(c_1, ..., c_n)$ nesatur vienlaicīgi $p(c_1, ..., c_n)$ un $\neg p(c_1, ..., c_n)$;
tad visas šīs kopas formulas ir vienlaicīgi izpildāmas (t.i. eksistē interpretācija, kurā visas kopas formulas ir patiesas).

[Kāpēc runa ir tikai par slēgtu formulu kopu? Tāpēc, ka pierādījumā ir būtiski, ka neviena formula nesatur brīvus mainīgos.]

Pierādījums. Vajadzīgās interpretācijas apgabals būs visas objektu konstantes, kas parādās kopā. Katras šādas konstantes interpretācija būs pati šī konstante. Predikātu konstantes interpretēsim šādi: ja kopā ieiet $p(c_1, ..., c_n)$, tad p šīm argumentu vērtībām ir "patiess", bet ja kopā ieiet $\neg p(c_1, ..., c_n)$, tad p šīm argumentu vērtībām ir "aplams". Ja kopā neieiet neviena no abām formulām, tad $p(c_1, ..., c_n)$ patiesuma vērtību varam izvēlēties patvaļīgi. Visas atomārās formulas (ar vai bez negācijām), kas ieiet kopā, tad būs patiesas.

Parādīsim, ka arī visas pārējās kopas formulas šajā interpretācijā būs patiesas. No pretējā: ja kopā kādas formulas ir aplamas, tad paņemsim vienu no tām aplamajām, kurās ir vismazākais konjunkciju, disjunkciju un kvantoru

kopskaits.

Ja šī aplamā formula ir $A \wedge B$, tad formulās A, B konjunkciju, disjunkciju un kvantoru kopskaits ir mazāks, t.i. tās abas (piederēdamas kopai) ir patiesas. Bet tad arī $A \wedge B$ ir jābūt patiesai. Pretruna. Arī otrajā gadījumā: ja aplama ir $A \vee B$, tad arī viena no formulām A, B (piederēdama kopai) ir patiesa. Bet tad arī $A \vee B$ ir jābūt patiesai. Pretruna.

Ja aplamā formula ir $\exists x B$, tad kopā ir arī kāda formula B(c), tajā ir par vienu kvantoru mazāk, tāpēc tai jābūt patiesai. Bet tad arī $\exists x B$ ir jābūt patiesai. Pretruna.

Ja aplamā formula ir $\forall x B$, tad kopā ir visas formulas B(c) visām konstantēm c, kas ieiet mūsu interpretācijas apgabalā. Visās šajās formulās ir par vienu kvantoru mazak, tāpēc tās visas ir patiesas, tātad patiesai jābūt arī $\forall x B$. Pretruna.

Līdz ar to esam pierādījuši, ka mūsu "uzbūvētajā" interpretācijā visas kopas formulas ir patiesas. Q.E.D.

Secinājums. 1) Algoritma **korektība.** Ja mūsu algoritma (iespējams, bezgalīgā) darba rezultāts ir pozitīvs, tad iegūtā formulu kopa apmierina Hintikkas lemmas nosacījumus, t.i. dotā formula ir izpildāma. 2) Algoritma **pilnība.** Ja dotā formula ir izpildāma, tad eksistē tāds mūsu algoritma darbības ceļš, kurā darba rezultāts ir pozitīvs.

Piezīme par konstruktivitāti. Vai pie šādas definēšanas mūsu interpretācijā katrs predikāts $p(x_1, ..., x_n)$ būs izrēķināms (rekursīvs)? To mēs nevaram garantēt. Tiešām, lai noteiktu $p(c_1, ..., c_n)$ patiesuma vērtību, mums jāgaida, kamēr algoritms uzģenerēs vai nu $p(c_1, ..., c_n)$, vai $\neg p(c_1, ..., c_n)$. Kamēr tas nav noticis, mēs nezinām, kādu lēmumu pieņemt. Tas nozīmē, ka predikātu konstanti p interpretējošais **predikāts būs efektīvi (rekursīvi) sanumurējams** (varam lemt arī citādi - rekursīvi sanumurējams būs p papildinājums). Vairāk garantēt mēs nevaram.

Uzdevums. a) Pamatojiet secinājumu 1 detalizēti. Sevišķi nopietni apdomājiet – kāda loma ir solī 5 paredzētajai formulas $\forall x B$ atkārtotajai ievietošanai sarakstā? Svarīgi, ka katram kopas formulu universālkvantoram solis 5 var tikt izpildīts bezgalīgi daudz reižu, t.i. pēc ikviena soļa 4, kas rada jaunu konstanti, kaut kad tiks izpildīts solis 5.

b) Pamatojiet secinājumu 2 detalizēti. Sākam ar interpretāciju J, kurā sākumformula ir patiesa. Solī 4, radot formulai $\exists x B$ jaunu konstanti c, mēs to tūlīt pat interpretējam kā tādu objektu x, kam formula B(x) interpretacijā J ir "patiesa". Tādā veidā turpinot, mēs valodu papildināsim ar ne vairāk kā sanumurējamu skaitu konstantu. Utt.

[*Piezīme*. Ja mēs interpretāciju J "iztīrīsim", izmetot no tās visus objektus, kas

neinterpretē nevienu no sākumformulas konstantēm (to ir ne vairāk kā galīgs skaits) un nevienu no jaunievestajām konstantē (to ir ne vairāk kā sanumurējams skaits), tad – kā varat pārliecināties paši – iegūsim interpretāciju, kuras apgabals būs ne vairāk kā sanumurējams, un kurā sākumformula būs patiesa. T.i. būtībā mēs te esam pierādījuši **Levenheima-Skolema teorēmu**, sk. kursa grāmatas sadaļus 4.3).]

c) Pārliecinieties, ka strādājot ar jau augstāk minēto formulu $\forall x \forall y \forall z (x<y \wedge y<z \rightarrow x<z) \wedge \forall x \neg(x<x) \wedge \forall x \exists y (x<y)$, tablo algoritms ieciklosies, un tomēr - "uzģenerēs" bezgalīgu interpretāciju, kurā šī formula ir patiesa. [Formulas pirmā daļa prasa, lai predikāts "<" būtu tranzitīvs, bet otrā daļa - lai tas nebūtu refleksīvs. Tas nozīmē, piemēram, ka starp 4 objektiem, kam c<d<e<f, nebūs divu vienādu.]

Determinētais tablo algoritms

Līdzīgi kā izteikumu valodas gadījumā, arī predikātu valodām determinētais tablo algoritms "audzē" koku. Formulu izmešanas vietā mēs tad praktizējam formulu atzīmēšanu par "apstrādātām". Un apstrādei vienmēr ņemam koka saknei tuvāko "neapstrādāto" "neatomāro" formulu. Koka sazarošanos (katrreiz - tikai divos zaros) izsauc tikai un vienīgi disjunkciju apstrāde. Tiklīdz kādā zarā pamanām divas pretī runājošas "atomāras" formulas, tā uzreiz pārtraucam šo zaru "audzēt".

