Petr Hájek; Pavel Kalášek; Petr Kůrka

O dynamické logice

Kybernetika, Vol. 16 (1980), No. Suppl, (1),3--41

Persistent URL: http://dml.cz/dmlcz/124858

Kybernetika

O dynamické logice

Petr Hájek, Pavel Kalášek, Petr Kůrka

ACADEMIA
PRAHA

ÚVOD

Slovy „dynamická logika" se rozumí různé formální systémy rozšiřující klasickou logiku tak, že vedle formálních výroků obsahuje i formální programy, přičemž z jednodušších výroků a programů lze tvořit složitější výroky a složitější programy. Zejména z každého programu α a každého výroku A lze v dynamické logice vytvořit výrok $[\alpha]A$, který, zhruba řečeno, znamená „kdykoli se zastaví (provede) program α, platí A".

V tomto pojednání se čtenář seznámí s některými přirozenými systémy dynamické logiky a s jejich základními vlastnostmi. Pozná zejména, jak lze aparátu dynamické logiky využít k vyjádření různých přirozených tvrzení o programech a nalezne zde prostředky, pomocí nichž lze taková tvrzení dokazovat. I když zde nalezne jen důkazy tvrzení o poměrně jednoduchých programech, poznání dynamické logiky mu, doufáme, dá jistý, pro něj možná nový, abstraktní pohled na programy a jejich logiku. V tom vidíme hlavní smysl článku.

Dynamická logika, kterou zde vykládáme, byla vytvořena v několika posledních letech převážně americkými pracovníky v computer science. Základní prací je Prattův článek [8]; termín „dynamická logika" se poprvé vyskytuje v článku [1] Fishera a Ladnera. Významné výsledky dosáhli dále Harel, Meyer, Parikh a jiní (viz seznam literatury). Ještě dříve než v USA byly podobné systémy studovány v Polsku pod názvem *algoritmická logika*. Jejím zakladatelem je Salwicki [12] a dnes v Polsku existuje početná škola algoritmické logiky (viz [11]). Algoritmická logika se v mnohém shoduje s dynamickou; základní rozdíl je v tom, že algoritmická logika studuje pouze deterministické programy, zatímco dynamická logika studuje programy nedeterministické, což dává větší obecnost a zároveň též větší eleganci a jednoduchost.

V tomto výkladu vycházíme především z Harelovy disertace [2] a Prattova článku [9]. V našem pojednání poněkud upravujeme symboliku a modifikujeme některé pojmy z právě citovaných prací, pokud se to pro srozumitelnost zdálo výhodné. Novým výsledkem je axiomatizace dynamického predikátového počtu s přiřazováním polím (Hájek, Kůrka [13]). Jak jsme již uvedli, jde především o to, aby čtenář získal ucelený přehled o dynamické logice. Aby bylo jasné, v jakém vztahu je dynamická logika ke klasické logice, vyložíme zde v jednotném pojetí kromě systémů dynamické logiky i základní pojmy klasického výrokového a predikátového počtu. Poslední část článku obsahuje tři podrobné příklady dokazování vlastností programů v dynamické logice. Děkujeme doc. J. Bečvářovi, CSc., a ing. J. Radovi, CSc., kteří pečlivě přečetli celý text, za řadu velmi cenných připomínek.

Tento článek vznikl revizí a rozšířením textu, který autoři napsali pro seminář SOFSEM 79. Autoři děkují programovému výboru semináře za svolení k této formě publikace.

OBSAH

1. LOGICKÉ SYSTÉMY

Formální logické systémy (kalkuly) vypadají zpravidla tak, že se nejprve definuje, co jsou *výrazy* takového systému. Tyto výrazy mají obvykle nějakou syntaktickou (kombinatorickou) strukturu, je možno je dělit na podvýrazy nebo kombinovat do složitějších výrazů, ale apriori nemají žádnou sémantiku, žádný význam, žádnou interpretaci. Druhý krok definice logického systému spočívá v popisu možných (přípustných) *interpretací*, tj. popisu toho, co mohou výrazy našeho systému *znamenat*.

Aby byl systém logicky a matematicky zajímavý, je třeba, aby byly splněny dva požadavky: Výrazy je možno interpretovat *různými* způsoby, ne jen jedním, ale přitom *ne zcela libovolně*: interpretace jsou podrobeny určitým omezením, která zpravidla popisují, jakým způsobem je interpretace složeného výrazu určena interpretací jeho podvýrazů. Už v elementární matematice se s takovými omezeními setkáváme: při „počítání s písmeny" víme, že písmena (proměnné) můžeme interpretovat jako libovolná (řekněme přirozená) čísla, tedy můžeme např. interpretovat a jako 17, b jako 9, anebo – při jiné interpretaci – a jako 0, b jako 30, ale že jsme zavázáni interpretovat $a + b$ jako *součet* interpretace obou proměnných, tj. v prvním případě jako 26 a v druhém jako 30. Podobně ve výrokovém počtu (viz níže) můžeme výrokové proměnné interpretovat libovolnými pravdivostními hodnotami (pišme 1 pro pravdivostní hodnotu „pravda" a 0 pro pravdivostní hodnotu „nepravda"), ale interpretace (ohodnocení) výrokových proměnných už jednoznačně určuje interpretaci každé formule z nich vytvořené; např. je-li p interpretováno jako pravdivé a q jako nepravdivé, jsme povinni konjunkci $(p \& q)$ interpretovat jako nepravdivou, tj. přiřadit jí hodnotu 0. Taková *sémantická omezení* jsou specifická pro každý konkrétní systém a je třeba je explicitně definovat (nerozumí se sama sebou).

Pokud některým výrazům logického systému říkáme formule, rozumíme tím, že tyto výrazy chápeme jako formální výroky (nebo schémata výroků) a že v každé

interpretaci jim mají být připisovány pravdivostní hodnoty. V celém výkladu se omezíme na dvě uvedené pravdivostní hodnoty, 1 a 0. V některých logických systémech (např. v klasickém výrokovém počtu) je každý výraz formule. V jiných je tomu jinak: v predikátovém počtu je např. výraz $x + y = y + x$ formule (formální výrok), ale $x + y$ už není formule, ale *term*, tj. výraz, jehož významem je nějaký konkrétní objekt, nikoli pravdivostní hodnota.

Hlavním předmětem našeho zájmu budou logické systémy, jejichž některé výrazy jsou chápány jako *formule* (formální výroky) a jiné výrazy jsou chápány jako *programy*, přičemž z jednodušších formulí a programů lze vytvářet složitější formule a složitější programy. Uvedeme sémantická omezení na interpretace takových výrazů, která umožňují užít tyto systémy k přirozenému vyjadřování důležitých vlastností programů. Tyto systémy se nazývají systémy *dynamické logiky* (nebo též systémy *algoritmické logiky*).

Třetí krok při budování logického systému spočívá ve snaze o *axiomatizaci*, tj. ve snaze vybudovat *důkazový aparát*. Tento důkazový aparát má splňovat dvě podmínky: (1) být nezávislý na sémantice, tj. definice důkazu nemá užívat pojmu interpretace formulí (nýbrž má vycházet z vnitřní struktury formulí), ale (2) má být v dobrém vztahu k sémantice, především má být *korektní*, tj. každá dokazatelná formule má být pravdivá v každé interpretaci. Formule pravdivé v každé interpretaci nazýváme *tautologie*. Na důkazový aparát můžeme položit také obrácený požadavek, požadavek *úplnosti*: každá tautologie je dokazatelná.

Pojem důkazu je vždy definován takto: udá se nějaká jednoduše popsaná množina axiomů (to jsou jisté speciální tautologie) a nějaká jednoduše popsaná dedukční pravidla tvaru „z předpokladů $A_1, \ldots, A_n$ bezprostředně odvoď B". Důkaz je pak libovolná konečná posloupnost $B_1, \ldots, B_k$ formulí, jejíž každý člen je buď axiom nebo jej lze bezprostředně odvodit z některých předchozích členů podle některého dedukčního pravidla.

Pro každý systém je důležitá otázka, (1) zda lze algoritmicky rozhodnout, zda posloupnost formulí je důkazem, či ne a (2) zda lze algoritmicky rozhodnout, zda daná formule je dokazatelná (tj. zda existuje důkaz, který má tuto formuli jako poslední člen). Je-li odpověď na (2) kladná, je systém rozhodnutelný.

Popíšeme některé systémy dynamické výrokové logiky a dynamické predikátové logiky 1. řádu včetně jejich důkazového aparátu. Uvedeme obsáhlé příklady důkazů vlastností konkrétních programů. Přitom také stručně popíšeme klasický výrokový a predikátový počet, takže četba článku nevyžaduje předběžnou znalost klasické formální logiky.

2. KLASICKÝ VÝROKOVÝ POČET

Formule se budují z *výrokových proměnných* $(p, q, r, p_1, r_1, q_1, \ldots)$ pomocí *logických spojek* $\&, \vee, \rightarrow, \equiv, \neg$ (nazývaných postupně konjunkce, disjunkce, implikace, ekvivalence, negace) a závorek podle těchto pravidel:

Každá výroková proměnná je formule. Jsou-li A, B formule, pak $(A \,\&\, B)$, $(A \vee B)$, $(A \rightarrow B)$, $(A \equiv B)$, $\neg A$ jsou formule. Čteme je po řadě: „A a B", „A nebo B", „A implikuje B", „A právě když B", „ne-A".

Příklady formulí: $p \rightarrow (q \vee r)$; $\neg\neg p \rightarrow p$; $\neg(p \,\&\, q) \equiv (\neg p \vee \neg q)$. (Vynecháváme nejvnějšjší závorky.)

Přípustná interpretace výrokového počtu je zobrazení $\boldsymbol{I}$ přiřazující každé formuli hodnotu 1 nebo hodnotu 0 a respektující následující podmínky: Pro všechny formule A, B platí:

$\boldsymbol{I}(A \,\&\, B) = 1$, právě když $\boldsymbol{I}(A) = 1$ a $\boldsymbol{I}(B) = 1$,
$\boldsymbol{I}(A \vee B) = 1$, právě když $\boldsymbol{I}(A) = 1$ nebo $\boldsymbol{I}(B) = 1$ (nebo obojí)
$\boldsymbol{I}(A \rightarrow B) = 1$, právě když $\boldsymbol{I}(A) = 0$ nebo $\boldsymbol{I}(B) = 1$ (nebo obojí)
$\boldsymbol{I}(A \equiv B) = 1$, právě když $\boldsymbol{I}(A) = \boldsymbol{I}(B)$,
$\boldsymbol{I}(\neg A) = 1 - \boldsymbol{I}(A)$.

První dvě podmínky můžeme zapsat také takto:

$$\boldsymbol{I}(A \,\&\, B) = \min(\boldsymbol{I}(A), \boldsymbol{I}(B)),\ \boldsymbol{I}(A \vee B) = \max(\boldsymbol{I}(A), \boldsymbol{I}(B)).$$

Slovy to můžeme vyjádřit takto: Konjunkce dvou formulí je pravdivá, právě když obě tyto formule jsou pravdivé; disjunkce dvou formulí je pravdivá, právě když aspoň jedna z těchto formulí je pravdivá, atd.

Tyto podmínky (tj. tato sémantická omezení) můžeme shrnout do známých pravdivostních tabulek.

A	$\neg A$
1	0
0	1

A	B	$A \,\&\, B$	$A \vee B$	$A \rightarrow B$	$A \equiv B$
1	1	1	1	1	1
1	0	0	1	0	0
0	1	0	1	1	0
0	0	0	0	1	1

Každá interpretace je zřejmě jednoznačně určena svými hodnotami na výrokových proměnných; z nich je možno hodnotu libovolné formule vypočítat. Např.:

p	q	r	$q \vee r$	$p \rightarrow (q \vee r)$
1	0	0	0	0

Vidíme, že formule $p \to (q \vee r)$ není tautologií – v uvedené interpretaci má hodnotu 0. Čtenář si může ověřit, že oba další výše uvedené příklady formulí, tj. $\neg\neg p \to p$, $\neg(p \& q) \equiv (\neg p \vee \neg q)$ jsou tatologie. V prvním případě stačí vyšetřit dva případy $I(p) = 1$ a $I(p) = 0$ a v druhém čtyři ($I(p)$ může být 1 nebo 0 a $I(q)$ také). Obecně: obsahuje-li formule n výrokových proměnných, je třeba vyšetřit 2^n případů, pro každou n-tici nul a jedniček jeden. Čtenář již vidí algoritmus, který umožňuje rozhodnout o každé formuli výrokového počtu, zda je či není tautologií.

O tautologičnosti se lze též přesvědčit dokazováním. K tomu je třeba udat axiomy a dedukční pravidla. Konkrétních axiomatizací výrokového počtu je celá řada; uvedeme jen fragment, axiomatizující tautologie vytvořené z výrokových proměnných pomocí spojek implikace a negace.

Schémata axiomů. Pro všechny formule A, B, C jsou následující formule axiomy:

(1) $A \to (B \to A)$

(2) $(A \to (B \to C)) \to ((A \to B) \to (A \to C))$

(3) $(\neg B \to \neg A) \to (A \to B)$

Dedukční pravidlo odloučení (modus ponens):

Z formule A a z formule $A \to B$ bezprostředně odvoď formuli B.

Čtenář snadno ověří, že každý z axiomů je tautologie a dále, je-li A tautologie také $A \to B$ je tautologie, pak i B je tautologie, tedy naše axiomatizace je korektní. Lze též dokázat (ale to už není snadné), že tato axiomatizace je pro formule uvedeného druhu úplná, tj. každá tautologie vybudovaná z výrokových proměnných pomocí implikace a negace je dokazatelná.

Příklad důkazu ve výrokovém počtu.

$$p \to ((p \to p) \to p) \qquad \text{(ax. 1)}$$

$$(p \to ((p \to p) \to p)) \to ((p \to (p \to p)) \to (p \to p)) \qquad \text{(ax. 2)}$$

$$(p \to (p \to p)) \to (p \to p) \quad \text{modus ponens}$$

$$p \to (p \to p) \qquad \text{(ax. 1)}$$

$$p \to p \qquad \text{modus ponens}$$

Tento tradiční příklad důkazu formule $p \to p$ ukazuje, že formální dokazování i jednoduchých formulí ve výrokovém počtu může být velmi pracné. Příklad však ilustruje dobře užití axiomů a dedukčních pravidel.

Všechny systémy studované dále budou tak či onak výrokový počet obsahovat; protože však množina všech tautologií výrokového počtu je algoritmicky rozhodnutelná, nebudeme se v dalším bránit tomu, brát rovnou všechny tautologie výrokového počtu za axiomy (vedle dalších axiomů specifických pro daný systém).

Shrňme tedy:

Formule výrokového počtu jsou tvořeny z výrokových proměnných (= atomických formulí) pomocí logických spojek. Každá interpretace výrokového počtu je plně určena hodnotami výrokových proměnných. Problém tautologičnosti pro formule výrokového počtu je algoritmicky řešitelný. Existuje jednoduchá axiomatizace výrokového počtu, která je korektní a úplná.

