Kapitola 2

Inferencia vo výrokovej logike

Pri použití výrokovej logiky pre reprezentáciu znalostí je možné vyčleniť tieto hlavné typy úloh, ktoré je potrebné riešiť:

Hlavné typy úloh

- dedukčná úloha,
- úloha validnosti,
- úloha splniteľnosti.

Pri *dedukčnej úlohe* sú znalosti o probléme (vyjadrené ako znalostná báza KB reprezentujúca súhrn dostupných znalostí) reprezentované vetou vo výrokovom počte. Cieľom je nájsť ďalšie znalosti, ktoré zo známych znalostí vyplývajú. Predpokladá sa, že báza znalostí KB je pravdivá (teda implicitne sa uvažujú iba tie interpretácie, v ktorých nadobúda hodnotu logickej pravdy) a hľadá sa nejaká znalosť F taká, pre ktorú bude platiť $KB \models F$, kde znak $\models$ reprezentuje *logické vyplývanie*. Dedukcia má charakter overovania – nová znalosť F reprezentuje hypotézu, ktorú je potrebné dokázať. V ilustračnom príklade na strane 2 by takýmito novými znalosťami mohli byť napríklad tvrdenia "ak Fero podvádzal tak aj Stano podvádzal" alebo "ak Ondro podvádzal tak potom aj Fero podvádzal".

Dedukčná úloha

Úloha určenia *validnosti* tiež používa reprezentáciu znalostí o nejakom probléme vo forme znalostnej bázy KB, vyjadrenej vetou výrokového počtu. Cieľom je rozhodnúť, či táto veta je validná – teda či každý svet je jej modelom a teda či pravdivosť danej vety nezávisí od jej interpretácie. Príkladom by mohlo byť, ak by sme v ilustračnom príklade na strane 2 skúmali či "z platnosti, že ak Stano podvádzal tak podvádzal aj Ondro, a

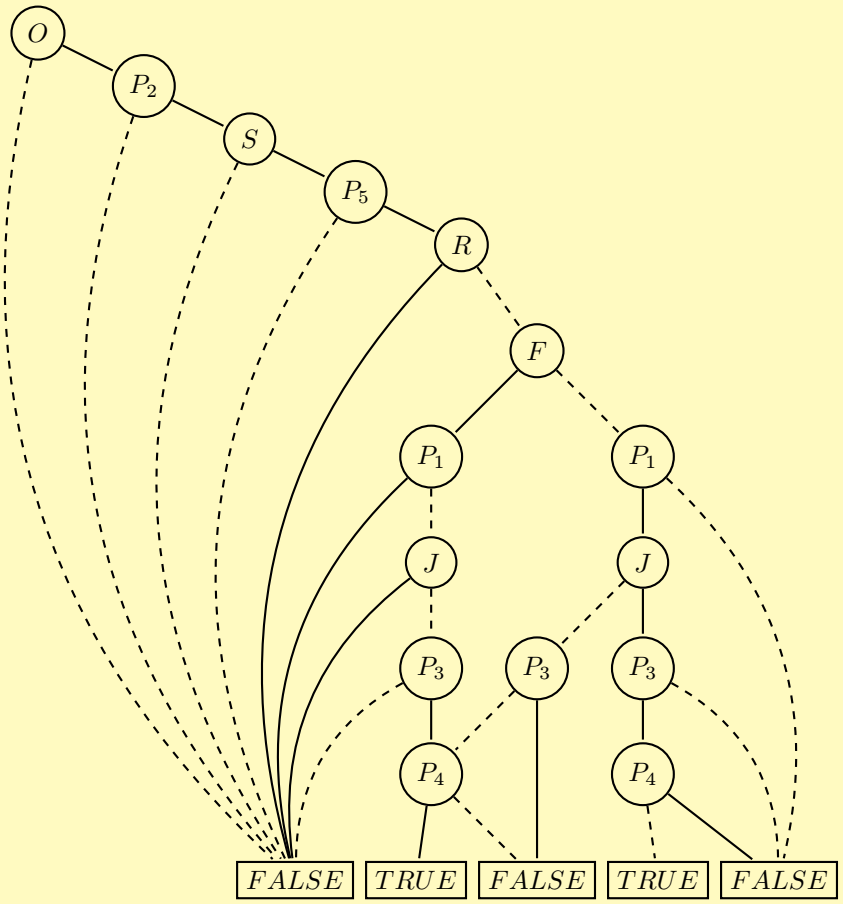
Validnosť

ILUSTRAČNÝ PRÍKLAD

Vráťme sa k podozrivým z predošlej kapitoly – Ferovi, Ondrovi, Stanovi, Rudovi a Jarovi.
Úloha 1: *Je potrebné určiť, ktorý z piatich uvedených študentov pri skúške podvádzal.*
Úloha 2: *Je potrebné na základe zistených výpovedí potvrdiť platnosť tvrdení:*

- *ak Fero podvádzal, tak potom podvádzal aj Stano,*
- *ak Ondro podvádzal, tak potom podvádzal aj Fero.*

Reprezentácia: *Uvedený problém daný šiestimi výpoveďami (minule reprezentovaný ako CNF) je možné reprezentovať pomocou binárneho rozhodovacieho diagramu podľa obrázku:*



Riešenie 1: *Do úvahy prichádzajú tri riešenia, keď podvádzajúcimi boli Ondro a Stano, a to buď sami alebo v spolupáchateľstve s Ferom alebo Jarom.*
Riešenie 2: *Z výpovedí vyplýva iba platnosť prvého tvrdenia (ak Fero podvádzal, tak potom podvádzal aj Stano), druhé tvrdenie z výpovedí nevyplýva.*

Ilustr. 2.1: Príklad pre inferenciu vo výrokovej logike

platnosti, že ak Stano podvádzal tak naopak Ondro nepodvádzal, vyplýva že Stano je nevinný, pretože nepodvádzal".

Spniteľnosť

V prípade *úlohy splniteľnosti*, ktorá je známa aj pod označením *SAT* (SATisfiability) problém, sú znalosti o riešenom probléme (znalostná báza KB reprezentujúca súhrn dostupných znalostí reprezentovaný ako konjunkcia týchto znalostí) opäť vyjadrené vo forme vety vo výrokovom počte. Cieľom je nájsť model znalostnej bázy – takú interpretáciu, pri ktorej daná veta reprezentujúca znalosti nadobúda hodnotu logickej pravdy. Každá takáto interpretácia, charakterizovaná nejakým priradením logických hodnôt symbolom, je riešením úlohy. V ilustračnom príklade na strane 2 by hľadaným modelom bolo zistenie, kto zo študentov pri skúške podvádzal a kto nie, pričom znalostnou bázou by bolo vyjadrenie znalostí, dostupných v zadaní úlohy, pomocou výrokového počtu.

Alternatívne sa niekedy za cieľ považuje namiesto nájdenia jednej vhodnej interpretácie nájsť všetky vhodné interpretácie alebo namiesto ich nájdenia určiť ich počet alebo dokonca iba rozhodnúť, či model vôbec existuje alebo nie.

Vzťahy medzi úlohami

Medzi jednotlivými úlohami existujú vzťahy, umožňujúce redukovať jeden typ úlohy na iný. Príkladom je redukcia ostatných úloh na úlohu hľadania splniteľnosti:

- Pri hľadaní validnosti je možné namiesto priameho hľadania validnosti KB skúmať, či existuje model pre $\neg KB$ (ak existuje, tak KB aspoň pre jednu interpretáciu nie je splniteľná a teda nie je validná).
- Na dokázanie platnosti $KB \models F$ stačí dokázať, že neexistuje taká interpretácia, ktorá by bola modelom bázy KB a súčasne by nebola aj modelom pre F – stačí dokázať, že veta $KB \wedge \neg F$ je nesplniteľná, nemá žiadnu interpretáciu ktorá by bola jej modelom. Naopak pre dokázanie neplatnosti $KB \models F$ stačí nájsť aspoň jeden taký model bázy KB, ktorý nie je súčasne aj modelom F – a teda dokázať, že veta $KB \wedge \neg F$ je splniteľná.

Iným príkladom je redukcia úloh na hľadanie validnosti:

- Pre dokázanie platnosti $KB \models F$ stačí dokázať, že $KB \rightarrow F$ je validné.
- Ak je KB validná, tak je aj splniteľná a je známy jej model (každá interpretácia symbolov). Ak však validná nie je, môže avšak nemusí byť splniteľná. Preto v tomto prípade pre potvrdenie splniteľnosti (avšak bez určenia konkrétnych modelov) je potrebné ešte potvrdiť, že $\neg KB$ nie je validná.

A posledným prípadom vzájomnej redukcie úloh je redukcia na dedukčnú úlohu:

- Ak sa potvrdí vyplývanie $\neg KB \models F \wedge \neg F$, tak neexistuje model pre $\neg KB$ a teda KB je validná.
- Dôkaz vyplývania $\neg KB \models F \wedge \neg F$ znamená súčasne aj splniteľnosť KB pre ľubovoľnú interpretáciu. Na druhej strane, ak sa podarí dokázať $KB \models F \wedge \neg F$, tak KB nie je splniteľná. Ak sa však nepodarí dôkaz ani jedného z daných dvoch vyplývaní, tak KB je síce splniteľná, avšak jej modely neboli určené.

VLASTNOSTI ODVODZOVACÍCH PROCEDÚR

Pre riešenie uvedených úloh je možné použiť rôzne odvodzovacie metódy. Pre každú z nich sú dôležité dve vlastnosti:

- správnosť, zachovávanie pravdivosti,
- úplnosť.

Každá procedúra musí zachovávať *pravdivosť*, teda jej závery musia byť správne – musia byť pravdivé v tom zmysle, že odvodí iba to, čo skutočne vyplýva z logických viet, nad ktorými robí odvodzovanie. Neodvodí nič, platnosť čoho by bola sporná vzhľadom na vstupné vety. Správna procedúra neposkytne ako riešenie niečo, čo v skutočnosti riešením nie je – a teda napríklad nenájde riešenie v prípade, že riešenie neexistuje.

Na druhej strane *úplnosť* znamená, že procedúra dokáže nájsť hľadané riešenie v tom prípade, že toto riešenie existuje. Na druhej strane, ak takáto procedúra riešenie nedokáže nájsť, znamená to, že riešenie skutočne neexistuje.

ODVODZOVANIE PRE BDD

Uvedené úlohy sú jednoduché pri použití reprezentácie pomocou binárneho rozhodovacieho diagramu. V tomto prípade nájsť model logickej vety znamená nájsť nejakú cestu od koreňového uzla až k TRUE terminálnemu uzlu, pričom značenie hrán pozdĺž tejto cesty predstavuje priradenie pravdivostných hodnôt jednotlivým symbolom, ktoré charakterizuje daný model reprezentovanej vety. Nájsť všetky takéto cesty znamená nájsť všetky modely.

Reprezentovaná logická veta je teda splniteľná, ak existuje terminálny uzol TRUE a existuje aspoň jedna cesta od koreňového uzla k tomuto terminálnemu uzlu. Veta je validná vtedy, ak neexistuje terminálny uzol FALSE resp. žiadna cesta od koreňového uzla k tomuto terminálnemu uzlu. Pre riešenie dedukčnej úlohy všetky cesty od koreňového uzla k TRUE terminálnemu uzlu je potrebné dodatočne analyzovať, či sú modelmi aj dokazovanej hypotézy.

