PIĄTA JESIENNA SZKOŁA PTI

WSPÓŁCZESNE KIERUNKI ROZWOJU INFORMATYKI

Organizowane przez POLSKIE TOWARZYSTWO INFORMATYCZNE

Do użytku
wewnętrznego

Mrągowo, 5—9 grudnia 1988 r.

PIĄTA JESIENNA SZKOŁA PTI

WSPÓŁCZESNE KIERUNKI ROZWOJU INFORMATYKI

Organizowane przez POLSKIE TOWARZYSTWO INFORMATYCZNE

Do użytku
wewnętrznego

Mrągowo, 5—9 grudnia 1988 r.

S P I S T R E S C I

Str.

1. Doc. Wojciech CELLARY (Politechnika Poznańska)
Sterowanie współbieżnością w systemach rozproszonych baz danych 1

2. Dr Jacek IRLIK (Instytut Systemów Sterowania)
Ochrona prawna programów na świecie 21

3. Dr Wacław ISZKOWSKI (Politechnika Warszawska)
Przyspieszanie obliczeń przez ich rozpraszanie 39

4. Mgr Andrzej KŁOPOTOWSKI (WZM PZL)
Sprawna praca małych zespołów 67

5. Prof. Laurent SIKLOSSY (Free University Amsterdam)
Active Databases: Theory and Application 76

6. Prof. Władysław M. TURSKI (Uniwersytet Warszawski)
Logiki nieklasyczne (dla informatyka pracującego) 103

7. Doc. Piotr DEMBINSKI
Protokoły komunikacyjne 133

SŁOWO WSTĘPNE

W programie Piątej Jesiennej Szkoły PTI dominuje tematyka współbieżności i systemów rozproszonych. Z ogólnej liczby osmiu wykładów aż cztery poświęcone są bezpośrednio lub posrednio właśnie jej. Fakt ten właściwie ilustruje swiatowe tendencje rozwoju informatyki i jej zastosowań. Sieci, systemy rozproszone, zrównoleglanie obliczeń dla ich przyspieszenia, to tematy zajmujące poczesne miejsce w pracach teoretyków i praktyków informatyki. Rowniez w Polsce mamy silne środowisko naukowe zajmujące się tymi zagadnieniami. Z praktyką jest, niestety, nieco gorzej. Czytelnikom tego tomu nie muszę wyjaśniać czego brak w naszym kraju, aby mobły go objąć krajowe i międzynarodowe sieci komputerowe. Ale przeciez w Mrągowie spotykamy się po to aby nabrać twórczego fermentu, no i trochę pomarzyć.

Wykład Antoniego Mazurkiewicza wprowadza w ogólną matematyczną teorię procesow współbieznych, stanowiąc dobry punkt wyjscia do następnych wykładów z tej tematyki. Dwa dalsze poświęcone są typowym zastosowaniom procesów współbieznych. Wacław Iszkowski omawia metody przyspieszania obliczeń przez ich rozpraszanie, a Wojciech Cellary - rozproszone bazy danych. Cykl współbieżnościowy zamyka wykład Piotra Dembińskiego na temat protokołów komunikacyjnych. Stanowią one zarówno ważne pole zastsowań procesów współbieżnych, jak i narzędzie organizowania współpracy pomiędzy rozproszonymi systemami.

Obok zagadnień związanych z organizacją przepływu danych i sterowania procesami, tematem o którym ostatnio coraz głośniej w informatyce są nieklasyczne metody wnioskowania. Wyrosły one z dwóch grup zastosowań: analiza poprawności programów oraz dedukcyjne banki danych. O wnioskowaniu nieklasycznym pisze Władysław Turski.

. Tegorocznym wykładowcą zagranicznym jest Laurent Siklóssy profesor Wolnego Uniwersytetu w Amsterdamie. Studiował matematykę i fizykę na uniwersytecie w Yale (USA) oraz Harvard (USA), na tym ostatnim uzyskując stopień magistra fizyki (1964). Doktoryzował się z informatyki na Uniwersytecie Carnegie-Mellon (USA) w roku 1968. Pracował jako adiunkt, a następnie profesor na uniwersytetach Kalifornijskim, Texaskim i Chicagowskim. Zajmuje się sztuczną inteligencją. Jest autorem około osiemdziesięciu prac naukowych. Jego wykład poświęcony został zastosowaniom metod sztucznej inteligencji w bazach danych.

Tradycyjnie do grona wykładowców naszych szkół zapraszani są, obok informatyków, również specjaliści z innych dziedzin. W tym roku po raz trzeci juz występuje Jacek Irlik, informatyk i prawnik w jednej osobie. Jego wykład poświęcony jest omówieniu stanu ochrony prawnej programów na świecie. Drugi wykład z tej grupy stanowi w historii naszych Szkół niewątpliwą nowość. Andrzej Kłopotowski, specjalista od organizacji pracy i zarządzania, wieloletni ekspert międzynarodowej organizacji ILO (International Labor Organization), omawia zagadnienia sprawnej organizacji pracy małych zespołów. Temat ten staje się dziś przebojem nowoczesnych przemysłów. Pracę w małych zespołach prowadzą jednak również programiści, projektanci systemów komputerowych i naukowcy. A więc temat dla nas wszystkich.

Warszawa, październik 1988 Andrzej Blikle

Jesienna Szkoła PTI
Mrągowo, grudzień 1988

STEROWANIE WSPÓŁBIEŻNOŚCIĄ W SYSTEMACH ROZPROSZONYCH BAZ DANYCH

doc. dr hab. inż. W.Cellary
Instytut Informatyki, Automatyki i Robotyki
Politechnika Poznańska
ul. Piotrowo 3A
60-965 Poznań

1. Wstęp

Sterowanie współbieżnością w systemach rozproszonych baz danych (SRBD) stanowi jeden z najistotniejszych i najtrudniejszych problemów stojących przed ich projektantami. Celem tego sterowania jest zapewnienie spójności rozproszonej bazy danych (RBD) i zagwarantowanie zrealizowania każdej transakcji zainicjowanej w systemie w sytuacji, gdy wielu użytkowników dokonuje dostępu do RBD, a zatem gdy wiele transakcji jest wykonywanych współbieżnie.

W badaniach problemów sterowania współbieżnością transakcji w SRBD można wyróżnić kilka kierunków, które dominowały w pewnych okresach. W pierwszym okresie badań dominowały próby adaptacji dla SRBD algorytmów metod sterowania opracowanych dla wielodostępnych, lecz scentralizowanych baz danych. Próby te zakończyły się niepowodzeniem, co wynikało z dwóch podstawowych faktów. Po pierwsze, w SRBD dowolna transakcja może żądać jednoczesnego dostępu do wielu lokalnych baz danych, zlokalizowanych na różnych stanowiskach komputerowych rozproszonych geograficznie. Wobec powyższego, w stosunku do problemu sterowania współbieżnością w

systemach scentralizowanych baz danych, który polegał na zapewnieniu spójności wewnętrznej pojedynczej lokalnej bazy danych, odpowiedni problem w SRBD jest rozszerzony o konieczność zapewnienia spójności zewnętrznej rozproszonej bazy danych, rozumianej jako identyczność wszystkich kopii tej samej danej logicznej. Po drugie, istotną trudność konstrukcji algorytmów sterowania współbieżnością w SRBD stanowi fakt, że w systemach tych żadne stanowisko komputerowe nie dysponuje pełną informacją o globalnym stanie całego systemu. Stąd kniecznosć podejmowania decyzji zarządzających wykonywaniem transakcji na każdym stanowisku komputerowym na podstawie niepełnej i nie w pełni aktualnej informacji o aktywności pozostałych stanowisk. Wymaga to uwzględnienia tych faktów w samych mechanizmach sterowania, stanowiących podstawę do opracowania konkretnych algorytmów.

W drugim okresie badań nad problemami sterowania współbieżnością transakcji w SRBD wykrystalizowały się trzy podstawowe metody dotyczące tak zwanego syntaktycznego modelu współbieżności przy kryterium jednowersyjnej uszeregowalności. Metody te to: metoda blokowania, metoda porządkowania transakcji według etykiet czasowych i metoda legalizacji. Badania w zakresie tego modelu i tego kryterium są ciągle aktualne. Skupiają się one obecnie nad integracją wyżej wymienionych metod.