Precīzāk:

0) Ievedam pilnīgi jaunu objektu konstanti c_0, kas nav sastopama dotajā formulā (formulu kopā).

1) Koka saknes virsotnē liekam doto formulu (negāciju normālformā). Ja formulas vietā mums ir dota formulu kopa, tad koka saknē liekam šo formulu konjunkciju.

2) Ja virsotnē ir formula $A \wedge B$, tad **katram zaram, kas iet caur šo virsotni**, "pieaudzējam" turpinājumu ar 2 virsotnēm A un B:

$A \wedge B$ ----- ...-----$-A-B$.

Un virsotni $A \wedge B$ atzīmējam kā "apstrādātu".

3) Ja virsotnē ir formula $A \vee B$, tad **katram zaram, kas iet caur šo virsotni**, "pieaudzējam" sazarojumu – attiecīgi ar virsotni A, un ar B:

$A \vee B$ -----...-----

$|-A$

$|-B$

Un virsotni AvB atzīmējam kā "apstrādātu".

4) Ja virsotnē ir formula $\exists x\, B$, tad **katram zaram, kas iet caur šo virsotni**, "pieaudzējam" turpinājumu – formulu B(c), kur c – pilnīgi jauna, tikko izveidota objektu konstante (t.i. konstante, kas **šī zara formulās** līdz šim nebija sastopama):

$$\exists x\, B \text{----- ...-----} - B(c)$$

Un virsotni $\exists x\, B$ atzīmējam kā "apstrādātu".

5) Ja virsotnē ir formula $\forall x\, B$, tad **katram zaram, kas iet caur šo virsotni**, "pieaudzējam" turpinājumu: pievienojam visas tādas formulas B(c), kur c – visas šajā zarā jau sastopamās objektu konstantes, kam B(c) vēl šajā zarā nav bijusi, **un aiz tām pievienojam atkal formulu** $\forall x\, B$ **:**

$$\forall x\, B \text{----- ...-----} - B(c) - B(d) \text{----...----} \forall x\, B$$

Un pirmo virsotni $\forall x\, B$ atzīmējam kā "apstrādātu".

6) Iterējam soļus 2)-5), kā nākošo apstrādājamo virsotni izvēloties jebkuru no tām "neapstrādātajām" virsotnēm, kas satur "neatomāras" formulas un kas koka saknei ir vistuvāk. Tiklīdz kādā zarā pamanām divas pretī runājošas "atomāras" formulas, tā uzreiz pārtraucam šo zaru "audzēt", visas zara formulas atzīmējot kā "apstrādātas". Kad vairs nav "neapstrādātu" virsotņu ar "neatomārām" formulām, darbu beidzam. (Vārdu "neatomārs" liekam pēdiņās, jo par "atomārām" formulām šeit jāuzskata gan atomi, gan atomi ar negācijām.)

Universālkvantoru dēļ koka audzēšanas process var beigties, bet var arī nebeigties, turpinoties līdz bezgalībai!

Visi aizvērtie zari ir galīgi.

Atvērtie zari var būt kā galīgi, tā bezgalīgi.

Piemērs.

Parādīsim, ka formula $\exists x \forall y\, p(x,y) \rightarrow \forall y \exists x\, p(x,y)$ ir izvedama klasiskajā loģikā, t.i. parādīsim, ka tās negācija nav izpildāma. Pārveidojam negāciju normālformā:

$\neg(A \rightarrow B)$ ir ekvivalenta $\neg(\neg A \vee B)$, $\neg\neg A \wedge \neg B$, un beidzot, $A \wedge \neg B$.

Tātad mūsu formula ir ekvivalenta $\exists x \forall y\, p(x,y) \wedge \neg \forall y \exists x\, p(x,y)$ un $\exists x \forall y\, p(x,y) \wedge \exists y \forall x \neg p(x,y)$.

Tablo algoritms šai formulai veidos koku, kas sastāv no viena zara. Sākumā būs abi konjunkcijas locekļi: $\exists x \forall y\, p(x,y)$; $\exists y \forall x \neg p(x,y)$. Apstrādājot pirmo eksistences kvantoru, iegūstam turpinājumu $\forall y\, p(a,y)$, aptrādājot otro eksistences kvantoru, iegūstam $\forall x \neg p(x,b)$, kur a, b ir konstantes. Apstrādājot pirmo universālkvantoru, iegūstam turpinājumu:

$p(a,a); p(a,b); \forall y\, p(a,y)$, bet apstrādājot otro – turpinājumu

$\neg p(a,b); \neg p(b,b); \forall x \neg p(x,b)$. Mūsu vienīgajā koka zarā nu ir radusies pretruna: $p(a,b); \neg p(a,b)$. Tātad mūsu kokā "visi zari ir aizvērti", kas nozīmē, ka sākotnējās formulas **negācija nav izpildāma.** Tātad pati formula ir loģiski vispārderīga un tātad – saskaņā ar Gēdela Pilnības Teorēmu – izvedama klasiskajā loģikā.

Piemēra beigas.

Ja visa procesa rezultātā visi zari iznāk aizvērti (t.i. tajos ir pretrunīgas "atomārās" formulas), tad katrs koka zars ir galīgs. No Deneša Kēniga lemmas (sk. augstāk) tad seko, ka arī viss koks kopumā ir galīgs. **Šīs situācijas iestāšanos mēs varam uzreiz konstatēt.**

Pretējais gadījums - procesa laikā kokā rodas vismaz viens galīgs vai bezgalīgs atvērts zars (t.i. bez pretrunīgām "atomārām" formulām). Galīgu atvērto zaru mēs konstatējam uzreiz, kad tas ir radies. Bet bezgalīgu zaru ne vienmēr var "atšifrēt" - kamēr tas turpinās, mēs parasti nevaram uzzināt, vai tas izvērtīsies aizvērtos zaros, vai arī beigsies atvērts, vai turpināsies bezgalīgi kā atvērts.

Kopsavilkums

Teorēma par tablo algoritma konverģenci, korektību un pilnību.

1) Tablo algoritma darba rezultātā rodas vai nu galīgs koks, kurā visi zari ir aizvērti, vai arī koks, kurā ir vismaz viens (galīgs vai bezgalīgs) atvērts zars.

2) Ja tablo algoritma uzģenerētajā kokā eksistē atvērts zars, tad koka saknē dotā formula ir **izpildāma**. (Šīs situācijas iestāšanos mēs varam arī neuzzināt.)

3) Ja tablo algoritma ģenerētajā kokā visi zari ir aizvērti, tad koka saknē dotā formula **nav izpildāma**. (Šīs situācijas iestāšanos mēs ar laiku uzzināsim.)

Pierādijums. Izmanto Hintikkas lemmu.