Odvodzovanie pre DNF

Podobne jednoduchá situácia je v prípade, keď je použitá reprezentácia pomocou DNF. Reprezentácia daná ako disjunkcia konjunktívnych klauzúl je vlastne vymenovanie všetkých modelov – každá konjunkcia literálov predstavuje jeden alebo viac modelov reprezentovanej vety. Neprázdna DNF znamená, že skúmaná veta je splniteľná s počtom modelov daným počtom tých interpretácií, ktoré sú pokryté klauzulami. Pre dôkaz validity je potrebné analyzovať klauzuly, či pokrývajú všetky možné svety – ak nie, tak existuje interpretácia, v ktorej skúmaná veta nie je splnená. A dedukčnú úlohu je potrebné riešiť rovnakým spôsobom ako pre prípad reprezentáciou BDD.

Odvodzovanie pre CNF

Pri riešení úloh v prípade, že je použitá všeobecná reprezentácia alebo reprezentácia v tvare CNF, je možné použiť jeden z dvoch nasledovných prístupov:

- priamy prístup,
- nepriamy prístup.

Metóda pravdivostných tabuliek

Pre odvodzovanie je možné použiť metódu pravdivostných tabuliek, ktorá je reprezentantom priameho prístupu, pričom zvláda ako všeobecný tvar logickej vety tak aj CNF. Táto metóda primárne cieli na úlohu splniteľnosti – explicitným spôsobom vymenúva všetky možné interpretácie na základe zoznamu použitých symbolov a následne určuje pravdivosť jednotlivých častí skúmanej vety a na ich základe pravdivosť tejto vety ako celku. Nedostatkom metódy je jej malá škálovateľnosť – zložitosť metódy závisí na počte použitých symbolov a explicitne je možné efektívne generovať a analyzovať iba obmedzený počet interpretácií.

Metóda ekvivalentných transformácií

Alternatívou primárne zameranou na dedukčnú úlohu je postupná transformácia logickej vety resp. iba jej časti pomocou transformácií zachovávajúcich ekvivalentnosť (Tab. 1.4) dovtedy, kým výsledkom transformácie nie je žiadaný výraz. Tým žiadaným výrazom môže byť hypotéza, ktorej vyplývanie z danej logickej vety je potrebné dokázať (priamy prístup), alebo žiadaným výrazom môže byť ľubovoľný výraz reprezentujúci spor pri dokazovaní nesplniteľnosti (nepriamy prístup). Vďaka veľkému množstvu rôznych ekvivalentných transformácií aj táto metóda je obmedzená iba na menšie logické vety.

Rezolučné odvodzovanie

Reprezentantom nepriameho prístupu je rezolučné odvodzovanie (rezolvenčná metóda bola objavená J. A. Robinsonom v 1965), ktoré cieli na hľadanie sporu pri riešení dedukčnej úlohy alebo úlohy validnosti. Na rozdiel od predchádzajúcej metódy využívajúcej rôzne ekvivalentné transformácie, využíva iba jednu transformáciu – rezolvenčné odvodzovacie pravidlo.

Tým, že sa snaží zo vstupnej vety dedukovať platnosť sporu, tak sa vlastne zameriava na nesplniteľnosť vyšetrovanej vety.

Samotné dokazovanie je založené na využití rezolvenčného odvodzovacieho pravidla a zachovávania splniteľnosti týmto pravidlom – použitie transpozície hovorí, že ak je rezolventa nesplniteľná, tak originálna množina klauzúl musí byť tiež nesplniteľná. Pravidlo sa používa takým spôsobom, keď celé úsilie smeruje k nájdeniu rezolvencie typu

$$\frac{P \qquad \neg P}{\oslash}$$

kde $\oslash$ reprezentuje nájdený spor. Disjunktívna klauzula je splniteľná, ak aspoň v jednej interpretácii aspoň jeden literál klauzuly je interpretovaný ako pravdivý. Prázdna klauzula $\oslash$ však nemá žiadny literál a teda nie je splniteľná.

DETEKCIA SPORU

Ak by sa rezolvencia vykonávala nad množinou klauzúl, ktoré sú navzájom protirečivé, tak po nejakom počte opakovaných rezolvencií bude detekovaný spor. To možno ilustrovať napríklad na klauzulách $\neg P \vee Q$, $\neg Q \vee R$, P a $\neg R$. V tomto prípade by rezolvencie mohli vyzerať

$$\dfrac{\dfrac{\dfrac{\neg P \vee Q \qquad \neg Q \vee R}{\neg P \vee R} \qquad P}{R} \qquad \neg R}{\oslash}$$

V prípade, že už nie je možné vytvoriť takú rezolventu, ktorá ešte nebola vytvorená, a spor ani v jednom prípade nenastal, tak nie je možné vyvrátiť platnosť skúmanej množiny klauzúl.

ALGORITMUS PRE REZOLUČNÉ HĽADANIE SPORU

Formálna podoba rezolučného algoritmu pre vyhľadávanie spornosti logickej vety vyjadrenej v CNF forme je uvedená ako Alg. 2.1. Jedná sa o cyklický proces, kde v každej iterácii sa z množiny uchovávaných klauzúl KB vyberajú dve klauzuly, vytvára sa z nich množina rezolvent (vytvárajú sa všetky rezolventy, ktoré je možné z daných dvoch vybraných klauzúl vytvoriť) a tieto rezolventy sa následne môžu pridať k uchovávaným klauzulám.

VYLÚČENIE KLAUZÚL S ČISTÝM LITERÁLOM

Takzvaný *čistý* literál je taký literál, ku ktorému v skúmanej KB neexistuje komplementárny literál. Keďže sa daný symbol vyskytuje iba v jednom tvare (či už ako pozitívny alebo negatívny literál), nie je možné na základe tohto symbolu vykonať žiadnu rezolvenciu. A teda nie je možné klauzulu, ktorá obsahuje literál tohto symbolu, postupnosťou viacerých rezolúcií redukovať na prázdnu klauzulu, reprezentujúcu spor. Z hľadiska cieľa algoritmu sú takéto klauzuly zbytočné a preto je ich možné vypustiť.

vstup: znalostná báza KB v klauzulárnom tvare
výstup: $TRUE$ v prípade dokázania sporu alebo $FALSE$ inak

1. $KB := KB \setminus \{$ *C: C obsahuje čistý literál* $\}$
2. **while** $\neg(\oslash \in KB)$ **do**
3. *vyber* $C_1, C_2 \in KB$
4. **if** *žiadny nový pár* C_1, C_2 *neexistuje* **then return** $FALSE$
5. $Rez := \{R : R$ *je rezolventa* C_1 a C_2, *nie je tautológia* $\}$
6. **for** $R \in Rez$ do
7. $R :=$ *redukcia viacnásobného výskytu v* R
8. **if** *žiadna klauzula z* KB *nezahŕňa* R
9. **then** $KB := \{R\} \cup KB \setminus \{C : R$ *zahŕňa* $C\}$
10. **return** $TRUE$

Alg. 2.1: Rezolučné hľadanie sporu

Pri vytváraní rezolvent sa uvažujú iba tie rezolventy, ktoré nie sú tautológiami – tautológia je validná a teda môže byť odstránená z logickej vety bez zmeny pravdivostnej hodnoty vety. Rezolventa je tautológiou, ak súčasne obsahuje nejaký literál v priamej aj negovanej podobe. To znamená, že vzniká iba vtedy, ak rezolvované klauzuly obsahujú viac než jeden pár komplementárnych literálov. V prípade tautológie by mohlo nastať

Vylúčenie tautológií

$$\dfrac{\dfrac{\dfrac{\neg P \vee Q \vee R \quad \neg Q \vee P}{\neg P \vee R \vee P} \quad \neg R}{\neg P \vee P} \quad \neg P}{\neg P}$$

kde je vidno, že tautológia v konečnom dôsledku neumožní získať prázdnu klauzulu (a teda nájdenie sporu nemôže byť založené na tautológii) a preto je vhodné sa takýchto rezolvent zbaviť a v ďalšom ich neuvažovať.

Vylúčenie viacnásobného literálu

Ak by vstupné klauzuly obsahovali rovnaký literál, tak potom by rezolventa obsahovala dva výskyty tohto literálu. V tomto prípade je potrebné rezolventu previesť na ekvivalentný tvar použitím $P \vee P \equiv P$. Ak by sa to neurobilo, tak by mohla nastať situácia podľa

$$\dfrac{\dfrac{P \vee Q \qquad \neg Q \vee P}{P \vee P} \qquad \neg \mathrm{P}}{P}$$

kedy nedochádza k detekcii sporu. Preto v uvažovaných rezolventách je potrebné odstrániť prípadný viacnásobný výskyt rovnakého literálu.

VYLÚČENIE ZAHRNUTÝCH KLAUZÚL

Pri pridávaní rezolvent k ostatným klauzulám ešte hrá úlohu vlastnosť *zahrnutia*. Kratšia klauzula môže zahŕňať dlhšiu klauzulu, napríklad klauzula C (označovaná ako zahŕňajúca) zahŕňa klauzulu D (označovanú ako zahrnutá) vtedy, ak klauzula D obsahuje tie isté literály ako klauzula C a okrem nich D môže obsahovať aj ďalšie literály. V prípade súčasného výskytu zahŕňajúcej aj zahrnutej klauzuly nie je potrebné aj naďalej uvažovať zahrnutú klauzulu, pretože tým istým postupom, ktorý redukuje zahrnutú klauzulu na prázdnu, je možné redukovať aj zahŕňajúcu klauzulu na prázdnu klauzulu a to tým skôr, čím je zahŕňajúca klauzula kratšia (a teda ponechanie zahrnutej klauzuly je zbytočné). Toto možno ilustrovať na príklade

$$\dfrac{\dfrac{\dfrac{\neg P \vee Q \vee R \qquad \neg Q}{\neg P \vee R} \qquad \neg R}{\neg P} \qquad P}{\oslash} \qquad \dfrac{\dfrac{Q \vee R \qquad \neg Q}{R} \qquad \neg R}{\oslash}$$

kde $Q \vee R$ je zahŕňajúcou a $\neg P \vee Q \vee R$ zahrnutou klauzulou.

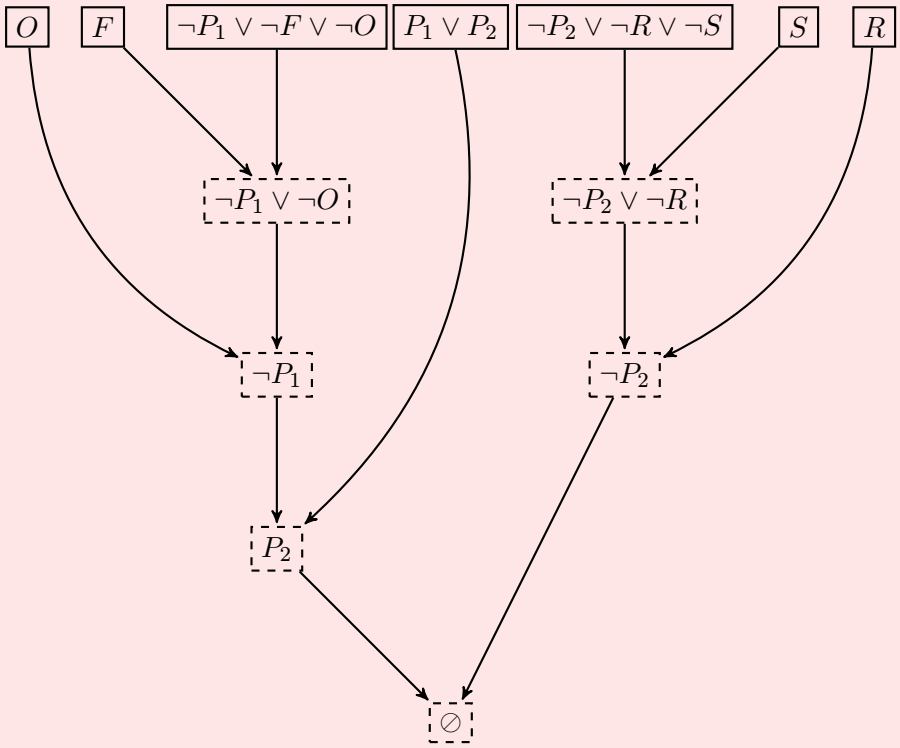
Na základe zahrnutia je možné, že niektorá rezolventa nie je pridaná k ostatným klauzulám KB alebo naopak, pridanie rezolventy spôsobí vypustenie nejakej klauzuly z KB.