W kolejnym etapie badań nastąpiło odejście od założenia o jednowersyjności danych w RBD. Przyjęto mianowicie, że każda dana stanowi sekwencję kolejnych wersji utworzonych przez kolejne operacje zapisu. SRBD, dla którego przyjętę taki model danych, nosi nazwę systemu wielowersyjnego. Atrakcyjność wielowersyjności wynika z tego, że pozwala ona na znaczne zwiększenie stopnia współbieżności transakcji. Ponadto pozwala ona połączyć w naturalny sposób mechanizmy sterowania współbieżnością z mechanizmami podwyższania niezawodności.

W ostatnich dwóch latach pojawił się nowy kierunek badań nad problemami sterowania współbieżnością charakteryzujący się odejściem od syntaktycznego modelu współbieżności i włączeniem do procesu synchronizacji informacji semantycznej o SRBD.

Dotychczsowe wyniki prac w tym zakresie są obiecujące.

Celem tego artykułu jest ukazanie zarysu problematyki sterowania współbieżnością w SRBD. W rozdziale 2. wprowadzamy pojęcia podstawowe, takie jak system rozproszonej bazy danych, rozproszona baza danych i jej spójność oraz transakcja. W rozdziale 3. omawiamy trzy modele sterowania współbieżnością, nazywane od kryteriów poprawności realizacji zbioru transakcji: jednowersyjną uszeregowalnością, wielowersyjną uszeregowalnością i wielopoziomową atomowością. Dwa pierwsze są modelami syntaktycznymi, a trzeci jest modelem semantycznym. W rozdziale 4. przedstawiamy pokrótce koncepcje metod sterowania współbieżnością: metody blokowania, szeregowania według etykiet czsowych i legalizacji. Artykuł kończy krótkie podsumowanie.

2. Model systemu rozproszonej bazy danych.

Systemem rozproszonej bazy danych SRBD nazywamy trójkę

$$(RBD, \tau, C(\tau)),$$

gdzie RBD jest rozproszoną bazą danych, $\tau = (T_1, T_2, \ldots, T_m)$ jest zbiorem transakcji, a $C(\tau)$ stanowi kryterium poprawności realizacji zbioru transakcji w RBD.

2.1. Rozproszona baza danych

Rozproszona baza danych (RBD) jest zbiorem *danych*. Każda dana ma *nazwę* i *wartość*. W konkretnym systemie, daną może być rekord, plik, strona, relacja, krotka relacji itp. Odróżniamy dane widziane przez użytkowników systemu od ich implementacji. Dane widziane przez użytkowników nazywamy *danymi logicznymi* i oznaczamy dużymi literami X, Y, Z. Zbiór wszystkich danych logicznych w RBD nazywamy *logiczną bazą danych*:

$$\mathcal{L} = (X_1, X_2, \ldots, X_n).$$

Każda dana logiczna X_i jest, w ogólności, implementowana jako zbiór *danych fizycznych* $x_{i1}, x_{i2}, \ldots, x_{ic_i}$, będących *kopiami*,

przechowywanych na stanowiskach RBD:

$$\mathcal{S} = (S_1, S_2, \ldots, S_N).$$

Zbiór wszystkich danych fizycznych w RBD nazywamy *fizyczną bazą danych*:

$$\mathcal{D} = (x_{11}, x_{12}, \ldots, x_{1c_1}, \ldots, x_{i_1}, x_{i_2}, \ldots, x_{ic_i}, \ldots, x_{n_1}, x_{n_2}, \ldots, x_{nc_n}).$$

Mówimy, że RBD jest *niedublikowana* jeśli żadna dana fizyczna nie ma kopii na innym stanowisku. Mówimy, że RBD jest *całkowicie dublikowana* jeśli każda dana fizyczna ma kopię na każdym stanowisku. Mówimy wreszcie, że RBD jest *częściowo dublikowana* jeśli istnieją dane fizyczne mające kopie na innych stanowiskach.

Każda dana fizyczna x jest, w ogólności, ciągiem wersji:

$$x = \langle x^0, x^1, \ldots, x^g \rangle,$$

przechowywujących jej kolejne wartości. Liczba wersji może być ograniczona lub nie. Jeśli jest nieograniczona, to każde uaktualnienie danej powoduje utworzenie jej następnej wersji. Jeśli jest ograniczona, to uaktualnienie danej może spowodować zastąpienie jednej z istniejących jej wersji nową wersją. Przyjmuje się, że istnieje co najmniej zerowa wersja każdej danej fizycznej. Wartości wszystkich wersji wszystkich danych fizycznych stanowią *stan* RBD.

Dana fizyczna jest *jednowersyjna* jeśli w każdej chwili istnieje tylko jedna jej wersja; w przeciwnym przypadku dana jest *wielowersyjna*. RBD jest *jednowersyjna* jeśli każda dana fizyczna $x \in \mathcal{D}$ jest jednowersyjna, w przeciwnym przypadku RBD jest wielowersyjna.

Reasumując, RBD jest zdefiniowana jako czwórka:

$$\langle \mathcal{L}, \mathcal{D}, \mathcal{S}, \mathcal{L}oc \rangle,$$

gdzie:

$\mathcal{L} = \langle X_1, X_2, \ldots, X_n \rangle$ jest zbiorem *danych logicznych*, nazywanym *logiczną bazą danych*;

$\mathcal{D} = \langle x_{11}, x_{12}, \ldots, x_{1c_1}, \ldots, x_{i_1}, \ldots, x_{ic_i}, x_{n_1}, \ldots, x_{nc_n} \rangle$ jest zbiorem *danych fizycznych*, nazywanym *fizyczną bazą danych*;

$\mathcal{S} = \langle S_1, S_2, \ldots, S_N \rangle$ jest zbiorem N stanowisk;

$\mathcal{L}oc : \mathcal{D} \longrightarrow \mathcal{S}$ jest funkcją określającą lokalizację danych fizycznych w zbiorze stanowisk.

2.2. Spójność RBD

Jednym z głównych problemów SBRD jest problem zapewnienia poprawności danych oraz wzajemnych związków pomiędzy nimi, to znaczy, zapewnienia *spójności* RBD. Zachowanie spójności RBD należy rozumieć jako ochronę danych przed możliwością wystąpienia błędu w wyniku niewłaściwego uaktualnienia, usunięcia lub wprowadzenia danych.

Spójność RBD może zostać naruszona zasadniczo z trzech powodów:

(i) na skutek błędnego zarządzania współbieżnością wykonywania transakcji żądających dostępu do tych samych danych;

(ii) na skutek wystąpienia awarii systemu powodującej jego częściowy lub całkowity upadek;

(iii) na skutek błędnego programu transakcji.

W dalszym ciągu będziemy rozważać wyłącznie możliwość naruszenia spójności RBD na skutek błędnego zarządzenia współbieżnym dostępem wielu transakcji do tych samych danych.

Wyróżniamy dwa rodzaje spójności RBD: spójność wewnętrzną i spójność zewnętrzną. *Spójność wewnętrzna* RBD jest definiowana w oparciu o zbiór ograniczeń integralnościowych nałożonych na semantykę logicznej bazy danych. W ogólności wyróżniamy dwa typy ograniczeń integralnościowych:
- ograniczenia typu statycznego, określające dopuszczalny zbiór stanów RBD, nazywany *zbiorem stanów spójnych*
- ograniczenia typu dynamicznego, określające dopuszczalny zbiór *przejść stanów spójnych* RBD. Problem zapewnienia spójności wewnętrznej RBD nie różni się w swej istocie od odpowiedniego problemu dla scentralizowanych baz danych. Polega on na weryfikowaniu ograniczeń integralnościowych. Specyficznym problemem SRBD jest problem zapewnienia spójności zewnętrznej.