3. DYNAMICKÝ VÝROKOVÝ POČET

Popíšeme nyní nejdřív neformálně logický systém, jehož výrazy se dělí na formule (podobné formulím klasického výrokového počtu) a programy. Náš pojem programu bude značně abstraktní; v dalších částech textu bude pojednáno o konkrétnějších typech programů. Všimneme si té základní skutečnosti, že program je věc, která, je-li provedena, může něco změnit: výrok, který byl pravdivý před spuštěním programu, může být pro zastavení (provedení) programu nepravdivý (např. „hodnota toho a toho paměťového místa je 0"). Toto pozorování je základem našeho způsobu intepretace programů: budeme mít vždy nějakou abstraktní množinu W *stavů* (stavů počítače, stavů světa, chcete-li: možných světů) a interpretace přiřadí každému výrazu α, který je programem (ve smyslu našeho logického systému) nějakou binární relaci I_α na W, tj. množinu jistých uspořádaných dvojic abstraktních stavů ($I_\alpha \subseteq W \times W$). Jsou-li stavy s, t v této relaci, značíme to $\langle s, t\rangle \in I_\alpha$, nebo též $s \xrightarrow{\alpha} t$ a chápeme to tak, že je-li program α spuštěn ve stavu s, může tento stav převést do stavu t (tj. po zastavení programu budeme, resp. bude počítač ve stavu t). Ujasněme si hned některé důsledky tohoto pojetí:

(1) Může se stát, že k danému s neexistuje žádné t takové, že $s \xrightarrow{\alpha} t$, např. když program α, spuštěn ve stavu s v intuitivním smyslu diverguje.

(2) K danému s může existovat jediné t takové, že $s \xrightarrow{\alpha} t$; ale připouštíme též možnost, že k danému s existuje více různých t takových, že $s \xrightarrow{\alpha} t$, to je studujeme nedeterministické programy. Důvody pro toto pojetí budou prodiskutovány za chvíli.

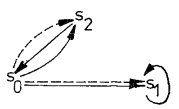
Zvolená interpretace programů ovlivní také interpretaci výroků: interpretace nebude připisovat výroku jedinou pravdivostní hodnotu, ale pravdivostní hodnota bude funkcí výroku a stavu. (To odpovídá tomu, co jsme řekli výše: tentýž výrok může být pravdivý v jednom stavu a nepravdivý v jiném stavu.)

Máme-li tedy množinu $AF = \{p_1, p_2, \ldots\}$ *atomických formulí* (= výrokových proměnných), množinu $AP = \{a_1, a_2, \ldots\}$ *atomických programů* a množinu W abstraktních *stavů*, pak *interpretace* I množin AF a AP ve W je dána tím, že každé atomické formuli p_i je přiřazeno zobrazení I_{p_i} množiny W do množiny $\{0, 1\}$ a každému atomickému programu a_i je přiřazena relace $I_{a_i} \subseteq W \times W$.

Jsou-li množiny AF, AP a W konečné a dost malé, můžeme takovou interpretaci vyjádřit tabulkou třeba takto:

	p	q	r	a	b
s_0	1	0	0	s_1, s_2	s_1, s_2
s_1	0	1	1	s_1	—
s_2	0	0	1	s_0	—

Zde tedy máme tři výrokové proměnné $AF = \{p, q, r\}$ dva atomické programy $AP = \{a, b\}$ a tři stavy: $W = \{s_0, s_1, s_2\}$. I_p je v prvním sloupci: např. $I_p(s_0) = 1$, tj. výrok p je pravdivý ve stavu s_0. Ve sloupci pod a je reprezentováno I_a: vidíme, že $s_0 \xrightarrow{a} s_1$, $s_0 \xrightarrow{a} s_2$, $s_1 \xrightarrow{a} s_1$, $s_2 \xrightarrow{a} s_0$ a žádné další dvojice už v relaci I_a nejsou. Interpretaci programů bychom alternativně mohli vyjádřit multigrafem, v němž plné šipky odpovídají programu a a čárkované šipky odpovídají programu b:



Základní formální konstrukce, které umožňují z formulí a programů vytvářet nové formule, jsou tyto dvě:

$[a]\, p$ — čteme *a-nutně p*

$\langle a \rangle\, p$ — čteme *a-možná p*

(To je vypůjčeno z modální logiky.) Formule $[a]\, p$ je pravdivá v nějakém stavu s, jestliže formule p je pravdivá v každém stavu, do kterého může být stav s programem a převeden. Formule $\langle a \rangle\, p$ je pravdivá ve stavu s, jestliže p je pravdivá v alespoň jednom stavu, do kterého může být stav s programem převeden. Tedy v našem případě formule $[a]\, p$ je nepravdivá ve stavu s_0 (neboť p není pravdivá v s_1, s_2, což jsou stavy, kam a převádí s_0), ale je pravdivá ve stavu s_2; formule $[a]\, r$ je pravdivá v s_0, formule $\langle a \rangle\, q$ je pravdivá v s_0, ale $[a]\, q$ je v s_0 nepravdivá. Všimněte si také, že formule $[b]\, p$ je pravdivá v s_1 (protože b nepřevádí s_1 vůbec nikam, tj. ve všech stavech, kam b převádí stav s_1, platí cokoli), ale $\langle b \rangle\, p$ je nepravdivá v s_1, neboť neexistuje stav, do něž b převádí s_1.

Formule budeme pochopitelně kombinovat do složitějších formulí pomocí logických spojek &, $\vee$, $\rightarrow$, $\equiv$, $\neg$. Tak jako lze formule kombinovat pomocí logických spojek, tak budeme i programy a formule kombinovat do složitějších programů pomocí programových konstruktů. Tento způsob pojetí programů pochází z teorie

strukturovaného programování. Mezi hlavní konstrukty, kterými se z programů α, β a formule A tvoří složitější, patří

BEGIN α; β END — *kompozice*
IF A THEN α ELSE β — *větvení*
WHILE A DO α — *cykl*

Z logického hlediska konstrukty větvení a cyklu nejsou primitivní, ale skládají se z jednodušších konstruktů testování, sjednocení a iterace:

A? — *test*
$\alpha \cup \beta$ — *sjednocení*
α^* — *iterace*
$\alpha; \beta$ — *kompozice.*

Dynamická logika umožňuje tyto koncepty isolovat a studovat samostatně.

Program $(\alpha; \beta)$ čteme „α a potom β"
$(\alpha \cup \beta)$ čteme „α nebo β"
α^* čteme „iterovaně α"

Program $(\alpha; \beta)$ funguje tak, že daný stav s nejprve převede do některého stavu t, do něhož může být s převeden programem α, a pak tento stav t převede do některého stavu, do něhož stav t může být převeden programem β. Jinými slovy, interpretací programu $(\alpha; \beta)$ je *složení* relací interpretujících α, β.

Program $(\alpha \cup \beta)$ může převést stav s do každého stavu, do kterého může být s převeden programem α nebo programem β. Interpretací programu $(\alpha \cup \beta)$ je tedy *sjednocení* interpretací programů α, β.

Konečně program α^* funguje jako α několikrát po sobě opakované (včetně nulkrát). Tj. α^* převede stav s do sebe (tj. do s), dále do každého stavu, do kterého může s být převeden programem α, programem $(\alpha; \alpha)$, programem $(\alpha; \alpha; \alpha)$, programem $(\alpha; \alpha; \alpha; \alpha)$ atd. Interpretací je *reflexivní transitivní obal* interpretace programu α.

Budeme mít také způsob, jak z formulí tvořit programy. Je-li A formule, pak $(A?)$ je program zvaný *test*, který funguje takto: Jestliže ve stavu s platí A, pak $(A?)$ převádí s do sebe; jestliže v s neplatí A, pak $(A?)$ nepřevádí s nikam.

Jsme plně připraveni k formální definici dynamického výrokového počtu.

Symboly: atomické formule $p, q, \ldots$
atomické programy $a, b, \ldots$
logické spojky $\&, \vee, \rightarrow, \equiv, \neg$
programové spojky $;, \cup, *$
modality $[\,], \langle\,\rangle$
test ?
závorky (,)

Formule a programy: (1) Každá atomická formule je formule a každý atomický program je program. (2) Jsou-li A, B formule, pak i $(A \& B)$, $(A \vee B)$, $(A \to B)$, $(A \equiv B)$, $\neg A$ jsou formule. (3) Jsou-li α, β programy, pak i $(\alpha; \beta)$, $(\alpha \cup \beta)$, α^* jsou programy. (4) Je-li A formule, pak $(A?)$ je program. (5) Je-li A formule a α program, pak $[\alpha] A$, $\langle\alpha\rangle A$ jsou formule. (6) Jiné formule a programy nejsou.

Buď W neprázdná množina. *Interpretace* (výrazů dynamického výrokového počtu) s množinou stavů W je zobrazení I přiřazující každé formuli A jisté zobrazení I_A množiny W do $\{0, 1\}$ a každému programu α jistou relaci $I_\alpha \subseteq W \times W$ a splňující výše uvedené podmínky závislosti interpretace složených programů a formulí na interpretaci složek, např. $I_{A\&B}(s) = \min(I_A(s), I_B(s))$, $I_{A\vee B}(s) = \max(I_A(s), I_B(s))$, $I_{\neg A}(s) = 1 - I_A(s)$, $I_{\alpha;\beta}$ je složení relací I_α a I_β. Dále: $I_{\alpha\cup\beta}$ je sjednocení relací I_α a I_β, $I_{[\alpha]A}(s) = 1$, právě když pro každé t takové, že $s \xrightarrow{\alpha} t$, jest $I_A(t) = 1$, atd.

Je-li počet atomických programů, formulí a abstraktních stavů dost malý, můžeme formule a programy vyhodnocovat ve formě tabulky.

Příklad: Mějme formule (1) $p \to [a](q \vee r)$, (2) $\neg p \& [a] p$, (3) $[a] p \to [b] p$ a výše uvedenou interpretaci. Spočítáme $I_{(1)}$, $I_{(2)}$, $I_{(3)}$.

	p	q	r	$q \vee r$	$[a](q \vee r)$	(1)	$\neg p$	$[a] p$	(2)	$[b] p$	(3)
s_0	1	0	0	0	1	1	0	0	0	0	1
s_1	0	1	1	1	1	1	1	0	0	1	1
s_2	0	0	1	1	0	1	1	1	1	1	1

Interpretace programů a, b viz výše

Jiný příklad: $AF = \{p, q\}$, $AP = \{a, b\}$, $W = \{1, 2, 3, 4, 5, 6\}$

	p	q	a	b	$[a] p$	$\langle a\rangle p$	$[b] p$	$[a] q$	$a; b$	$a \cup b$	a^*	$p?$
1	1	1	2, 3	1	0	1	1	1	2, 3	1—3	1—6	1
2	1	1	4. 5, 6	2	0	1	1	0	—	2, 4—6	1—6	2
3	0	1	—	3	1	0	0	1	—	3	3	—
4	1	0	1, 4	—	1	1	1	0	1	1, 4	1—6	4
5	1	0	3	—	0	0	1	1	3	3	3, 5	5
6	0	0	6	—	0	0	1	0	—	6	6	—

základní interpretace — interpretace některých složených formulí — interpretace některých složených programů

Všimněte si, jak je možno vyjádřit klasické programové konstrukty:

IF A THEN α ELSE β vyjádříme jako $(A?; \alpha) \cup ((\neg A)?; \beta)$

WHILE A DO α vyjádříme jako $((A?; \alpha)^*; \neg A?)$

První program převádí každý stav s, v němž platí výrok A, do týchž stavů, do kterých je stav s převeden programem α, a každý stav t, v němž platí výrok $\neg A$, převede tento program do týchž stavů, do kterých je stav t převeden programem β.

Druhý program převede stav s do stavu t, jestliže buďto ve stavu s neplatí A a přitom $t = s$, nebo existuje posloupnost $t_1, \ldots, t_n$ stavů takových, že v $t_1, \ldots, t_{n-1}$ platí A, v t_n neplatí A a $s \xrightarrow{\alpha} t_1 \xrightarrow{\alpha} \ldots \xrightarrow{\alpha} t_{n-1} \xrightarrow{\alpha} t_n = t$.

Všimněte si, že pokud jsou programy α, β deterministické, tj. každý z nich převádí každý stav nejvýše do jednoho stavu, pak i programy „IF A THEN α ELSE β" a „WHILE A DO α" jsou deterministické, přestože obsahují v našem vyjádření nedeterministické složky.

Příklad:

	p a b	$p?$	$\neg p?$	$p?; a$	$\neg p?; b$	if p then a else b	$(p?;a)^*$	while p do a
1	1 2 4	1	—	2	—	2	1, 2, 3	3
2	1 3 1	2	—	3	—	3	2, 3	3
3	0 4 2	—	3	—	2	2	3	3
4	0 — 3	—	4	—	3	3	4	4

Nyní je čas, abychom uvedli důvody pro studium nedeterministických programů.

(1) V dynamické logice jsou islovány primitivní řídící konstrukty, umožňující zapsat elegantně deterministické i nedeterministické řídící konstrukty známých programovacích jazyků. Tak zejména v konstruktech jako „IF THEN ELSE", „WHILE DO" jsou spojeny koncepty testování a výběru resp. testování a iterace. Nedeterministická dynamická logika umožňuje tyto koncepty isolovat a studovat samostatně.

(2) V praktických úvahách o deterministických programech může být výhodné dokázat obecnější tvrzení o nějakých ne nutně deterministických programech. (Např. pozorování, že program „WHILE $x < 0$ DO $x \leftarrow x + 2$" nemění paritu čísla, závisí na rozpoznání, že provedu-li libovolněkrát $x \leftarrow x + 2$, tj. provedu-li $(x \leftarrow y + 2)^*$, parita x se nezmění.)

Uvádějí se ještě další důvody, jako např. vztah k paralelním procesům, vztah k jazykům umělého intelektu (STRIPS, PLANNER); uvedené dva jsou však, zdá se, nejzávažnější.

Fakt, že formule A dynamické logiky je pravdivá ve stavu s interpretace $\boldsymbol{I}$, značíme $\boldsymbol{I} \models A[s]$ nebo stručně jen $s \models A$, pokud je $\boldsymbol{I}$ jasné z kontextu. Formule A dynamické logiky je *pravdivá v interpretaci* $\boldsymbol{I}$, je-li pravdivá v každém jejím stavu. Formule dynamické logiky je *tautologie*, jestliže je pravdivá v každém stavu každé interpretace. Formule A dynamické logiky je *splnitelná*, jestliže existuje interpretace $\boldsymbol{I}$ a stav s tak,

že A je pravdivá v interpretaci $\boldsymbol{I}$ ve stavu s. (Tj. A je nesplnitelná, právě když $\neg A$ je tautologie.)

Platí následující, naprosto ne zřejmý výsledek:

Každá splnitelná formule A je splnitelná v interpretaci mající jen konečný počet stavů; přesněji, je-li A posloupnost symbolů délky n a je-li splnitelná, pak existuje interpretace $\boldsymbol{I}$, jejíž množina stavů má nejvýše mohutnost 2^n, a stav s takový, že A je pravdivá v interpretaci $\boldsymbol{I}$ ve stavu s.

Z toho bezprostředně vyplývá, že také pro dynamický výrokový počet je problém tautologičnosti algoritmicky řešitelný.

Uvedeme také příklad korektní a úplné axiomatizace dynamického výrokového počtu.

Schémata axiomů

(1) Všechny tautologie klasického výrokového počtu (přesněji: formule, které vzniknou z tautologií klasického výrokového počtu dosazením formulí dynamického výrokového počtu za výrokové proměnné).