Ukončenie algoritmu môže nastať z dvoch dôvodov – alebo bola vygenerovaná prázdna klauzula (a dôkaz bol úspešný) alebo prázdna klauzula nebola vygenerovaná a zároveň už nie je možné vytvoriť takú rezolventu, ktorá ešte vytvorená nebola.

Príklad 2.1 Ako ukážku rezolvenčného uvažovania opäť zoberme ilustračný príklad na strane 2 s tým, že sa pokúsime overiť predpoklad, že aspoň jeden zo študentov nepodvádzal a teda ku klauzulám pribudne ešte päť nových klauzúl, čiže overovaná veta bude mať výsledný tvar $\ldots \wedge (\neg R \vee O) \wedge F \wedge O \wedge S \wedge R \wedge J$ kde pre overenie že aspoň jeden z nich nepodvádzal sme pridali opačné tvrdenie že všetci podvádzali, pretože budeme hľadať spornosť (hypotézu teda negujeme a snažíme sa v dôkaze získať prázdnu klauzulu).

DERIVAČNÝ GRAF

Obr. 2.1 zobrazuje jeden možný spôsob nájdenia sporu vo forme *derivačného grafu*, pričom sú zobrazené iba tie rezolvencie, ktoré prispievajú k



Obr. 2.1: Derivačný graf

redukcii na prázdnu klauzulu (a teda vďaka tomu má graf stromovú štruktúru). Uzly grafu reprezentujú klauzuly dvojakého typu:

- vstupné klauzuly (klauzuly vybraté zo znalostnej bázy),
- rezolventy.

Prvý typ klauzúl je znázornený plnými uzlami, ktoré nemajú predchodcov, zatiaľ čo čiarkované uzly s predchodcami reprezentujú druhý typ klauzúl. Hrany reprezentujú uskutočnené rezolvencie, keď vždy dve hrany spájajú vstupné klauzuly s ich rezolventou.

Okrem uvedených rezolvencií však mohli byť vykonané aj ďalšie. Takými mohli byť napríklad rezolvencia klauzuly $P_1 \vee P_2$ s klauzulou $\neg P_2$, ktorej výsledok by po zložení s $\neg P_1$ vytvoril alternatívnu rovnako dlhú cestu v derivačnom grafe vedúcu na prázdnu klauzulu.

Redundantné a zbytočné rezolvencie

K prázdnej klauzule by sa dalo dopracovať aj inak – trebárs postupom znázorneným derivačným grafom pre vytvorenie rezolventy P_2 by sa dali

vytvoriť aj rezolventy P_1, P_3, P_4 a P_5 a táto pätica pridaných klauzúl by spolu so vstupnou klauzulou $\neg P_1 \vee \neg P_2 \vee \neg P_3 \vee \neg P_4 \vee \neg P_5$ viedli na prázdnu klauzulu. Toto by si však vyžiadalo niekoľkokrát vyšší počet realizovaných operácií než čo znázorňuje derivačný graf. •

Vytváranie rôznych rezolvent môže viesť na rôzne dlhé cesty nájdenia sporu. Dokonca niekedy vytvorenie nejakej rezolventy môže viesť do slepej uličky, pretože pri aktuálnych znalostiach neumožní redukciu na prázdnu klauzulu. Je teda zrejmé, že výber dvojíc klauzúl, ktoré majú byť rezolvované, má veľký vplyv na to, koľko rezolvent bude generovaných dovtedy, kým sa podarí odvodiť prázdnu klauzulu (a teda koľko rezolvent bude generovaných zbytočne, pretože alebo neprispejú k odvodeniu prázdnej klauzuly alebo prispejú k nájdeniu alternatívnych ciest).

STRATÉGIA VOĽBY KLAUZÚL

Je možné používať rozličné stratégie voľby klauzúl, ktoré by mali byť navzájom rezolvované (riadok 3 algoritmu Alg. 2.1), s cieľom minimalizovať počet rezolvencií. Najznámejšími stratégiami sú

- usporiadanie podľa veľkosti,
- jednotková preferencia,
- oporná množina,
- hĺbková saturácia.

STRATÉGIA USPORIADANIA PODĽA VEĽKOSTI

Pri *usporiadaní podľa veľkosti* sa preferujú malé klauzuly pred klauzulami väčšími, pričom veľkosť klauzúl sa určuje podľa počtu literálov v klauzulách. Preferencia malých klauzúl je založená na snahe, aby produkované rezolventy boli čo najmenšie (a tým pádom čo najrýchlejšie redukovateľné na prázdnu klauzulu).

STRATÉGIA JEDNOTKOVEJ PREFERENCIE

Pri *jednotkovej preferencii* sa používajú jednotkové klauzuly s jediným literálom. Stratégia je založená na fakte, že v prípade že jedna z klauzúl obsahuje iba jeden literál, je rezolventa kratšia než druhá použitá klauzula. Použitie tejto stratégie je založené na nádeji, že explicitný dôraz na skracovanie klauzúl umožní skôr dosiahnuť prázdnu klauzulu. Táto stratégia však nie je úplná, nie vždy sa dá dosiahnuť prázdna rezolventa napriek tomu, že vstupná veta je sporná.

STRATÉGIA OPORNEJ MNOŽINY

Pri stratégii *opornej množiny* sa vychádza z myšlienky, že časť logickej vety je neprotirečivá a teda nie je možné odvodiť prázdnu klauzulu iba na základe klauzúl z tejto časti. Protirečivosť je spôsobená druhou časťou vety (napríklad negáciou dokazovanej hypotézy). Preto v každej rezolvencii by ako jedna z klauzúl mala byť vybratá klauzula buď priamo zo spornej časti vety alebo nejaká klauzula, ktoré bola z tejto spornej časti odvodená.

Stratégia hĺbkovej saturácie

Pri *hĺbkovej saturácii* všetky klauzuly vo vstupnej množine sú umiestnené v hĺbke 0. Rezolventa je v hĺbke $n+1$ vtedy, ak jedna jej rodičovská klauzula je v hĺbke n a druhá v hĺbke menšej alebo rovnej n. Stratégia predpisuje najprv odvodiť všetky možné rezolventy v hĺbke 1, potom všetky v hĺbke 2, atď. Je to systematická stratégia – je vlastne analógiou *prehľadávania do šírky*.

Stratégia vstupnej a lineárnej rezolvencie

Pri stratégii *vstupnej* rezolvencie každá operácia rezolvencie kombinuje jednu zo vstupných klauzúl (zadaných na začiatku) s nejakou inou klauzulou. Táto stratégia nie je úplná (iba v prípade, ak klauzuly majú tvar Hornových klauzúl). Zovšeobecnením je *lineárna* rezolvencia, keď dve klauzuly môžu byť kombinované iba v prípade, že jedna je alebo jednou zo vstupných klauzúl alebo je predkom druhej klauzuly usporiadaní danom derivačným grafom.

Tieto stratégie robia rezolvenčné odvodzovanie správnym, pretože zachovávajú splniteľnosť – ak originálna množina klauzúl je splniteľná, tak aj rezolventy sú splniteľné (a nebude odvodená prázdna klauzula $\oslash$) a ak prázdna klauzula bude odvodená, tak iba vtedy ak vstupné klauzuly nie sú splniteľné. Niektoré stratégie sú úplné (ak množina klauzúl nie je splniteľná, tak bude odvodená prázdna klauzula), iné však úplné nie sú (negarantujú odvodenie prázdnej klauzuly v prípade nesplniteľnosti originálnej množiny klauzúl).

Tablo kalkul

Príkladom priameho prístupu je *Tablo kalkul*, ktorý je zameraný na problém splniteľnosti. Patrí do skupiny sémantických metód, ktoré pri odvodzovaní zohľadňujú pravdivostné hodnoty. Vytvára tzv. *tablo* čo je stromová štruktúra, ktorá obsahuje skúmanú logickú vetu vo svojom koreňovom uzle. Jednotlivé vetvy stromu obsahujú uzly, z ktorých každý reprezentuje nejaký podvýraz výrazu reprezentovaného niektorým vyššie zaradeným uzlom v tej istej vetve. Dochádza tak k postupnej dekompozícii logickej vety, pričom sú skúmané všetky možnosti dekompozície. Pozdĺž vetiev stromu dochádza k zjednodušovaniu pôvodnej vety až na jednotlivé literály. Takáto štruktúra umožňuje určiť splniteľnosť logickej vety a v prípade splniteľnosti aj jej modely. Štruktúra samotného algoritmu je v Alg. 2.2.

Pod neredukovanou formulou sa rozumie taký logický výraz, umiestnený v nejakom uzle stromu, ktorého časti, na ktoré je možné daný výraz redukovať, sa nenachádzajú v každej vetve stromu, ktorá prechádza uzlom v ktorom je ten výraz.

Aby boli spracované všetky výrazy, používa sa nejaký systematický spôsob ich výberu. Jednoduchým príkladom je výber takého, ktorý sa nachádza čo najbližšie koreňu a v prípade viacerých takýchto preferencia toho, ktorý je umiestnený v najľavejšej vetve stromu.

vstup: veta F v ľubovoľnom tvare
výstup: tablo T

1. T := *vytvor_koreň(F)*
2. **while** neredukovaná_formula $\in T$
3. $\quad R$:= *vyber_formulu(T)*
4. $\quad T$:= *pridaj_subformuly(T,R)*
5. $\quad$ **if** kontradikcia **then** T := *uzavri_vetvu(T)*

Alg. 2.2: Tablo algoritmus

TABLO DEKOMPOZIČNÉ PRAVIDLÁ

Redukcia výrazov sa deje podľa dekompozičných pravidiel uvedených v Tab. 2.1. Tieto pravidlá sa delia na dve skupiny – vetviace a nevetviace pravidlá. Vetviace pravidlá redukujú výraz na podvýrazy, z ktorých postačuje splnenie jedného aby bol splnený pôvodný výraz. Preto pravidlo takéhoto typu spôsobuje vetvenie a do každej novej vetvy vkladá jednu z alternatív. Príkladom pravidla takéhoto typu je

$$\frac{\neg(P \wedge Q)}{\neg P \mid \neg Q}$$

hovoriace, že ak má byť konjunkcia nesplnená, potom musí byť nesplnený aspoň jeden z argumentov tejto konjunkcie. Toto pravidlo po aplikácii na nejakú vetvu $[\ldots, \neg((\mathbf{A} \rightarrow \mathbf{B}) \wedge \mathbf{C})]$ rozvetví túto vetvu podľa podvýrazov do dvoch vetiev zdieľajúcich svoj začiatok $[\ldots, \neg((A \rightarrow B) \wedge C), \neg(\mathbf{A} \rightarrow \mathbf{B})]$ a $[\ldots, \neg((A \rightarrow B) \wedge C), \neg\mathbf{C}]$.