Mówimy, że RBD jest w stanie *spójności zewnętrznej*, jeżeli, przy założeniu, że zbiór aktualnie wykonanych transakcji τ jest pusty, stany wszystkich kopii fizycznych $x_{i1},\ldots,x_{ic_i}$ każdej danej logicznej $X_i \in \mathcal{X}$ są identyczne.

Jak wynika z powyższych definicji, mechanizm zapewnienia spójności zewnętrznej RBD nie może polegać na testowaniu stanu RBD. Wymagałoby to bowiem, w każdym przypadku uaktualnienia danej blokowania całego systemu. W celu rozwiązania problemu spójności, zarówno wewnętrznej jak i zewnętrznej, wprowadza się koncepcję *transakcji*.

2.3. Transakcja

Transakcja jest wykonaniem programu zawierającego operacje na RBD, ujęte w logiczne nawiasy o znaczeniu początku i końca tarnsakcji.

Dwie szczególnie ważne operacje na RBD to *odczyt* i *zapis*. Operacja odczytu r(x) udostępnia wartość jednej z wersji danej x, zazwyczaj ostatniej wersji. Operacja zapisu w(x) tworzy nową wersję danej x. Wersja ta zastępuje starą wersję - w przypadku jednowersyjnych RBD, zastępuje jedną z starych wersji - na przykład najstarszą - w przypadku wielowersyjnych RBD z

ograniczoną liczbą wersji, lub jest dodawana do RBD - w przypadku wielowersyjnych RBD z nieograniczoną liczbą wersji.

Transakcje nie zawierające operacji zapisu są nazywane *zapytaniami*; transakcje zawierające operacje zapisu są nazywane *transakcjami uaktualniającymi*.

Operacje na RBD składające się na transakcję mogą być wykonywane współbieżnie. Zbiór tych operacji może być nieznany w momencie inicjacji transakcji, lecz tworzony dynamicznie w trakcie jej wykonywania. Mówimy w takim przypadku o transakcji *zależnej od danych*.

Transakcja może zakończyć się normalnie lub nienormalnie. W pierwszym przypadku mówimy, że jest ona *zaakceptowana*, a w drugim, że jest *odrzucona*. Dopóki transakcja nie jest zaakceptowana to może być odrzucona. Transakcja może być odrzucona z własnej inicjatywy - na przykład na skutek błędnych danych wejściowych, lub przez system zarządzenia rozproszoną bazą danych - na przykład ze względu na wymagania sterowania współbieżnością.

Transakcja może być częścią większego programu użytkowego zawierającego wiele transakcji. Istotne jest to, że żadna operacja dostępu do RBD nie może być wykonana poza transakcją.

Transakcje są niezależne od siebie, to znaczy nie komunikują się ze sobą bezpośrednio. Jedynym sposobem ich komunikacji jest zapisywanie i odczytywanie tych samych danych.

O transakcji zakładamy, że cechują ją trzy podstawowe właściwości:

1. *Spójność*; która oznacza, że transakcja odwzorowuje stan spójny RBD w stan spójny, przy czym RBD nie musi być w stanie spójnym w trakcie wykonywania transakcji.
2. *Atomowość*; która oznacza, że w przypadku gdy w skład transakcji wchodzą operacje zapisu, muszą zostać wykonane wszystkie te operacje lub żadna z nich. Innymi słowy, jeżeli transakcja musi zostać wycofana, na skutek błędu operatora lub awarii systemu,

to wszystkie zmiany, jakie transakcja wprowadziła w bazie danych muszą być cofnięte.

3. *Trwałość*; która oznacza, że po zaakceptowaniu transakcji, zmiany dokonane przez nią w RBD nie zostaną utracone nawet w przypadku wystąpienia awarii.

Formalnie, *transakcją* T_i nazywamy uporządkowaną parę $(\bar{T}_i, <_{T_i})$, gdzie $\bar{T}_i = \{T_{ij} : 1 \leq j \leq m_i\}$ jest zbiorem operacji, natomiast $<_{T_i}$ jest relacją poprzedzania porządkującą zbiór T_i.

W celu określenia typu operacji na RBD, danej, do której dokonuje dostępu i stanowiska, na którym jest wykonywana będziemy stosować następującą notację:

$r_i^{S_k}(x)$ - dla operacji odczytu oraz

$w_i^{S_k}(x)$ - dla operacji zapisu,

gdzie S_k oznacza stanowisko, na którym operacja jest wykonana, a x oznacza daną fizyczną, taką że $\mathcal{L}oc(x) = S_k$.

3. Modele sterowania współbieżnością w RBD

W rozdziale 2 przedstawiliśmy model RBD i model transakcji oraz wprowadziliśmy pojęcie spójności RBD. W tym rozdziale będziemy zajmować się zbiorem transakcji τ.

Jak wspomnieliśmy, jedną z zasadniczych przyczyn naruszenia spójności RBD jest niewłaściwe zarządzanie współbieżnym wykonywaniem zbioru transakcji. Współbieżne wykonywanie transakcji wymaga odpowiedniego *sterowania*, polegającego na szeregowaniu operacji na RBD pochodzących z różnych transakcji. Formalny opis wykonania zbioru transakcji τ w SRBD nazywamy *realizacją zbioru transakcji* τ. Uwzględniamy w nim tylko zaakceptowane transakcje, gdyż transakcje odrzucone nie mają wpływu na stan RBD. Dowolna realizacja zbioru transakcji jest wynikiem działania określonego

algorytmu sterowania współbieżnym wykonywaniem transakcji przyjętego w SRBD. Podstawową miarą jakości działania takiego algorytmu jest rozmiar zbioru wszystkich poprawnych realizacji generowanych przez ten algorytm. Zbiór ten jest funkcją informacji o SRBD jaką rozporządza algorytm sterowania współbieżnym wykonaniem transakcji.: o strukturze i organizacji RBD, o ograniczeniach integralnościowych nałożonych na RBD, o zbiorze operacji składających się na transakcje oraz o charakterze obliczeń wykonywanych przez transakcje. Zakres i charakter tych informacji jest określony przez *model sterowania współbieżnością*. Model ten określa kryterium poprawności C(τ) realizacji zbioru transakcji τ .

W kolejnych punktach przedstawimy trzy zasadnicze kryteria poprawności realizacji zbioru transakcji τ: kryterium uszeregowalności jednowersyjnej - krócej - uszeregowalności, kryterium uszeregowalności wielowersyjnej i kryterium wielopoziomowej atomowości. Pierwsze dwa kryteria poprawności odnoszą się do przypadku, gdy dysponujemy informacją czysto syntaktyczną o SRBD; kryterium trzecie odnosi się do przypadku gdy dysponujemy również informacją semantyczną o zbiorze transakcji τ.

3.1. Uszeregowalność

W tym punkcie będziemy rozpatrywać problem poprawności realizacji transakcji w jednowersyjnych SRBD. W celu zdefiniowania kryterium uszeregowalności najpierw formalnie zdefiniujemy jednowersyjną realizację zbioru transakcji.

Jednowersyjną realizacją s zbioru transakcji τ nazywamy częściowo uporządkowany zbiór $(T(\tau), <_s)$, gdzie $T(\tau) = \bigcup_i T_i \cup T_o \cup T_f$ jest zbiorem wszystkich operacji składających się na transakcje należące do zbioru τ rozszerzonego o operacje $T_o = \langle w_{oj}(x) : x \in \mathcal{D} \rangle$ i $T_f = \langle r_{fj}(x) : x \in \mathcal{D} \rangle$ dwóch hipotetycznych transakcji: początkowej i końcowej, które odpowiednio zapisują początkowy stan RBD i odczytują jej stan końcowy, natomiast $<_s$ jest relacją częściowego porządku, dla której spełnione są

następujące warunki

(i) $<_s = \sum T_i$;

(ii) dla każdej pary operacji $T_{ij} \in T_i$, $T_{kl} \in T_k$, żądających dostępu do tej samej danej, jeśli co najmniej jedna z nich jest operacją zapisu, to zachodzi:

$$T_{ij} <_s T_{kl} \vee T_{kl} <_s T_{ij}.$$

Wyróżniamy sekwencyjne i współbieżne realizacje zbioru transakcji. Realizację r zbioru transakcji τ nazywamy *realizacją sekwencyjną*, jeżeli żadne dwie transakcje nie są wykonywane współbieżnie. w przeciwnym przypadku mówimy o *realizacji współbieżnej*.