Piezīme. Būtībā mēs esam ieguvuši arī alternatīvu pierādījumu kādai svarīgai lietai, ko parasti izved no Gēdela Pilnības Teorēmas: **katrā predikātu valodā visu loģiski vispārderīgo formulu kopa ir rekursīvi sanumurējama**. Tiešam, formula F ir loģiski vispārderīga tad un tikai tad, ja $\neg F$ nav izpildāma. Tāpēc, ja mēs formulai $\neg F$ darbināsim tablo algoritmu, tad tās neizpildāmību mēs konstatēsim pēc galīga soļu skaita (algoritma veidotais koks iznāks galīgs un tajā visi zari būs aizvērti). T.i. eksistē algoritms, kas apstājas un atbild "jā" tad un tikai tad, ja formula F ir loģiski vispārderīga. Q.E.D.

No Gēdela Pilnības Teorēmas šo faktu izved tā: tā kā formula F ir loģiski vispārderīga tad un tikai tad, ja to var izvest no loģikas aksiomām, tad ģenerējot visus iespējamos izvedumus, mēs iegūsim visas loģiski vispārderīgās formulas. Q.E.D.

No **Čērča-Kalmāra teorēmas** seko, ka tablo algoritms predikātu formulām ne

vienmēr izdos rezultātu galīgā laikā. Ja formula F ir loģiski vispārderīga, tad $\neg F$ nav izpildāma. Tāpēc, ja mēs formulai $\neg F$ darbināsim tablo algoritmu, tad **tās neizpildāmību mēs konstatēsim pēc galīga soļu skaita** (algoritma veidotais koks iznāks galīgs un tajā visi zari būs aizvērti). Bet, ja F nav loģiski vispārderīga, tad $\neg F$ ir izpildāma, un tātad tablo algoritma koks dažos gadījumos augs bezgalīgi, bet mēs to nekad neuzzināsim.

Prof. Gunta Bārzdiņa otrais eksperiments, programmējot valodā Prolog tablo algoritmu predikātu valodām.

Tablo algoritms predikātu valodām ar funkciju konstantēm

Uzdevums. Vispāriniet Hintikkas lemmu predikātu valodām, kas satur **funkciju konstantes**. Interpretācijas apgabals tad būs visi iespējamie termi, ko var izveidot no dotajā formulu kopā ieejošajām objektu konstantēm un valodas funkciju konstantēm. Tādus termus (t.i. termus, kas nesatur mainīgos) ir pieņemts saukt par **konstantiem termiem**. Funkciju konstantu interpretācijas (t.i. funkcijas) veidojiet kā "sintaksiskus konstruktorus", t.i. $f(t_1, ..., t_n)$ vērtība būs pats terms $f(t_1, ..., t_n)$ (kā simbolu virkne). Tad ikviena konstanta terma t intepretācija būs pats šis terms t.

Utt.

3.3. Tablo algoritms loģikai ALC

Izmantosim mums labi zināmo ALC izteiksmju translāciju predikātu valodas formulās. Par T-kastes formulām mēs zinām šādas lietas:

a) Valodā ir tikai vienvietīgas un divvietīgas predikātu konstantes (klases un lomas);
b) Atomārās formulas ir tikai vienvietīgie predikāti: p(x), ...;
c) Ja ir viena vai divas formulas B(x), C(x) ar vienu brīvu mainīgo x, tad arī B(x)&C(x), B(x)vC(x) un ~B(x) ir formulas ar vienu brīvu mainīgo x;
d) Katrs kvantors ir ierobežots ar divvietīgu predikātu konstanti: Ey(r(x,y)&B(y)), Ay(r(x,y)->B(y)), un formulā B ir tikai viens brīvs mainīgais x;
e) Tādā veidā var uzbūvēt tikai formulas ar vienu brīvu mainīgo x. Objektu konstantes šajās formulās parādīties nevar.

T-kastes **aksiomas** fiksē jēdzienu pakļautību: C1<=C2 divām izteiksmēm C1, C2. Translējot uz predikātu formulām, sanāk formula Ax (C1(x)->C2(x)), kas ALC valodai nepieder (jo kvantors Ax nav ne ar ko ierobežots). Bet pagaidām uzskatīsim, ka mūsu **T-kaste ir tukša**, t.i. ka šādu aksiomu nav.

Ja vēlamies aplūkot arī A-kasti, tad papildus nāk klāt:

f) Objektu konstantes (indivīdu vārdi): c, d, ...
g) Apgalvojumi par indivīdiem, t.i. formulas F(c), kur c ir indivīda vārds.

Ievērosim, ka formulā F(c) ir tikai **viena** objektu konstante c - tā, kuru formulā F(x) ievieto vienīgā brīvā mainīgā x vietā.
h) Apgalvojumi par lomām (pozitīvi un negatīvi), t.i. formulas r(c, d) un ~r(c, d), kur r ir lomas vārds, bet c, d - indivīdu vārdi.
i) Apgalvojumi par indivīdu atšķirību, t.i. formulas ~(c=d). Ja par c un d tas nav atklāti pateikts, tad uzskatīsim, ka šīs konstantes var apzīmēt arī vienu un to pašu objektu.

Piezīme. Vairumā avotu punkta i) vietā izmanto t.s. **vārdu vienīguma pieņēmumu** (unique name assumption), uzskatot, ka dažādas konstantes vienmēr apzīmē atšķirīgus objektus. S.Tobies savā disertācijā iztiek bez šī pieņēmuma. Arī mēs iztiksim.

Tā kā T-kaste mums ir tukša, tad uzdevums "A-Box consistency check" ("A-kastes saderības noskaidrošana") mums reducējas uz **ALC valodas slēgtu formulu galīgas kopas izpildāmības noskaidrošanu** (vēlreiz atcerēsimies, ka T-kastes aksiomas nav ALC valodas formulas).

Otrs populārais uzdevums - "Concept satisfiability check" ("jēdziena saderības noskaidrošana") nozīmē, ka mums ir dota ALC izteiksme C, un ir jānoskaidro, vai šis jēdziens nav tukšs. C translācijas rezultātā rodas predikātu formula C(x), kurai jānoskaidro izpildāmība (jēdziens nav tukšs tad un tikai tad, ja formula ir izpildāma). Bet C(x) ir izpildāma tad un tikai tad, ja izpildāma ir formula C(d), kur d ir objektu kontante. Tātad arī otrais uzdevums reducējas uz **ALC valodas slēgtu formulu galīgas kopas izpildāmības noskaidrošanu** (šoreiz kopā tad ir tikai viena formula C(d)).

Ar šo uzdevumu (A-Box consistency check: vai ALC valodas slēgtu formulu dotā galīgā kopa S ir izpildāma?) tad arī nodarbosies mūsu tablo algoritma pirmā versija.

Nedeterminētais algoritms

Mērķis: noteikt, vai dotā ALC valodas slēgtu formulu galīga kopa S ir izpildāma, vai nav (izpildāma - tas nozīmē, ka eksistē valodas interpretācija, kurā kopas S visas formulas ir patiesas).

Vispirms mums kopas S formulas jāpārveido **negāciju normālformā**.