Nevetviace pravidlá redukujú výraz na podvýrazy, z ktorých všetky musia byť splnené aby bol splnený pôvodný výraz. Preto pravidlo takéhoto typu pokračuje v danej vetve, do ktorej vkladá všetky podvýrazy. Príkladom pravidla takéhoto typu je

$$\frac{\neg(P \rightarrow Q)}{\begin{array}{c} P \\ \neg Q \end{array}}$$

hovoriace, že ak má byť implikácia nesplnená, tak predpoklad musí byť interpretovaný ako pravdivý a záver zase ako nepravdivý. Toto pravidlo

$\dfrac{P \wedge Q}{\begin{array}{c}P\\Q\end{array}}$	$\dfrac{\neg(P \vee Q)}{\begin{array}{c}\neg P\\\neg Q\end{array}}$	$\dfrac{\neg(P \to Q)}{\begin{array}{c}P\\\neg Q\end{array}}$
$\dfrac{P \vee Q}{P \mid Q}$	$\dfrac{P \to Q}{\neg P \mid Q}$	$\dfrac{\neg(P \wedge Q)}{\neg P \mid \neg Q}$
$\dfrac{P \leftrightarrow Q}{P \wedge Q \mid \neg P \wedge \neg Q}$ alebo $\neg P \vee Q$ $P \vee \neg Q$	$\dfrac{P \oplus Q}{P \wedge \neg Q \mid Q \wedge \neg P}$ alebo $P \vee Q$ $\neg P \vee \neg Q$	$\dfrac{\neg(P \leftrightarrow Q)}{P \wedge \neg Q \mid \neg P \wedge Q}$ alebo $P \vee Q$ $\neg P \vee \neg Q$
$\dfrac{\neg\neg P}{P}$	$\dfrac{\neg(P \downarrow Q)}{P \mid Q}$	$\dfrac{\neg(P \oplus Q)}{P \wedge Q \mid \neg P \wedge \neg Q}$ alebo $\neg P \vee Q$ $\neg Q \vee P$
$\dfrac{P \uparrow Q}{\neg P \mid \neg Q}$	$\dfrac{P \downarrow Q}{\begin{array}{c}\neg P\\\neg Q\end{array}}$	$\dfrac{\neg(P \uparrow Q)}{\begin{array}{c}P\\Q\end{array}}$

Tab. 2.1: Pravidlá dekompozície pre Tablo algoritmus

nejakú vetvu [..., $\neg((\mathbf{A} \wedge \mathbf{B}) \to \neg\mathbf{C})$] predĺži vložením podvýrazov na tvar [..., $\neg((A \wedge B) \to \neg C)$, $\neg\neg(\mathbf{A} \wedge \mathbf{B})$, $\neg\neg\mathbf{C}$)].

Niektoré pravidlá môžu vystupovať ako vetviace aj ako nevetviace vďaka distributívnosti disjunkcie nad konjunkciou a konjunkcie nad disjunkciou (napríklad $P \oplus Q$, ktoré možno vyjadriť ako $(P \wedge \neg Q) \vee (\neg P \wedge Q)$ alebo ako $(P \vee Q) \wedge (\neg P \vee \neg Q)$). V týchto prípadoch je možné zvoliť ľubovoľnú z foriem.

Každá vetva stromu (a v prípade vetvenia aj jej podvetvy) je predlžovaná dovtedy, kým nastane niektorá zo situácií

- na koniec vetvy bol pridaný výraz, ktorý je komplementom pre nejaký výraz už zaradený v danej vetve (ak F reprezentuje nejaký výraz, tak komplementom k F je $\neg F$),
- vetva obsahuje rozklad každého výrazu, ktorý je v nej zaradený.

Ak nastane jedna z nich, vetva sa považuje za ukončenú. V prvom prípade vetva obsahuje spor (požaduje sa súčasné splnenie každého výrazu v rámci vetvy – teda aj oboch kontradikčných komplementov). Takáto vetva sa

označí ako *uzavretá*. V druhom prípade už jednoducho nie je čo do vetvy pridať. Takáto vetva sa označí ako *otvorená*. Ak už nie je možné predlžovať ziadnu vetvu, strom sa označuje ako *úplný*. Ak v úplnom strome všetky vetvy sú uzavreté, tak aj tablo je uzavreté. Ak existuje aspoň jedna otvorená vetva, tablo sa označí ako otvorené.

Po skončení vytvárania úplného stromu každá otvorená vetva reprezentuje model logickej vety, ktorá je v koreni stromu. Navyše k potvrdeniu existencie modelu je z danej vetvy možné zistiť aj to, ktorú interpretáciu daný model reprezentuje – stačí z danej vetvy vybrať všetky literály. Ak je tablo uzavreté a teda v strome neexistuje žiadna otvorená vetva ale naopak všetky vetvy sú uzavreté, tak logická veta v koreni stromu nie je splniteľná.

TABLO KALKUL A DNF

Tablo kalkul je vlastne grafická metóda systematickej transformácie výrokovej vety do tvaru DNF – disjunkcie konjunktívnych klauzúl. Využíva to, že

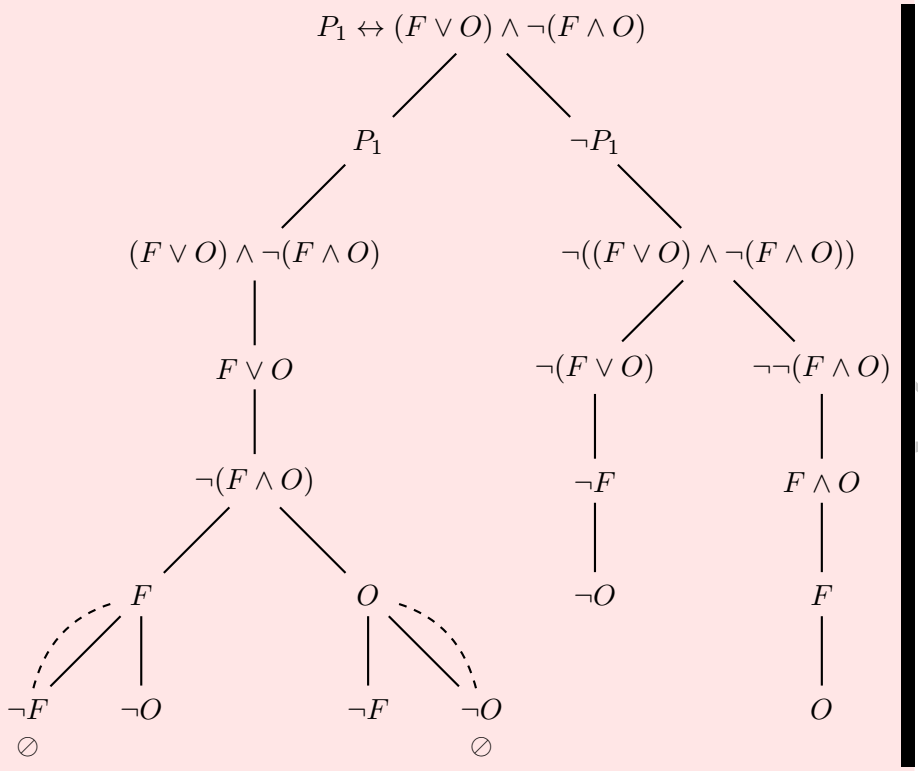
- konjunktívna klauzula je interpretovaná ako nepravdivá, ak obsahuje nejaký literál a súčasne aj jeho negáciu,
- DNF je interpretovaná ako nepravdivá, ak každá jej klauzula je interpretovaná ako nepravdivá,
- DNF je interpretovaná ako pravdivá, ak aspoň jedna jej klauzula nie je interpretovaná ako nepravdivá.

Každá vetva stromu (generovaného tabla) reprezentuje vytváranie jednej konjunktívnej klauzuly – samotná klauzula je daná konjunkciou všetkých tých literálov, ktoré sa v príslušnej vetve nachádzajú. Výhoda metódy je v tom, že

- akonáhle sa zistí kontradikcia v niektorej vetve, už sa nepokračuje v rozvíjaní danej vetvy,
- klauzuly sa nerozvíjajú každá úplne osobitne ale je snaha ich čo najdlhšie rozvíjať spoločne (teda aby vetvy mali čo najdlhšie spoločné časti).

Metóda tabiel je pre výrokovú logiku vhodnou rozhodovacou procedúrou, umožňujúcou rozhodnúť o (ne)splniteľnosti a (ne)validnosti logických viet.

Príklad 2.2 Pre ilustráciu činnosti algoritmu opäť použime Stanov výrok, ktorý má tvar $P_1 \leftrightarrow F \oplus O$. Tento výrok sa umiestni do koreňa stromu, pričom operácia $\oplus$ bola nahradená pomocou iných operandov, pretože dekompozičné pravidlá v Tab. 2.1 túto operáciu nezohľadňujú.



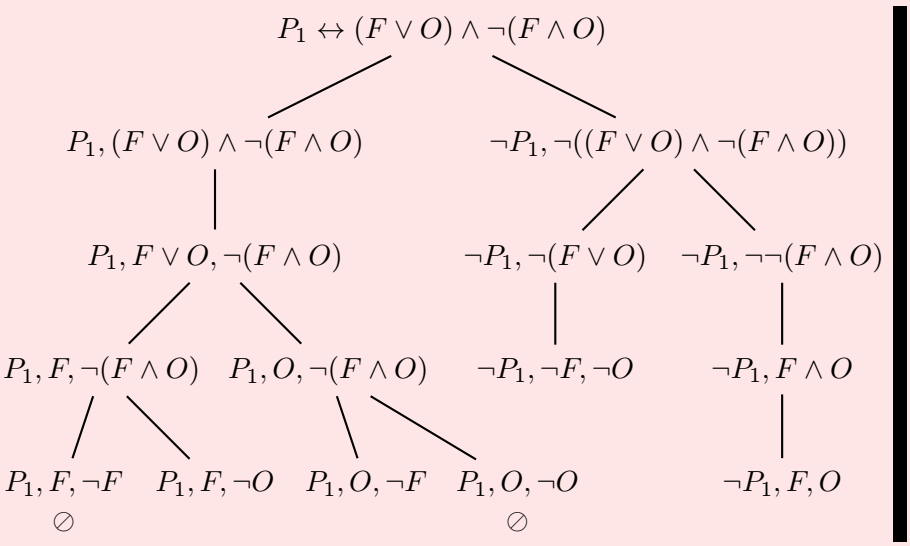
Obr. 2.2: Splniteľnosť pomocou analytického tabla

Vyvorený výsledný strom je na obr. 2.2. Celkovo bolo použitých šesť rôznych dekompozičných pravidiel, z ktorých tri spôsobili vetvenie. Strom obsahuje šesť vetiev, z ktorých dve sú označené ako uzavreté (označené znakom $\oslash$ reprezentujúcim výskyt sporu) – v jednom prípade kvôli komplementárnym literálom symbolu F a v druhom prípade kvôli komplementárnym literálom symbolu O (pozitívne verzie týchto literálov boli pridané dekompozíciou výrazu $F \vee O$ a ich negatívne literály zase dekompozíciou $\neg(F \wedge O)$).

