Jiří Boleslav
TOS Kuřim

ROZHODOVACÍ TABULKY JAKO PROSTŘEDEK USNADNĚNÍ VERIFIKACE PROGRAMŮ

1. Úvod

Současně s pozitivním jevem, jakým je bezesporu rychlý rozvoj výpočetní techniky v posledních letech, začíná se stále častěji projevovat i jeden z jeho důsledků, který pro nás, spojence počítače, zdaleka už není pozitivní není. Jedná se o neustále se zmenšující ochotu uživatelů spokojovat se s nepovedenými produkty výpočetních center jenom proto, že pocházejí od "Jeho Výsosti Počítače". Za co horšího, začínají se objevovat dokonce tendence (zatím naštěstí pouze ojedinělé)postihovat programátory a analytiky za jejich chybně pracující výtvory, tedy programy. Pod tíhou těchto okolností se v poslední době můžeme stále častěji setkat se snahami o zvýšení spolehlivosti softwarových produktů. Cílem tohoto referátu je přispět do diskuze na dané téma.

2. Verifikace programů.

V tomto odstavci se především pokusíme definovat několik základních pojmů z oblasti verifikace programů. V e r i f i k a c í programu rozumíme činnost (prováděnou programátorem či automatizovaně), jejímž cílem je

prokázat správnost či nesprávnost zkoumaného programu. S p r á v n ý m budeme nazývat program, neobsahující žádné logické chyby; jako s p o l e h l i v ý označíme program, který se za všech okolností chová tak, jak bylo jeho tvůrcem původně zamýšleno. Je tedy zřejmé, že správnost programu je nutnou podmínkou pro jeho spolehlivost, že spolehlivost programu je silnější pojem. Spolehlivost programu totiž kromě jeho logické správnosti zahrnuje navíc ještě celou řadu dalších komponent , jako například schopnost eliminace různých chyb, vzniklých

- poruchami hardware
- chybami v operačním systému
- nesprávným použitím programu
- vinou obsluhy počítače
- chybami v datech
- při komunikaci mezi uživatelem, analytikem, programátorem a operátorem

a pod.(viz např./6/)

Metody verifikace programů se od svého počátku ubíraly v podstatě dvěma různými směry. První z nich, snad by ho bylo možno též nazvat cestou empirickou, je znám pod pojmem ladění programu. Jeho princip,důvěrně známý všem programátorům, spočívá obvykle ve snaze vyloudit ze zkoumaného programu za pomocí určité množiny vstupních dat jinou, předem známou množinu dat výstupních. Je-li tato, obvykle značně úporná snaha, korunována úspěchem, bývá takový program prohlášen svým autorem za správný a jakákoliv zmínka o jeho chybách je přijímána se značnou nevolí. Ke cti naší programátorské obce neslouží příliš doba, která uplynula do zjištění, že pomocí této metody lze dokázat pouze nesprávnost zkoumaného programu, nikoliv jeho správnost.

Druhý z těchto směrů, snad by mu slušel přívlastek exaktní, se snaží teoretickým rozborem programu dokázat jeho správnost pomocí exaktního matematického aparátu. Výsledky této druhé metody zatím bohužel rovněž nejsou zcela uspokojivé. Její základní problém spočívá v nalezení vhodných prostředků matematické interpretace základních pojmů z oblasti programování, jako jsou například: vývojový diagram, proměnná, soubor, příkaz a podobně. V dalším se budeme snažit ukázat, že obě dříve zmíněné metody verifikace lze na rozhodovací tabulky dobře aplikovat.

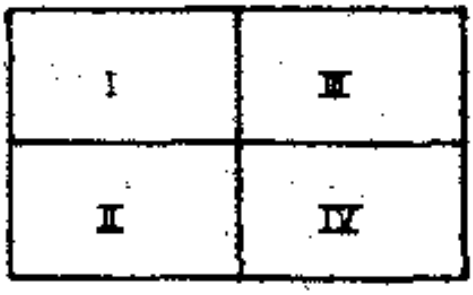
Na problematiku tvorby správných programů můžeme pohlédnout i z poněkud širšího úhlu, vezmeme-li v úvahu různá období jejich "života": etapu analýzy daného problému, návrhu algoritmu, výběru jazyka, zápisu algoritmu, překladu a vlastního chodu programu. Ve všech těchto etapách existuje značná pravděpodobnost výskytu chyb, s níž je nutno počítat. Programátor-analytik, který tento fakt odmítá akceptovat, by měl snad podle mého soudu raději změnit profesi. Pro každou z výše uvedených etap totiž už existují prostředky, které nám usnadňují naši úlohu, je pouze třeba je znát a umět vhodně použít.