Problem poprawności danej realizacji będziemy rozpatrywać w tym punkcie w kontekście informacji czysto syntaktycznej. Oznacza to, że algorytm sterowania współbieżnością dysponuje wyłącznie informacją o zbiorach danych, do których transakcje żądają dostępu, natomiast poprawność współbieżnej realizacji r zbioru τ winna być zachowana dla dowolnego zbioru ograniczeń integralnościowych nałożonych na RBD oraz dla dowolnego zbioru transakcji. Mówimy w tym przypadku o *syntaktycznym modelu współbieżności*.

Kryterium poprawności realizacji w syntaktycznym modelu współbieżności dla jednowersyjnych SRBD nosi nazwę *jednowersyjnej uszeregowalności*, w skrócie *uszeregowalności*. Wprowadzimy teraz kilka pojęć niezbędnych dla formalnego zdefiniowania tego kryterium.

Stanem RBD nazywamy zbiór wartości wszystkich danych fizycznych należących do RBD. *Obrazem RBD* widzianym przez transakcję T w realizacji s, nazywamy zbiór wartości danych odczytanych przez tę transakcję.

Mówimy, że realizacje $s = (\bar{T}(\tau), <_s)$ i $s' = (\bar{T}(\tau), <_{s'})$ są *obrazowo równoważne*, jeżeli obraz RBD widziany przez dowolną

transakcję $T \in \tau$ w realizacji s jest identyczny z obrazem RBD, widzianym przez tę transakcję w realizacji s'.

Mówimy, że realizacje $s=(\bar{T}(\tau), <_s)$ i $s'=(\bar{T}(\tau), <_{s'})$ są *stanowo równoważne*, jeżeli dla dowolnego stanu początkowego RBD, stany końcowe osiągnięte w ich wyniku są identyczne. Mówimy ogólnie, że realizacje $s=(\bar{T}(\tau), <_s)$ i $s'=(\bar{T}(\tau), <_{s'})$ są *równoważne*, jeżeli są obrazowo i stanowo równoważne.

Możemy obecnie formalnie zdefiniować kryterium uszeregowalności.

Kryterium uszeregowalności

Realizacja s zbioru transakcji τ jest poprawna wtedy i tylko wtedy, gdy jest ona równoważna dowolnej realizacji sekwencyjnej zbioru τ.

3.2. Uszeregowalność wielowersyjna

W tym punkcie będziemy rozpatrywać problem poprawności realizacji transakcji w wielowersyjnych SRBD, o nieograniczonej liczbie wersji. W systemach tych operacje odczytu danej x stoją przed możliwością dostępu do jednej z wielu aktualnie istniejących jej wersji. Specyfikacja wielowersyjnej realizacji zbioru transakcji musi zatem zawierać odwzorowanie operacji odczytu "danej" w operację odczytu "wersji danej".

Wielowersyjną realizacją zbioru transakcji τ nazywamy trójkę $mvs=(\bar{T}(\tau), h, <_{mvs})$, gdzie:

1. $\bar{T}(\tau) = U_i \bar{T}_i \cup T_o \cup T_f$ jest zbiorem wszystkich operacji składających się na transakcje należące do zbioru τ, rozszerzonego o operacje dwóch hipotetycznych transakcji: początkowej i końcowej;
2. h jest funkcją, która odwzorowuje każdą operację odczytu $r_{ij}(x) \in \bar{T}(\tau)$ w operację zapisu $w_{kl}(x) \in \bar{T}(\tau)$;
3. $<_{mvs} = U_i <_{T_i}$ jest relacją częściowego porządku na zbiorze $\bar{T}(\tau)$ spełniającą warunki:

jeśli $T_{ij} <_{T_i} T_{ik}$. to $T_{ij} <_{mvs} T_{ik}$;

jeśli $h(r_{ij}(x)) = w_{kl}(x)$ to $w_{kl}(x) <_{mvs} r_{ij}(x)$.

Zdefiniowana powyżej funkcja h odwzorowuje zbiór operacji odczytu danej w zbiór operacji zapisu wersji danej, co należy interpretować jako odwzorowanie odczytu danej w odczyt wersji danej.

Mówimy, że realizacje wielowersyjne $mvs(\tau) = (\bar{T}(\tau), h, <_{mvs})$ i $mvs'(\tau) = (\bar{T}(\tau), h', <_{mvs'})$ są *równoważne*, jeżeli są obrazowo i stanowo równoważne. Dwie realizacje wielowersyjne $mvs(\tau)$ i $mvs'(\tau)$ są obrazowo równoważne wtedy i tylko wtedy, gdy $h = h'$. Dwie realizacje wielowersyjne $mvs(\tau)$ i $mvs'(\tau)$ są stanowo równoważne jeśli stany końcowe RBD osiągnięte w ich wyniku są identyczne.

W celu sformułowania kryterium wielowersyjnej uszeregowalności wprowadzimy pojęcie standardowej realizacji sekwencyjnej, w której są nałożone odpowiednie warunki na funkcję h. Zauważmy bowiem, że sam fakt sekwencyjności realizacji wielowersyjnej nie gwarantuje jeszcze jej poprawności.

Sekwencyjna wielowersyjna realizacja $mvs(\tau) = (\bar{T}(\tau), h, <_{mvs})$ jest *standardowa* jeśli każda operacja odczytu $r_{ij}(x) \in \bar{T}(\tau)$ dokonuje dostępu do wersji danej x utworzonej przez ostatnią operację $w_{kl}(x) \in \bar{T}(\tau)$ poprzedzającą $r_{ij}(x)$.

Kryterium uszeregowalności wielowersyjnej

Realizacja wielowersyjna mvs zbioru transakcji τ jest poprawna, jeżeli jest ona równoważna dowolnej standardowej wielowersyjnej realizacji sekwencyjnej zbioru τ.

3.3. Wielopoziomowa atomowość

W punktach 3.1 i 3.2 rozważaliśmy syntaktyczny model współbieżności, przyjmując za kryterium poprawności współbieżnej realizacji zbioru transakcji uszeregowalność. Mówiąc, że dana

realizacja jest poprawna w sensie kryterium uszeregowalności rozumiemy, że dla dowolnego zbioru ograniczeń integralnościowych nałożonych na bazę danych oraz dla dowolnego zbioru transakcji realizacja ta zachowuje spójność bazy danych. Jednakże, w odniesieniu do wielu SRBD kryterium uszeregowalności jest zbyt silne i za bardzo ogranicza stopień współbieżności wykonywania transakcji możliwy do osiągnięcia w systemie. Ograniczenie to wynika z faktu, że uszeregowalność "aproksymuje" w sposób pesymistyczny zbiór możliwych ograniczeń integralnościowych nałożonych na RBD w tym sensie, że zakłada wystąpienie zawsze najgorszego przypadku tych ograniczeń. Kryterium to jest zatem jedynie warunkiem dostatecznym zachowania przez dowolną realizację współbieżną ogólnie rozumianej poprawności bazy danych. Dysponując dodatkową informacją o SRBD, kryterium uszeregowalności można osłabić uzyskując w wyniku realizacje nieuszeregowalne, ale poprawne z punktu widzenia spójności bazy danych i cechujące się większym stopniem współbieżności. Informacje te można sklasyfikować następująco:

-informacje o ograniczeniach integralnościowych,
-informacje syntaktyczne o zbiorze przetwarzanych transakcji,
-informacje semantyczne o zbiorze przetwarzanych transakcji.