Piemēram, aplūkosim negāciju formulai *Virietis ^ E berns.(Sieviete ^ A berns.Bagats)* - vīrietis, kam ir meita, kurai visi bērni ir bagāti:

-(Virietis ^ E berns.(Sieviete ^ A berns.Bagats))
-Virietis v *-E berns.(Sieviete ^ A berns.Bagats)*
-Virietis v *A berns.-(Sieviete ^ A berns.Bagats)*
-Virietis v *A berns.(-Sieviete* v *-A berns.Bagats)*
-Virietis v *A berns.(-Sieviete* v *E berns.-Bagats)*

Pēdējā formula ir negāciju normālformā - negācijas ir tikai pie pamatklasēm.

Vispārīgajā gadījumā redukcijas likumi ir tādi paši kā predikātu valodām:

-(B^C) var pārveidot par (-B)v(-C),
-(BvC) var pārveidot par (-B)^(-C),
-Er.C var pārveidot par Ar.(-C),
-Ar.C var pārveidot par Er.(-C),
--C var pārveidot par C.

Un arī šeit redukcija ir izpildāma **lineārā laikā** (no formulas garuma).

Tālāk sekojošo tablo algoritmu atkal sākumā aprakstīsim kā nedeterminētu procesu, kas pārveido **formulu kopu** (vairs ne sarakstu - tas te vairs nav obligāti):

1) Sākumā kopā ir tikai formulas no dotās kopa S (visas formulas ir negāciju normālformā).

2) Ja kopā ir formula A(c)&B(c), bet tajā nav kādas no formulām A(c), B(c), tad kopai šo (šīs) trūkstošo(ās) formulu(as) pievienojam.

3) Ja kopā ir formula A(c)vB(c), bet tajā nav nevienas no formulām A(c), B(c), tad kopai pievienojam formulu A(c) vai formulu B(c) (būtiski **nedeterminēta izvēle)**.

4) Ja kopā ir formula C(d) = Ey(r(d,y)&B(y)), bet nevienai objektu konstantei e mūsu kopā nav abu formulu r(d, e) un B(e), tad kopai pievienojam formulas r(d, e) un B(e), kur e - pilnīgi jauna objektu konstante. **[Kāpēc mēs tā darām?]**

5) Ja kopā ir formula C(d) = Ay(r(d,y)->B(y)), un kādai objektu konstantei e formula r(d, e) jau ir kopā, bet B(e) vēl nav, tad kopai pievienojam B(e). **[Kāpēc mēs tā darām?]**

6) Patvaļīgā secībā iterējam soļus 2) - 5), kamēr vien kāds no tiem ir lietojams. Algoritms beidz darbu, kad neviens no soļiem 2)-5) vairs nav lietojams.

Piemērs. Vai jēdziens "laimīgs tēvs" ir saderīgs? Runa ir par izteiksmi *Virietis ^ E berns.Cilveks ^ A berns.Bagats*. Principā vajadzīgo interpretāciju te varam uzminēt: "definējam" vīrieti vārdā John un viņa vienīgo bērnu - bagāto meitu vārdā Parisa. Bet tādu pašu rezultātu mums dod arī tablo algoritms. Tiešām, dotā izteiksme predikātu valodā izskatās tā (c - konstante): *Virietis(c) & Ey (berns(c,y)&Cilveks(y)) & Ay(berns(c,y)->Bagats(y)).* Jānoskaidro, vai šī formula ir izpildāma. Solis 2 to sadala 3 formulu kopā:
Virietis(c);
Ey (berns(c,y)&Cilveks(y));
Ay(berns(c,y)->Bagats(y)).
Tālāk otrajai formulai izmantosim soli 4 - ievedīsim jaunu konstanti d, un pievienosim kopai divas formulas:
berns(c,d);
Cilveks(d).
Pēc tam trešajai formulai izmantosim soli 5 - pievienosim kopai formulu:

Bagats(d).
Vairāk ar soļiem 2, 3, 4 te nekas nav izdarāms, tāpēc algoritms darbu beidz. Esam ieguvuši 4 atomāras formulas (pirmā un pēdējās 3 formulas), kas mums dod interpretāciju, kurā sākumformula ir patiesa (c ir John, un d ir viņa vienīgais bērns - bagātā Parisa).

Kāpēc apstrādātās formulas mēs no kopas neizmetam? Kāpēc solis 5) pievieno tikai pa vienai formulai (nevis visas, ko šai brīdī būtu "loģiski" pievienot)? Tas ieteikts

Stephan Tobies http://www.stephan-tobies.net/ disertācijā http://lat.inf.tu-dresden.de/research/phd/Tobies-PhD-2001.pdf.

Kā novērojis kolēģis Tobies, *neatkarīgi* no soļu 2)-5) izpildes secības, process **vienmēr** beidzas pēc galīga soļu skaita. Pierādījumā izmantoti diezgan smalki novērojumi (tālākais šīs sadaļas teksts ideju ziņā seko Tobies disertācijas 3.nodaļā teiktajam).

Katrā algoritma darbības momentā aplūkosim šādu "augošu" orientētu grafu G[S] (te ir **pierādījuma netriviālā vieta**, tālāko ir viegli "piedomāt klāt"):

- Grafa virsotnes: visas objektu konstantes, kas šobrīd sastopamas kopas formulās; katrai virsotnei d tiek piekārtota "augoša" kopa L(d) no visām tām formulām izskatā C(d), kas šobrīd ir algoritma uzģenerētajā formulu kopā. Tādā veidā visas sākumkopas S formulas un visas formulas, kas tiek uzģenerētas algoritma darba laikā, tiek sagrupētas pa kopām L(d), t.i. pa grafa virsotnēm.

- No virsotnes d uz virsotni e tiek vilkta šķautne, ja algoritma uzģenerētajā kopā ir (vai parādās) kāda formula r(d, e) vai ~r(d, e); un šķautnei tiek piekārtota visu šādu formulu kopa (no algoritma uzģenerētās kopas).

Novērojumi:

1) Algoritma darba sākumā šis grafs būtībā ir patvaļīgs galīgs orientēts grafs. Tā virsotnes atbilst sākumkopā S atrodamajām objektu konstantēm: d, e, ... Ja kopā S ir formula d: C, tad kopā L(d) tiks iekļauta formula C(d). Ja kopā S ir formula (d, e): r vai (d, e): r, tad mūsu grafā no virsotnes d uz virsotni e būs novilkta šķautne, kurai tiks pierakstīta formula r(d, e) vai formula ~r(d, e). Vienai šķautnei var būt pierakstītas vairākas šādas formulas. Nekādu ierobežojumu te nav, tāpēc mūsu grafa sākumstāvoklis būtībā ir patvaļīgs galīgs orientēts grafs.

Kas notiek tālāk?

2) Ja virsotnei c kopā L(c) ir formula A(c)&B(c), un tai tiek izpildīts solis 2, tad tai pašai kopai L(c) tiek pievienota viena vai divas formulas - A(c) un/vai B(c). Skaidrs, ka formulai A(c)&B(c) neviens no algoritma soļiem turpmāk vairs lietojams nekad nebūs, tāpēc šo formulu mēs varētu atzīmēt kā "apstrādātu". Otrkārt, pievienoto formulu "loģiskais dziļums" ir par vienu mazāks nekā formulas A(c)&B(c) "loģiskais dziļums", un tās ir formulas

A(c)&B(c) apakšformulas.