3. R T a vývojové diagramy

Vývojovým diagramem budeme rozumět konečný graf, jehož uzly představují příkazy a hrany předávání řízení. Tento diagram může kromě jediného počátečního příkazu START obsahovat libovolné množství koncových příkazů STOP, přiřazovacích příkazů a rozhodovacích bloků. Cesta diagramem je posloupnost uzlů, z nichž první je START a každé dva další jsou spojeny hranou s příslušnou orientací. Poznamenejme, že tentýž uzel se může na jedné cestě vyskytnout vícekrát. Větev je posloupnost uzlů, z nichž první je buď START nebo rozhodovací, poslední

STOP nebo rozhodovací a uvnitř žádný rozhodovací není.

Rozhodovací tabulkou, resp.jejím jádrem, rozumíme obvykle strukturu, sestávající ze 4 kvadrantů (viz obr.1)



obr. 1

v níž
I.kvadrant se nazývá formulace podmínek,
II. - " - formulace činností
III. - " - volba podmínek
IV. - " - volba činností

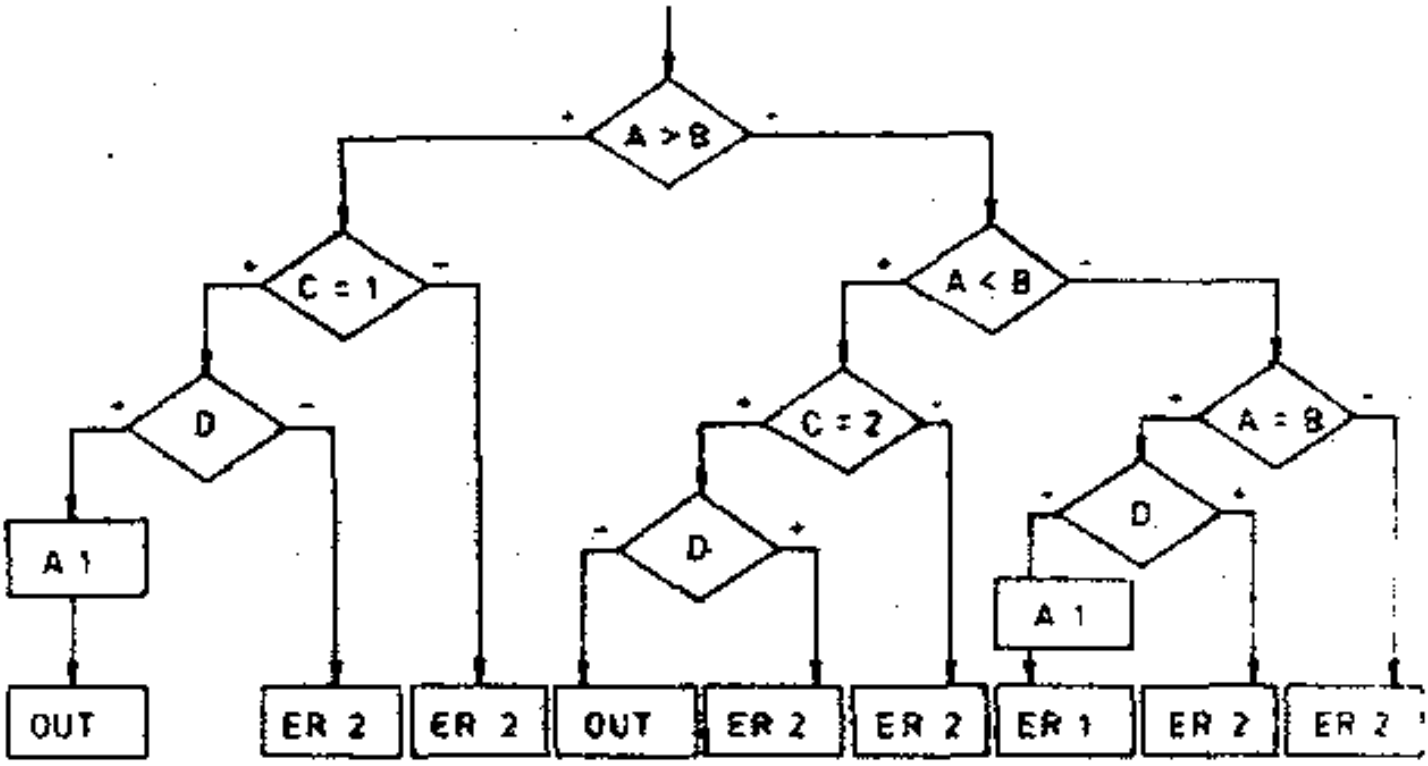
Jako příklad uveďme rozhodovací tabulku, v níž hostujícím jazykem je COBOL.