(2) $[\alpha](A \to B) \to ([\alpha] A \to [\alpha] B)$

(3) $[\alpha \cup \beta] A \equiv ([\alpha] A \,\&\, [\beta] A)$

(4) $[\alpha; \beta] A \equiv [\alpha][\beta] A$

(5) $[\alpha^*] A \to A$

(6) $[\alpha^*] A \to [\alpha] A$

(7) $[\alpha^*] A \to [\alpha^*][\alpha^*] A$

(8) $(A \,\&\, [\alpha^*](A \to [\alpha] A)) \to [\alpha^*] A$

(9) $[A?] B \equiv (A \to B)$

(10) $\langle\alpha\rangle A \equiv \neg[\alpha] \neg A$

Pravidla dedukce: (1) *Modus ponens* (z A a $A \to B$ bezprostředně odvoď B), (2) *Generalizace*: Z A bezprostředně odvoď $[\alpha] A$.

Čtenář může pro kterýkoli axiom ověřit, že je tautologií a že platí-li v nějaké interpretaci předpoklady některého z našich pravidel (tj. platí-li ve všech stavech té interpretace), pak tam platí i závěr. Tj. axiomatizace je *korektní*. Důkaz tvrzení o *úplnosti* je ovšem značně náročný.

Tvrzení o úplnosti je velmi důležité, ale má pojistný charakter: na nic důležitého jsme v axiomatizaci nezapomněli. Praktický význam axiomatizace spočívá patrně především v tom, že nám umožňuje výslovně si uvědomit schémata některých pravdivých výroků o programech a schémata korektních důkazových kroků pro tvrzení o programech.

Jako jednoduchý příklad ukažme skicu důkazu formule, pravící, že po ukončení programu „WHILE A DO α" platí $\neg A$. Postupně provádíme ekvivalentní úpravy:

$[(A?; \alpha)^*; (\neg A?)]\ \neg A$
$[(A?; \alpha)^*]\,[(\neg A?)]\ \neg A$ dle axiomu (4)
$[A?; \alpha)^*]\,(\neg A \to \neg A)$ dle axiomu (9)

Stačí tedy dokázat poslední formuli. Ale $\neg A \to \neg A$ je tautologie klasického výrokového počtu, tedy axiom; z něho dostaneme formuli $[(A?; \alpha)^*]\,(\neg A \to \neg A)$ podle pravidla generalizace. (Zde vidíme intuitivní význam pravidla generalizace: platí-li některá formule ve všech stavech, pak platí speciálně ve všech stavech, do nichž se může dostat nějaký program.)

Podobně může čtenář dokázat formuli

$$A \to ([\alpha]\, B \equiv [(A?; \alpha) \cup (\neg A?; \beta)]\, B)\,,$$

která praví, že platí-li A, pak po provedení α bude platit B právě tehdy, když po provedení „IF A THEN α ELSE β" bude platit B. (Tj. platí-li A, chová se „IF A THEN α ELSE β" jako α.) *Návod*: ekvivalentně upravte $[(A?; \alpha) \cup (\neg A?; \beta)]\, B$ podle axiomů (3), (4), (9).

Shrňme:

Formule dynamického výrokového počtu jsou tvořeny z atomických formulí pomocí logických spojek, modalit a programů. Programy jsou tvořeny z atomických programů, programových spojek, testu a z formulí. Každá interpretace je dána množinou abstraktních stavů a interpretací atomických formulí a atomických programů závislou na stavech. Problém tautologičnosti pro formule dynamického výrokového počtu je algoritmicky řešitelný. Existuje jednoduchá axiomatizace dynamického výrokového počtu, která je korektní a úplná.

---

4. VÝPOČTOVÉ STROMY

Sémantika definovaná v předchozím odstavci umožňuje definovat pravdivost formulí týkajících se začátku a konce výpočtu. Tak např. jsou-li A, B formule a α program, pak

(1) Formule $A \to [\alpha]\, B$ vyjadřuje parciální korektnost programu vzhledem k počáteční podmínce A a koncové podmínce B ($A\{\alpha\}\, B$ v Hoarově značení): znamená, že pokud v nějakém stavu s platí A a t je libovolný stav takový, že $s \xrightarrow{\alpha} t$, pak v t platí B.

(2) Formule $A \to \langle\alpha\rangle\, B$ říká, že pro každý stav s, v němž platí A, existuje aspoň jeden stav t takový, že $s \xrightarrow{\alpha} t$ a v t platí B. (Pokud je program α deterministický, je tím vyjádřena totální korektnost.)

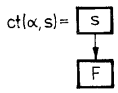
(3) Formule $A \to (\langle\alpha\rangle\, true \,\&\, [\alpha]\, B)$ říká, že α každý stav s, v němž platí A, převádí alespoň do jednoho stavu t, a že pro každý stav t takový, že $s \xrightarrow{\alpha} t$ a v s platí A, je B splněno v t. (Zde a dále je „*true*": zkratka za libovolnou formuli pravdivou v každém stavu každé interpretace (tj. tautologii), např. $A \vee \neg A$.) To pro deterministické programy opět dává totální korektnost; pro obecné programy je však třeba hlubší analýza, ke které nyní přistupujeme.

V tomto odstavci budeme vyšetřovat bohatší sémantiku výpočtových stromů, která umožňuje vyjadřovat i vlastnosti programů týkající se celého výpočtu.

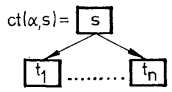
Výpočtový strom příslušný k programu α a stavu s zachycuje všechny možné výpočty, běhy programu α začínající ve stavu s. Nechť AF, AP, W jsou množiny atomických formulí, atomických programů a stavů a nechť I je interpretace AF a AP ve W. Výpočtový strom $\boldsymbol{ct}(\alpha, s)$ příslušný k programu α a stavu s je strom, jehož vrcholy jsou označeny buďto stavy z W nebo symbolem F (*failure*-selhání a jehož kořen je označen stavem s, definovaný rekursivně takto:

1. Je-li $\alpha \in AP$ nebo α je $A?$, kde A je formule, pak

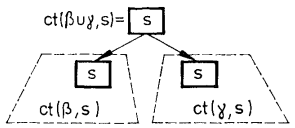
   (a) neexistuje-li stav t tak, že $s \xrightarrow{\alpha} t$, pak



   (b) jsou-li $\{t_1, \ldots, t_n\}$ $n \geqq 1$ všechny stavy, pro které $s \xrightarrow{\alpha} t_i$, pak

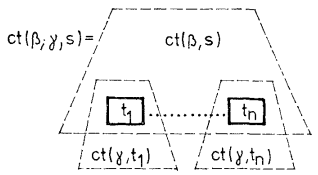


2. Je-li α program $\beta \cup \gamma$ pak $\boldsymbol{ct}(\alpha, s)$ vznikne z $\boldsymbol{ct}(\alpha, s)$ a $\boldsymbol{ct}(\gamma, s)$ přidáním dalšího vrcholu (kořene) označeného s a hran z tohoto kořene do kořene $\boldsymbol{ct}(\beta, s)$ a $\boldsymbol{ct}(\gamma, s)$:

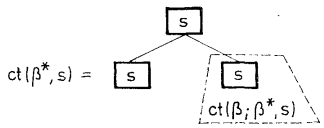


3. Je-li α program $\beta; \gamma$, pak $\boldsymbol{ct}(\alpha, s)$ vznikne z $\boldsymbol{ct}(\beta, s)$ nahrazením každého listu (vrcholu bez následníka) označeného stavem (např. $t \in W$) stromem $\boldsymbol{ct}(\gamma, t)$.

Přitom hrana, která vedla do listu označeného t, se vede do kořene stromu $\boldsymbol{ct}(\gamma, t)$:

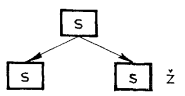


4. Je-li α program β^*, pak $\boldsymbol{ct}(\alpha, s)$ je strom splňující rovnici:

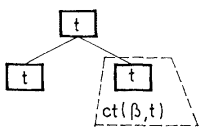


Tento strom může být nekonečný.

Poznámka: Uvedenou rovnici je nutno chápat jako návod k postupnému sestrojování stromu $\boldsymbol{ct}(\beta^*, s)$ (za předpokladu, že jsou definovány stromy $\boldsymbol{ct}(\beta, t)$ pro všechna t). Podrobněji bychom to mohli formulovat takto: Nechť $\boldsymbol{ct}_{\check{z}}(\beta, t)$ vznikne ze stromu $\boldsymbol{ct}(\beta, t)$ tak, že ke každému listu označenému stavem připíšeme symbol $\check{z}$ (živý). Buď $\boldsymbol{T}_0$ strom



a pro každé n buď $\boldsymbol{T}_{n+1}$ strom, který vznikne z $\boldsymbol{T}_n$ tak, že každý živý list tvaru $\boxed{t}^{\check{z}}$ nahradíme stromem



Nyní jsou dvě možnosti: buďto existuje n takové, že $\boldsymbol{T}_n$ už neobsahuje živé listy, pak $\boldsymbol{T}_n = \boldsymbol{T}_{n+1}$ a definujeme $\boldsymbol{ct}(\beta^*, s) = \boldsymbol{T}_n$. Nebo každý strom $\boldsymbol{T}_n$ obsahuje živé

listy; pak je-li $\boldsymbol{T}'_n$ strom $\boldsymbol{T}_n$ s odstraněnými symboly $\check{z}$, jest $\boldsymbol{T}'_n \subseteq \boldsymbol{T}'_{n+1}$ a definujeme $\boldsymbol{ct}(\beta^*, s)$ jako sjednocení stromů $\boldsymbol{T}'_n$.

Uvedeme příklad speciálního dynamického výrokového počtu a výpočtových stromů v něm. Podobným způsobem budeme později definovat i dynamický predikátový počet.

Nechť $N = \{0, 1, 2, \ldots\}$ je množina přirozených čísel,

$V = \{x, y, z, \ldots\}$ je množina proměnných (konečná nebo nekonečná),

AF, AP jsou množiny výrazů tvaru

$$AF = \{x = y,\ x = y + z,\ x = y \,.\, z,\ x < y \mid x, y, z \in V \cup N\}$$
$$AP = \{x \leftarrow y,\ x \leftarrow y + z,\ x \leftarrow y \,.\, z,\ x \leftarrow y \div z \mid x \in V,\ y, z \in V \cup N\}\,.$$

Stav je určen přiřazením hodnot všem proměnným. Pro jednoduchost dalších definic budeme předpokládat, že stavy jsou definovány i na N a to identicky, tj.

$$W = \{s : V \cup N \to N \mid n \in N \Rightarrow s(n) = n\}\,.$$

Definujeme dále interpretaci $\boldsymbol{I}$ předpisem

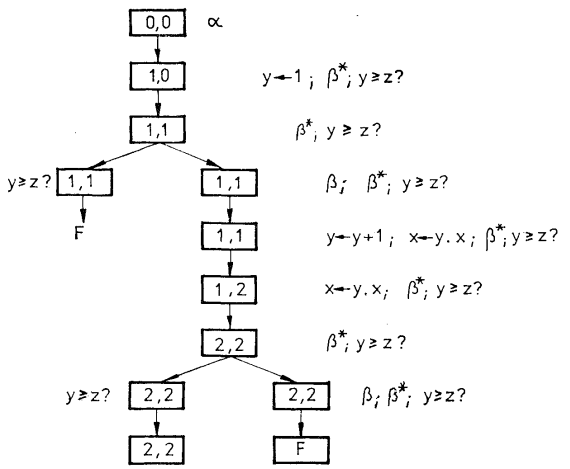
$$\boldsymbol{I}_{x=y}(s) = 1\,, \quad \text{právě když} \quad s(x) = s(y)$$
$$\boldsymbol{I}_{x=y+z}(s) = 1\,, \quad \text{právě když} \quad s(x) = s(y) + s(z)$$
$$\boldsymbol{I}_{x=y.z}(s) = 1\,, \quad \text{právě když} \quad s(x) = s(y) \,.\, s(z)$$
$$\boldsymbol{I}_{x<y}(s) = 1\,, \quad \text{právě když} \quad s(x) < s(y)$$
$$\boldsymbol{I}_{x\leftarrow y} = \{(s, t) \mid t(x) = s(y);\ w \in V - \{x\} \Rightarrow t(w) = s(w)\}$$
$$\boldsymbol{I}_{x\leftarrow y+z} = \{(s, t) \mid t(x) = s(y) + s(z);\ w \in V - \{x\} \Rightarrow t(w) = s(w)\}$$
$$\boldsymbol{I}_{x\leftarrow y.z} = \{(s, t) \mid t(x) = s(y) \,.\, s(z);\ w \in V - \{x\} \Rightarrow t(w) = s(w)\}$$
$$\boldsymbol{I}_{x\leftarrow y\div z} = \{(s, t) \mid t(x) = s(y) \div s(z);\ w \in V - \{x\} \Rightarrow t(w) = s(w)\}$$

(Definujeme $a \div b = a - b$ pokud $a \geqq b$, jinak $a \div b = 0$.)

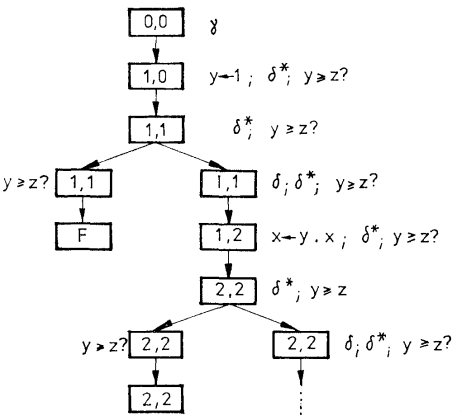
Takový systém lze považovat za jednoduchý programovací jazyk. Na příklad program α, který v proměnné x vypočítává faktoriál čísla z lze zapsat takto:

$$x \leftarrow 1;\ y \leftarrow 1;\ \beta^*;\ y \geqq z?\,, \quad \text{kde } \beta \text{ je program} \quad y < z?;\ y \leftarrow y + 1;\ x \leftarrow y \,.\, x\,.$$

Nechť s je libovolný stav, který přiřazuje proměnným x, b, z hodnoty 0, 0, 2 (na hodnotách ostatních proměnných nám nezáleží). Uvedeme výpočtový strom programu α ve stavu s. U každého vrcholu je stav příslušný tomuto vrcholu označen hodnotami proměnných x, y. Vedle každého vrcholu (kromě listů) je napsán program, jehož výpočtový strom má za kořen daný vrchol.



V dalším případě je výpočtový strom nekonečný. Nechť γ je program $x \leftarrow 1$; $y \leftarrow 1$; δ^*; $y \geqq z$?, kde δ je program $y \leftarrow y + 1$; $x \leftarrow y \,.\, x$. Výpočtový strom γ ve stavu s (viz předchozí příklad) nyní obsahuje listy tvaru $a!$, a pro $a \geqq 2$ a je tedy nekonečný.



Snadno se nahlédne, že program α převádí stav s do stavu t, právě když $\boldsymbol{ct}(\alpha, s)$ obsahuje list označený t.

Představme si nyní procesor (překladač), který provádí programy dynamické logiky. Zadáním pro takový procesor je program α, a stav $s \in W$, jeho cílem je nalézt nějaký stav t tak, že $s \xrightarrow{\alpha} t$. To je tedy ekvivalentní úloze nalézt v $\boldsymbol{ct}(\alpha, s)$ list označený stavem (tj. nikoliv F).

Při prohledávání výpočtového stromu se procesor buď nezastaví, nebo se zastaví, ale nenajde list označený stavem – v těchto případech výpočet skončí neúspěšně – nebo se zastaví a najde list označený stavem.