Mogłoby się wydawać, że problem specyfikacji spójnych realizacji nieuszeregowalnych, a więc, innymi słowy, problem specyfikacji zbioru C(τ), może być rozwiązany przez podanie zbioru ograniczeń integralnościowych nałożonych na RBD. Rozwiązanie takie nie jest jednak możliwe. Po pierwsze, nie jest praktycznie możliwe sterowanie współbieżnością w trybie "on-line" w oparciu wyłącznie o znajomość zbioru ograniczeń integralnościowych. Z założenia bowiem, testowanie poprawności współbieżnej realizacji zbioru transakcji byłoby wówczas możliwe dopiero po zakończeniu realizacji całego zbioru transakcji w oparciu o końcowy stan RBD, a więc w trybie "off-line". Po drugie, najczęściej, zbiór wszystkich rzeczywistych ograniczeń integralnościowych nałożonych na RBD: (i) nie jest w pełni znany, a nawet identyfikowalny; (ii) nie jest formalnie zadeklarowany; (iii) nie jest praktycznie weryfikowalny. Po trzecie, można pokazać, że znajomość zbioru ograniczeń integralnościowych nie w każdym przypadku jest warunkiem dostatecznym, gwarantującym semantyczną poprawność

współbieżnej realizacji dowolnego zbioru transakcji. Dlatego do specyfikacji zbioru C(τ) wykorzystuje się informację syntaktyczną i semantyczną o zbiorze transakcji τ wykonywanych w SRBD. Informacja ta pozwala na podanie reguł, zgodnie z którymi określone transakcje lub grupy transakcji mogą być współbieżnie realizowane. Istotnym argumentem przemawiającym za takim rozwiązaniem jest fakt, że dla wielu zastosowań użytkownik RBD lub jej administrator potrafi stosunkowo precyzyjnie określić zbiór realizowanych transakcji oraz sklasyfikować i scharakteryzować ich wzajemne związki mimo niepełnej znajomości zbioru ograniczeń integralnościowych. Mówimy w tym przypadku o *semantycznym modelu współbieżności*.

Przedstawimy teraz semantyczny model współbieżności oparty na koncepcji kompatybilności transakcji, opracowany przez Garcię-Molinę. W modelu tym kryterium poprawności realizacji transakcji jest dane w postaci specyfikacji realizacji spójnych, lecz niekoniecznie uszeregowalnych.

Najpierw wprowadzimy formalną definicję pojęcia realizacji spójnej, z którego korzystaliśmy dotychczas w sposób intuicyjny.

Niech $\mathcal{D}_0$ oznacza początkowy, spójny stan bazy danych $\mathcal{D}$, niech s oznacza realizację zbioru transakcji τ. Mówimy, że realizacja s jest *semantycznie spójna* wtedy i tylko wtedy, gdy:
(i) realizacja s odwzorowuje stan spójny $\mathcal{D}_0$ w stan spójny $\mathcal{D}_f$, oraz
(ii) każda transakcja $T \in \tau$ widzi spójny stan bazy danych $\mathcal{D}$.

Krokiem transakcji nazywamy sekwencję elementarnych operacji wykonywanych na pojedynczym stanowisku SRBD. Krok jest *atomowy* jeżeli sekwencja operacji składająca się na ten krok jest wykonana w sposób niepodzielny. Realizację zbioru transakcji, w której kroki wszystkich transakcji są atomowe, nazywamy *realizacją krokową*.

Wprowadzimy teraz pojęcie *semantycznego typu transakcji* T, jako zbioru kroków, który będziemy oznaczać przez *typ*(T). Semantyczny typ transakcji jest określany przez użytkownika SRBD dla danego zastosowania. Zbiór wszystkich semantycznych typów

transakcji dla danego zastosowania będziemy oznaczać przez *TYP*.

Dla każdego semantycznego typu $Y \in TYP$ definiujemy *zbiór kompatybilnych typów semantycznych* oznaczony przez $CSC(Y)$. Elementy zbioru $CSC(Y)$ nazywane *deskryptorami współbieżności* zbioru kompatybilnych typów semantycznych $CSC(Y)$, gdzie $dw \in CSC(Y)$ i $Y \in TYP$, nazywamy zbiór semantycznych typów transakcji spełniający następujące warunki:

(i) $dw \in TYP$;

(ii) dla dowolnych transakcji T_1, T_2 takich, że $typ(T_1) \in dw$ i $typ(T_2) \in dw$, dowolna krokowa realizacja s transakcji T_1, T_2 zachowuje spójność RBD.

Mówimy, że zbiór $CSC(Y)$ jest *nieredundantny*, jeżeli nie istnieją deskryptory $dw_1 \in CSC(Y)$ i $dw_2 \in CSC(Y)$ takie, że $dw_1 \subseteq dw_2$. Mówimy, że dwa zbiory kompatybilnych typów semantycznych są *koherentne*, jeżeli dla dowolnych typów semantycznych Y_1 i Y_2 takich, że istnieje $dw \in CSC(Y_2)$ i $Y_1 \in dw$, również $dw \in CSC(Y_1)$. W dalszym ciągu będziemy zakładać przypadek zbiorów nieredundantnych i koherentnych. Mówimy, że transakcja T_2 jest *kompatybilna* z transakcją T_1 wtedy i tylko wtedy, gdy istnieje deskryptor współbieżności dw taki, że $dw \in CSC(typ(T_1))$ i $typ(T_2) \in dw$. Jak wynika z powyższego, jeśli dwie transakcje są kompatybilne to żadna ich współbieżna realizacja nie narusza spójności RBD.

W rozważanym modelu współbieżności, zbiór poprawnych realizacji $C(\tau)$ jest ograniczony do realizacji krokowych, w których jedynie transakcje kompatybilne są wykonywane współbieżnie.

Przedstawiony powyżej semantyczny model współbieżności nie jest modelem najbardziej ogólnym, w tym sensie, że nie gwarantuje on maksymalnego stopnia współbieżności. Zauważmy, że wzrost stopnia współbieżności w stosunku do modelu syntaktycznego został uzyskany w nim w wyniku włączenia do procesu sterowania współbieżnością informacji pochodzącej od użytkownika o kompatybilności semantycznych typów transakcji. Jednakże współbieżność kompatybilnych transakcji nie jest dowolna, lecz może być realizowana wyłącznie w pewnych wybranych punktach realizacji transakcji, które wyznacza podział transakcji na kroki.

Dla przedstawionego powyżej semantycznego modelu współbieżności przyjęty podział na kroki jest stały i jednolity dla każdego semantycznego typu transakcji. Bardziej efektywne jest przyjęcie dla każdego semantycznego typu transakcji, różnych podziałów na kroki dla różnych typów transakcji kompatybilnych z nim.

Takie uogólnienie powyższego modelu współbieżnością zostało dokonane przez Lynch. W modelu tym zdefiniowano wielopoziomową strukturę klas semantycznych typów transakcji. Klasy zdefiniowane na wyższych poziomach struktury zawierają, jako podzbiory, klasy definiowane na niższych poziomach struktury. Definicja klasy typów transakcji na danym poziomie struktury określa kompatybilność tych typów transakcji. Typy transakcji należące do tej samej klasy zdefiniowanej na najniższym poziomie struktury cechują się najmniejszymi rozmiarami kroków, gdzie przez rozmiar kroku rozumiemy liczbę operacji składających się na krok, a zatem mogą być one realizowane z maksymalnym stopniem współbieżności. Typy transakcji należące do tych samych klas, definiowanych na wyższych poziomach struktury, cechują się odpowiednio mniejszym stopniem współbieżności realizacji.

Dla zamodelowania struktury zbioru semantycznych klas typów transakcji oraz struktur pojedynczych semantycznych typów transakcji wprowadzimy pojęcie abstrakcyjnej *K-poziomowej struktury* Π *określonej na zbiorze* X=(x).

K-poziomową strukturę Π określoną na zbiorze X definiujemy w oparciu o zbiór relacji równoważności (Π (i): 1≤ i ≤ k), w taki sposób, że dla dowolnego i-tego poziomu struktury Π relacja Π(i) ustala następujący podział zbioru X na rozłączne i niepuste klasy równoważności:

(i) Π(1) ustala dokładnie jedną klasę równoważności zawierającą wszystkie elementy x ∈ X,

(ii) Π(k) ustala dokładnie |X| jednoelementowych klas równoważności, (gdzie |X| oznacza rozmiar zbioru X),

(iii) klasy równoważności relacji Π(i) zawierają klasy równoważności relacji Π(i+1).