M	RT - 1	1	2	3	E
C	IF A ? B	>	<	=	
C	IF C = ?	1	2	-	
C	IF D	Y	N	N	
A	PERFORM A1	*	-	*	-
A	PERFORM ?	OUT	OUT	ER1	ER2

Poznámka. C na začátku řádku značí podmínku, A činnost, přerušované čáry představují dělení RT na jednotlivé kvadranty.

Všimněme si nyní v krátkosti formálního vztahu mezi rozhodovacími tabulkami a vývojovými diagramy. Je-li z nějakých důvodů (např.kvůli závislosti podmínek) pořadí podmínek pevně dáno, je každé rozhodovací tabulce

logicky ekvivalentní právě jednomu vývojovému diagramu; například výše uvedené RT-1 diagramu:



obr. 2

Předchozí příklad nám jednak ilustruje způsob převodu mezi oběma strukturami, jednak vhodně dokumentuje rozdíl v jejich složitosti a tím i srozumitelnosti. Jeho pomocí dále snadno nahlédneme, že pojmu cesta u vývojového diagramu odpovídá u rozhodovací tabulky pojem pravidlo.

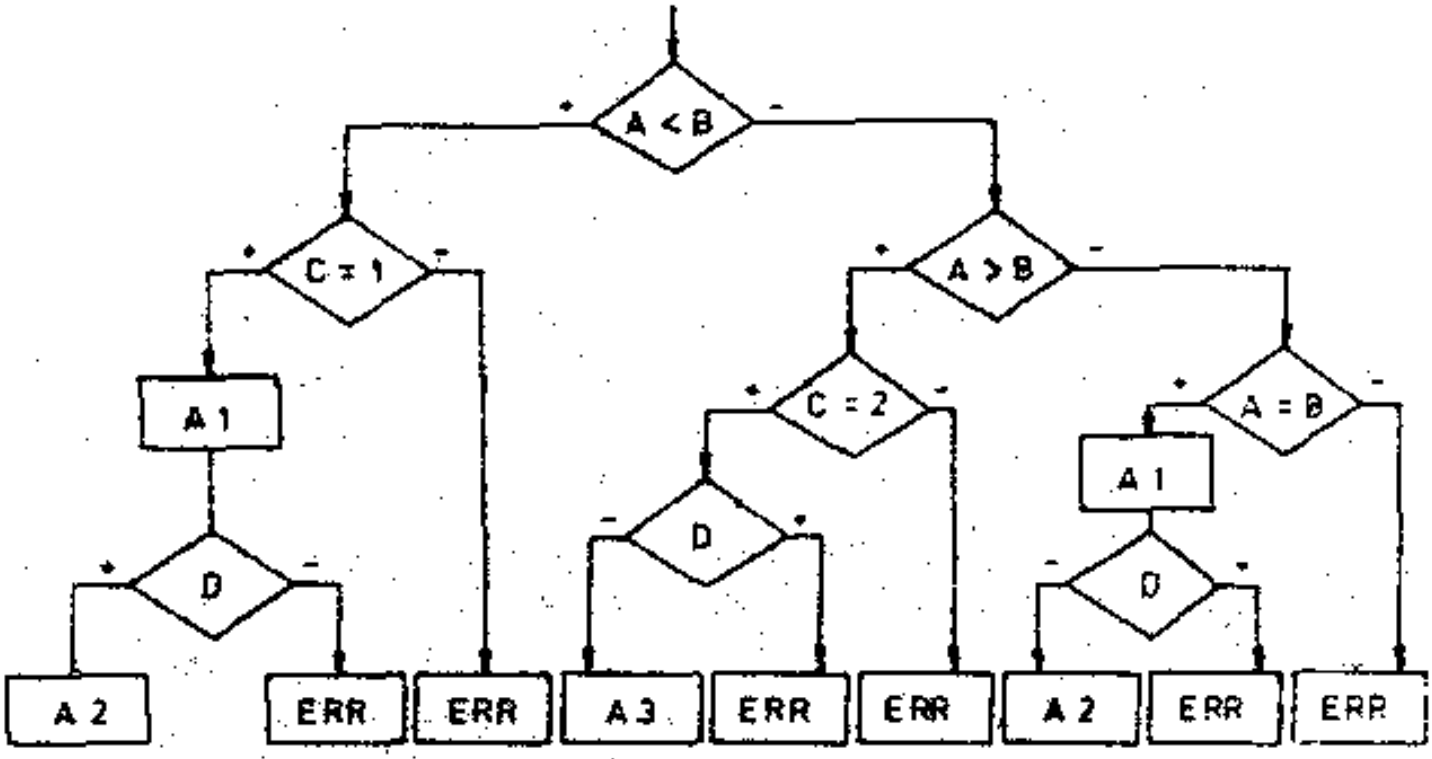
O rozhodovacích tabulkách, definovaných výše uvedeným způsobem, je známo, že jsou výborným pomocníkem programátorů a analytiků při řešení složitých rozhodovacích struktur, v nichž lze predikáty předřadit činnostem. V těchto případech jsou nesporně daleko lepším prostředkem formalizace programu než vývojové diagramy, především z důvodů své prosté algebraické struktury a neprocedurálnosti. Přestože těchto případů je v praxi daleko víc, než by se dalo na první pohled očekávat, ponévadž zařazování