Jsou možné různé *strategie výpočtu* a v závislosti na zvolené strategii lze definovat totální správnost programu.

Program α je *totálně správný vzhledem k formuli A a dané strategii*, jestliže v každém stavu s je splněna formule $\langle\alpha\rangle \mathit{true} \,\&\, [\alpha] A$, a jestliže výpočtová strategie najde ke každému stavu s nějaký stav t tak, že $s \xrightarrow{\alpha} t$. Uvedeme některé výpočtové strategie. U každé strategie uvedeme třídu programů, pro které *zaručuje totální správnost*, tj. pro každý program α z této třídy platí: jakmile je v každém stavu splněna formule $\langle\alpha\rangle \mathit{true}$, pak ke každému stavu s najde strategie nějaký stav t takový, že $s \xrightarrow{\alpha} t$. (Pokud strategie připouští nedeterministická rozhodování, pak slovem „najde" rozumíme „najde při jakémkoli přípustném rozhodování".)

D (*Depth search*)

Začni na kořeni stromu a z každého vrcholu pokračuj k jeho následníku. Má-li vrchol více následníků, vyber nedeterministicky některý z nich. Nemá-li vrchol následníka, tj. je-li to list, vydej jeho označení jako výsledek. Tato strategie zaručuje totální správnost jenom těch programů, jejichž výpočtové stromy jsou konečné a neobsahují selhání.

DI (*Depth search with backtracking*)

Postupuj stejně jako v případě *D*, je-li však list označen F, vrať se zpátky na nejbližší větvení a postupuj z něj jinou cestou. Byly-li již všechny cesty vyzkoušeny, vrať se na předcházející větvení. Vydej výsledek F, neexistuje-li předchozí větvení. Tato strategie zaručuje totální správnost programů, jejichž výpočtové stromy jsou konečné, je však náročná na paměť, protože procesor si musí pamatovat všechny stavy, které navštívil.

*DI*1

Postupuj stejně jako v případě *DI*, ale backtracking prováděj jen k nejbližšímu větvení. Tato strategie zaručuje totální správnost deterministických programů, které obsahují *, $\cup$ jen v konstruktech IF THEN ELSE, WHILE DO, a jejichž atomické programy převádějí každý stav do nejvýše jednoho stavu. Processor si nemusí pamatovat celou historii výpočtu, ale jen současný stav.

B (*Bredth search*)

Začni na kořeni stromu a postupuj z každého vrcholu k jeho následníku. Má-li vrchol více následníků, sleduj všechny cesty současně. Najdeš-li list, vydej jeho označení jako výsledek. Tato strategie zaručuje totální správnost programů, jejichž výpočtové stromy neobsahují selhání.

BT (*Bredth search with backtracking*)

Postupuj stejně jako v případě *B*, ale ignoruj listy označené *F*. Tato strategie zaručuje totální správnost všech programů, ale je stejně jako předchozí náročná na paměť a čas.

V příkladu programu na výpočet faktoriálu budou úspěšné strategie *DT*, *DT*1, *BT*, v druhém příkladu (program γ) budou úspěšné strategie *BT*.

Úvahy o výpočtových stromech a o totální správnosti lze provádět přímo v syntaktickém systému dynamické logiky. Za tím účelem rozšíříme definici formulí dynamického výrokového počtu tak, že přidáme následující pravidla tvoření formulí: Nechť α je program, A je formule. Pak

1. FL (α) (α neobsahuje selhání) je formule.
   Formule FL (α) je splněna ve stavu s interpretace $\boldsymbol{I}$, jestliže $\boldsymbol{ct}(\alpha, s)$ neobsahuje list označený F.
2. LOOP (α) (α neobsahuje cyklus) je formule.
   $s \models$ LOOP (α), jestliže $\boldsymbol{ct}(\alpha, s)$ je konečný.
3. MAINT (α, A) (α udržuje A) je formule.
   $s \models$ MAINT (α, A), jestliže pro každý stav t, který se vyskytuje jako označení v $\boldsymbol{ct}(\alpha, s)$, platí $t \models A$.
4. PRES (α, A) (α zachovává A) je formule.
   $s \models$ PRES (α, A), jestliže pro každé dva po sobě jdoucí vrcholy v $\boldsymbol{ct}(\alpha, s)$ označené stavy t_1, t_2 platí: jestliže $t_1 \models A$, pak $t_2 \models A$.

Stejně jako v případě dynamického výrokového počtu definujeme i zde, že formule je pravdivá v interpretaci $\boldsymbol{I}$, je-li pravdivá v každém jejím stavu $s \in W$, a formule je tatologie, je-li pravdivá v každé interpretaci.

Některé fragmenty tohoto rozšířeného dynamického výrokového počtu lze axiomatizovat. Například rozšíření o formule typu PRES a MAINT lze úplně a korektně axiomatizovat tak, že k axiomatickému systému dynamického výrokového počtu přidáme následující axiomy a dedukční pravidlo.

(11) $A \,\&\, \text{PRES}(\alpha, A) \,\&\, \text{PRES}(\alpha, A \to B) \to \text{PRES}(\alpha, B)$

(12) $A \equiv B \,\&\, \text{PRES}(\alpha, A \equiv B) \,\&\, \text{PRES}(\alpha, A) \to \text{PRES}(\alpha, B)$

(13) $A \,\&\, \text{PRES}(\alpha, A) \to \text{PRES}(\alpha, \neg A)$

(14) $\text{PRES}(\alpha, A) \,\&\, \text{PRES}(\alpha, B) \to \text{PRES}(\alpha, A \,\&\, B)$

(15) PRES (α, A) & PRES $(\alpha, B) \rightarrow$ PRES $(\alpha, A \vee B)$

(16) PRES $(A?, B)$

(17) PRES $(\alpha \cup \beta, A) \equiv$ PRES (α, A) & PRES (β, A)

(18) PRES $(\alpha, \beta, A) \equiv$ PRES (α, A) & $[\alpha]$ PRES (β, A)

(19) PRES $(\alpha^*, A) \equiv [\alpha^*]$ PRES (α, A)

(20) MAINT $(\alpha, A) \equiv A$ & PRES (α, A)

Dedukční pravidlo:

(3) Z A odvoď PRES (α, A)

Existuje více strategií provádění programů dynamické logiky. Dynamický výrokový počet lze obohatit o formule, které vypovídají o celých výpočtech a tyto formule lze interpretovat v sémantice výpočetních stromů.

---

5. KLASICKÝ PREDIKÁTOVÝ POČET

Klasický predikátový počet se liší od výrokového, zhruba řečeno, tím, že nevychází z nedělitelných a nestrukturovaných atomických formulí, nýbrž že umožňuje popsat strukturu atomických formulí jako jistých schemat výroků o nějakých objektech, jejich vlastnostech a vztazích mezi nimi. Nám půjde vposledu o dynamický predikátový počet, který umožňuje také popsat strukturu atomických programů jako jistých přiřazení (assignment). Nicméně popíšeme nejprve klasický predikátový počet; dynamický predikátový počet je jeho přirozeným rozšířením.

Přirozenou formalizací systému objektů s nějakými vlastnostmi a vztahy je matematický pojem *relační struktury*. Relační struktura je dána (1) neprázdnou množinou M (universum neboli nosič struktury), (2) nějakými relacemi a operacemi na M, (3) nějakými vytčenými (význačnými) prvky množiny M. Tak např. relační struktura *přirozených čísel* je struktura $\boldsymbol{N} = \langle N, =, \leqq, +, \cdot, 0, 1\rangle$ s relací rovnosti a „menší-rovno", operacemi sčítání a násobení a vytčenými prvky 0, 1. Má stejný *typ* jako struktura

$$\boldsymbol{Re} = \langle Re, =, \leqq, +, \cdot, 0, 1\rangle$$

reálných čísel, tj. má stejný počet relací a operací stejné odpovídající četnosti a stejný počet vytčených prvků. Každá grupa je struktura $\langle G, =, \cdot, 1\rangle$ s jednou binární operací násobení a s jedním vytčeným prvkem (jednotkou) splňující známé předpoklady. Podobně každá Booleova algebra může být chápána jako relační struktura s rovností, a operacemi průseku, spojení a komplementu.

Predikátový jazyk určitého typu obsahuje symbolická jména relací, operací a vytčených prvků libovolné struktury zvoleného typu. Symbolická jména relací

jsou *predikátové symboly* (stručně *predikáty*), jména operací jsou *funkční symboly*, jména vytčených prvků jsou *individuové konstanty*. Každý predikát a funkční symbol má jednoznačně stanovenou četnost – přirozené číslo, udávající četnost relace (operace), kterou může pojmenovávat.

Predikátový jazyk dále obsahuje *individuové proměnné*

$$x_0, x_1, x_2, \ldots$$

logické spojky (&, $\vee$, $\rightarrow$, $\equiv$, $\neg$) a dva *kvantifikátory*: $\forall$ (universální, čteme: „pro každé") a $\exists$ (existenční, čteme: „existuje").

Např. predikátový jazyk aritmetiky obsahuje binární predikáty $\approx$, $\lessapprox$, binární funkční symboly $\dot{+}$, $\sim$ a konstanty $\mathbf{0}$, $\mathbf{1}$. Těmito symboly mohou být pojmenovány příslušné složky nejen struktury přirozených čísel, ale každé struktury téhož typu. Zvolíme-li strukturu správného typu jako prostředek k interpretaci predikátového počtu, je tím dán význam všech predikátů, funkčních symbolů a konstant a také je tím dán obor proměnnosti proměnných: chápeme je jako proměnné pro libovolný prvek nosiče zvolené struktury.

Z proměnných, konstant a funkčních symbolů (a závorek) můžeme zřejmým způsobem tvořit symbolické složené názvy prvků, zvané termy. Např. v jazyce aritmetiky jsou následující výrazy termy: $(x_0 \dot{+} x_1) \sim (\mathbf{1} \dot{+} \mathbf{1})$, $\mathbf{1} \dot{+} \mathbf{1} \dot{+} \mathbf{1} \dot{+} \mathbf{1}$, $x_0 \sim x_0 \sim x_0$ atd. Pokud takový term neobsahuje žádné proměnné, je při dané struktuře jednoznačně určena jeho hodnota; např. hodnota termu $\mathbf{1} \dot{+} \mathbf{1} \dot{+} \mathbf{1} \dot{+} \mathbf{1}$ ve struktuře přirozených čísel je 4. Pokud term obsahuje proměnné, závisí jeho hodnota na tom, jaké hodnoty přiřadíme jednotlivým proměnným; pokud bychom si představovali, že proměnným odpovídají nějaká paměťová místa, mohli bychom říci, že hodnota závisí na *stavu* těchto paměťových míst. To zní velmi podobně jako závislost pravdivostní hodnoty formule dynamického výrokového počtu na abstraktním stavu a tato podobnost není náhodná. Máme-li zvolenu nějakou relační strukturu $\boldsymbol{M}$, pak stavem této struktury (přesněji: stavem příslušného predikátového jazyka) nazveme každé zobrazení s, které každé proměnné x_i přiřadí nějaký prvek s_i nosiče struktury $\boldsymbol{M}$. Je intuitivně jasné (a lze lehko formálně definovat), co je to hodnota termu v nějakém stavu; např. je-li $s_0 = 7$ a $s_1 = 9$, pak hodnota termu $(x_0 \dot{+} x_1) \sim (\mathbf{1} \dot{+} \mathbf{1})$ v tomto stavu je jistě 32. (Stále mlčky předpokládáme, že struktura, o níž jde, je struktura $\boldsymbol{N}$ přirozených čísel.)

Hodnotu termu t ve stavu s struktury $\boldsymbol{M}$ značíme $(t)_{\boldsymbol{M},s}$.

Atomická formule predikátového jazyka sestává z predikátu a tolika termů, kolik udává četnost tohoto predikátu. Tak např. $x_0 \approx x_1$, $x_0 \dot{+} x_1 \lessapprox \mathbf{1} + \mathbf{1}$ jsou atomické formule jazyka aritmetiky.

Buď $\boldsymbol{M}$ struktura a s nějaký její stav. Predikát v atomické formuli pojmenovává nějakou relaci ze struktury $\boldsymbol{M}$ a $\boldsymbol{M}$ spolu se stavem s určuje hodnoty všech termů z naší atomické formule. Můžeme tedy definovat: Atomická formule $P(t_1, \ldots, t_n)$ je *pravdivá* ve struktuře $\boldsymbol{M}$ ve stavu s, jestliže hodnoty termů

$$(t_1)_{M,s}, \ldots, (t_n)_{M,s}$$

jsou v relaci, kterou pojmenovává predikát P.

Z atomických formulí tvoříme formule podle těchto pravidel: (1) Každá atomická formule je formule. (2) Jsou-li A, B formule, pak $A \,\&\, B$, $A \vee B$, $A \to B$, $A \equiv B$, $\neg A$ jsou formule. (3) Je-li A formule, pak pro každou proměnnou x_i jsou $(\forall x_i)\, A$, $(\exists x_i) A$ formule.

Např. v jazyce aritmetiky máme tyto formule:

$$(\forall x_1)\,(x_1 \perp \mathbf{0} \approx x_1)\,, \quad \neg(x_1 \approx \mathbf{0}) \to (\exists x_2)\,(x_1 \approx x_2 \perp \mathbf{1})$$

apod. Slovní čtení takových formulí nebude jistě čtenáři dělat obtíže.

Čtenář vidí, že odlišujeme např. relaci rovnosti $(=)$ a predikát, který je jejím jménem $(\approx)$; podobně odlišujeme relaci $\leqq$ a její jméno $\lessapprox$, operace $+$, $.$ a jejich jména $\perp$, $\sim$ atd. Později pro jednoduchost od tohoto rozlišení upustíme a budeme např. psát $x_1 = x_2 + x_3$ místo $x_1 \approx x_2 \perp x_3$ atd.

Intuitivně je jasné, co znamená, že jedna formule je *podformulí* jiné (např. $x_1 \approx \approx x_2 \perp \mathbf{1}$ je podformulí druhé z výše uvedených formulí). Definujeme, jak závisí pravdivostní hodnota formule (pro danou strukturu $\boldsymbol{M}$ a její stav s) na hodnotách jejích podformulí.

Pro atomické formule jsme již definici podali (pochopitelně „formule je pravdivá" znamená totéž, co „pravdivostní hodnota formule je 1"). Ve shodě s dřívějším značením bychom mohli pravdivostní hodnotu formule A v $\boldsymbol{M}$, s značit $\boldsymbol{M}_A(s)$; místo $\boldsymbol{M}_A(s) = 1$ je zvykem psát též $\boldsymbol{M} \models A[s]$ (a číst: v $\boldsymbol{M}$ platí A ve stavu s). Je jasné, jak je definována pravdivostní hodnota formulí vznikajících z jednodušších pomocí logických spojek (např. $\boldsymbol{M}_{A\&B}(s) = \min(\boldsymbol{M}_A(s), \boldsymbol{M}_B(s))$, tj. $\boldsymbol{M} \models (A \,\&\, B)\,[s]$, právě když $\boldsymbol{M} \models A[s]$ a $\boldsymbol{M} \models B[s]$ atd.).