K-poziomową strukturą E klas typów transakcji będziemy nazywać K-poziomową strukturę Π określoną na zbiorze semantycznych typów transakcji *TYP*.

K-poziomową strukturą krokową B_Y semantycznego typu transakcji nazywamy K-poziomową strukturę Π określoną na zbiorze operacji Y taką, że dla każdego i, $1 \leq i \leq k$, $B_Y(i)$ jest podziałem typu Y na kroki.

Realizację $s=(T(\tau), <_s)$ dla danej K-poziomowej struktury klas typów transakcji E oraz danego zbioru K-poziomowych struktur krokowych $\mathcal{B}$ będziemy nazywać *realizacją wielopoziomowo atomową* jeżeli spełnia ona następujące warunki:

(i) $<_s$ jest relacją częściowego porządku na zbiorze $\bar{T}(\tau)$ taką, że

$$<_s = \sum_{T_i \in \tau} <_{T_i} ,$$

(ii) jeżeli dla dowolnych transakcji T_i i T_k, których semantyczne typy Y_i i Y_k są kompatybilne na poziomie $E(l)$, a nie są kompatybilne na poziomie $E(l')$ dla żadnego $l'>l$, istnieją takie operacje O_{ip}, O_{iq} i O_{kr}, że $(O_{ip}, O_{iq}) \in B_{Y_i}(l)$, to spełniony jest następujący warunek:

$$(T_i: I_{ip} <_s T_k: O_{kr}) \Rightarrow (T_i: O_{iq} <_s T_k: O_{kr}).$$

Intuicyjna interpretacja powyższej definicji jest następująca. Jeśli transakcje T_i i T_k są kompatybilne na l-tym poziomie struktury E, i operacja $O_{ip} \in T_i$ poprzedza w realizacji s operację $O_{kr} \in T_k$, to operację O_{kr} muszą również poprzedzać wszystkie operacje O_{iq} takie, że O_{ip} i O_{iq} należą do tego samego kroku transakcji T_i określonego na l-tym poziomie struktury B_{Y_i}.

Kryterium wielopoziomowej atomowości można obecnie sformułować następująco:

Kryterium wielopoziomowej atomowości

Dana współbieżna realizacja s zbioru transakcji τ jest poprawna jeśli jest ona wielopoziomowo atomowa.

4. Metody sterowania współbieżnością

4.1 Metoda blokowania

Jak wiadomo w trakcie wykonywania transakcji ograniczenia integralnościowe nałożone na RBD mogą być przejściowo naruszone. Celem każdej metody sterowania współbieżnością wykonywania transakcji jest niedopuszczenie do odczytu lub zapisu danych, których spójność jest chwilowo naruszona. Aby ten cel osiągnąć w metodzie blokowania z każdą daną $x \in \mathcal{D}$ jest związana *blokada*. Zakłada się, że transakcja przed odczytem lub zapisem danej musi uzyskać założenie odpowiedniej blokady tej danej oraz że blokady wszystkich danych, do których transakcja żądała dostępu, zostaną zdjęte przed zakończeniem jej wykonywania. Zakładanie blokad danych, do których transakcja żąda dostępu eliminuje dostęp do nich przez inne transakcje w czasie, gdy ograniczenia integralnościowe RBD mogą być przejściowo naruszone.

Wyróżniamy dwa podstawowe typy blokad: *blokadę dla odczytu*, oraz *blokadę dla zapisu*.

Mówimy, że dwie blokady są *kompatybilne*, jeżeli mogą być jednocześnie założone na tę samą daną przez dwie różne transakcje. W przeciwnym razie mówimy o blokadach *niekompatybilnych*.

Najszerzej stosowany w praktyce algorytm metody blokowania to *algorytm blokowania dwufazowego* (ang. (Two-Phase Locking, 2PL). Istotą tego algorytmu jest wymaganie, aby wykonywanie każdej transakcji przebiegało w dwóch fazach: w fazie blokowania oraz w fazie odblokowania. W fazie blokowania transakcja musi uzyskać blokady wszystkich danych, do których będzie dokonywać dostępu. Moment założenia wszystkich żądanych blokad, równoznaczny z zakończeniem fazy blokowania, jest momentem (punktem) akceptacji. W algorytmie 2PL odczyt danej jest możliwy natychmiast po założeniu blokady tej danej, a więc w fazie blokowania, natomiast zapis jest możliwy dopiero po osiągnięciu przez transakcję punktu akceptacji, a więc w fazie odblokowywania. Ściśle biorąc, operacja zapisu jest realizowna następująco. Założenie blokady dla zapisu jest równoznaczne z wykonaniem tzw. *zapisu wstępnego* w obszarze

roboczym związanym z zapisywaną daną. Zapis właściwy, jest realizowany w fazie odblokowania, w momencie zdejmowania blokady tej danej. Polega on na uaktualnieniu w katalogu wskaźnika do danej, tak aby wskazywał on na obszar roboczy, który staje się miejscem przechowywania danej. Taka procedura gwarantuje atomowość transakcji.

4.2 Metoda porządkowania według etykiet czasowych

Podstawowym pojęciem wykorzystywanym w metodzie porządkowania według etykiet czasowych (ang. Time-stamp Ordering, T/O) jest pojęcie *etykiety czasowej*. Z każdą transakcją jest związana etykieta czasowa jej inicjowania oznaczona przez TS(T$_i$), która jest konkatenacją stanu zegara fizycznego w chwili inicjowania transakcji i jednoznacznego identyfikatora stanowiska komputerowego, na którym transakcja T$_i$ jest inicjowana. Również z każdą daną $x \in D$ są związane dwie etykiety czasowe: etykieta czasowa odczytu R_ts(x) oraz etykieta czasowa zapisu W_ts(x). Etykiety te przyjmują wartość etykiety czasowej inicjowania transakcji, która jako ostatnia zrealizowała odczyt lub zapis tej danej.

W metodzie porządkowania według etykiet czasowych zezwala się transakcji na odczyt danej, jeśli jej etykieta czasowa inicjowania jest większa niż etykieta czasowa zapisu tej danej. Natomiast na zapis zezwala się wówczas, gdy etykieta czasowa inicjowania transakcji jest większa od obu etykiet czasowych danej.

4.3 Metoda legalizacji

Podstawową przesłanką, która legła u podstaw metody legalizacji, jest założenie, że konflikty dostępu pomiędzy transakcjami występują na tyle rzadko, że większość współbieżnych realizacji jest poprawna. Zamiast więc odrzucać lub opóźniać wykonywanie transakcji, jak to ma miejsce w metodach blokowania oraz porządkowania według etykiet czasowych, należy wykonać każdą transakcję, a po jej wykonaniu sprawdzić, czy w trakcie wykonywania transakcji nie wystąpiły konflikty dostępu mogące

prowadzić do naruszenia spójności RBD. W przypadku, gdy wynik testu jest negatywny transakcja jest wycofywana i restartowana.

W metodzie legalizacji wprowadza się trzy fazy realizacji transakcji: fazę odczytu, legalizacji i zapisu. W fazie odczytu są dokonywane odczyty, i wstępne zapisy. W momencie zakończenia wykonywania transakcji logicznej, następuje przejście do fazy drugiej - fazy legalizacji. W fazie tej sprawdza się, czy zaakceptowanie transakcji nie naruszy spójności RBD, innymi słowy czy po zaakceptowaniu transakcji wynikowa realizacja pozostanie uszeregowalna. Jeżeli realizacja pozostanie uszeregowalna to transakcja jest akceptowana i przechodzi do fazy zapisu. W przeciwnym razie następuje wycofanie transakcji i jej restart.

5. Podsumowanie

Z kombinacji modeli i metod sterowania współbieżnością w SRBD wynikają konkretne algorytmy sterowania. Skonstruowano ich do chwili obecnej kilkaset. Reprezentatywny przegląd oraz dyskusję tych algorytmów można znaleźć w [1].

Bibliografia

[1] W.Cellary, E.Gelenbe, T.Morzy: "Concurrency Control in Distributed Database Systems", North Holland, Amsterdam, New York, Oxford, Tokyo, 1988.