Vetva stromu (druhá zľava) $[P_1 \leftrightarrow (F \vee O) \wedge \neg(F \wedge O)$, P_1, $(F \vee O) \wedge \neg(F \wedge O)$, $F \vee O$, $\neg(F \wedge O)$, F, $\neg O]$ je otvorená a reprezentuje splnenie výroku pri interpretácii $I = \{P_1^I = TRUE, F^I = TRUE, O^I = FALSE\}$. Stanov výrok je celkovo splniteľný pri štyroch interpretáciách zo všetkých $2^3 = 8$ možných.

Uvedené tablo sa niekedy označuje aj ako *analytické* tablo. Inou mož-

nosťou je *sémantické tablo*, ktorého ilustrácia pre ten istý prípad (Samov výrok) ako v predošlom je na obr. 2.3. •



Obr. 2.3: Splniteľnosť pomocou sémantického tabla

Rozdiel medzi analytickou a sémantickou formou je:

- pri sémantickom variante dochádza ku kopírovaniu výrazov medzi uzlami, analytický variant v popise uzlov nekopíruje predchádzajúce výrazy,
- analytický variant vytvára dlhšie vetvy, pretože nevetviace pravidlo predlžuje vetvu o viac uzlov zatiaľ čo u sémantického variantu iba o jeden uzol,
- pri testovaní uzavretosti vetvy je pri analytickom variante potrebné sledovať celú vetvu zatiaľ čo pri sémantickom stačí posledný uzol,
- pri hľadaní výrazu, ktorý je potrebné dekomponovať, stačí u sémantického variantu skúmať posledný uzol vo vetve zatiaľ čo pri analytickom variante je potrebné skúmať viacero uzlov vetvy.

VÝBER PRAVIDIEL

Vo všeobecnosti ak v nejakej vetve je možné aplikovať vetviace aj nevetviace pravidlo, tak často sa realizuje najprv nevetviace a až potom vetviace – v strome sa tak generuje menej uzlov. Napríklad pri vetve $[\ldots, A \vee B,$

$C \wedge D$] by sa získali dve vetvy [..., $A \vee B$, $C \wedge D$, C, D, A] a [..., $A \vee B$, $C \wedge D$, C, D, B] s pridaním štyroch uzlov do stromu na rozdiel od opačného prípadu keď by sa získali dve vetvy [..., $A \vee B$, $C \wedge D$, A, C, D] a [..., $A \vee B$, $C \wedge D$, B, C, D] s pridaním šiestich uzlov do stromu (teraz je spoločná časť oboch vetiev kratšia než v predošlom poradí dekompozícií).

TABLO PRE CNF

Príklad ukázal, že metódu je možné použiť pre logickú vetu vo všeobecnom tvare. Ak by sa použilo CNF vyjadrenie vety, tak do stromu sa budú pridávať iba literály – jednotlivé vetvy stromu budú reprezentovať priamo kombinácie hodnôt symbolov, teda ich interpretácie. Problémom však je, že nejaká kombinácia hodnôt symbolov môže byť reprezentovaná viacerými vetvami stromu, ak literály budú v nich uvedené v rôznom poradí, čo zbytočne zvyšuje počet ciest. Ak by skúmanou vetou bolo napríklad

$$(A \vee B) \wedge (A \vee \neg B) \wedge (\neg A \vee B)$$

tak algoritmus by vygeneroval strom so siedmimi vetvami, z ktorých dve by boli otvorené ([.., A, B] [..., B, A]) a päť uzavretých ([..., A, $\neg A$], [..., A, $\neg B$, $\neg A$], [..., A, $\neg B$, B], [..., B, A, $\neg A$], [..., B, $\neg B$]). A to napriek tomu, že pre dva symboly je potrebné preskúmať iba štyri rôzne interpretácie. Je zrejmé, že zložitosť Tablo algoritmu závisí od syntaktickej štruktúry skúmanej logickej vety.

PREHĽADÁVACIE METÓDY

Iným z existujúcich priamych prístupov ako riešiť úlohu splniteľnosti je prehľadávanie priestoru všetkých možných interpretácií s cieľom nájsť interpretáciu, ktorá je zároveň aj modelom. Tieto prehľadávacie metódy škálujú lepšie ako predchádzajúce metódy, preto sú základom pre moderné nástroje pre odvodzovanie, ktoré sú vhodné aj v prípade zložitejších logických viet. Zvyčajne stavajú na použití CNF reprezentácie, pretože ju chápu ako konjunkciu ohraničení, ktoré je potrebné splniť voľbou vhodnej interpretácie hodnôt použitých symbolov – a na základe toho môžu využiť postupy z riešenia úloh s ohraničeniami pomocou prehľadávania.

Toto prehľadávanie môže byť *lokálne* alebo *systematické*. Lokálny prístup (ktorý je zo svojej podstaty neúplný a teda nezaručuje nájdenie interpretácie, pri ktorej je skúmaná veta pravdivá) reprezentuje použitie rôznych prehľadávacích algoritmov (napr. stochastické lokálne prehľadávanie, prírodne inšpirované metaheuristiky, atď.). Systematický prístup (ktorý na rozdiel od lokálneho je úplným a teda garantuje nájdenie vhodnej interpretácie ak taká existuje) je založený na postupnom prehľadávaní s navracaním na základe stratégie *prehľadávania do hĺbky*. Zosobnením tohto prístupu je *DPLL algoritmus*.

DPLL ALGORITMUS

DPLL algoritmus (spojený s menami Davis, Putnam, Logemann a Loveland – je založený na základoch daných algoritmom DP navrhnutým Da-

vstup: veta F v CNF tvare
výstup: interpretácia I v prípade úspechu alebo {} pri neúspechu

```
1.  s := 0, I := {}
2.  loop
3.      switch F^I
4.          case TRUE
5.              return I
6.          case FALSE
7.              while s > 0  and flip[s] := 1
8.                  s := s − 1
9.              I := I \ {X_t^I : t > s}
10.             if s = 0 then return {}
11.                     else flip[s] := 1
12.                          X_s^I := ¬X_s^I
13.         default
14.             s := s + 1
15.             X := select_symbol(F,I)
16.             flip[s] := 0
17.             I := I ∪ X_s^I
```

Alg. 2.3: DPLL skeleton

visom a Putnamom v 1960 a algoritmom DLL navrhnutým Davisom, Logemannom a Lovelandom v 1962), sa nachádza v jadre mnohých dnešných *SAT solverov*. Základná schéma algoritmu je uvedená v Alg. 2.3. Jedná sa o postupné rekurzívne budovanie parciálnej (neúplnej) interpretácie. Začína sa z prázdnej interpretácie $I = \{\}$, do ktorej sa postupne v nasledujúcich krokoch pridávajú interpretácie symbolov X_s^I, kde X reprezentuje interpretovaný symbol, X^I jeho interpretáciu a s reprezentuje číslo kroku, v ktorom bola daná interpretácia pridaná. Je možné pridať interpretáciu symbolu ako pravdivého alebo nepravdivého – záznam o tom, aká interpretácia bola pridaná, sa udržiava v poli $flip$. Hodnota $flip[s] = 0$ znamená, že v kroku s bola pridaná prvá interpretácia zvoleného symbolu, zatiaľ čo hodnota

$flip[s] = 1$ signalizuje, že pre daný symbol bola jeho interpretácia zmenená na druhú interpretáciu (a teda už boli vyčerpané všetky možnosti pre daný symbol). Takýmto postupom sa parciálna interpretácia rozširuje až nastane jedna zo situácií:

- na základe interpretácie symbolov je možné interpretovať celú vetu F ako pravdivú ($F^I = TRUE$),
- vetu F na základe interpretácie symbolov je možné interpretovať ako nepravdivú ($F^I = FALSE$),
- na základe vytvorenej parciálnej interpretácie symbolov vetu F ešte nie je možné interpretovať ako celok.

Týmto trom situáciám zodpovedajú tri vetvy algoritmu. V prípade prvej situácie (riadok 4) vytvorená interpretácia (či už parciálna alebo úplná) predstavuje hľadané riešenie a preto prehľadávanie končí. Druhá situácia (riadok 6) znamená, že aktuálnu parciálnu interpretáciu nie je možné rozšíriť tak, aby reprezentovala hľadané riešenie. Je potrebné sa vrátiť spätne po vykonaných krokoch (s odstránením posledne pridaných interpretácií symbolov) až ku kroku so symbolom, pre ktorý neboli vyskúšané obe možné pravdivostné hodnoty. Ak taká možnosť nie je (pokus o *navracanie* skončí prípadom $s = 0$), signalizuje sa nemožnosť nájdenia riešenia. Ak sa však podarí nájsť nejaký predošlý krok so symbolom, pre ktorý bola vyskúšaná iba jedna z možných interpretácií, tak pre daný symbol sa zmení jeho interpretácia a hľadanie opäť pokračuje od tohto symbolu avšak už s jeho zmenenou interpretáciou. V tretej situácii (riadok 13) dochádza k ďalšiemu rozšíreniu parciálnej interpretácie symbolov. Zvolí sa ďalší symbol (či už ľubovoľne alebo ďalší vo vopred dohodnutom poradí) a jemu priradená pravdivostná hodnota (prvá vo vopred dohodnutom poradí). Voľba sa zaregistruje v poli $flip$ a zaradí do interpretácie.

Skeleton algoritmu v sebe obsahuje dve slučky:

- hlavná slučka (riadky 2 až 17),
- navracacia slučka (riadky 7 až 8),

pričom v rámci hlavnej slučky sa parciálna interpretácia symbolov rozširuje o nové symboly, zatiaľ čo v navracacej slučke sa parciálna interpretácia symbolov skracuje vypúšťaním posledne pridaných symbolov.

Moderné SAT solvery založené na systematickom prehľadávaní tento základný algoritmus dopĺňajú o ďalšie prvky, umožňujúce zrýchlenie nájdenia riešenia. Ako možné doplnenia sa najčastejšie používajú:

Rozšírenia DPLL

- jednotková propagácia a propagácia čistého literálu,
- učenie klauzúl riadené konfliktami,
- nechronologické navracanie,
- heuristiky pre výber symbolu a hodnoty.

JEDNOTKOVÁ PROPAGÁCIA

Jednotková propagácia je založená na využití konceptu jednotkovej rezolvencie pri výskyte jednotkovej klauzuly – klauzuly, ktorá obsahuje iba jeden literál. Aby takáto klauzula bola pravdivá, musí byť pravdivý jej literál a teda je zrejmé, aká pravdivostná hodnota musí byť zvolená pre interpretáciu daného symbolu (v závislosti od toho, či symbol je v literále negovaný alebo nie). Keďže priradenie pravdivostnej hodnoty symbolu môže mať za následok, že nejaká iná klauzula sa stane jednotkovou, tak takáto propagácia môže mať rekurzívny charakter. Príkladom sú klauzuly

$$(\neg P \vee Q) \quad \wedge \quad (\neg P \vee Q \vee \neg R) \quad \wedge \quad P$$

kde tretia klauzula je jednotková a preto $P^I = TRUE$. Následkom toho prvý literál v prvej klauzule je nepravdivý (a teda môže byť z klauzuly vypustený) a prvá klauzula sa tak stáva tiež jednotkovou s následkom $Q^I = TRUE$. Výsledkom je, že všetky tri klauzuly sú interpretované ako pravdivé (bez špecifikovania interpretácie symbolu R, pretože v splnených klauzulách nie je potrebné skúmať zostávajúce literály symbolov bez interpretácie).