určitých podmínek až za činnosti bývá velmi často pouze formální záležitostí, kterou lze snadno odstranit bez újmy na správnosti, je na druhé straně nezvratným faktem existence struktur, které takto upravit nelze. Bývá tomu tak obecně v případech, kdy některé z činností ovlivňují jisté podmínky. Je-li tabulka procházena cyklicky, lze tento jev většinou eliminovat zavedením dalších pomocných proměnných, případně uvědoměním si faktu, že se inkriminované akce „zpožďují" za podmínkami.

Přesto se zdá vhodné obohatit stávající pojem RT o možnost libovolného umístění podmínek a činností, tedy odstranit pevnou hranici mezi jejich podmínkovou a činnostní částí. Je sice zřejmé, že i po této úpravě zůstanou vývojové diagramy obecnějším nástrojem, na druhé straně se však třída programů, v nichž budou moci být takové RT použity, nesporně rozšíří. RT kromě toho svůj předchozí handicap vyrovnávají řadou jiných výhod, z nichž jednou je, jak se budeme snažit dále dokázat, jejich daleko větší vhodnost pro účely verifikace. Než k tomu přistoupíme, ukažme si v krátkosti příklad takové zobecněné RT

H	RT-2		1	2	3	E
C	IF	A ?B	<	>	=	
C	IF	C= ?	1	2	-	
A	PERFORM	A1	*	-	*	
C	IF	D	Y	N	N	
A	PERFORM	?	A2	A3	A2	ERR

a s ní ekvivalentního vývojového diagramu



obr 3

Zdá se, že s převodem mezi takto zobecněnými RT a vývojovými diagramy by neměly vzniknout žádné problémy. Dále na předchozím příkladě lze postřehnout analogii pojmu větev z vývojového diagramu, který u normální RT splývá s pojmem cesty. Zde tomuto pojmu odpovídají všechny činnosti určitého pravidla, které jsou buď na počátku, nebo na konci tabulky, nebo jsou uvnitřny mezi dvěma podmínkami.

4. Verifikace RT

Cílem této kapitoly je nastínit alespoň v hrubých rysech prostředky, pomocí nichž by překladač RT mohl a tedy i měl usnadnit uživateli práci s verifikací programu, obsahujícího RT. Hovoříme-li zde o překladači RT, máme především na mysli předkompilátor do některého z vyšších programovacích jazyků, t.zv.hostujícího jazyka, poněvadž tato jeho forma se zdá být z mnoha důvodů nejvhodnější (viz např. /8/). O jednom z těchto důvodů bude ještě řeč později. Nyní několik slov k vlastní otázce

ověřování správnosti RT. Z chronologického i metodologického hlediska by tento proces měl sestávat ze dvou etap:

- etapy překladu RT
- etapy chodu programu, který RT obsahuje.