Zbývá definovat hodnotu formule $(\forall x_i)\, A$ a formule $(\exists x_i)\, A$. Formule $(\forall x_i)\, A$ říká, že A platí pro každé x_i, tedy je přirozené definovat, že $(\forall x_i)\, A$ platí (je pravdivé) v $\boldsymbol{M}$, s, jestliže A platí v $\boldsymbol{M}$ v každém stavu, který se liší od s nejvýše hodnotou proměnné x_i. Podobně definujeme: $\boldsymbol{M} \models (\exists x_i)\, A[s]$, jestliže existuje stav s' lišící se od s nejvýše v hodnotě proměnné x_i a takový, že $\boldsymbol{M} \models A[s']$.

Všimněte si, že v této poslední definici vůbec nezáleží na tom, jakou hodnotu stav s přiřazuje proměnné x_i (neboť tato proměnná je ve formuli $(\forall x_i)\, A$ a ve formuli $(\exists x_i)\, A$ *vázána* kvantifikátorem); zato však pravdivostní hodnota obou těchto formulí nezávisí jen na stavu s, ale na všech stavech, které jsou se stavem s svázány uvedenou podmínkou. (Všimněte si podobnosti s definicí $\boldsymbol{I}_{[\alpha]A}(s)$.) Je-li speciálně nějaká formule uzavřená, tj. každý výskyt každé proměnné x_i je výskyt v nějaké podformuli tvaru $(\forall x_i)\, A$ nebo $(\exists x_i)\, A$, pak její pravdivostní hodnota je stejná pro všechny stavy a můžeme prostě říci, že je v dané struktuře pravdivá nebo nepravdivá.

Formule je *tautologie*, jestliže je pravdivá v každé struktuře v každém jejím stavu. Např. formule $(\exists x_1)\,(\forall x_2)\, P(x_1, x_2) \to (\forall x_2)\,(\exists x_1)\, P(x_1, x_2)$ je tautologie.

Interpretací predikátového jazyka *danou strukturou* $\boldsymbol{M}$ rozumíme zobrazení,

které každé formuli A a každému stavu s struktury $\boldsymbol{M}$ přiřadí hodnotu 1 nebo 0 podle toho, zda je A ve stavu s pravdivá nebo nepravdivá. Tuto hodnotu můžeme značit $\boldsymbol{M}_A(s)$ (viz výše).

Popíšeme jednu z mnoha možných axiomatizací predikátového počtu.

Axiomy:

(1) Všechny tautologie výrokového počtu (přesněji: formule, které vzniknou z tautologií výrokového počtu dosazením formulí predikátového počtu za výrokové proměnné).

(2) $(\forall x_i)\, A(x_i) \rightarrow A(t)$,

kde $A(x_i)$ je formule predikátového počtu, $A(t)$ vzniká z $A(x_i)$ dosazením t za všechny volné výskyty proměnné x_i a je splněna jistá syntaktická podmínka „čistoty proměnných" (např. stačí, že žádná proměnná z t se nevyskytuje v $A(x_i)$).

(3) $(\forall x_i)\,(A \rightarrow B(x_i)) \rightarrow (A \rightarrow (\forall x_i)\, B(x_i))$,

pokud x_i nemá žádný volný výskyt v A.

(4) $(\exists x_i)\, A \equiv \neg(\forall x_i)\, \neg A$.

Odvozovací pravidla:

Modus ponens (viz výše)

Generalizace: Z A bezprostředně odvoď $(\forall x_i)\, A$.

Tato axiomatizace predikátového počtu má tyto vlastnosti: (1) je *korektní*, tj. každý axiom je tautologií a dedukční pravidla zachovávají tautologičnost, (2) je *úplná*, tj. formule je tautologií, právě když je dokazatelná, (3) je *efektivní*, tj. lze algoritmicky rozhodnout o každé formuli, zda je axiom, a o každé posloupnosti formulí, zda je důkazem, ale (4) obecně nelze algoritmicky rozhodovat, zda formule je či není tautologií, tj. zda je či není dokazatatelná. To je klasický výsledek o *algoritmické nerozhodnutelnosti* klasického predikátového počtu. (Problém tautologií je algoritmicky rozhodnutelný jen při omezení na jisté velmi chudé jazyky, např. jazyky obsahující jen unární (jednočetné) predikáty a neobsahující žádné funkční symboly).

Klasický predikátový počet je velmi výhodným prostředkem pro studium axiomatických teorií; tím se však zde nebudeme zabývat.

Shrňme:

Formule klasického predikátového počtu jsou tvořeny z atomických formulí pomocí logických spojek a kvantifikátorů. Atomické formule jsou tvořeny z predikátů a termů. Každá interpretace je dána jistou relační strukturou, která určuje interpretaci všech funkčních symbolů a konstant, a též určuje množinu všech stavů; každý stav určuje spolu s onou strukturou interpretaci všech výrazů. Problém tautologičnosti je algoritmicky neřešitelný (až na jisté „singulární případy"), existuje však efektivní korektní a úplná axiomatizace.

---

6. DYNAMICKÝ PREDIKÁTOVÝ POČET

Dynamický predikátový počet rozšiřuje klasický predikátový počet podobně jako dynamický výrokový počet rozšiřuje klasický výrokový počet. Formulím klasického predikátového počtu budeme od této chvíle říkat *klasické formule*; definujeme, co jsou programy a formule. Systém, který zde popíšeme, se někdy nazývá regulární dynamický predikátový počet.

Interpretace výrazů dynamické predikátové logiky (DPL) bude vždy dána nějakou relační strukturou, přičemž proměnné budeme nyní velmi přirozeně chápat jako jména nějakých paměťových míst. *Stav* struktury, který byl definován jako libovolné zobrazení množiny všech proměnných do nosiče struktury, lze chápat jako momentální stav paměti; interpretovat program jako relaci na těchto stavech je pak velmi přirozené (program převádí jeden stav do druhého).

Dynamický predikátový jazyk sestává z predikátů, funkčních symbolů, individuových konstant, individuových proměnných, logických spojek a kvantifikátorů (jako klasický případ) a dále z programových spojek, znaku přiřazení $\leftarrow$ a testu ?.

Termy a *atomické formule* se definují jako v klasickém případě; *atomický program* je výraz tvaru $x_i \leftarrow e$, kde x_i je proměnná a e je term. Formule a programy se tvoří podle těchto pravidel:

(1) Každá atomická formule je formule a každý atomický program je program.

(2) Je-li A klasická formule, pak $A?$ je program (test).

(3) Jsou-li α, β programy, pak $(\alpha; \beta)$, $(\alpha \cup \beta)$, α^* jsou programy.

(4) Jsou-li A, B formule, x_i proměnná, α program, pak

$$(A \,\&\, B)\,, \quad (A \vee B)\,, \quad (A \to B)\,, \quad (A \equiv B)\,, \quad \neg A\,,$$

$$(\forall x_i)\, A\,, \quad (\exists x_i)\, A\,, \quad [\alpha]\, A\,, \quad \langle \alpha \rangle\, A$$

jsou formule.

Buď $\boldsymbol{M}$ relační struktura (odpovídajícího typu, tj. interpretující predikáty, funkční symboly a individuové konstanty jazyka). Pro každý stav s a každou formuli A bude definována *pravdivostní hodnota* formule A ve stavu s a pro každý program α bude definována *množina stavů, do nichž α převádí stav s*. Skoro všechny části definice můžeme převzít z klasického predikátového počtu nebo z dynamického výrokového počtu; např. pro každou klasickou formuli je $\boldsymbol{M}_s(A)$ definováno. Víme také, jak interpretovat $A?$ a známe-li interpretaci programů α, β, víme, jak interpretovat programy z nich složené.

Zbývá vlastně definovat jen význam atomických programů; vše ostatní lze převzít. Je-li α přiřazení $x_i \leftarrow e$, pak tento program převádí stav s_1 do stavu s_2 (symbolicky: $s_1 \xrightarrow{\alpha} s_2$ nebo $\langle s_1, s_2 \rangle \in \boldsymbol{M}_\alpha$), jestliže

(i) $s_2(x_i) = s_1(e)$,

(ii) pro $j \neq i$ jest $s_2(x_j) = s_1(x_j)$,

tj. hodnota x_i v s_2 je stejná jako hodnota termu e ve stavu s_1 jinak se s_1 a s_2 neliší.

Tím je pro každou formuli A a každý program α definována jeho interpretace; zejména můžeme říci, co je to *tautologie*, tj. formule pravdivá v každé interpretaci v každém jejím stavu.

Každému programu α můžeme přiřadit množinu $var(\alpha)$ jeho vytčených proměnných: jsou to proměnné, které se vyskytují bezprostředně vlevo od symbolu $\leftarrow$. Tak např. je-li α program

$$(x = z \,\&\, y = u)?\,;\quad (x \leftarrow f(x) \cup y \leftarrow f(y))\,,$$

pak $var(\alpha) = \{x, y\}$. Tento program se skládá z testu $(x = z \,\&\, y = u)?$ a následujícího nedeterministického sjednocení: $x \leftarrow f(x) \cup y \leftarrow f(y)$. Proměnné z, u slouží jako vstupní proměnné: je-li ve stavu s splněna podmínka $x = z \,\&\, y = u$, pak celý program α převede tento stav jednak do stavu s_1, do něhož je s převeden programem $x \leftarrow f(x)$, jednak do stavu s_2, do něhož je s převeden programem $y \leftarrow f(y)$. Přitom s_1 se liší od s pouze hodnotou proměnné x (a má stejnou hodnotu proměnné y) a stav s_2 se liší od s pouze hodnotou proměnné y (a má stejnou hodnotu proměnné x). Tedy po provedení programu α bude platit $x = z$ nebo $y = u$. Pokud ve stavu s neplatí $x = z \,\&\, y = u$, pak jej program α nepřevádí nikam. Tedy celkem: Formule

$$[(x = z \,\&\, y = u)?;\ x \leftarrow f(x) \cup y \leftarrow f(y))]\,(x = z \vee y = u)$$

je tautologie.

Pomocí výsledků matematické logiky lze snadno dokázat, že množina tautologií DPL nejen že není rozhodnutelná, ale *není ani efektivně axiomatizovatelná*, tj. neexistuje algoritmicky rozhodnutelný systém axiomů a dedukčních pravidel, který by byl korektní a úplný. To je v podstatě věci. Lze však podat axiomatizaci, v níž je neefektivnost dobře „lokalizována", a která umožňuje dobré formální dokazování. Tu nyní popíšeme. Přidržíme se přitom jistého speciálního případu, který pro naše potřeby postačí.

Aritmetickou relační strukturou rozumějme strukturu

$$\boldsymbol{M} = \langle N, =, \leqq, +, \cdot, 0, 1, \ldots\rangle\,,$$

jejíž nosič je množina přirozených čísel a která mimo jiné obsahuje relace rovnosti a „menší-rovno", operace sčítání a násobení přirozených čísel a vytčené prvky 0, 1. (Dále může obsahovat libovolné další relace, operace a vytčené prvky.) Formule je *aritmetická tautologie*, platí-li ve všech stavech každé aritmetické struktury. (Obecněji bychom mohli předpokládat, že přirozená čísla tvoří pouze definovatelnou část nosiče a že na nich je k dispozici jejich aritmetika. Zůstaneme však při našem speciálním pojmu.) Omezení na aritmetické struktury znamená, že se soustřeďujeme na

jazyky, které umožňují explicitně mluvit o aritmetice přirozených čísel a že část našeho jazyka, která mluví o přirozených číslech, je interpretována standardně. Pomocí jazyka aritmetiky můžeme definovat řadu dalších obvyklých funkcí, např. $x \dot{-} 1$ $(0 \dot{-} 1 = 0, (x + 1) \dot{-} 1 = x)$. Pak např. formule $\langle (x \leftarrow x \dot{-} 1)^* > (x = 0)$ je aritmetická tautologie; říká, volně řečeno, že ať je hodnota x jakákoli, pak opakovaným snižováním o 1 dospějeme po jistém počtu iterací k hodnotě 0.

Uvedeme nyní slíbenou axiomatizaci DPL a formulujeme její vlastnosti.

Axiomy:

(1) Všechny tautologie klasického výrokového počtu.

(2) Všechny klasické formule, které jsou aritmetickými tautologiemi.

(3) $[x \leftarrow e] A(x) \equiv A(e)$, $\leftarrow$ kde A je klasická formule.

(4) $[B?] A \equiv (B \rightarrow A)$ pro libovolné formule A, B.

(5) $[\alpha; B] A \equiv [\alpha][\beta] A$.

(6) $[\alpha \cup \beta] A \equiv ([\alpha] A \,\&\, [\beta] A)$.

(7) $\langle \alpha \rangle A \equiv \neg [\alpha] \neg A$.

Dedukční pravidla:

(1) $$\frac{A,\ A \rightarrow B}{B} \quad \text{(modus ponens)},$$

(2) $$\frac{A \rightarrow [\alpha] A}{A \rightarrow [\alpha^*] A},$$

(3) $$\frac{A \rightarrow B}{[\alpha] A \rightarrow [\beta] B},$$

(4) $$\frac{A(n+1) \rightarrow \langle \alpha \rangle A(n)}{A(n) \rightarrow \langle \alpha^* \rangle A(0)}$$ pokud A je klasická formule a $n \notin var(\alpha)$.

Vlastnosti této axiomatizace:

(1) Je korektní (každý axiom je aritmetická tautologie a každé pravidlo zachovává aritmetickou tautologičnost).

(2) Je úplná, tedy formule je dokazatelná, právě když je aritmetickou tautologií (platí ve všech stavech všech aritmetických struktur).

(3) Kromě axiomů skupiny (2) je efektivní (algoritmicky rozhodnutelná); je však algoritmicky nerozhodnutelné, zda nějaká formule aritmetiky patří pod (2) nebo ne. (O velké řadě formulí víme, že patří pod (2), např. všechny formule, které byly dokázány ve formální (Peanově) aritmetice.)

(4) Problém tautologičnosti je ovšem pro DPL algoritmicky neřešitelný.

(5) Každá formule A dynamické predikátové logiky je aritmeticky ekvivalentní nějaké klasické formuli, tj. ke každé formuli A existuje klasická formule A' taková, že ekvivalence $A \equiv A'$ je aritmetickou tautologií.

Podotkněme, že „vada krásy" spočívající v neefektivnosti axiomů (2) je podstatná; neexistuje žádná efektivní axiomatizace aritmetických tautologií. Výhoda uvedené axiomatizace je v tom, že neefektivnost je zde dobře „lokalizována" a díky vlastnosti (3) můžeme podané axiomatizace dobře užívat ke konkrétnímu dokazování.

Na základě uvedených axiomů a dedukčních pravidel lze odvodit řadu dalších odvozených dedukčních pravidel, např.

$$\frac{A \to B}{\langle\alpha\rangle A \to \langle\alpha\rangle B}, \tag{5}$$

$$\frac{C \to A, \quad A \to [\alpha] A, \quad A \to B}{C \to [\alpha^*] B}, \tag{6}$$

$$\frac{C \to (\exists n) A(n), \quad A(n+1) \to \langle\alpha\rangle A(n), \quad A(0) \to B}{C \to \langle\alpha^*\rangle B} \tag{7}$$

(v posledním pravidlu se předpokládá, že A je klasická formule, přičemž $n \notin var(\alpha)$).

Dále lze v DPL dokázat analogie axiomů (3), (4) klasického predikátového počtu (vlastnosti kvantifikátorů), tj. pro libovolné formule A, B, C dynamického predikátového počtu, C neobsahující proměnnou x, lze dokázat v DPL:

$$(\forall x)(C \to B) \to (C \to (\forall x) B), \tag{8}$$

$$(\exists x) A \equiv \neg(\forall x) \neg A. \tag{9}$$

Důkaz není zřejmý, užívá vlastnosti (5) nahoře. Čtenář může, chce-li, přidat k axiomům DPL formule (8), (9) a k dedukčním pravidlům pravidla (5), (6), (7), jakož i pravidlo generalizace z klasického predikátového počtu; tím se nemění třída dokazatelných formulí.