Dochádza teda k identifikácii jednotkovej klauzuly s určením vhodnej interpretácie príslušného symbolu, tvoriaceho jednotkový literál. Následne dochádza k vypusteniu tých klauzúl, ktoré obsahujú pravdivý literál daného symbolu a k skráteniu tých klauzúl, ktoré obsahujú nepravdivý literál daného symbolu, vypustením tohto nepravdivého literálu.

PROPAGÁCIA ČISTÉHO LITERÁLU

Podobný charakter má *propagácia čistého literálu*. O čistom literále sa hovorí vtedy, ak nejaký symbol sa v klauzulách vyskytuje iba v jednej podobe, buď iba priamo ako jednoduchý symbol alebo iba negovane ako negovaný symbol. Vtedy je možné daný symbol interpretovať tak, aby vyskytujúci sa literál zapríčinil pravdivosť tých klauzúl, v ktorých sa nachádza. Takáto interpretácia nemá za následok nepravdivosť žiadneho literálu, pretože druhá podoba literálu daného symbolu sa v klauzulách nenachádza. Príkladom sú klauzuly

$$(\neg P \vee Q) \quad \wedge \quad (\neg P \vee S) \quad \wedge \quad (\neg Q \vee R) \quad \wedge \quad (\neg S \vee \neg R)$$

kde symbol P sa vyskytuje iba v podobe negovaného literálu, zatiaľ čo ostatné symboly vytvárajú literály oboch podôb. Interpretácia $P^I = FALSE$

spôsobí, že prvé dve klauzuly sú interpretované ako pravdivé (a teda literály Q a S nemusia byť skúmané). Následkom toho budú $\neg Q$ a $\neg S$ považované za ďalšie čisté literály a teda bude môcť byť použitá interpretácia $Q^I = FALSE$ a $S^I = FALSE$ so splnením zostávajúcich dvoch klauzúl. Tento typ propagácie môže mať teda tiež rekurzívny charakter.

KOMBINOVANIE PROPAGÁCIÍ

Jednotková propagácia môže vytvoriť podmienky pre propagáciu čistého literálu (naopak to nie je možné – spĺňaním klauzúl propagáciou čistého literálu nie je možné vytvoriť jednotkovú klauzulu). Toto je možné ilustrovať pomocou

$$(\neg P \vee \neg Q \vee \neg S) \quad \wedge \quad (P \vee Q) \quad \wedge \quad P \quad \wedge \quad (S \vee \neg R) \quad \wedge \quad (\neg Q \vee R)$$

kde na základe jednotkovej propagácie bude $P^I =$ TRUE, druhá klauzula bude interpretovaná ako pravdivá a prvá klauzula sa zmení na $(\neg Q \vee \neg S)$. Vďaka neuvažovaniu druhého literálu v druhej klauzule sa symbol Q vyskytuje iba v podobe negovaného literálu a teda je možné na základe propagácie čistého literálu použiť $Q^I = FALSE$.

DOPLNENIE DPLL O PROPAGÁCIE

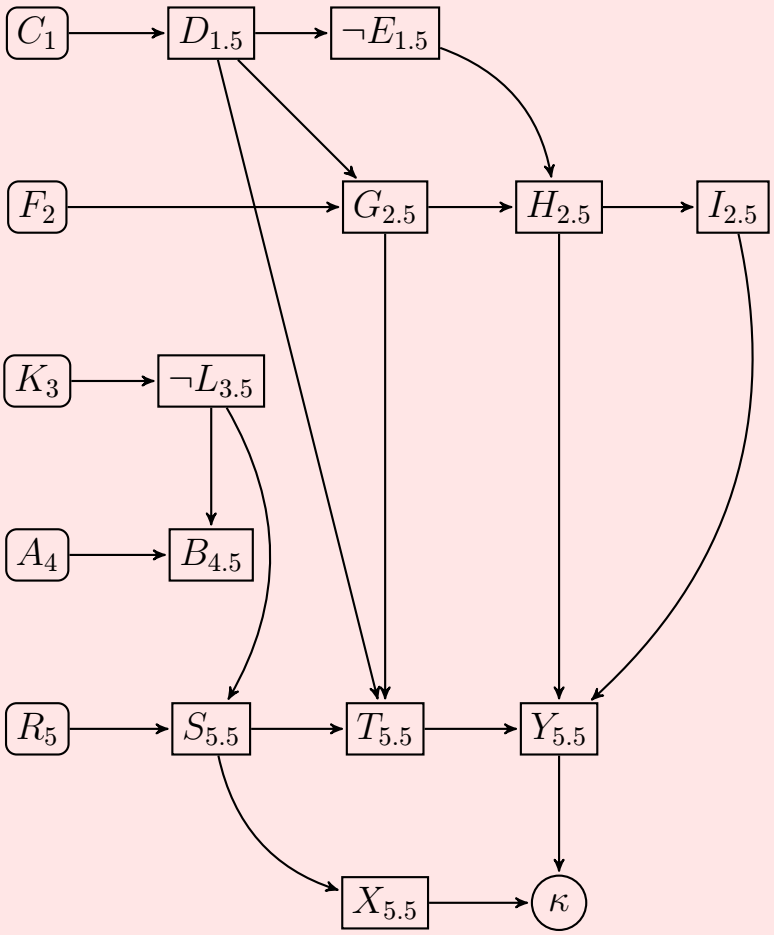
Oba typy propagácie reprezentujú orezávanie alternatív, ktoré je potrebné prehľadať, pretože pre symboly interpretované v rámci týchto propagácií nemá zmysel pri neúspechu skúmať ich alternatívne interpretácie. Je ich možné zaradiť do algoritmu Alg. 2.3 medzi riadky 2 a 3 v tvare

```
2.4.   while X^I_{s+0.5} := unit_propagation(F)
2.5.       I := I ∪ X^I_{s+0.5}
2.6.   while X^I_{s+0.5} := pure_literal_propagation(F)
2.7.       I := I ∪ X^I_{s+0.5}
```

pričom aby sa využil vzťah medzi obomi typmi propagácie, tak je potrebné najprv zaradiť jednotkovú propagáciu a až po nej propagáciu čistého literálu. Aby bolo možné odlíšiť, interpretácia ktorého symbolu bola pridaná do I na základe niektorej z propagácií a ktorá na základe voľby v riadkoch 15 až 17, interpretácie symbolov z propagácií sú pridávané s neceločíselným krokom (aby navracanie v riadkoch 7 a 8, pracujúce iba s celočíselnými hodnotami kroku, sa navracalo iba k tým symbolom, ktorých interpretácie pochádzajú z voľby na riadku 17).

IMPLIKAČNÝ GRAF

Propagácia je prirodzeným spôsobom asociovaná s *implikačným grafom*, ktorý zachytáva všetky spôsoby, ktorými boli určené interpretácie symbolov. Ukážka takéhoto grafu je na Obr. 2.4. Jedná sa o orientovaný acyklický graf, ktorého uzlami sú interpretácie symbolov, pričom každému uzlu zodpovedá práve jeden symbol (interpretácii $X^I_s = TRUE$ vytvorenej na úrovni



Obr. 2.4: Implikačný graf

s zodpovedá uzol s popisom X_s, zatiaľ čo interpretácii $X_s^I = FALSE$ zase uzol s popisom $\neg X_s$). Hrany vyjadrujú závislosti medzi interpretáciami, napr. podľa obrázku symbol Y bol interpretovaný na základe interpretácií symbolov H, I a T (očividne vďaka klauzule $\neg T \vee \neg H \vee \neg I \vee Y$ alebo dvojici klauzúl $\neg T \vee \neg H \vee Y$ a $\neg T \vee \neg I \vee Y$). Voliteľne môžu byť tieto hrany označené množinami tých klauzúl, ktoré danú závislosť reprezentujú.

Na obrázku sú interpretácie na základe voľby reprezentované obdĺžnikmi s oblými hranami a celočíselnými krokmi, zatiaľ čo interpretácie na základe propagácie sú reprezentované obdĺžnikmi s ostrými hranami a neceločíselnými krokmi. Obrázok teda reprezentuje postup, keď najprv bola

Rozšírenie: Dimacs CNF formát

Dimacs CNF formát

Tento formát je zameraný na zápis logických viet v tvare CNF klauzúl. Je široko akceptovaným formátom pre SAT solvery, ktoré ho používajú pre formátovanie svojho vstupného súboru. Jedná sa o jednoduchý formát, ktorého gramatika v BNF je:

< riadky >	::=	< komentár >* < parametre > < klauzula >+
< komentár >	::=	"c" < text >
< parametre >	::=	"p cnf " < počet symbolov > < počet klauzúl >
< klauzula >	::=	< literál >+ "0"
< literál >	::=	< symbol > | -< symbol >
< symbol >	::=	1 | 2 | ...

Formát je teda zložený zo za sebou idúcich komentárových riadkov, jedného parametrického riadku a klauzulárnych riadkov (niekedy je komentárový riadok povolený na ľubovoľnom mieste):

- komentárový riadok začína písmenom *c*, počet takýchto riadkov je ľubovoľný (vrátane žiadneho),
- parametrický riadok definuje počet použitých symbolov a počet použitých klauzúl, začína písmenom *p* nasledovaným identifikátorom *cnf*,
- klauzulárny riadok reprezentuje klauzulu (disjunkciu literálov) a jeho koniec je indikovaný znakom *0*. Počet literálov je ľubovoľný avšak vždy aspoň jeden.

Jednotlivé symboly sú reprezentované numericky pomocou prirodzených čísel z intervalu *1* až *počet symbolov*. Priamy (pozitívny) literál je vyjadrený ako daný symbol, zatiaľ čo negovaný (negatívny) literál je vyjadrený ako záporná hodnota príslušného symbolu.

zvolená interpretácia symbolu C, následne pomocou propagácie boli určené interpretácie symbolov D a E, nasledovala voľba pre symbol F, za ňou na základe propagácií interpretácie symbolov G, H a I, voľba pre K, atď.

Konfliktný graf

Ak by nejaký symbol mal v tomto grafe nadobudnúť rôzne interpretácie, znamená to konflikt a namiesto týchto interpretácií je v grafe vložený konfliktný uzol κ (konflikt na obrázku vznikol kvôli klauzulám $Z \vee \neg X$ a $\neg Z \vee \neg Y$). Implikačný graf môže obsahovať žiadny, jeden alebo aj viac

konfliktných uzlov. V prípade konfliktu je z takéhoto grafu možné odvodiť *konfliktný graf* ako podgraf implikačného grafu, ktorý obsahuje iba jeden konfliktný uzol a súčasne obsahuje iba tie uzly, z ktorých je daný konfliktný uzol dosiahnuteľný. Ak implikačný graf obsahuje viacero konfliktných uzlov, tak potom je možné z neho odvodiť viac navzájom rôznych konfliktných grafov. Na obrázku obr. 2.4 nie je implikačný graf totožný s konfliktným grafom (uzly A_4 a $B_{4.5}$ sú súčasťou implikačného grafu ale nie konfliktného).