Piezīme par "loģisko dziļumu". ALC formulas "loģisko dziļumu" definējam kā operāciju skaitu - izņemot negācijas, kas nepieciešams tās uzbūvēšanai. Visas formulas mums ir negāciju normālformā, un negācijas tajās parādās tikai pie atomiem, tāpēc loģiskā dziļuma definīcijā tās nevajag iekļaut: šo dziļumu veido tikai konjunkcijas, disjunkcijas un kvantori.

3) Ja virsotnei c kopā L(c) ir formula A(c)vB(c), un tai tiek izpildīts solis 3, tad tai pašai kopai L(c) tiek pievienota viena no formulām - A(c) vai B(c). Skaidrs, ka formulai A(c)vB(c) neviens no algoritma soļiem turpmāk vairs lietojams nekad nebūs, tāpēc šo formulu mēs varētu atzīmēt kā "apstrādātu". Otrkārt, pievienotās formulas "loģiskais dziļums" ir par vienu mazāks nekā formulas A(c)vB(c) "loģiskais dziļums", un tā ir formulas A(c)vB(c) apakšformula.

4) Ja virsotnei d kopā L(d) ir formula C(d) = Ey(r(d,y)&B(y)), un tai tiek izpildīts solis 4, tad grafam tiek pievienota **jauna virsotne** e, no d uz e tiek novilkta jauna šķautne, kam tiek pierakstīta formula r(d, e), bet kopā L(e) tiek ievietota tās pirmā formula B(e). Skaidrs, ka formulai C(d) neviens no algoritma soļiem turpmāk vairs lietojams nekad nebūs, tāpēc šo formulu mēs varētu atzīmēt kā "apstrādātu". Otrkārt, pievienotās formulas B(e) "loģiskais dziļums" ir par vienu mazāks nekā formulas C(d) "loģiskais dziļums", un tā ir formulas C(d) apakšformulas B(y) instance.

Vēl kas: tā kā solī 4 vienmēr tiek radīta jauna virsotne e (un citi algoritma soļi ne jaunas virsotnes, ne jaunas šķautnes nerada), tad, pirmkārt, jaunajā virsotnē ieies tikai viena šķautne - tā, kuru tikko novilkām, un otrkārt, pie šīs šķautnes būs pierakstīta tikai viena formula - r(d,e). Un no e izejošās šķautnes (kas, varbūt, radīsies vēlak un varēs būt vairākas) vedīs tikai uz jaunradītām virsotnēm. Tātad šis jaunu virsotņu un šķautņu pievienošanas process būtībā ir "koku augšana" (ar "zarošanos"), kas sākas no virsotnēm, kas grafā bija jau sākumstāvoklī. Kopējā aina tātad ir šāda: sākumā mums ir patvaļīgs orientēts grafs, no kura virsotnēm "aug koki".

5) Ja virsotnei d kopā L(d) ir formula C(d) = Ay(r(d,y)->B(y)), un tai tiek izpildīts solis 5, tas nozīmē, ka no d uz virsotni e ved šķautne ar formulu r(d, e) un ka kopai L(e) tiek pievienota formula B(e). Šoreiz jāatzīmē, ka formulai C(d) solis 5 var tikt pielietots atkārtoti: ja no d uz citu virsotni e' ir novilkta šķautne r(d, e') (šī šķautne varēja būt jau sākumgrafā, vai arī to radīja solis 4), tad solis 5 formulai C(d) būs lietojams atkal - kopai L(e') pievienojot formulu B(e'). Otrkārt, pievienotās formulas B(e) "loģiskais dziļums" ir par vienu mazāks nekā formulas C(d) "loģiskais dziļums", un tā ir formulas C(d) apakšformulas B(y) instance.

Novērojumi-secinājumi:

a) Katra kopas L(d) formula ir formā C(d), un tā ir dotās formulu kopas S formula, vai arī kādas S formulas kādas apakšformulas C(x) instance C(d).

Tātad **formulu skaits kopā L(d) nekad nepārsniegs skaitli |S|** (kopas S formulu visu apakšformulu kopskaitu).

b) Jauna grafa virsotne e rodas tikai, izpildot soli 4, piekārtojot kādai virsotnei d pilnīgi jaunu e, un novelkot šķautni r(d, e). **Un nekādas citas šķautnes uz virsotni e novilktas netiks!** Tātad katrai grafa šķautnei, kas ved uz **jaunradītu** virsotni e, tiek piekārtota **tieši viena formula** r(d, e). Un tātad šai virsotnei e, "ejot atpakaļ" atbilst tikai viena virsotne d. Tātad, ja neskaita to grafa daļu, kas atbilst dotās kopas S elementiem, tad **mūsu grafs sastāv no galīga skaita koku.** Katra koka sakne ir kāda konstante c, kas ir dota kopas S formulās (sk. soli 1).

c) Pieņemsim, ka grafa virsotnes d, e savieno šķautne r(d, e), pie tam e ir jaunradīta virsotne. Ko var teikt par formulu kopām L(d) un L(e), kas piekārtotas virsotnēm d un e? Kopā L(e) jauna formula var rasties vai nu soļos 2, 3 (t.i. no citas formulas, kas jau ir kopā L(e)), vai arī soļos 4, 5 - no kādas formulas, kas ir kopā L(d). (Par soli 5: ja no virsotnes d' formulas Ay(q(d', y)->B(u)) virsotnē e parādās formula B(e), tad no d' uz e ved šķautne ar formulu q(d', e), kas saskaņā ar novērojumu (b) ir iespējams tikai tad, ja d'=d un q=r.) Secinājums: katra L(e) formula vai nu "atnāk no L(d)" kā apakšformula, vai rodas kā apakšformula no citas L(e) formulas, t.i. katrai L(e) formulai eksistē L(d) formula, kuras "loģiskais dziļums" ir lielāks. Un tāpēc: **vissarežģītākā L(e) formula ir vienkāršāka par vissarežģītāko L(d) formulu.** Bet ejot pa kādu koka zaru mūsu grafā (t.i. radot arvien jaunas virsotnes e) šī formulu sarežģītība nevar dilt neierobežoti - pirms nulles procesam ir jāapstājas. Tātad minētajos mūsu **grafa kokos visi zari ir galīgi** (un to garums nepārsniedz |S| - kopas S formulu visu apakšformulu skaitu).

d) No katras grafa virsotnes d virzienā uz jaunradītajām virsotnēm **var iziet tikai galīgs skaits šķautņu**. **[Kāpēc tas ir svarīgi?]** Tiešām, šādas šķautnes rodas tikai solī 4, no kopas L(d) formulām C(d), un nevienai formulai C(d) = Ey(r(d,y)&B(y)) šis solis netiks pielietots divreiz. Un to, ka kopas L(d) apjoms, kura satur formulu C(d), ir ierobežots (ar kopas S formulu visu apakšformulu skaitu |S|), mēs jau zinām. Tātad no katras grafa virsotnes d virzienā uz jaunradītajām virsotnēm var iziet ne vairāk kā |S| šķautnes.