Uveďme ještě, jak by bylo možno DPL obohatit.

(1) Je možno přidat atomické programy tvaru $x_i \leftarrow random$; program $x_i \leftarrow random$ převede stav s do libovolného stavu t, který se od s liší nejvýše v hodnotě proměnné x_i. Všimněte si, že formule $[x_i \leftarrow random] A$ má pak týž význam $(\forall x_i) A$ a že formule $\langle x_i \leftarrow random\rangle A$ má týž význam jako formule $(\exists x_i) A$.

(2) Obecněji, je-li $B(\boldsymbol{Z}, \boldsymbol{W})$ klasická formule o $2n$ volných proměnných $\boldsymbol{Z} = z_1, \ldots, z_n$, $\boldsymbol{W} = w_1, \ldots, w_n$, je možno zavést program

$$\boldsymbol{Z} \leftarrow random\,(\boldsymbol{W} : B(\boldsymbol{Z}, \boldsymbol{W})),$$

který pracuje se stavem s takto: zjistí hodnoty $s(z_1), \ldots, s(z_n)$ (které stav s přiřazuje proměnným $z_1, \ldots, z_n$); zvolí libovolně hodnoty $w_1, \ldots, w_n$, které jsou k $s(z_1), \ldots, s(z_n)$ ve vztahu B; výsledný stav t splňuje $t(z_i) = w_i$ a jinak se t shoduje se stavem s.

Tedy $z_1 \leftarrow random$ je totéž co $z_1 \leftarrow random\,(w_1 : w_1 = w_1)$, program $z \leftarrow$ $\leftarrow random\,(w : w > z)$ změní stav s tak, že hodnotu proměnné z libovolným způsobem *zvětší* (a jinak nic nezmění) atd. Tento typ programů bude později velmi užitečný.

Shrňme:

V dynamickém predikátovém počtu tvoříme formule z atomických formulí a libovolných programů pomocí logických spojek, kvantifikátorů a modalit; programy tvoříme z atomických programů (přiřazení) a libovolných klasických formulí pomocí programových spojek a testů. Interpretace je dána relační strukturou a jejími stavy podobně jako interpretace formulí v klasickém predikátovém počtu a interpretace programů v dynamickém výrokovém počtu. Při omezení na aritmetické relační struktury získáme pojem aritmetických tautologií; pro ně máme k dispozici axiomatizaci, která je korektní a úplná, a ačkoli není (z principiálních důvodů) efektivní (nelze algoritmicky rozhodnout, zda daná formule je axiom), je v ní neefektivnost redukována pouze na nemožnost rozhodnout, zda formule klasické aritmetiky je pravdivá či ne. Tato axiomatizace skýtá dobrý důkazový aparát, protože o spoustě formulí víme, že jsou axiomy.

---

7. DYNAMICKÝ PREDIKÁTOVÝ POČET S DIVERGENCÍ

Tak jako u dynamického výrokového počtu, lze i u dynamického predikátového počtu vyšetřovat bohatší sémantiku výpočtových stromů (definice z odstavce 4 zůstává nezměněna) a formule, které o výpočtových stromech vypovídají. V tomto odstavci uvedeme axiomatizaci systému dynamické logiky obohaceného o formule LOOP(α) s významem „program α neobsahuje nekonečný cyklus". Z formálních důvodů budeme uvažovat její modifikaci $[\alpha]^+ A$, která je ekvivalentní formuli $[\alpha] A$ & & LOOP(α). Syntaktickou definici formulí a jejich sémantiku tedy rozšiřujeme o následující bod.

Je-li α program a A formule, pak $[\alpha]^+ A$ je formule. Formule $[\alpha]^+ A$ je splněna ve stavu s ($s \models [\alpha]^+ A$), jestliže $s \models [\alpha] A$ a výpočtový strom $\boldsymbol{ct}(\alpha, s)$ je konečný.

Zavedeme-li označení $\langle\alpha\rangle^+ A \equiv \neg[\alpha]^+ \neg A$, pak $s \models \langle\alpha\rangle^+ A$, jestliže buď $s \models \langle\alpha\rangle A$ nebo $\boldsymbol{ct}(\alpha, s)$ je nekonečný. Dále je vidět, že $s \models [\alpha]^+ \boldsymbol{true}$, právě když $\boldsymbol{ct}(\alpha, s)$ je konečný, a $s \models \langle\alpha\rangle^+ \mathit{false}$, právě když $\boldsymbol{ct}(\alpha, s)$ je nekonečný.

Odpovídající axiomatický systém pro dynamický predikátový počet s divergencí vznikne tak, že k axiomatickému systému dynamického predikátového počtu přidáme následující axiomy a dedukční pravidla.

Axiomy:

1) $[\alpha]^+ A \equiv [\alpha] A \,\&\, [\alpha]^+ \mathit{true}$
2) $[x \leftarrow t]^+ \mathit{true}$
3) $[B?]^+ \mathit{true}$
4) $[\alpha; \beta] \mathit{true} \equiv [\alpha]^+ [\beta]^+ \mathit{true}$
5) $[\alpha \cup \beta]^+ \mathit{true} \equiv [\alpha]^+ \mathit{true} \,\&\, [\beta]^+ \mathit{true}$

Dedukční pravidla:

1) $$\frac{A(n+1) \to [\alpha]^+ A(n), \neg A(0)}{A(n) \to [\alpha^*]^+ \mathit{true}},$$

kde A je klasická formule a $n \notin \mathit{var}(\alpha)$

2) $$\frac{A \to \langle\alpha\rangle^+ A}{A \to \langle\alpha^*\rangle^+ \mathit{false}}$$

O této axiomatizaci se nyní dá opět ukázat, že je korektní a úplná.

8. DYNAMICKÝ PREDIKÁTOVÝ POČET S REKURSÍ

V kapitole 6 jsme ukázali, že dynamický predikátový počet je rozšířením predikátového počtu o určité prostředky, které umožňují vytvářet formule popisující vlastnosti jisté třídy programů. K zavedení predikátového počtu s rekursí vedla potřeba studovat vlastnosti programů, umožňujících rekursivní volání „procedur", což prostředky dynamického predikátového počtu není možné. Jazyk dynamické logiky s rekursí se liší od jazyka dynamického predikátového počtu pouze tím, že operátor iterace je zde nahrazen operátorem rekurse. Bude tedy možné najít jisté analogie.

Pro dynamický predikátový počet s rekursí budeme v dalším textu používat zkratku CFDL vycházející z anglického pojmenování (context free dynamic logic) zavedeného Harelem.

Jazyk CFDL sestává z predikátových a funkčních symbolů, individuových konstant a proměnných, z logických spojek a kvantifikátorů a dále z jedné *programové proměnné* (označované X) programových spojek, znaku přiřazení $\leftarrow$, testu $?$ a z operátoru rekurse označovaného μ.

Jak jsme již zdůraznili, liší se CFDL od DPL v zásadě jen rozdílnou strukturou programů (přesněji programových termů). *Množina programů* v CFDL je vytvořena podle těchto pravidel:

(1) Je-li x individuová proměnná, e term a A formule klasického predikátového počtu, pak každé přiřazení $x \leftarrow e$, test $A?$ a programová proměnná X jsou programové termy.

(2) Jsou-li τ, τ_1, τ_2 programové termy a X programová proměnná, pak $(\tau_1; \tau_2)$, $(\tau_1 \cup \tau_2)$ a $\mu X(\tau)$ je programový term. Programový term $\mu X(\tau)$ lze intuitivně chápat jako program, podle něhož počítáme tak, že kdykoliv narazíme při výpočtu na výskyt programové proměnné X, substituujeme na místo „označené" X celý program τ. V rekursivních programech je tedy prostředek umožňující iteraci nahrazen schopností programů volat samy sebe. Operátor rekurse umožňuje tedy jistým způsobem vázat programové proměnné. Řekneme, že výskyt programové proměnné v termu τ je vázaný, je-li obsažen v podtermu tvaru $\mu X(\tau_1)$; v opačném případě je výskyt programové proměnné X v τ volný. Programový term neobsahující žádné volné výskyty programové proměnné nazveme *uzavřeným*.

Označíme CP množinu *rekursivních programů* (dále jen programů). Rekursivní programy jsou právě uzavřené programové termy. Označíme-li nyní $\tau(X)$ term obsahující volnou proměnnou X, pak budeme pro term $\mu X(\tau(X))$ užívat v dalším testu zkratku τ^*. Tato zkratka je ve formulích daleko přehlednější a samotný zápis je pohodlný. Je navíc jasné, že nemůže dojít k záměně se značkou pro iteraci v DPL, neboť rekursivní term obsahuje programovou proměnnou. Programová proměnná nemá však jiný význam, než že „označuje místo", kde bude program volat sám sebe.

Množina formulí v CFDL je vytvářena pomocí těchto pravidel:

(1) Každá atomická formule predikátového počtu je formulí v CFDL.

(2) Jsou-li A, B formule, x individuová proměnná a τ program, pak $(A \& B)$, $(A \vee B)$, $(A \to B)$, $(A \equiv B)$, $\neg A$, $(\forall x) A$, $(\exists x) A$, $[\tau] A$, $\langle \tau \rangle A$ jsou formule.

Bude-li nyní $\boldsymbol{M}$ relační struktura interpretující predikátové a funkční symboly a individuové konstanty, pak sémantika CFDL je definována stejně jako pro DPL s tím rozdílem, že zbývá nahradit sémantiku operátoru iterace sémantikou rekurse. Ukažme tedy jak lze interpretovat term τ^*. Označíme-li $\tau(X)$ programový term obsahující právě jednu volnou proměnnou X, můžeme definovat $\tau(\alpha)$ jako zkratku označující výsledek substituce programu α za všechny volné výskyty X v τ. Definujeme-li dále

$$\tau^0(\alpha) = \alpha \quad \text{a} \quad \tau^{i+1}(\alpha) = \tau(\tau^i(\alpha))$$

pak definujeme sémantiku programu τ^* opět jako množinu dvojic stavů takto:

$$\langle s_1, s_2 \rangle \in \boldsymbol{M}_{\tau^*}, \text{ právě když } \langle s_1, s_2 \rangle \in \bigcup_{i=0}^{\infty} \boldsymbol{M}_{\tau^i(false?)} .$$

Přitom program „$false$?" je ovšem prázdný, tj. nepřevádí žádný stav do žádného stavu. Můžeme tedy říci, že program τ^* převádí stav s_1 do stavu s_2 tehdy a jen tehdy, existuje-li přirozené číslo n takové, že program $\tau^n(false?)$ převádí stav s_1 do stavu s_2.

Příklad:

Uvažujme program

$$\alpha : z \leftarrow x;\ ((z = 0?;\ y \leftarrow 1) \cup (z \neq 0;\ z \leftarrow z - 1;\ X;\ z \leftarrow z + 1;\ y \leftarrow y \,.\, z))^*$$

jenž je tvaru $z \leftarrow x;\ \tau^*$. Uveďme příklad programu $\tau^3(false?)$:

$$\begin{aligned}
&((z = 0;\ \ y \leftarrow 1) \cup (z \neq 0?\ z \leftarrow z - 1\ ; \\
&\qquad ((z = 0?;\ y \leftarrow 1) \cup (z \neq 0?;\ z \leftarrow z - 1\ ; \\
&\qquad\qquad ((z = 0?;\ y \leftarrow 1) \cup (z = 0?;\ z \leftarrow z - 1\ ; \\
&\qquad\qquad\qquad false?\ ; \\
&\qquad\qquad z \leftarrow z + 1;\ y \leftarrow y \,.\, z))\ ; \\
&\qquad z \leftarrow z + 1;\ y \leftarrow y \,.\, z))\ ; \\
&z \leftarrow z + 1;\ y \leftarrow y \,.\, z))\,.
\end{aligned}$$

Není obtížné si všimnout, že v každém stavu s_1, v němž je proměnné x přiřazena hodnota 2, bude pravdivé

$$\langle z \leftarrow x;\ \tau^3(false?)\rangle\,(y = 2)\,.$$

Pro každé $n < 3$ bude $\boldsymbol{M} \models ([z \leftarrow x;\ \tau^n\, false?)]\ false\,[s_1]$. Podobně se ukáže, že interpretací programu $z \leftarrow x;\ \tau^n(false?)$ je relace

$$\{\langle s, t\rangle : s(x) \leqq n - 1 \,\&\, t(y) = (s(x))!\}\,,$$

takže program α počítá faktoriál na přirozených číslech.

Všimněme si nyní, že programy $\alpha \in$ CF mohou změnit nejvýše hodnoty proměnných obsažených ve $var\,(\alpha)$, tj. každý program realizuje nějakou relaci na hodnotách proměnných ve $var\,(\alpha)$ (vztah mezi hodnotami „vstupními" a „výstupními". K tomu, abychom takové relace dokázali popisovat a studovat, je vhodné jazyk CFDL poněkud rozšířit. Rozšíření provedeme obohacením množiny programů CF, přičemž ostatní složky CFDL ponecháme beze změny. Množinu programů CF′ definujeme takto:

(1) Je-li $B(\boldsymbol{Z}, \boldsymbol{W})$ klasická formule predikátového počtu obsahující navzájem různé volné proměnné $z_1, z_2, \ldots, z_n$ a $w_1, w_2, \ldots, w_n$ (přičemž píšeme $\boldsymbol{Z}$ místo $z_1, \ldots, z_n$ a $\boldsymbol{W}$ místo $w_1, \ldots, w_n$), pak $\boldsymbol{Z} \leftarrow random\,(\boldsymbol{W} : B(\boldsymbol{Z}, \boldsymbol{W}))$ je programový term v CF′.

(2) Každé přiřazení $x \leftarrow e$ a test $A?$ je programový term v CF′.

(3) Každý jednoduchý uzavřený term je v CF′.

(4) Jsou-li $\alpha, \beta \in$ CF′, pak $(\alpha; \beta)$ a $(\alpha \cup \beta)$ jsou v CF′. Neformální popis sémantiky $\boldsymbol{Z} \leftarrow random\,(\boldsymbol{W} : B(\boldsymbol{Z}, \boldsymbol{W}))$ byl uveden v závěru kapitoly pojednávající o DPL. V dalším textu budeme používat zkratku $\boldsymbol{Z} \leftarrow rand\,\boldsymbol{B}$, nebo $\boldsymbol{Z} \leftarrow rand\,B(\boldsymbol{Z}, \boldsymbol{W})$. Formálně lze sémantiku tohoto „příkazu" popsat následovně.

$\langle s_1, s_2\rangle \in \boldsymbol{M}_{\boldsymbol{Z} \leftarrow rand\boldsymbol{B}}$ právě tehdy, když

(1) s_1 a s_2 se shodují v hodnotách proměnných různých od $\boldsymbol{Z}$

(2) zaměníme-li v s_1 hodnoty proměnných $\boldsymbol{W}$ hodnotami, které mají proměnné $\boldsymbol{Z}$ v s_2, pak vznikne stav $s_1(\boldsymbol{W}/\boldsymbol{Z}_{s_2})$, pro nějž platí $\boldsymbol{M} \models B(\boldsymbol{Z}, \boldsymbol{W})\,[s_1(\boldsymbol{W}/\boldsymbol{Z}_{s_2})]$.