ANALÝZA KONFLIKTU

Konfliktný graf sa používa pre analýzu konfliktu a vyhľadávanie konfliktných klauzúl. Keďže podľa obrázku bezprostrednou príčinou konfliktu bola interpretácia symbolov Y a X (oba boli interpretované ako pravdivé), tak pre zabránenie konfliktu je možné odvodiť platnosť klauzuly

$$\neg(X \wedge Y) \equiv \neg X \vee \neg Y$$

Z grafu je zrejmé, že interpretácia symbolu Y sa dá získať pomocou implikácie

$$T \wedge H \wedge I \rightarrow Y \equiv \neg T \vee \neg H \vee \neg I \vee Y$$

Z posledne uvedených dvoch klauzúl je možné odvodiť pomocou rezolvenčného odvodzovacieho pravidla (zohľadnením výskytu oboch literálov symbolu Y) ďalšiu klauzulu, ktorá musí platiť aby nedošlo ku konfliktu

$$\frac{\neg X \vee \neg Y \qquad \neg T \vee \neg H \vee \neg I \vee Y}{\neg X \vee \neg T \vee \neg H \vee \neg I}$$

REZY KONFLIKTNÝM GRAFOM

Obe odvodené klauzuly reprezentujú *rezy* konfliktným grafom, keď ho delia na dva podgrafy, pričom v ľavom podgrafe ostávajú interpretácie označené celočíselnou úrovňou (pochádzajúce z rozhodnutia algoritmu Alg. 2.3 na riadkoch 15 až 17) a v pravom podgrafe ostáva konfliktný uzol. Interpretácie z propagácií sa v rôznom pomere rozdeľujú medzi oba podgrafy. Takýmto spôsobom je možné postupným posúvaním rezu smerom nahor a doľava získať ďalšie klauzuly, niektoré z nich sú $\neg S \vee \neg Y$, $\neg X \vee \neg T \vee \neg H$ či $\neg X \vee \neg S \vee \neg G \vee \neg D \vee \neg H \vee \neg I$. V množine možných rezov jedným extrémom je už spomínaný $\neg X \vee \neg Y$, zatiaľ čo opačným extrémom je $\neg R \vee \neg K \vee \neg F \vee \neg C$ obsahujúci iba tie symboly, ktorých interpretácia bola získaná voľbou a nie propagáciou.

VÝBER KONFLIKTNEJ KLAUZULY

Principiálne každá z týchto rezových konfliktných klauzúl môže zabrániť analyzovanému konfliktu. Aby sa mu v budúcnosti predišlo, stačí vybranú klauzulu zaradiť k pôvodným klauzulám riešeného problému. Otázkou však zostáva, ktoré z klauzúl vybrať, pretože takýchto klauzúl je možné generovať veľké množstvo. Tomu zodpovedá aj existencia viacerých schém výberu. Jedna z najznámejších je založená na tzv. uzloch *UIP* (Unique Implication

Point). UIP v implikačnom grafe je taký uzol na aktuálnej úrovni s (na ktorej vznikol konflikt), že každá cesta od uzla s celočíselným označením tejto úrovne ku konfliktnému uzlu vedie cez tento uzol. Ak takýchto uzlov existuje viac, preferuje sa najľavejší (označovaný ako prvý UIP).

V našom prípade na Obr. 2.4 konflikt vznikol na piatej úrovni. Je zrejmé, že existuje iba jeden UIP – $S_{5.5}$. Cez tento uzol prechádza viacero rezov. Jeden z týchto rezov bude pridaný k pôvodným klauzulám ako nová naučená klauzula. Buď to bude prvý nájdený rez prechádzajúci cez UIP, alebo nejaký iný tiež prechádzajúci cez UIP, ktorý je od neho kratší. Je samozrejme možné objaviť všetky takéto rezy a vybrať z nich najkratší (ale počet všetkých rezov môže byť značne veľký), alebo nájsť prvý a tento podrobiť dodatočnej procedúre, v rámci ktorej sa daná klauzula bude minimalizovať (bude sa znižovať počet literálov v nej obsiahnutých).

MINIMALIZÁCIA KONFLIKTNEJ KLAUZULY

V našom prípade, ak sa použije nájdená klauzula $\neg X \vee \neg T \vee \neg H \vee \neg I$, tak je možné dospieť napríklad k rezu $\neg X \vee \neg S \vee \neg D \vee \neg G \vee \neg H \vee \neg I$, ktorý je následne možné postupne zjednodušiť pomocou série rezolvencií

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\neg X \vee \neg S \vee \neg D \vee \neg G \vee \neg H \vee \neg I \quad S \rightarrow X}{\neg S \vee \neg D \vee \neg G \vee \neg H \vee \neg I} \quad H \rightarrow I}{\neg S \vee \neg D \vee \neg G \vee \neg H} \quad G \wedge \neg E \rightarrow H}{\neg S \vee \neg D \vee \neg G \vee E} \quad D \rightarrow \neg E}{\neg S \vee \neg D \vee \neg G}$$

DOPLNENIE DPLL O NOVÚ KLAUZULU

Takúto analýzu konfliktov spolu s učením novej klauzuly a jej zaradením k pôvodným klauzulám je možné zaradiť do algoritmu Alg. 2.3 medzi riadky 6 a 7

6.2. $< C, G > := \mathit{conflict\_analysis}(F, I)$
6.3. $C' := \mathit{clause\_minimization}(C, G)$
6.4. $F := F \wedge C'$

keď sa najprv vygeneruje konfliktný graf G a na jeho základe sa naučí klauzula C, ktorej splnenie zabráni danému konfliktu, táto klauzula sa následne zjednoduší a zaradí k ostatným klauzulám.

NÁVRAT RIADENÝ KONFLIKTOM

Učenie konfliktných klauzúl môže byť využité aj pre riadenie navracania. Naivná verzia navracania podľa riadkov 7 a 8 algoritmu Alg. 2.3 realizuje návrat vždy iba o jednu úroveň a iba ak to nie je možné, tak sa pokračuje na úroveň ešte pred tým (schéma je označovaná ako *chronologické* navracanie). Túto schému je možné nahradiť niektorou zo schém, reagujúcich na výsledky analýzy konfliktov. Príkladom je schéma založená na intuitívnom chápaní, že ak pre nejaký symbol na úrovni s nastal konflikt pre obe jeho

interpretácie, pričom pre jednu interpretáciu najbližšou príčinou bola voľba na úrovni a a pre druhú interpretáciu zase voľba na úrovni b, nie je nutné sa navrátiť k úrovni $s-1$ ale je možné spätne *skočiť* priamo na úroveň $max(a,b)$.

Uvažujme opäť prípad implikačného grafu podľa Obr. 2.4, keď interpretácia $R^I = TRUE$ mala za následok konflikt. Z grafu možno naučiť klauzulu $\neg R \vee \neg K \vee \neg F \vee \neg C$ odvodenú iba z tých interpretácií, ktoré boli určené voľbou ale nie propagovaním. Pretože došlo ku konfliktu, interpretácia symbolu R bola zmenená na $R_5^I = FALSE$. Predpokladajme, že následkom toho opäť nastal konflikt a bola odvodená klauzula $R \vee \neg F \vee \neg C$.

Ak by sa algoritmus teraz navrátil k symbolu A, tak po zmene jeho interpretácie $A_4^I = TRUE$ na $A_4^I = FALSE$ by opäť došlo k skúmaniu vhodnosti interpretácií symbolu R – s už známym výsledkom, pretože odvodené klauzuly signalizujú, že ani jeden z neúspechov nezávisel na interpretácii symbolu A. Preto je možné bezpečne skočiť na voľbu interpretácie toho symbolu, ktorý figuruje v odvodenej klauzule aspoň pre jeden neúspech a je zároveň najbližšie k aktuálnej úrovni (pre náš prípad teda skok spätne na úroveň 3 k interpretácii symbolu K).

DOPLNENIE DPLL O SPÄTNÝ SKOK

Takúto analýzu konfliktov pre určenie cieľa spätného skoku je možné zaradiť do algoritmu Alg. 2.3 medzi riadky 6 a 7

```
6.6.      j := jump_level(G)
6.7.      if flip[s] = 0 then jump := j
6.8.                     else s := max(jump,j)
7.        while s > 0  and flip[s] = 1
8.            s := s − 1
```

keď sa vždy po analýze konfliktného grafu G vygeneruje úroveň, na ktorej bola zvolená tá interpretácia spomedzi interpretácií relevantných voči konfliktu, ktorá je k aktuálnej úrovni najbližšie. Ak sa jedná ešte len o prvú interpretáciu symbolu na aktuálnej úrovni (teda $flip[s] = 0$), tak táto úroveň sa bude pamätať. Ak sa jedná už o konflikt pre druhú interpretáciu symbolu na aktuálnej úrovni (teda *flip[s] = 1*), tak sa použije bližšia z úrovní (odpamätanej a aktuálne určenej). Pri potrebe navracania sa najprv skočí na určenú úroveň a až z tejto úrovne v prípade potreby nasleduje navracanie po jednotlivých úrovniach.

VÝBEROVÉ HEURISTIKY

Pri rozširovaní parciálnej interpretácie symbolov (riadok 15 v Alg. 2.3) je možné naivný prístup výberu podľa vopred určeného statického poradia symbolov a priradzovaných pravdivostných hodnôt nahradiť dynamickým prístupom, reagujúcim na priebeh prehľadávania, ktorý je založený na

využití nejakých merateľných charakteristík. V tomto smere existuje veľa *heuristík*. Ako ukážku uveďme zopár používaných heuristík:

- DLIS (Dynamic Largest Individual Sum),
- DLCS (Dynamic Largest Combined Sum),
- MOM (Maximum Occurence on clauses of Minimum size),
- VSIDS (Variable State Independent Decaying Sum).

HEURISTIKA DLIS

Heuristika *DLIS* vyberá symbol a k nemu pravdivostnú hodnotu (teda literál) tak, aby voľba umožnila interpretovať čo najviac klauzúl spomedzi tých, ktoré ešte nemajú interpretáciu, ako pravdivých. Je to dynamická heuristika, pretože to, ktoré klauzuly už sú interpretované ako pravdivé (pretože ak je niektorá klauzula interpretovaná ako nepravdivá, tak nasleduje fáza navracania) a ktoré ešte interpretáciu nemajú, závisí od aktuálnej parciálnej interpretácie symbolov. Na jednej strane to je jednoduchá heuristika, na druhej strane však vyžaduje pri každej voľbe kontrolovať všetky klauzuly, čo je výpočtovo náročné. *DLCS* je podobná s tým rozdielom, že výber je rozdelený na dve etapy – najprv na výber symbolu a až potom na výber pravdivostnej hodnoty. Symbol vyberá tak, aby sa vyskytoval v čo najväčšom počte tých klauzúl, ktoré ešte nemajú interpretáciu, bez ohľadu na to či v nich vystupuje v priamom tvare alebo v negovanom tvare. Hodnotu vyberá následne podľa toho, v akom tvare sa vybraný symbol v tých klauzulách vyskytoval častejšie.

HEURISTIKA DLCS

HEURISTIKA MOM

Heuristika *MOM* k myšlienke interpretácie čo najväčšieho počtu doposiaľ neinterpretovaných klauzúl pridáva ďalšie dve idey: preferenciu krátkych klauzúl (pretože tam je väčšia šanca na vytvorenie jednotkových klauzúl) a preferenciu rovnomerného rozdelenia priameho a negovaného výskytu (analogicky k hľadaniu v rámci intervalu jeho delením na dve časti – delenie na polovicu je najrýchlejšie v najhoršom prípade). Metrika, ktorú je potrebné maximalizovať pri výbere symbolu je

$$2^k[\#(X) + \#(\neg X)] + \#(X) * \#(\neg X)$$

kde # reprezentuje početnosť (frekvenciu) výskytu v krátkych klauzulách a k je hodnota zvolená heuristicky. Vyberá sa teda ten symbol X, pre ktorý je táto hodnota najväčšia. Hodnota sa vyberá podľa toho, či je $\#(X) > \#(\neg X)$ (výsledkom je priamy výskyt v literále) alebo naopak (výsledkom je negovaný literál).