Přitom v první z obou etap by překladač RT kromě jejich běžného překladu vykonával navíc ještě další dvě činnosti:

1. Kontrolu logické správnosti dané RT, t.j.kontrolu její úplnosti, redundance, kontradikce a t.d. Bližší podrobnosti o této kontrole se lze dočíst například ve /7/, /2/,/3/. Snad jenom v krátkosti poznamenejme, že úplnou tabulkou se rozumí ta, která ve III. kvadrantě obsahuje všechny možné kombinace podmínek, redundancí že rozumíme v podstatě jev, kdy dvě nebo více pravidel obsahují stejné podmínky i činnosti, zatím co kontradikcí jev, kdy se pravidla se stejnými podmínkami liší v činnostech.

H	RT - 3			
C	IF P1	Y	Y	Y
C	IF P2	-	-	-
C	IF P3	N	N	N
A	PERFORM A 1	*	*	-
A	PERFORM A 2	-	-	*

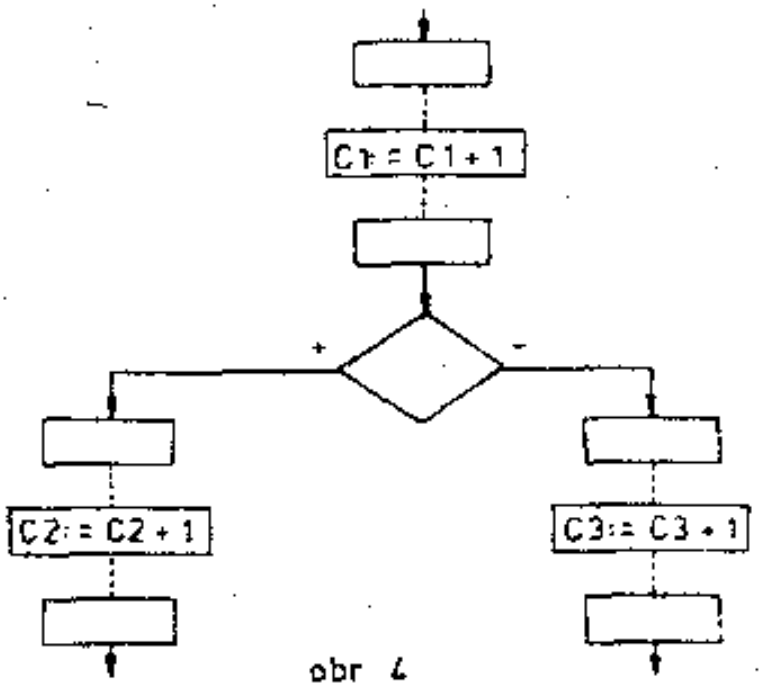
Například výše uvedená tabulka RT-3 zřejmě není úplná a navíc pravidla 1 a 2 jsou redundantní, pravidla 1 a 3 nebo 2 a 3 kontradiktonická. V poměrně snadno realizovatelné automatizaci kontrol tohoto druhu tkví snad největší přednost rozhodovacích tabulek před vývojovými diagramy v oblasti verifikace.

2. Generování příkazů hostujícího jazyka, sloužících k testování dané RT v době chodu programu.

Dříve, než přistoupíme k jejich bližší specifikaci, poznamenejme ještě, že z praktických důvodů se zdá být účelné oddělit obě dvě dříve zmíněné funkce překladače od ostatních i od sebe navzájem, tak, aby je bylo možno požadovat

volitelně. Je totiž zřejmé, že první z nich je do značné míry nezávislá na hostujícím jazyce a bylo by ji tedy možno s výhodou použít už například v etapě analýzy či v době, kdy ještě celý program není k dispozici, nebo kdy ani vůbec vytvářen nebude. V tomto kontextu se jeví překladač RT ve formě předkompilátoru jako mimořádně výhodný. Rovněž druhá z obou funkcí nemusí být (například následkem nemístné sebedůvěry programátora) vůbec požadována.