Jesienna Szkoła PTI
Mrągowo, Grudzień 1988

OCHRONA PRAWNA PROGRAMÓW NA ŚWIECIE

Jacek Irlik
Instytut Systemów Sterowania
ul. Armii Czerwonej 101
40-157 Katowice

1. Na forum Polskiego Towarzystwa Informatycznego podkreślano niejednokrotnie stwierdzenie, że jedną z podstawowych przyczyn hamujących rozwój przedsięwzięć w zakresie produkcji oprogramowania komputerowego jest brak jednoznacznego stanu prawnego w odniesieniu do stosunków własnościowych w sferze tej kategorii dóbr. /Dobra te mają charakter niematerialny i kwalifikacja wspomnianych stosunków wymaga odwołania się do uregulowań w zakresie tzw. własności intelektualnej, w szczególności do prawa autorskiego, które nie wypowiada się jednoznacznie w kwestii oprogramowania komputerowego/.

Wytwórca, który inwestuje w opracowanie systemu oprogramowania, chce w oczywisty sposób wiedzieć, że skalkulowana o przewidywaną wielkość kręgu nabywców cena kopii systemu bę-

dzie naprawdę ceną wyrobu jednostkowego. Nie hamowana możliwość kopiowania programów i rozpowszechniania kopii bez udziału i wiedzy wytwórcy nie daje podstaw rozsądnej kalkulacji ani bezpieczeństwa w obrocie.

Wytwórca, który inwestuje w opracowanie rozwiązań stanowiących o wartości tworzonego oprogramowania, chce wyeliminować możliwość wykorzystania rozpowszechnianych przez siebie kopii, udostępnianych dla bezpośredniego użytku nabywców, jako źródła dla innych wytwórców, wykorzystujących te rozwiązania do tworzenia produktów konkurencyjnych bez ponoszenia nakładów na takie opracowania.

Wytwórca nie mający takiej pewności nie jest skłonny do ponoszenia nakładów na nowe rozwiązania.

Niezależnie od oceny sytuacji z punktu widzenia wytwórcy istnieje ocena sytuacji z punktu widzenia użytkowników, przekonanych, że oprogramowanie nic nie kosztuje /poza kosztami reprodukcji/. Rozpowszechnianie się takich przekonań stanowi dodatkowy element hamujący rozwój.

2. Problem "piractwa" software'owego pojawił się w Polsce z całą wyrazistością dopiero w okresie ostatnich pięciu lat - w związku z gwałtownym rozwojem rynku mikrokomputerowego. Do tego okresu rozważania na temat stanu prawnego w zakresie stosunków własnościowych w odniesieniu do programów komputerowych miały bardziej teoretyczny charakter. Wynikało to w dużym stopniu z charakteru stosunków gospodarczych, zgodnych z modelem

nakazowo-rozdzielczym. Rozważania te podejmowane w literaturze prawniczej koncentrowały się głównie na kwestii, czy programy komputerowe można uznać za dzieła chronione przepisami obowiązującego w Polsce prawa autorskiego. Wykaz literatury poświęconej tej kwestii znaleźć można w książce B.Czachórskiej p.t. "Umowy w zakresie informatyki i ochrona programów komputerowych", wydanej przez Ossolineum w 1980 r.

Pomijając rozbieżność pomiędzy prezentowanymi we wspomnianej kwestii stanowiskami należy zaznaczyć - co podkreślali również ich autorzy - że prawo autorskie w swym dotychczasowym ujęciu nie w pełni odpowiada bogactwu form, w jakich występowa- może oprogramowanie komputerowe.

W czasopiśmie The Computer Journal, vol. 31, nr 1 z 1988 r. publikowany jest artykuł R.A.Clarke'a p.t. "Judicial Understanding of Information Technology:The Case of the Wombat ROM", opisujący przebieg procesu wszczętego w Australii w 1983 r. przez Apple'a przeciwko firmie Computer Edge o naruszenie praw autorskich do dwóch programów zapisanych w ROMie komputera Wombat, będącego tajwańską kopią komputera Apple II. Sprawa ta, jej przebieg oraz niejednogłośne werdykty wydawane w rozprawach kolejnych instancji jest znakomitą ilustracją trudności, na jakie można napotkać przy kwalifikowaniu prawnym produktów software'owych w oparciu o tradycyjnie interpretowane prawo autorskie.

Dotychczasowe rozważania w odniesieniu do wspomnianej

kwestii nie obejmowały ponadto innej kwestii z poprzednią związanej: jaka jest wzajemna relacja prawa przysługującego producentowi oprogramowania /inwestorowi, pracodawcy zespołu informatyków/ do prawa przysługującego rzeczywistym autorom lub współautorom oprogramowania /projektantom, programistom/. Obowiązująca w Polsce ustawa o prawie autorskim /Dz.U. z roku 1952, nr 34, poz. 234/ stanowi, że prawo autorskie przysługuje twórcy chyba, że przepis szczególny stanowi inaczej. W przypadku oprogramowania brak jest takich przepisów szczególnych z czego wynika, że jeśli uznać prawo autorskie jako przysługujące do programu, to prawo to jest prawem twórcy, a nie przedsiębiorstwa.

Wychodząc z założenia, że program komputerowy jest przedmiotem stosunków prawnych jakościowo odmiennym od tradycyjnie rozumianego utworu "literackiego, naukowego lub artystycznego", należy postawić kwestię nowego uregulowania praw autorskich w zakresie obejmującym programy komputerowe, określającego jednoznacznie sytuację prawną zarówno autorów, jak i producentów oprogramowania.

3. W Polskim Towarzystwie Informatycznym podjęto dyskusję na temat znaczenia uregulowania stosunków własnościowych w odniesieniu do oprogramowania komputerowego dla rozwoju zastosowań informatyki w Polsce.

Stanowisko PTI w tej sprawie wyrażone zostało w uchwale II Walnego Zjazdu Delegatów PTI z maja 1987 r.

Reasumując dotychczasowe dyskusje, w PTI uważa się, że:

- uregulowanie stanu prawnego w zakresie stosunków własnościowych w odniesieniu do oprogramowania komputerowego jest warunkiem koniecznym dla rozwoju przemysłu oprogramowania w Polsce, bez istnienia którego nie będzie można mówić o postępie w zastosowaniach informatyki,
- zjawisko "piractwa" w zakresie oprogramowania powoduje wiele i może spowodować jeszcze więcej szkód w skali gospodarki narodowej poprzez fakt stosowania niekompletnego, niewłaściwie udokumentowanego lub uszkodzonego w wyniku łamania zabezpieczeń,
- kopiowanie na szeroką skalę oprogramowania zagranicznego bez zgody uprawnionych i wbrew ich woli stanowi łamanie zasad już przyjętych w obrocie w krajach producentów i jest poważną barierą dla rozwoju prawidłowych stosunków handlowych z tymi producentami,
- sytuacja, w które można za darmo otrzymać pirackie kopie oprogramowania, stwarza społeczne przekonanie, że oprogramowanie nic nie kosztuje lub kosztuje niewiele, przekonanie błędne i mogące mieć poważne skutki dla podejmowanych przez podmioty gospodarcze decyzji.

W styczniu b.r. Zarząd Główny PTI powołał stałą komisję ZG d/s prawnych uwarunkowań produkcji oprogramowania. Komisja ta stanowi organ, który m.in. wypracowywać ma stanowisko PTI w odniesieniu do pożądanego zdaniem Towarzystwa kształtu re-

gulacji prawnych w obszarze produkcji oprogramowania. Deklaracja gotowości współpracy Towarzystwa z organami państwowymi, prowadzącymi prace legislacyjne w tym zakresie, wyrażona została w liście Prezesa PTI, Prof. dr hab. A.Bliklego, skierowanym do Marszałka Sejmu PRL w maju b.r.
Założenia wstępne dla pożądanych uregulowań są, jako materiał do dyskusji w Towarzystwie, załączone do tekstu niniejszego referatu.