HEURISTIKA VSIDS

Pri *VSIDS* pre každý možný literál každého symbolu existuje počítadlo. Všetky počítadlá sú na začiatku inicializované na nulu. Vždy pri vložení nejakej klauzuly sa inkrementujú počítadlá tých literálov, ktoré sa nachádzajú

v tejto vkladanej klauzule. Vždy po určitej dobe sú hodnoty všetkých počítadiel znížené na polovicu. Pri voľbe sa vyberá ten literál (výber sa deje iba medzi literálmi tých symbolov, ktoré ešte neboli interpretované), ktorého počítadlo má aktuálne najvyššiu hodnotu.

Keďže počítadlá sa inkrementujú iba pri vkladaní nových klauzúl (na začiatku to sú klauzuly reprezentujúce problém, neskôr to sú odvodené konfliktné klauzuly), výpočtová náročnosť je menšia. Periodické zmenšovanie hodnôt počítadiel znamená, že sa viac preferujú výskyty v posledne vkladaných klauzulách oproti výskytom v klauzulách vložených skoršie – teda voľba viac reaguje na posledné dianie.

Cvičenia

1. Pomocou Tablo algoritmu dokážte
   - validnosť vety $((P \wedge Q) \rightarrow R) \rightarrow (P \rightarrow (Q \rightarrow R))$
   - validnosť vety $\neg((P \vee Q) \rightarrow R) \vee (R \rightarrow (P \wedge Q))$
   - splniteľnosť vety $((P \rightarrow Q) \rightarrow R) \rightarrow ((P \wedge Q) \rightarrow R)$
   - splniteľnosť vety $\neg(P \rightarrow (Q \rightarrow R)) \wedge ((P \rightarrow Q) \rightarrow R)$
   - vyplývanie vety $((P \vee Q) \rightarrow R) \models ((P \rightarrow Q) \rightarrow R)$
   - vyplývanie vety $(P \rightarrow (Q \rightarrow R)) \models ((P \wedge Q) \rightarrow R)$
2. Overte, či pre tie pravidlá tablo dekompozície z Tab. 2.1, ktoré môžu byť použité ako vetviace alebo ako nevetviace (pre $P \leftrightarrow Q$, $\neg(P \leftrightarrow Q)$, $P \oplus Q$ a $\neg(P \oplus Q)$) je výhodnejšia vetviaca alebo nevetviaca forma.
3. Pre výrok "Podvádzali Fero alebo Ondro, avšak nie obaja", ktorého pravdivosť je neznáma, nájdite všetky modely pomocou tablo algoritmu. Uvažujte
   - čo najkompaktnejšiu (najkratšiu) reprezentáciu,
   - reprezentáciu v tvare CNF.
4. Pomocou rezolvenčného odvodzovania dokážte nesplniteľnosť množín klauzúl
   - $P \rightarrow (Q \vee R) \wedge \neg(Q \wedge R)$, $P \rightarrow (S \vee T) \wedge \neg(S \wedge T)$, $S \rightarrow Q$, $\neg R \rightarrow T$, $T \rightarrow S$, P
   - $S \leftrightarrow \neg(A \wedge B) \wedge (A \vee B)$, $C \leftrightarrow A \wedge B$, A, B, $\neg S$, $\neg C$
   - $\neg P \vee Q$, $P \vee \neg Q$, $P \vee R \vee S$, $\neg P \vee \neg R \vee S$, $\neg P \vee R \vee \neg S$, $P \vee \neg R \vee \neg S$, $\neg S \vee T$, $S \vee \neg T$, $\neg Q \vee R \vee T$, $Q \vee \neg R \vee T$, $Q \vee R \vee \neg T$, $\neg Q \vee \neg R \vee \neg T$

5. Pre priateľov, ktorí radi komunikujú (cvičenia kapitoly 1), sa pokúste zo znalostí o nich v tvare CNF odvodiť, že

   a. Klára je online
   b. Klára nie je online
   c. Rudolf alebo Vladimír sú online, nie však obaja
   d. ak je online Rudolf, tak je online aj Vladimír
   e. ak je online Vladimír, tak je online aj Rudolf
   f. Anna a Hedviga sú alebo obe online alebo ani jedna
   g. ak je online nejaký chlapec, tak je online aj nejaké dievča
   h. najviac dve dievčatá sú online
   i. aspoň dve dievčatá sú online

   Pri odvodení použite stratégiu opornej množiny. Vyskúšajte aj iné stratégie.

6. Pre výraz v tvare CNF

   $$(P \vee Q \vee R) \wedge (P \vee \neg Q \vee \neg R) \wedge (P \vee \neg W) \wedge (\neg Q \vee \neg R \vee \neg W)$$
   $$\wedge(\neg P \vee \neg Q \vee R) \wedge (U \vee X) \wedge (U \vee \neg X) \wedge (Q \vee \neg U) \wedge (\neg R \vee \neg U)$$

   nájdite vhodnú interpretáciu symbolov manuálnou simuláciou

   - naivnej verzie algoritmu DPLL podľa Alg. 2.3,
   - ako v predchádzajúcom s pridaním jednotkovej propagácie a propagácie čistého literálu,
   - ako v predchádzajúcom s pridaním učenia nových klauzúl.

   Pri simulácii vytvárajte implikačný graf.

7. V "pigeon-hole" probléme ide o umiestnenie $n+1$ holubov do n dier, pričom každá diera môže obsahovať najviac jedného holuba. Pre $n = 3$ je problém popísaný pomocou 34 klauzúl (symbol ij vyjadruje umiestnenie i-teho holuba do j-tej diery)

   $\neg 11 \vee \neg 21$, $\neg 11 \vee \neg 31$, $\neg 11 \vee \neg 41$, $\neg 21 \vee \neg 31$, $\neg 21 \vee \neg 41$, $\neg 31 \vee \neg 41$,

   $\neg 12 \vee \neg 22$, $\neg 12 \vee \neg 32$, $\neg 12 \vee \neg 42$, $\neg 22 \vee \neg 32$, $\neg 22 \vee \neg 42$, $\neg 32 \vee \neg 42$,

   $\neg 13 \vee \neg 23$, $\neg 13 \vee \neg 33$, $\neg 13 \vee \neg 43$, $\neg 23 \vee \neg 33$, $\neg 23 \vee \neg 43$, $\neg 33 \vee \neg 43$,

   $\neg 11 \vee \neg 12$, $\neg 11 \vee \neg 13$, $\neg 12 \vee \neg 13$, $11 \vee 12 \vee 13$,

   $\neg 21 \vee \neg 22$, $\neg 21 \vee \neg 23$, $\neg 22 \vee \neg 23$, $21 \vee 22 \vee 23$,

   $\neg 31 \vee \neg 32$, $\neg 31 \vee \neg 33$, $\neg 32 \vee \neg 33$, $31 \vee 32 \vee 33$,

$\neg 41 \vee \neg 42$, $\neg 41 \vee \neg 43$, $\neg 42 \vee \neg 43$, $41 \vee 42 \vee 43$

Dokážte nesplniteľnosť tejto množiny klauzúl pomocou

- manuálnej simulácie DPLL algoritmu,
- zakódovaním problému do DIMACS formátu s následným použitím SAT solvera.

8. Reprezentácie dvoch problémov v tvare CNF z cvičení kapitoly 1 (kto podvádzal a priatelia ktorí radi navzájom komunikujú) zakódujte do DIMACS formátu a použite niektorý SAT solver pre zistenie všetkých riešení týchto problémov.

9. Vytvorte výrazy vo výrokovom počte, reprezentujúce nasledovné tvrdenia:

   a. Ak pôjdeš do Austrálie, budeš príliš ďaleko.
   b. Nepočujem ťa keď si príliš ďaleko.
   c. Keď ťa nepočujem, zabudnem ako vyzeráš.
   d. Ak prídem do Austrálie a nebudem vedieť ako vyzeráš, tak ťa tam nenájdem.
   e. Teda ak pôjdeš do Austrálie, tak ťa tam nestretnem.

   Overte, či záver (tvrdenie e) platí alebo nie.

10. Vyriešte prípad záhadnej vraždy. K tomu potrebujete zistiť, kde boli jednotlivé osoby a ktoré zbrane mali k dispozícii. Podozrivými sú traja muži (Fero, Jano, Ondro) a tri ženy (Anna, Hedviga, Klára). Každá osoba bola v inej miestnosti (kúpeľňa, jedáleň, kuchyňa, obývačka, komora, pracovňa). Podozrivá zbraň bola nájdená v každej miestnosti (vrece, puška, plyn, nôž, jed, povraz).

    1. Muž v kuchyni nemal povraz, nôž, pušku ani vrece.
    2. Anna bola v pracovni alebo v kúpeľni, Klára bola v druhej z týchto miestností.
    3. Osoba s vrecom, ktorá nie je Anna ani Fero, nebola v kúpeľni ani v jedálni.
    4. Žena s povrazom bola v pracovni.
    5. Zbraň v obývačke mal Jano alebo Fero.
    6. Nôž nebol v jedálni.
    7. Klára nemala zbraň z tých, čo boli nájdené v pracovni a v komore.

8. Puška bola v miestnosti s Ferom.

Bolo zistené, že zavraždený bol udusený plynom v komore. Podozrivý, ktorý bol nájdený v tejto miestnosti, je vrah. Na koho je treba ukázať prstom ?

Literatúra

1. Darwiche, A. – Pipatsrisawat, K.: Complete Algorithms. In: Handbook of satisfiability, Biere et al. (eds.). IOS Press, Amsterdam, 2009, 99–126.

2. Kostelník, P.: Praktický úvod do symbolickej umelej inteligencie. In: Umelá inteligencia a kognitívna veda I, Kvasnička, V. et al. (eds.), STU, Bratislava, 2009, 87–138.

3. Košík, M.: Prirodzená dedukcia. In: Umelá inteligencia a kognitívna veda II, Kvasnička, V. et al. (eds.), STU, Bratislava, 2010, 125–146.

4. Kroening, D. – Strichman, O.: Decision procedures (An algorithmic point of view). Springer, Berlin Heidelberg, 2008.

5. Kvasnička, V.: Logika a využitie sémantických tabiel pre inteligentné systémy. Univerzita sv. Cyrila a Metoda, Trnava, 2018.

6. Marek, V.: Introduction to Propositional Satisfiability. Studies in Logic, vol. 49. College Publications, Milton Keynes, 2014.

7. Marques-Silva, J. – Lynce, I. – Malik, S.: Conflict-Driven Clause Learning SAT Solvers. In: Handbook of satisfiability, Biere et al. (eds.). IOS Press, Amsterdam, 2009, 131–153.

8. Mordechai B.-A.: Mathematic Logic for Computer Science (3. vyd.). Springer, London, 2012.

9. Návrat, P. a kol.: Umelá inteligencia. Slovenská Technická Univerzita, Bratislava, 2002.

10. Russel, S. – Norvig, P.: Artificial Intelligence: A Modern Approach (3. vydanie). Pearson, Upper Saddle River, NJ, 2010.

11. van Harmelen, F. – Lifschitz, V. – Porter, B.: Handbook of Knowledge Representation. Elsevier, Amsterdam, 2008.