Všimněme si nyní ale blíže činností, uvedaných v bodě 2. a snažme se je blíže specifikovat. Pokusme se přitom vyjít z některých známých metod verifikace, užívaných pro vývojové diagramy. Jednou z nich je například metoda čítačů. Čítače jsou v podstatě celočíselné proměnné, registrující počet průchodů programu jeho určitým místem pomocí přiřazovacího příkazu c: = c + 1 (viz obr.4). Užívá se jich ke stanovení frekventovanosti jednotlivých částí programu; „mrtvých" větví programu, t.j.těch, jimiž výpočet nikdy neprochází a je tedy možno je pominout a dále části velmi frekventovaných, které se vyplatí optimalizovat.



obr 4

Aby čítače plnily optimálně svou funkci, bývá zvykem umisťovat je do každé větve vývojového diagramu; analogií

pojmu větev u RT jsme již zavedli v kapitole 3. V případě normálních RT to tedy především znamená vytvořit:

a)Statistiku průchodů jednotlivými pravidly. Její hodnoty nám totiž mohou pomoci, jak při verifikaci dané tabulky, především odhalením „mrtvých" pravidel, t.j. pravidel s nulovou frekvencí, které může být zapříčiněna jejich logickou (pouze v případě nedokonalého překladače) či sémantickou irracionalitou, tak při optimalizaci jejího překladu (viz/2/).Pro realizaci tohoto záměru je třeba do výstupního kódu vygenerovat:

- do pracovní oblasti programu pole s rozsahem v souladu s počtem pravidel
- mezi inicializační činnosti programu jeho nulování
- mezi činnosti, příslušné určitému pravidlu, zvětšení příslušného prvku tohoto pole o 1
- mezi finální činnosti programu výpis obsahu tohoto pole.

Kromě toho by bylo možné umístit do tohoto místa programu na přání uživatele úpravu těchto hodnot, pokud jsou součástí dané RT a tato je umístěna na př.ve zdrojové knihovně. Tímto způsobem by mohla pro uživatele být pohodlně vyřešena otázka optimalizace RT.

Podobně jako počet průchodů pravidly by bylo možné zjišťovat:

b) Počet průchodů celou tabulkou
c) Počet provádění testů jednotlivých podmínek
d) Počet provádění jednotlivých akcí

atd.

Bližší způsob realizace těchto dalších prostředků ponechme prozatím stranou; poznamenejme pouze, že na rozdíl od pomůcek, uvedených v dalším, mají statický charakter. Dynamický charakter mají naproti tomu:

e) Výpis pořadového čísla právě prováděného pravidla
f) Výpis momentálních hodnot všech podmínek

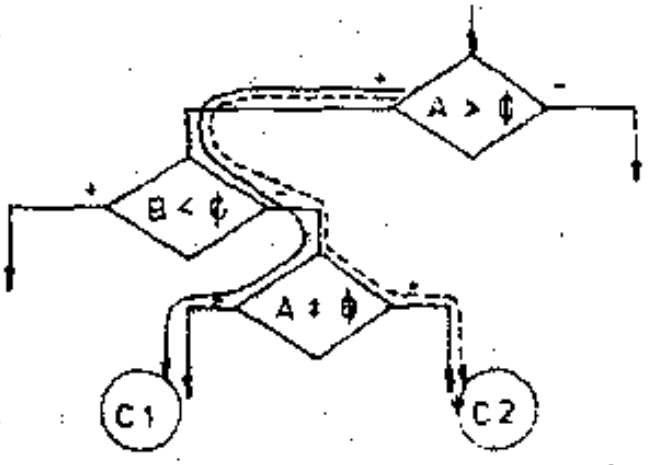
g) Výpis pořadového čísla právě testované podmínky

h) Výpis pořadového čísla právě prováděné akce

Tyto pomůcky navíc nemají žádný bližší vztah ke dříve zmíněné metodě čítačů ; přesto se zdají být rovněž pro RT vhodné.

Snadnost a poměrně značná efektivita metody čítačů vedla k návrhům, aby bylo standardizováno jejich zařazování do kompilátorů. Situace zatím bohužel vypadá tak, že tento, dle mého názoru jistě přinejmenším pozoruhodný požadavek, akceptován nebyl. Domnívám se, že hlavní důvod tohoto faktu je třeba hledat především v ne zcela triviálních redundantních činnostech, které je nutno provést za účelem separace jednotlivých větví programu. Jak snad bylo dříve ukázáno, u rozhodovacích tabulek a jejich kompilátorů by bylo řešení tohoto problému značně jednodušší.