Saskaņā ar Kēniga lemmu tas viss kopā nozīmē, ka algoritma darba laikā mūsu **grafā** G[S] **virsotņu skaits palielinās tikai līdz noteiktai robežai** (t.i. sava darba laikā algoritms ievedīs tikai galīgu skaitu jaunu konstantu e).

Galīgais secinājums:

Teorēma. Neatkarīgi no soļu 2-5 izpildes secības, mūsu nedeterminētais **algoritms vienmēr "konverģē", t.i. beidz darbu pēc galīga soļu skaita.**

Pierādījums. Tiešām, katrā reāli izpildītā algoritma solī rodas vismaz viena jauna formula B(d), un neviena formula nekad netiek izmesta, tāpēc, ja algoritma izpildīto soļu skaits būtu bezgalīgs, tad visu kopu L(d) apvienojums arī būtu bezgalīgs. Bet tas nav iespējams, jo mēs jau zinām, ka: 1) katra kopa

L(d) ir galīga, 2) dažādo d skaits grafā G[S] arī ir galīgs. Q.E.D.

Algoritma darba beigās ir iespējamas divas situācijas:

a) Formulu kopā kāds atoms p(c) vai r(c, d) ir sastopams divreiz - ar un bez negācijas, piemēram, p(c), ~p(c) vai r(c, d), ~r(c, d). Tad algoritma darba rezultātu sauksim par **negatīvu**.

b) Formulu kopā katrs atoms p(c) un r(c, d) sastopams tikai vienreiz - vai nu ar negāciju, vai bez tās. Tad algoritma darba rezultātu sauksim par **pozitīvu** (citos tekstos: *clash free*).

Algoritma korektība

Teorēma. Ja algoritma darba rezultāts ir pozitīvs, tad dotā formulu kopa S ir izpildāma. Pie tam attiecīgajā interpretācijā:
a) objektu apgabals ir galīgs (t.s. ***Finite model property***);
b) ja A-kastē nav apgalvojumu par lomām, tad modeļa grafs G[S] ir koku kolekcija (t.s. ***Tree model property***).

Pierādījums. Ja algoritma darba rezultāts ir pozitīvs, tad aplūkosim formulu kopu, kas darba laikā radusies, un uz tās pamata izveidosim šādu interpretāciju J.

Interpretācijas apgabals būs visas objektu konstantes, kas ir dotās kopas S formulās vai ir ievestas darba laikā (t.i. augstāk aplūkotā grafa G[S] virsotnes - to skaits ir galīgs). Katras objektu konstantes interpretācija būs pati šī konstante. Tādā veidā mums kļūst patiesas visas formulas ~(c=d), kur c un d ir dažādas konstantes.

Vienvietīgo predikātu q interpretācijas definēsim šādi: ja formula q(d) atrodas kopā, tad q(d) būs "patiess", bet ja kopā atrodas formula ~q(d), tad q(d) būs "aplams". Ja kopā nav ne vienas, ne otras formula, tad q(d) patiesuma vērtību varam izvēlēties patvaļīgi. Divvietīgo (t.i. lomu) predikātu r(d, e) interpretācijas definēsim šādi: ja formula r(d, e) atrodas kopā, tad r(d, e) būs "patiess", ja neatrodas - tad r(d, e) būs "aplams".

Pirmkārt, šīs interpretācijas apgabals ir galīgs (jo galīgs ir augstāk aplūkotais grafs G[S]). Otrkārt, ja A-kastē nav apgalvojumu par lomām, tad, saskaņā ar novērojumu (b), modeļa grafs ir koku kolekcija.

Treškārt, interpretācijā J visas uzģenerētās kopas (t.i. arī dotās kopas S) formulas ir patiesas. Tiešām, sāksim ar atomārajām formulām (ar vai bez negācijām). Tās visas ir patiesas. Tālāk pierādīsim ar indukciju pa kopas formulu loģisko dziļumu. Aplūkosim kādu kopas formulu G, kas nav atomāra (ar vai bez negācijas).

1) Ja G ir A(c)&B(c), un solis 2 nav lietojams, tad kopā ir arī abas formulas A(c), B(c). Pēc indukcijas pieņēmuma, tās ir patiesas, tātad patiesa ir arī G.

2) Ja G ir A(c)vB(c), un solis 3 nav lietojams, tad kopā ir arī viena no formulām A(c), B(c). Pēc indukcijas pieņēmuma, šī formula ir patiesa, tātad

patiesa ir arī G.

3) Ja G ir Ey(r(d,y)&B(y)), un solis 4 nav lietojams, tad kādai konstantei e kopā ir abas formulas r(d, e) un B(e). Pēc indukcijas pieņēmuma, šīs formulas ir patiesas, tātad patiesa ir arī G.

4) Ja G ir Ay(r(d,y)->B(y)), un solis 5 nav lietojams, tad jebkurai konstantei e, ja kopā ir formula r(d, e), tad tajā ir arī B(e). Pēc indukcijas pieņēmuma, formula B(e) ir patiesa, tātad patiesa ir arī G.

Tātad jāsecina, ka interpretācijā J patiesas ir visas uzģenerētās kopas formulas, t.i. arī dotās kopas S visas formulas. Q.E.D.

Algoritma pilnība

Teorēma. Ja formulu kopa S ir izpildāma (jebkādā interpretācijā), tad solī 3 vienmēr "pareizi izvēloties ceļu", algoritma darba rezultāts būs pozitīvs.

Pierādījums. Aplūkosim jebkuru interpretāciju J, kurā kopas S visas formulas ir patiesas. Algoritma darba laikā paplašināsim šo interpretāciju ar mūsu jaunievestajām konstantēm, bet šo konstantu interpretācijas vienmēr ņemsim no interpretācijas J apgabala (t.i. J apgabalu mēs nepaplašināsim).

Parādīsim, ka tad visas algoritma darba laikā radušās formulas būs patiesas mūsu definētajā interpretācijas J paplašinājumā. Sākuma formulu kopā S visas formulas ir patiesas. Aplūkosim tagad algoritma soļus.

1) Ja formula A(c)&B(c) interpretācijā J ir patiesa, tad patiesas būs arī solī 2 pievienojamās formulas A(c), B(c).

2) Ja formula A(c)vB(c) interpretācijā J ir patiesa, tad patiesa būs arī viena no formulām A(c), B(c). To arī pievienosim solī 3 (tā arī būs mūsu "nedeterminētā izvēle").

3) Ja formula Ey(r(d,y)&B(y)) interpretācijā J ir patiesa, un nevienai konstantei e mūsu ģenerētājā kopā nav abu formulu r(d, e) un B(e), tad algoritms kopai pievieno formulas r(d, e) un B(e), kur e - jauna objektu konstante. Konstantes e interpretāciju definēsim kā **to y vērtību** (no interpretācijas J apgabala), kam formula r(d,y)&B(y) ir patiesa. Tad abas pievienotās formulas r(d, e) un B(e) arī būs patiesas.