Podmínku (2) lze jinými slovy vyjádřit takto: zaměníme-li v s_2 hodnoty proměnných W hodnotami, které mají proměnné Z v s_1, vznikne stav $s_2(W/Z_{s_1})$, pro nějž $M \models B(W, Z)[s_2 W/Z_{s_1})]$. Máme-li nyní program α, $var(\alpha) = Z$, pak pro každé $\langle s_1, s_2 \rangle \in M_\alpha$ platí podmínka (1). Podmínka (2) vyjadřuje skutečnost, že mezi hodnotami Z v s_1 a hodnotami Z v s_2 platí vztah B.

Nyní popíšeme některé důležité vlastnosti CF′DL. Stejně jako u DPL bude neefektivnost axiomatického systému „dobře lokalizována" aritmetickými tautologiemi. Bude-li M libovolná aritmetická relační struktura, pak ke každé formuli A z CF′DL existuje klasická formule A' predikátového počtu taková, že $M \models A \equiv A'$.

Další zajímavou vlastností programů DPL a CF′DL je, že množina programů DPL je podmnožinou množiny programů CF′DL. Operátor iterace můžeme totiž přepsat v CF′DL následovně. Je-li M nějaká relační struktura, pak platí

$$M_{\alpha^*} = M_{(true?\cup(\alpha;X))^*} = M_{true?\cup(X;\alpha)^*}.$$

Všimněte si, že z vlastností příkazu $Z \leftarrow rand\ A$ vyplývá, že formule

$$W = Z \rightarrow [Z \leftarrow rand\ A(Z, W)]\ A(W, Z)$$

je tautologií. (Pozor na pořadí proměnných!) Toho využijeme k vyslovení dvojice tvrzení o CF′DL, která se bezprostředně promítnou do axiomatického systému. Tato tvrzení svazují pomocí příkazu $Z \leftarrow rand\ B$ vlastnosti rekursivních programů a relací vyjádřitelných pomocí klasických formulí predikátového počtu.

Pro každou relační strukturu M, pro každý program $\alpha \in$ CF′ a formuli A predikátového počtu platí:

(1) $M \models (W = Z \rightarrow [\alpha]\ A(W, Z)$ právě tehdy, když $M_\alpha \subseteq M_{Z \leftarrow rand A(Z,W)}$.

(2) $M \models (A(Z, W) \rightarrow \langle\alpha\rangle\ (Z = W)$ právě tehdy, když $M_{Z \leftarrow rand A(Z,W)} \subseteq M_\alpha$.

Intuitivní význam tohoto tvrzení lze popsat takto:

(1) Po provedení programu α platí vždy formule A, vyjadřující vztah mezi vstupními a výstupními hodnotami proměnných ve $var(\alpha)$, právě když každý výpočet podle programu α lze vyjádřit náhodným přiřazením podle formule A, tj. výsledek práce programu můžeme „uhádnout" formulí predikátového počtu.

(2) Naopak existuje výpočet podle α, který lze „předpovědět" formulí A vyjadřující vztah vstupních a výstupních hodnot proměnných ve $var(\alpha)$, právě když každé náhodné přiřazení podle formule A je vyčíslitelné programem α.

Další dvě tvrzení, která uvedeme, potvrzují oprávněnost vyjádření sémantiky rekursivních programů technikou minimálního pevného bodu. Totiž pro každou relační strukturu M, pro každé dva programy α, α' z CF′ a každý programový term $\tau(X)$ obsahující X jako volnou proměnnou platí

(1) $M_\alpha \subseteq M_{\alpha'} \rightarrow M_{\tau(\alpha)} \subseteq M_{\tau(\alpha')}$ (monotonie),

(2) $M_{\tau(\alpha)} \subseteq M_\alpha \rightarrow M_{\tau^*} \subseteq M_\alpha$ (existence horní meze),

(3) $M_{\tau(\tau)^*} = M_{\tau^*}$ (pevný bod).

Nyní můžeme popsat axiomatický systém pro CF'DL.

Axiomy:

Axiomy (1)–(7) jsou totožné s axiomy DPL.

(8) $[Z \leftarrow rand\ A]\ B \equiv (\forall W')\,(A(Z, W') \rightarrow B_Z^{W'}$,

kde A, B jsou klasické formule predikátového počtu a zkratka $B_Z^{W'}$ označuje výsledek substituce W' za Z ve formuli B.

(9) $(A \rightarrow [\tau^*]\ B) \rightarrow (A \,\&\, C \rightarrow [\tau^*]\,(B \,\&\, C))$,

jestliže C neobsahuje žádné proměnné z $var\,(\tau)$.

Dedukční pravidla:

(I) $\dfrac{A,\ A \rightarrow B}{B}$ (modus ponens)

(II) $\dfrac{A \rightarrow B}{[\alpha]\ A \rightarrow [\alpha]\ B}$

(III) $\dfrac{W = Z \rightarrow [\tau(Z \leftarrow rand\ A(Z, W))]\ A(W, Z)}{W = Z \rightarrow [\tau^*]\ A(W, Z)}$,

kde $Z = var\,(\tau)$

(IV) $\dfrac{A(n+1, Z, W) \rightarrow \langle \tau(Z \leftarrow rand\ A(n, Z, W))\rangle\,(Z = W),\quad \neg A(0, Z, W)}{A(n, Z, W) \rightarrow \langle \tau^* \rangle\,(Z = W)}$,

kde A je klasická formule predikátového počtu, $Z = var\,(\tau)$ a $n \notin var\,(\tau)$. V dalším textu bude velmi užitečné odvozené dedukční pravidlo

(III') $\dfrac{W = Z \rightarrow [\tau(Z \leftarrow rand\ A(Z, W))]\ A(W, Z),\quad B \rightarrow [Z \leftarrow rand\ A(Z, W)]\ C}{B \rightarrow [\tau^*]\ C}$

kde A, B, C jsou formule predikátového počtu a $Z = var\,(\tau)$. Pro rozšířený dynamický predikátový počet s rekursí CF'DL platí samozřejmě, že axiomatický systém je korektní. Platí také věta o úplnosti, tedy:

A je aritmetická tautologie tehdy a jen tehdy, když A je dokazatelná.

Shrneme-li vlastnosti této axiomatizace, pak platí analogicky jako pro DPL:

---

(1) Axiomatizace je korektní, úplná a kromě axiomů skupiny (2) je efektivní.

(2) Axiomatizace je algoritmicky nerozhodnutelná.

(3) Problém tautologičnosti pro CF'DL je algoritmicky neřešitelný.

(4) Každá formule A dynamického predikátového počtu s rekursí je aritmeticky ekvivalentní nějaké klasické formuli predikátového počtu.

(5) Dynamický predikátový počet s rekursí CF'DL je rozšířením DPL v tom smyslu, že operátor iterace je vyjádřitelný pomocí operátoru rekurse.

Možné zobecnění:

Mnohý čtenář se jistě v průběhu četby této kapitoly pohoršoval nad tím, že široký problém rekursivních programů je zde omezen pouze na studium programů s rekursivním voláním sebe sama. Obecně je však možno připustit operátor rekurse ve tvaru $\mu_j X_1 \dots X_n(\tau_1, \dots, \tau_n)$. Lze opět axiomatický systém analogickým způsobem jako systém pro CF'DL. (Podrobnosti viz [2].)

9. DYNAMICKÝ PREDIKÁTOVÝ POČET S PŘIŘAZOVÁNÍM POLÍM

V další variantě dynamické logiky jsou kromě individuových proměnných také pole proměnných (obdobně jako u programovacích jazyků) a přiřazovací příkaz může měnit hodnoty v těchto polích uložené. Formálně lze pole zavést pomocí logiky druhého řádu, ve které jsou kromě individuových proměnných $x, y, z, \dots$ také proměnné pro funkce $X, Y, Z, \dots$ s danou četností. Pojem stavu dané relační struktury $\boldsymbol{M}$ je nutno zobecnit takto: Stav je zobrazení, které (1) každé individuové proměnné x přiřadí nějaký prvek a nosiče M struktury $\boldsymbol{M}$ a (2) každé funkční proměnné X četnosti k přiřadí operaci četnosti k na M, tj. zobrazení množiny $\underbrace{M \times M \times \dots \times M}_{k\text{-krát}}$ do M. Funkční proměnné se také mohou objevovat v termech a tedy i ve formulích, a lze je kvantifikovat: Je-li A formule a X funkční proměnná, pak $(\forall X)\, A$ je formule. Definice programů se rozšiřuje o přiřazování funkčním proměnným. Je-li X n-ární funkční proměnná a $t, t_1, \dots, t_n$ termy, pak $X(t_1, \dots, t_n) \leftarrow t$ je program. Sémantika tohoto programu je obdobná jako u přiřazování individuovým proměnným. Program $X(t_1, \dots, t_n) \leftarrow t$ převádí stav s do jediného stavu $\bar{s}$, který vznikne z s tak, že hodnota proměnné X v souřadnicích $(t_1)_s, \dots, (t_n)_s$ se změní na $(t)_s$ a všechno ostatní zůstává stejné. ($(t)_s$ značí hodnotu termu t ve stavu s.)

V dynamickém predikátovém počtu s přiřazováním polím nebudeme axiomatizovat množinu všech pravdivých formulí, ale jen množinu pravdivých formulí, ve kterých se nekvantifikují funkční proměnné (tzv. Δ_0^1 formule) Axiomatický systém je zde rozsáhlejší, protože musíme přidat některé logické axiomy druhého řádu. Uvedeme zde z tohoto systému jen axiom, který se přímo týká přiřazování polím.

Axiom

$$[X(t_1, \dots, t_n) \leftarrow t]\, A \equiv (\forall Y)(\mathrm{MODIF}(Y, X, t_1, \dots, t_n, t) \rightarrow A_X^Y)\,.$$

Přitom MODIF $(Y, X, t_1, \ldots, t_n, t)$ je zkratka za

$$Y(t_1, \ldots, t_n) = t \,\&\, (\forall z_1, \ldots, z_n)(z_1 \neq t_1 \vee \ldots \vee z_n \neq t_n \to Y(z_1, \ldots, z_n) = \\ = X(z_1, \ldots, z_n)) .$$

Pro tento axiomatický systém se dá opět ukázat, že je korektní a úplný.

10. PŘÍKLADY

V tomto odstavci uvedeme příklady ilustrující využití prostředků uvedených axiomatických systémů.

I. Dokažme v axiomatickém systému dynamického predikátového počtu správnost programu na výpočet faktoriálu. Buď α program

$$x \leftarrow 1 \,;\; y \leftarrow 0 \,;\; \beta^* \,;\; y = z? ,$$

kde β je program $y < z?; y \leftarrow y + 1; x \leftarrow y \,.\, x$.

Program α je deterministický (snadno vidíme, že β^*; $y = z?$ je program "*while* $y < z$ *do* $(y \leftarrow y + 1; x \leftarrow y \,.\, x)$"), takže stačí dokázat formuli

$$\langle \alpha \rangle \, true \,\&\, [\alpha](x = z!) .$$

Formuli $\langle \alpha \rangle \, true$ lze ekvivalentně upravit takto:

$$\langle x \leftarrow 1 \rangle \langle y \leftarrow 0 \rangle \langle \beta^* \rangle \langle y = z? \rangle \, true ,$$

$$\equiv \langle x \leftarrow 1 \rangle \langle y \leftarrow 0 \rangle \langle \beta^* \rangle (y = z) .$$

Důkazy skončení v DPL vycházejí z Floydovy metody, podle které potřebujeme veličinu, která se po jednom provedení cyklu zmenší, a jejíž nulová hodnota zaručuje skončení programu. V našem případě je to veličina $z - y$ a formule A, na kterou budeme aplikovat odvozené dedukční pravidlo (7), je $n = z - y$. (Tato formule odpovídá intervalu ve Floydově metodě důkazu parciální správnosti.) Dokážeme-li

(1) $(\exists n) A(n)$,

(2) $A(n + 1) \to \langle \beta \rangle A(n)$,

(3) $A(0) \to y = z$,

je tím dokázáno $\langle \beta^* \rangle (y = z)$, tedy (dle odvozeného pravidla (5)) $\langle x \leftarrow 1 \rangle \langle y \leftarrow 0 \rangle \,.\, \langle \beta^* \rangle (y = z)$, tedy $\langle \alpha \rangle \, true$. Přitom důkazy formulí (1) a (3) jsou snadné, zbývá dokázat (2). Ale $A(n + 1) \to \langle \beta \rangle A(n)$ je ekvivalentní formuli

$$(n + 1 = z - y) \to (y < z \,\&\, n = z - y - 1) ,$$

což je klasická formule pravdivá v přirozených číslech. Tím je dokázáno $\langle \alpha \rangle \, true$.

Formuli $[\alpha](x = z!)$ lze ekvivalentně upravit na

$$[x \leftarrow 1][y \leftarrow 0][\beta^*][y = z?](x = z!),$$

což dává

$$[x \leftarrow 1][y \leftarrow 0][\beta^*](y = z \rightarrow x = z!).$$

V tomto případě musíme najít formuli A, která zůstává v platnosti po průchodu cyklem, a na kterou můžeme aplikovat dedukční pravidlo (2). Vezměme za A formuli $x = y!$. Chceme dokázat $A \rightarrow [\beta] A$. Tato formule je postupně ekvivalentní následujícím formulím:

$$x = y! \rightarrow [y < z?][y \leftarrow y + 1][x \leftarrow y \,.\, x](x = y!)$$

$$x = y! \rightarrow [y < z?][y \leftarrow y + 1](y \,.\, x = y!)$$

$$x = y! \rightarrow [y < z?]((y + 1) \,.\, x = (y + 1)!)$$

$$x = y! \rightarrow (y < z \rightarrow (y + 1) \,.\, x = (y + 1)!)$$

a poslední formule je klasická formule pravdivá pro přirozená čísla. Je tedy podle pravidla (2) dokázáno

$$x = y! \rightarrow [\beta^*] x = y!.$$

Podle pravidla (3) je

$$[x \leftarrow 1][y \leftarrow 0](x = y!) \rightarrow [x \leftarrow 1][y \leftarrow 0][\beta^*](x = y!),$$

a tedy $1 = 0! \rightarrow [x \leftarrow 1][y \leftarrow 0][\beta^*](x = y!)$. Protože ovšem platí $x = y! \rightarrow$ $\rightarrow (y = z \rightarrow x = z!)$, je podle pravidla (3) a předchozí formule dokázáno

$$[x \leftarrow 1][y \leftarrow 0][\beta^*] y = z \rightarrow x = z!),$$

a tedy jsme dokázali $[\alpha](x = z!)$.

II. V dalším příkladě ukážeme použití axiomů pro dynamický predikátový počet s divergencí.

Nechť α je opět program z příkladu I. Dokážeme formuli $[\alpha]^+ \mathit{true}$. Obdobně jako u důkazu formule $\langle\alpha\rangle \mathit{true}$ hledáme zde veličinu, která se po průchodu cyklem zmenší o jednotku, avšak nikdy nenabývá hodnoty 0. Takovou veličinou je $z - y + 1$ a formule $A(n)$ pro dedukční pravidlo 1 má tvar $n = z - y + 1 \,\&\, n > 0$.