Další, již dříve zmíněnou metodou ladění programů, je jejich prověřování pomocí jisté množiny kontrolních dat. Tato množina se nazývá spolehlivá, jestliže správná funkce programu na jejich prvcích zaručuje správnost programu na všech datech. Nutnou podmínkou této spolehlivosti je, aby každý realizovatelný příkaz programu byl alespoň jednou proveden; postačující podmínkou pak je průchod všemi realizovatelnými cestami (např. plně vyznačená cesta C 1 na obr.5 není realizovatelná v důsledku implikace $A>\phi \Rightarrow A \neq \phi$, což je obecně úkol prakticky nezvládnutelný.



obr. 5

množinu vstupních dat dále nazýváme úplnou, jestliže zabezpečí zasažení všech realizovatelných příkazů, větví a cest. Problém vytvoření takové množiny vstupních dat do značné míry souvisí s pojmem podmínky cesty. Například pro vývojový diagram na obr.5 je podmínka cesty C 2 : $A > \emptyset \wedge \neg B < \emptyset \wedge A \neq \emptyset$, tedy $A > \emptyset \wedge \neg B < \emptyset$. Vytvoříme-li rozhodovací tabulku, ekvivalentní předchozímu vývojovému diagramu, lépe řečeno její podmínkovou část,

			C1	C2		
C	IF	$A > \emptyset$	Y	Y	Y	N
C	IF	$B < \emptyset$	Y	N	N	-
C	IF	$A \neq \emptyset$	-	N	Y	-

vidíme, že jednotlivým podmínkám cest odpovídají konjugované hodnoty podmínek v příslušných pravidlech. Kromě toho se zdá být účelnější místo příslušné podmínky cesty uvažovat pouze kombinace voleb v jednotlivých pravidlech. Potom by se proces vytvoření úplné množiny kontrolních dat stal triviální, snadno automatizovatelnou záležitostí. Předpokládejme například, že zkoumaná rozhodovací tabulka obsahuje n podmínek a tyto mohou mít po řadě $m_1,\ldots,m_n$ různých voleb. Pak vytvoření úplné množiny kontrolních dat odpovídá vygenerování $m_1 x \ldots x\ m_n$ ($m_i \geq 2$ pro $i = 1,\ldots,n$) různých entic, odpovídajících všem možným kombinacím voleb podmínek, což by vzhledem k paměťovým kapacitám současných počítačů a vzhledem k faktu, že obvykle platí $m_i = 2$, nemělo být ani u rozsáhlejších tabulek problémem. Navíc by snad bylo vhodné obohatit daný systém o čtecí modul, který by při každém svém vyvolání předal systému hodnotu další entice. Zdá se však, že generování celé reálné úplné množiny kontrolních dat a její následující čtení je v tomto případě zbytečné. Daleko efektivnější by bylo vytvoření modulu, který by na základě vektorů všech možných voleb podmínek vytvářel jejich kombinace pouze uvnitř sebe sama a jehož výstupem by při každém jeho dalším vyvolání byla

další entice. Aplikace takto vytvořené úplné množiny vstupních dat, na rozhodovací tabulku, obohacenou (ať už zásluhou programátora či překladače) o dříve popsané ladicí prostředky by pak představovala velmi účinnou zbraň v boji za tvorbu správných programů.

5. Závěr

Jak bylo již dříve řečeno, požadavek tvorby spolehlivého a tedy i správného software se stává stále naléhavějším. Cílem tohoto příspěvku bylo ukázat rozhodovací tabulky jako jeden z prostředků, jimiž lze tuto problematiku do značné míry pozitivně ovlivnit.

Literatura

1.Hořejš J.: Ladění programů, sborník VVS OSN Bratislava ze semináře SOFSEM '76

2.Humbry E.: Programy na základe rozhodovacích tabuliek, Alfa, 1976

3.Chvalovský V.:Rozhodovací tabulky, SNTL, 1974

4.Jiříček P.:Problémy využívání rozhodovacích tabulek, sborník ze semináře Programování '78

5.Kešner J.: Rozhodovací tabulky a jejich využití v budování ASŘ, Výběr, 1976/3-5

6.Lacko B. : Spolehlivost programů, sborník ze semináře Programování '77

7.Zelenka J.:Programové overovanie správnosti RT-programov, sborník ze semináře Programování '78

8.Zelenka J.:Strojový preklad rozhodovacích tabuliek, sborník ze semináře Aplikace metody RT