4) Ja formula Ay(r(d,y)->B(y)) interpretācijā J ir patiesa, un kādai objektu konstantei e formula r(d, e) jau ir kopā (t.i. tā ir patiesa), bet B(e) vēl nav, tad algoritms kopai pievieno formulu B(e). Arī šī formula automātiski būs patiesa.

Līdz ar to esam noskaidrojuši, ka - ar "pareizām izvēlēm solī 3" - algoritma darba laikā rodas tikai patiesas formulas. Tāpēc darba rezultātā izveidojas kopa, kurā nevar vienlaicīgi būt q(d) un ~q(d). T.i. šai gadījumā algoritma darba rezultāts ir pozitīvs. Q.E.D.

Secinājums. Ja ALC valodas slēgtu formulu kopa S ir izpildāma (jebkādā interpretācijā), tad tā ir izpildāma arī kādā interpretācijā, kuras objektu apgabals ir galīgs.(t.s. *Finite model property*). Un ja A-kastē nav apgalvojumu

par lomām, tad lomu predikātu interpretācijas var izveidot tā, lai modelis būtu koku kolekcija (t.s. *Tree model property*).

Pierādījums. Paņemsim no interpretācijas J apgabala to daļu J', kas tiek izmantota mūsu algoritma darba laikā vajadzīgo konstantu interpretācijai. Klašu un lomu predikātu interpretācijas sašaurināsim līdz J'. Tad algoritma uzģenerētājā kopā visas atomārās formulas interpretācijā J' būs patiesas. Un beidzot, atkārtojot korektības pierādījuma spriedumus, secinām, ka arī visas pārējās algoritma ģenerētās kopas formulas būs patiesas interpretācijā J'. Tātad tādas būs arī visas sākumkopas S formulas. Q.E.D.

Visu modeli nedrīkst glabāt atmiņā!

Tikko aprakstītā tablo algoritma versija savā atmiņā būtībā būvē **pilnu dotās formulu kopas S model**i. Izrādās, ka var uzbūvēt ALC formulu, kuras garums būs polinoms no n, un kurai jebkurš modelis noteikti saturēs bināru koku ar n līmeņiem, t.i. šādā modelī būs vismaz 2^n-1 objektu. Tāpēc minētajai formulai tikko aprakstītā tablo algoritma versija neizbēgami izmantos **eksponenciālu atmiņu**, t.i. piederēs klasei NEXPTIME (nevis PSPACE kā solīts Zoļina navigatorā). Šis rezultāts ir publicēts rakstā:

Joseph Y. Halpern, **Yoram Moses**: A Guide to Completeness and Complexity for Modal Logics of Knowledge and Belief. *Artificial Intelligence*, 54(2): 319-379 (1992).

Uzdevums. Jau iepriekš zinot, ka tas ir iespējams, uzbūvējiet paši ALC formulu d: C, kuras garums būtu polinoms no n, un kurai jebkurš modelis noteikti saturētu bināru koku ar n līmeņiem. [Ja mūsu rīcībā būtu lomu inversija, kvantori ar skaitliskiem ierobežojumiem un aksiomas, tad šis uzdevums nebūtu grūts. Piemēram, sadalām visus iestādes darbiniekus n līmeņos (klasēs), i-tā līmeņa darbiniekiem ir katram vismaz divi padotie i+1-jā līmenī, un nevienam darbiniekam nav vairāk par vienu priekšnieku (kur priekšnieks = padotais^{-1}). Bet kā tikt galā, ja mūsu rīcībā ir tikai loģikas ALC līdzekļi?]

Secinājums. Optimizētajā versijā mums ir jāatsakās no visa modeļa pilnīgas glabāšanas atmiņā.

Atsakoties no visa modeļa glabāšanas atmiņā, aprakstīto tablo algoritmu var nopietni optimizēt, panākot, ka tas darbojas polinomiālā atmiņā (no doto formulu kopgaruma). T.i. var pierādīt, ka **loģikai ALC A-kastes saderības uzdevums (tukšas T-kastes gadījumā) ir** NPSPACE-complete **(**un līdz ar to - arī **PSPACE-complete** - Saviča teorēma)**. Finite model property un Tree model property tiek garantētas.** (Sk. minētās Stephan Tobies disertācijas 3.nodaļu).

Šo rezultātu pirmie ieguva 1991.gadā

Manfred Schmidt-Schauss, Gert Smolka: Attributive concept descriptions with complements. *Artificial Intelligence*, 48:1-26, 1991.

NPSPACE algoritms loģikas ALC A-kastei ar tukšu T-kasti

Tagad mums jaseko līdz ne tikai formulu skaitam, kas rodas tablo algoritma darba laikā, bet arī šo **formulu kopgarumam** (jo tās taču jāglabā atmiņā). Mūsu grafa G[S] virsotnei c formulu skaits kopā L(c) nevarēja pārsniegt |S| - kopas S formulu visu apakšformulu skaitu. Bet kopas L(c) formulu kopgarums - cik liels var būt tas?

Tikko izgudroju šādu lemmu (droši vien, citiem tā jau sen bija zināma):

Lemma. Formulas visu apakšformulu kopgarums nepārsniedz šīs formulas garuma kvadrātu: $kg(F)<=g^2(F)$.

Pierādījums. Atomārām formulām garums ir 1, tātad te lemma izpildās. Tālāk seko indukcijas soļi.

Ja formulu veido ar vienvietīgu operāciju o, tad $kg(oF) = g(oF)+kg(F) <= 1 + g(F) + g^2(F) <= (1+g(F))^2 = g^2(oF)$.

Ja formulu veido ar divvietīgu operāciju o, tad $kg(FoG) = g(FoG)+kg(F)+kg(G) <= 1 + g(F) + g(G) + g^2(F) + g^2(G)<= (1+g(F)+g(G))^2 = g^2(FoG)$.

Līdz ar to lemma ir pierādīta. Q.E.D.

Šo lemmu var iegli vispārināt, vienas formulas vietā aplūkojot formulu kopu S.

Secinājums 1. Ja kopas S formulu kopgarums ir g(S), tad tās visu apakšformulu kopgarums $kg(S)<=g^2(S)$.

Tiešām, ja $kg(S)<=g^2(S)$ un kopai S pievieno vēl vienu formulu F, tad $kg(Sv\{F\}) <= kg(S) + kg(F) <= g^2(S) + g^2(F) <= (g(S)+g(F))^2 = g^2(Sv\{F\})$.

Secinājums 2. Jebkurā tablo algoritma darbības brīdī grafa G[S] jebkurai virsotnei c: $kg(L(c))<=g^2(S)$.

Šis ir mūsu pirmais nopietnais solis ceļā uz NPSPACE.