Z axiomů 2, 3 vyplývá $[\beta]^+ \mathit{true}$, a dále $[\beta] A(n)$ je ekvivalentní

$$y < z \rightarrow n = z - y \,\&\, n > 0.$$

Formule $A(n + 1) \rightarrow [\beta]^+ A(n)$ je tedy ekvivalentní formuli

$$n + 1 = z - y + 1 \,\&\, n + 1 > 0 \rightarrow (y < z \rightarrow n = z - y \,\&\, n > 0),$$

která neobsahuje programy a je pravdivá. Protože platí také $\neg A(0)$, můžeme použít dedukční pravidlo 1 a dostaneme $A(n) \to [\beta^*]^+ \; true$. Z dedukčního pravidla generalizace obdržíme formuli $[x \leftarrow 1;\; y \leftarrow 0;\; \beta^*]^+ \; true$, a protože *true* je ekvivalentní $[y = z?]^+ \; true$, platí i $[\alpha]^+ \; true$.

III. Nyní předvedeme důkaz částečné správnosti programu počítajícího faktoriál prostředky CF′DL. Dokažme formuli

$$[z_2 \leftarrow u;\; \tau^*]\,(z_1 = u!)\,,$$

kde $\tau(X)$ je

$$(z_2 = 0?;\; z_1 \leftarrow 1) \cup (z_2 \neq 0?;\; z_2 \leftarrow z_2 - 1;\; X;\; z_2 \leftarrow z_2 + 1,\; z_1 \leftarrow z_1 \,.\, z_2)\,.$$

Máme tedy dokázat $[z_2 \leftarrow u]\,[\tau^*]\,(z_1 = u!)$.

K tomu stačí dokázat

$$[z_2 \leftarrow u]\,(z_2 = u)\,, \tag{1}$$

$$z_2 = u \to [\tau^*]\,(z_1 = u!)\,. \tag{2}$$

První formule má tvar $[\alpha]\, G$ a druhá $G \to H$; z $G \to H$ dostaneme $[\alpha]\, G \to [\alpha]\, H$ dle pravidla II a tedy dostaneme $[\alpha]\, H$ dle pravidla I (modus ponens). Přitom $[\alpha]\, H$ je dokazovaná formule.

Důkaz formule (1) je triviální, použijeme axiomu (3). Abychom dokázali (2), použijeme odvozeného dedukčního pravidla (III′) takto: Všimneme si, že je $var\,(\tau) = (z_1, z_2)$. Označíme-li $\boldsymbol{Z} = (z_1, z_2,)$, $\boldsymbol{W} = (w_1, w_2)$, $\boldsymbol{W}' = (w_1', w_2')$, a dále

$$B\colon\; z_2 = u\,, \qquad C\colon\; z_1 = u!\,,$$

$$A(\boldsymbol{Z}, \boldsymbol{W})\colon\; w_2 = z_2 \,\&\, w_1 = z_2!\,,$$

pak k důkazu formule $B \to [\tau^*]\, C$ (což je (2)) stačí dokázat

$$B \to [Z \leftarrow rand\; A(\boldsymbol{Z}, \boldsymbol{W})]\, C \tag{3}$$

a

$$\boldsymbol{Z} = \boldsymbol{W} \to [(z_2 = 0?;\; z_1 \leftarrow 1) \cup (z_2 \neq 0?;\; z_2 \leftarrow z_2 - 1\,;$$
$$(\boldsymbol{Z} \leftarrow rand\; A(\boldsymbol{Z}, \boldsymbol{W}));\; z_2 \leftarrow z_2 + 1;\; z_1 \leftarrow z_1 \,.\, z_2)]\; A(\boldsymbol{W}, \boldsymbol{Z})\,. \tag{4}$$

K důkazu (3) použijeme axiom (8), čímž dostaneme

$$z_2 = u \to (\forall w_1', w_2')\,((w_2' = z_2 \,\&\, w_1' = z_2!) \to w_1' = u!)\,,$$

což je axiom skupiny (2).

K důkazu (4) stačí dokázat

$$\boldsymbol{Z} = \boldsymbol{W} \to [z_2 = 0?;\; z_1 \leftarrow 1]\,(z_2 = w_2 \,\&\, z_1 = w_2!)\,, \tag{5}$$

což se lehko redukuje na axiomy skupiny (2), a dále klíčové tvrzení

$$(6) \qquad Z = W \to [z_2 \neq 0?;\ z_2 \leftarrow z_2 - 1]\,[Z \leftarrow rand\, A(Z, W)]$$
$$(z_2 + 1 = w_2 \,\&\, z_1 \,.\, (z_2 + 1) = w_2!)\,.$$

Formule (4) má tvar $K \to [\alpha \cup \beta]\, L$, tedy stačí dokázat $K \to [\alpha]\, L$ a $K \to [\beta]\, L$. Formule $(z_2 = w_2 \,\&\, z_1 = w_2!)$ je $A(W, Z)$; formule $(z_2 + 1 = w_2 \,\&\, z_1 \,.\, (z_2 + 1) = w_2!)$ vznikne z $A(W, Z)$ tak, že nejprve za z_1 dosadíme $z_1 \,.\, z_2$ a pak ve vzniklé formuli za z_2 dosadíme $z_2 + 1$.

Důkaz formule (6) se redukuje na důkaz dvou následujících formulí:

$$(7) \qquad Z = W \to [z_2 \neq 0?;\ z_2 \leftarrow z_2 - 1]\,(w_1 = z_1 \,\&\, w_2 = z_2 + 1)\,,$$

$$(8) \qquad z_1 = w_1 \,\&\, w_2 = z_2 + 1 \to$$
$$\to [Z \leftarrow rand\, A(Z, W)]\,(z_2 + 1 = w_2 \,\&\, z_1 \,.\, (z_2 + 1) = w_2!)\,.$$

(Srv. úvahu o formulích (1) a (2) nahoře.)

Přitom (7) je zřejmé; k důkazu (8) užijeme axiom (8); dostaneme

$$(9) \qquad z_1 = w_1 \,\&\, w_2 = z_2 + 1 \to$$
$$\to (\forall w_1', w_2')\,(A(Z, W') \to w_2 = w_2' + 1 \,\&\, w_1' \,.\, (w_2' + 1) = w_2!)\,.$$

Přitom $A(Z, W')$ jest $w_2' = z_2 \,\&\, w_1' = z_2!$. Formule (9) tedy tvrdí, že je-li $w_2 = z_2 + 1$, $w_2' = z_2$ a $w_1' = z_2!$, pak $w_2 = w_2' + 1$ a $w_2! = w_1' \,.\, (w_2' + 1)$, což je pravda, neboť

$$w_1' \,.\, (w_2 + 1) = z_2!\,(z_2 + 1) = (z_2 + 1)! = w_2!\,.$$

Tedy (9) je dokázáno a tím je dokončen celý důkaz.

IV. Poslední příklad podává důkaz částečné správnosti třídícího algoritmu v axiomatice dynamického predikátového počtu s přiřazováním polím. Algoritmus uspořádává prvních n členů jednorozměrného pole Z. Algoritmus používá proměnnou x jako pointer tak, že $Z(1), \ldots, Z(x)$ jsou vždy neklesající. Dále uloží hodnotu $Z(x + 1)$ do z, a posunuje $Z(x + 1) \leftarrow Z(x)$, $Z(x) \leftarrow Z(x - 1), \ldots$, dokud nenajde místo pro z. Algoritmus lze zapsat následujícím způsobem:

$$\delta : \quad x \leftarrow 1;\ (\alpha; \beta^*; \gamma)^*;\ x = n?,$$

kde

$$\alpha : \quad x < n?;\ y \leftarrow x;\ x \leftarrow x + 1;\ z \leftarrow Z(x)$$

$$\beta : \quad y > 0?;\ Z(y) > z?;\ Z(y + 1) \leftarrow Z(y);\ y \leftarrow y - 1$$

$$\gamma : \quad (y = 0 \vee Z(y) \leqq z)\,?;\ Z(y + 1) \leftarrow z\,.$$

Označme $\mathrm{ORD}(Y, y)$ formuli $(\forall z)(1 \leqq z < y \to Y(z) \leqq Y(z+1))$ a dokažme formuli $[\delta]\,\mathrm{ORD}(Z, n)$.

1. $\mathrm{ORD}(Z, y) \to [\beta]\,\mathrm{ORD}(Z, y+2)$. Platí $[\beta]\,\mathrm{ORD}(Z, y+2) \equiv$ $\equiv [y > 0?;\ Z(y) > z?;\ Z(y+1) \leftarrow Z(y)]\,\mathrm{ORD}(Z, y+1)$ a tato formule plyne z $[Z(y+1) \leftarrow Z(y)]\,\mathrm{ORD}(Z, y+1)$. Zbývá tedy dokázat $\mathrm{ORD}(Z, y) \to (\forall Y)$ $(\mathrm{MODIF}(Y, Z, y+1, Z(y)) \to \mathrm{ORD}(Y, y+1))$. Je-li $1 \leqq v \leqq y-1$, je $Y(v) =$ $= Z(v) \leqq Z(v+1) = Y(v+1)$. Je-li $v = y$, je $Y(v) = Z(y) = Y(y+1) = Y(v+1)$, takže platí $\mathrm{ORD}(Y, y+1)$.

2. $\mathrm{ORD}(Z, x) \,\&\, y \leqq x \to [\beta]\,(\mathrm{ORD}(Z, x) \,\&\, y < x \,\&\, z < Z(y+1))$. Pravá strana této implikace je ekvivalentní formuli

$$y > 0 \,\&\, Z(y) > z \to (\forall Y)\,(\mathrm{MODIF}(Y, Z, y+1, Z(y)) \to (\mathrm{ORD}(Y, x) \,\&$$
$$\&\ y - 1 < x \,\&\, z < Y(y)))\,.$$

Zbývá dokázat $Y(v) \leqq Y(v+1)$ pro $1 \leqq v < x$.

(a) Je-li $v \neq y$, $v \neq y+1$, pak $Y(v) = Z(v) \leqq Z(v+1) = Y(v+1)$.

(b) Je-li $v = y+1$, pak $Y(v) = Z(y) \leqq Z(y+1) \leqq Z(y+2) = Y(v+1)$.

(c) Je-li $v = y$, pak $Y(v) = Z(v) = Y(v+1)$.

Zřejmě také platí $y - 1 < x$ a $z < Z(y) = Y(y)$.

3. $\mathrm{ORD}(Z, x) \,\&\, y < x \,\&\, z < Z(y+1) \to [\gamma]\,\mathrm{ORD}(Z, x)$. Platí

$$[\gamma]\,\mathrm{ORD}(Z, x) \equiv y = 0 \vee Z(y) \leqq$$
$$\leqq z \to (\forall Y)\,(\mathrm{MODIF}(Y, Z, y+1, z) \to \mathrm{ORD}(Y, x))\,.$$

To lze opět dokázat rozborem případů.

4. $\mathrm{ORD}(Z, x) \to [\alpha; \beta; \beta^*; \gamma]\,\mathrm{ORD}(Z, x)$. Platí

$$\mathrm{ORD}(Z, x) \to [\alpha]\,(\mathrm{ORD}(Z, y) \,\&\, y = x - 1) \to$$
$$\to [\alpha; \beta]\,(\mathrm{ORD}(Z, y+2) \,\&\, y = x - 2 \,\&\, z < Z(y+1)) \to \quad (\text{podle } 1)\,,$$
$$\to [\alpha; \beta]\,(\mathrm{ORD}(Z, x) \,\&\, y < x \,\&\, z < Z(y+1)) \to$$
$$\to [\alpha; \beta; \beta^*]\,(\mathrm{ORD}(Z, x) \,\&\, y < x \,\&\, z < Z(y+1)) \to \quad (\text{podle } 2)\,,$$
$$\to [\alpha; \beta; \beta^*; \gamma]\,\mathrm{ORD}(Z, x) \quad (\text{podle } 3)\,.$$

5. $\mathrm{ORD}(Z, x) \to [\alpha; \gamma]\,\mathrm{ORD}(Z, x)$. Platí

$$[\alpha; \gamma]\,\mathrm{ORD}(Z, x) \equiv [\alpha]\,(y = 0 \vee Z(y) \leqq$$
$$\leqq z \to (\forall Y)\,(\mathrm{MODIF}(Y, Z, y+1, z) \to \mathrm{ORD}(Y, x)))\equiv$$
$$\equiv y < n \,\&\, (x = 0 \vee Z(x) \leqq Z(x+1)) \to$$

$$\rightarrow (\forall Y)(\mathrm{MODIF}(Y, Z, x+1, Z(x+1)) \rightarrow \mathrm{ORD}(Y, x+1))$$

$$x < n \,\&\, (x = 0 \vee Z(x) \leqq Z(x+1)) \rightarrow (\forall Y)(Z = Y \rightarrow \mathrm{ORD}(Y, x+1)),$$

což plyne z $\mathrm{ORD}(Z, x)$.

Podle 4. a 5. platí $\mathrm{ORD}(Z, x) \rightarrow [\alpha; \beta^*; \gamma]\,\mathrm{ORD}(Z, x)$, tedy $\mathrm{ORD}(Z, x) \rightarrow$ $\rightarrow [(\alpha; \beta^*; \gamma)^*]\,\mathrm{ORD}(Z, x)$, a protože $[x \leftarrow 1]\,\mathrm{ORD}(Z, x)$, platí i $[x \leftarrow 1;$ $(\alpha; \beta^*; \gamma)^*]\,\mathrm{ORD}(Z, x)$, $[\delta](x = n \,\&\, \mathrm{ORD}(Z, x))$ a $[\delta]\,\mathrm{ORD}(Z, n)$.

LITERATURA

[1] M. J. Fisher, R. E. Ladner: Propositional modal logic of programs. Proc. 9th Annual ACM Symp., Boulder 1977, 286—294.

[2] D. Harel: First order dynamic logic. (Lecture Notes in Computer Science Vol. 68.) Springer-Verlag, Berlin—Heidelberg—New York 1979.

[3] D. Harel, A. R. Meyer, V. R. Pratt: Computability and completeness in logics of programs. Proc. 9th Annual ACM Symp., Boulder 1977, 261—268.

[4] D. Harel, V. R. Pratt: Nondeterminism in logics of programs. Proc. 5th ACM Symp. on Principles of Programming Languages, Tucson, Ariz. 1978, 203—213.

[5] D. Litvintchouk, V. R. Pratt: A proof-checker for dynamic logic. Proc. 5th IJCAI, Boston 1978, 552—558.

[6] R. Parikh: A completeness result for propositional dynamic logic. MFCS 1978, J. Winkowski (ed.). (Lect. Notes in Comp. Sci. 64.) Springer-Verlag, Berlin—Heidelberg—New York 1978, 405—416.

[7] R. Parikh: A decidability result for second order process logic. Preprint MIT/LCS/TM-112 (1978).

[8] V. R. Pratt: Semantic considerations of Floyd-Hoare logic. Proc. 17th IEEE Symp. on Found. of Comp. Sci. (1976), 109—121.

[9] V. R. Pratt: Six lectures on dynamic logic. Preprint MIT/LCS/TM-117 (1978).

[10] V. R. Pratt: A near-optimal method for reasoning about action. Preprint MIT/LCS/TM-113 (1978).

[11] H. Rasiowa: Algorithmic logic. Prace IPI PAN 281, Warszawa 1977.

[12] A. Salwicki: Formalized algorithmic languages. Bull. Acad. Polon. Sci., Sér. Math., Phys. Astron. *18* (1970), 227—232.

[13] P. Hájek, P. Kúrka: A second order dynamic logic with array assignments. (Zasláno do tisku.)