Pamatideja tablo algoritma optimizācijai ir šāda - atteiksimies no iespējas pielietot soļus 2), 3), 4), 5) pilnīgi patvaļīgā secībā, un noteiksim šo secību tā: vispirms izmantojam visas iespējas pielietot soļus 2), 3), 5), un tikai tad, kas tas vairs nav iespējams, pielietojam soli 4). Kāpēc tā?

Pieņemsim, ka mēs kādā brīdī pārstājam izmantot soli 4 (kurš vienīgais rada jaunas grafa G[S] virsotnes). Ar V_0 apzīmēsim to virsotņu kopu, kas šajā brīdī ir grafā G[S]. Tagad līdz galam izmantojam **visas** iespējas pielietot soļus 2, 3 un 5. Virsotņu kopa paliek tā pati V_0. Pēc tam atsāksim lietot soli 4 (kopā ar 2, 3 un 5). Tad grafā atkal sāks rasties jaunas virsotnes, kas nepieder kopai V_0

(no kopas V_0 virsotnēm "augs koki"). Bet virsotnēm d no V_0 formulu kopas L(d) tad **vairs nepapildināsies**! Jo neviens no soļiem nepapildina nevienu kopu L(d) "uz atpakaļ" no jaunradītajām virsotnēm. [Pārliecināsimies!]

Šis novērojums mums dod iespēju nedeterminēto tablo algoritmu darbināt rekursīvi - katram no augošo koku zariem atsevišķi. Līdz ar to **atmiņā būs jāglabā nevis viss grafs, bet tikai viens koka zars!** To var realizēt tā:

a) Sākumā (kad mums ir tikai formulu sākumkopai S atbilstošais grafs), līdz galam izmantojam **visas** iespējas pielietot soļus 2, 3 un 5. Ja jau ir atradusies pretruna, tad **ziņojam par negatīvu rezultātu** un darbu beidzam. Ja pretrunas nav, varēsim turpināt. Līdz šīm brīdim (kad soļus 2, 3, 5 vairs nav iespējams lietot) tablo algoritms ir "saražojis" formulas ar kopējo garumu, kas (saskaņā ar secinājumu 2) nepārsniedz $k(S)*g^2(S)$, kur k(S) ir objektu konstantu kopskaits kopas S formulās. Tā kā $k(S)<=g(S)$, tad $k(S)*g^2(S) <= g^3(S)$. Tātad šis mūsu pirmais solis tiek galā, izmantojot tikai polinomiālu atmiņu.

b) Tālāk sameklējam mūsu grafā jebkuru tādu virsotni d, kurai kopā L(d) ir formula Ey(r(d,y)&B(y)), kurai var pielietot soli 4. Ja neatrodam, tad **ziņojam par pozitīvu rezultātu** un algoritma darbu beidzam. Ja atrodam , tad solis 4 mums "uzģenerēs" formulas r(d, e) un B(e), kur e - pilnīgi jauna objektu konstante. Mūsu grafā tas nozīmē, ka tiek radīta jauna virsotne e, un no virsotnes d uz to tiek novilkta šķautne r(d, e), bet kopa L(e) tagad ir {B(e)}.

c) Pēc tam atkal izmantojam **visas** iespējas pielietot soļus 2, 3, 5. Kādas ir šīs iespējas? Soli 5 varam cerēt pielietot tikai iepriekšējās virsotnes d kopā L(d), ja tur ir kāda formula izskatā Ay(r(d,y)->C(y)). Tad kopai L(e) būs jāpievieno formula C(e), t.i. papildināties var tikai kopa L(e). Savukārt, soļus 2, 3 varam cerēt pielietot tikai kopas L(e) formulām izskatā D&E un DvE. Tātad arī šai gadījumā papildināties var tikai kopa L(e). Ja pēc visa tā jau ir atradusies pretruna, tad **ziņojam par negatīvu rezultātu** un algoritma darbu beidzam. Ja pretrunas nav, tad varam turpināt.

d) Tālak mums ir jāorganizē **rekursija**, kas iet uz priekšu pa tikko iesākto zaru. Meklējam zara galā - kopā L(e) kādu formulu Ey(q(e,y)&D(y)), kurai varētu pielietot soli 4. Ja tādu formulu neatrodam, tad virsotni e likvidējam un ejam pa zaru atpakaļ (t.i. ejam uz soli f) - sk. zemāk). Ja atrodam , tad solis 4 mums "uzģenerēs" formulas q(e, f) un D(f), kur f - pilnīgi jauna objektu konstante. Mūsu grafā tas nozīmē, ka tiek radīta jauna virsotne f, un no virsotnes e uz to tiek novilkta šķautne q(e, f), bet kopa L(f) tagad ir {D(f)}.

e) Ejam atpakaļ uz soli c), tikai virsotņu pāra (d, e) vietā mums tagad ir pāris (e, f).

f) Ja ar kārtējo virsotni mums nav paveicies, un solis 4 tai vairs nav lietojams, tad kāpjamies atpakaļ uz iepriekšējo virsotni un ejam uz soli d). Nedaudz atšķirīgi jārīkojas, ja esam atkāpušies jau līdz sākumgrafa virsotnei d - tad ir jāiet uz soli b).

Tādā veidā mūsu algoritma darbs visu laiku notiek tikai vienā zarā, un neko daudz vairāk par šī zara virsotņu formulu kopām L(e) mums atmiņā nav jāglabā. Vienas kopas L(e) formulu kopgarums, kā jau zinām, nepārsniedz $g^2(S)$. Bet cik daudz virsotņu var saveidoties vienā zarā? No augšminētajiem novērojumiem mēs jau zinām, ka ja e ir jaunradīta virsotne, kas solī 4 radusies no virsotnes d, tad vissarežģītākā L(e) formula ir vienkāršāka par vissarežģītāko L(d) formulu. Tātad vienā zarā var sarasties ne vairāk kā g(S) virsotņu, un formulu kopgarums, kas mums jāglabā, tātad nevar pārsniegt $g^3(S)$.

Kopā ar sākotnējā soļa a) vajadzībām, mūsu nedeterminēta algoritma darbam nepieciešama atmiņa, kuras izmērs daudz nepārsniedz $2g^3(S)$, t.i. polinomu no sākumkopas S formulu kopgaruma. Tātad - NPSPACE - Q.E.D.

Saskaņā ar Saviča teorēmu[Wikipedia], NPSPACE nedeterminētu algoritmu vienmēr var determinizēt tā, ka iegūtais determinētais algoritms arī izmantos tikai polinomiālu atmiņu. Līdz ar to esam parādījuši, ka **loģikas ALC A-kastei ar tukšu T-kasti saderības uzdevums (A-Box consistency) pieder klasei PSPACE.**

Walter J. Savitch savu teorēmu publicēja 1970.gadā:

Walter J. Savitch. Relationships between Nondeterministic and Deterministic Tape Complexities. *Journal of Computer and System Sciences*, Vol. 4, No. 2 (April 1970), pp. 177-192.

3.4. Tablo algoritms ALC paplašinājumiem