4. Dzięki oficjalnym kontaktom PTI z Brytyjskim Towarzystwem Komputerowym /British Computer Society/ komisja jest w posiadaniu opracowania przeglądowego na temat ochrony prawnej producentów software u w różnych krajach świata, wydanego przez współpracującą z BCS firmą prawniczą Baker McKenzie. Opracowanie zaktualizowane jest zgodnie ze stanem prawnym do końca 1987 r. W niniejszym referacie omówiona zostanie sytuacja w zakresie ochrony prawnej producentów oprogramowania w USA oraz niektórych państwach Europy Zachodniej /Wielka Brytania, RFN, Francja/ - w oparciu o to opracowanie.

5. Producenci oprogramowania w USA oraz wielu krajach Europy Zachodniej mogą na gruncie obowiązującego w ich krajach stanu prawnego szukać ochrony przed działaniami, o których mowa była na początku niniejszego wystąpienia. Dostępne dla nich środki prawne nie ograniczają się do jednej, ale na-

leżą do kilku sfer uregulowań.

W krajach tych producenci oprogramowania poszukiwać mogą ochrony /ostatecznie w drodze postępowania sądowego/ w oparciu o:

- obowiązujący stan prawny w zakresie prawa autorskiego,
- przepisy dotyczące tzw. nieuczciwej konkurencji,
- przepisy dotyczące ochrony tajemnicy przemysłowej /handlowej/,
- przepisy dotyczące ochrony zarejestrowanych znaków handlowych,
- postanowienia kontraktów zawieranych w obrocie oprogramowaniem.

W USA, niezależnie od drogi sądowej, producenci mogą żądać ochrony przed naruszeniami ich praw wnioskując do Służby Celnej /Customs Service/ lub do Komisji Handlu Międzynarodowego /International Trade Comission/ - w przypadku, jeśli produkt naruszający ich prawa jest przedmiotem importu.

W dalszej części niniejszego referatu przedstawione zostaną bliżej kwestie dotyczące możliwości wykorzystania stanu prawnego we wspomnianych sferach uregulowań - w USA, Wlk. Brytanii, RFN i Francji.

6. W poszukiwaniu ochrony swoich interesów na gruncie prawa autorskiego, w najpewniejszej sytuacji są producenci w USA. W sprawie Williams Electronics przeciwko Artic International /1982/ Sąd Apelacyjny USA stwierdził, że "the co-

pyrightability of computer programs is firmly established after 1980 amendment to the Copyright Act 1976". Ponadto w sprawie Apple przeciwko Franklin /1983/ stwierdzono, że "program komputerowy, niezależnie od tego, czy w wersji źródłowej, czy w wynikowej, jest dziełem literackim ...Cil...program komputerowy w postaci wynikowej zapisanej w ROM jest przedmiotem prawa autorskiego."

W 1984 r. Komisja Handlu Międzynarodowego USA w sprawie wszczętej przez Apple'a wykluczyła z rynku amerykańskiego niektóre komputery tajwańskie i singapurskie na podstawie stwierdzenia, że ich syatemy operacyjne naruszają prawa autorskie Apple'a do systemu operacyjnego.

Interesujące są przesłanki stwierdzenia, że określony produkt podobny stanowi bezprawne opracowanie /adaptację/ produktu oryginalnego.

We wspomnianej wyżej sprawie z 1984 r. Komisja stwierdziła naruszenie pomimo, że jedynie 18% kodu wynikowego w porównywanych systemach było identyczne, a 25% było identyczne przy zignorowaniu dyslokacji niektórych rozkazów.

Komisja, orzekając o naruszeniu, wzięła pod uwagę, że identycznych było 23 z 32 najbardziej istotnych procedur systemowych - przy 70 procedurach w ogóle.

W sprawie SAS Institute, Inc. przeciwko S H Computer Systems, Inc. /1985/ czy w sprawie Whelan Associates, Inc. przeciwko Jaslov Dental Laboratory, Inc. /1985/ program oryginalny oraz program uznany za naruszającą prawa autorskie adaptację, były

programami przeznaczonymi dla różnych rodzajów komputerów, napisanymi w różnych językach. Orzekając o naruszeniu w pierwszym z przypadków sąd stwierdził, że nastąpiło "bezprawne skopiowanie organizacji programu oraz szczegółów struktury". W drugim z przypadków sąd orzekł, że "sposób przedstawienia idei zawarty w programie komputerowym jest chroniony prawem autorskim nawet wówczas, gdy musi być zmieniony, a nawet ulepszony po to, aby mógł być zastosowany na innym rodzaju komputera - ze względu na inną organizację jego sterowania." Informacje te pozwalają uświadomić sobie fakt, że sam zapis ustawowy, obejmujący ochroną programy komputerowe nie może być interpretowany w oderwaniu od rozważań o charakterze "softwarowo-inżynierskim" - pomijając oczywiste przypadki prostego kopiowania.

Sytuacja w Europie Zachodniej jest ugruntowana w mniejszym stopniu. W krajach tych były już rozstrzygane sprawy, w których potwierdzono, że prawo autorskie ma zastosowanie w przypadku programów komputerowych, choć istnieje nadal wiele wątpliwości co do zakresu tej ochrony.

Sądy francuskie potwierdzały, że oprogramowanie jest przedmiotem chronionym prawem autorskim np. w sprawach:

- Atari przeciwko Valadon /1984/,
- Apple przeciwko Golem /1983/,
- Apple przeciwko Segimex /1983/.

Jednoznaczna ochrona wynika z ustawy z 3 lipca 1985 r., która

weszła w życie z dniem 1 stycznia 1986 r.

W odróżnieniu od ustawy o prawie autorskim z 1957 r. regulacja z 1985 r. stanowi w odniesieniu do programów, że np.:

- programy komputerowe tworzone przez pracownika w ramach zatrudnienia należą do pracodawcy /art. 45/
- zabronione jest używanie programów komputerowych bez upoważnienia, a kopiowanie dozwolone jedynie dla celów archiwalnych /art. 47/.

W następstwie kilku spraw rozstrzyganych przez sądy krajowe RFN w 1983 r. Federalny Sąd Najwyższy orzekł w 1985 r. /Inkasso-Programm/, że prawo autorskie ma w zasadzie zastosowanie w sprawach dotyczących programów komputerowych.

W stanowisku tym zwrócono jednak uwagę na konieczność wymagania "odpowiedniego stopnia oryginalności, wykraczającej poza rzemiosło przeciętnego programisty - nie tylko w odniesieniu do programów prostych, ale również w odniesieniu do programów złożonych i pracochłonnych."

Stanowisko Federalnego Sądu Najwyższego potwierdzone zostało nowelą prawa autorskiego, uchwaloną 24 czerwca 1985 r., wprowadzającą programy nie listę dzieł chronionych.

Od 16 września 1985 r. wszedł w życie w Wielkiej Brytanii akt prawny "Copyright /Computer Software/ Amendment Act" nowelizujący prawo autorskie z 1956 r. i zawierający uregulowanie dotyczące programów komputerowych. W uregulowaniu tym ustanawia się np., że:

- ustawa z 1956 r. stosuje się do programów komputerowych tak, jak do dzieł literackich,

- wersja programu, która powstaje w wyniku przetłumaczenia go na inny komputer lub język programowania, jest adaptacją stanowiącą naruszenie prawa autorskiego,
- utworzenie programu bezpośrednio w pamięci komputera rodzi jego ochronę, a stąd, że skopiowanie programu do pamięci komputera wymaga zgody właściciela prawa autorskiego.

W odniesieniu do stanu prawnego w Europie Zachodniej D.G. Jerrard /firma Baker McKenzie/ stwierdza: "pomimo braku orzecznictwa wyższych instancji sądowych, orzecznictwo instancji niższych oraz opinia doktryny wskazują, że zakres ochrony powinien objąć większość postaci software'u nawet w tych krajach, w których prawo autorskie nie było dotychczas nowelizowane. Nawet więc, jeśli uznać, że sytuacja nie jest wszędzie całkowicie jasna w tej materii, można na ogół oczekiwać, że ochrona obejmie software aplikacyjny i systemowy, zarówno w postaci źródłowej, jak i wynikowej, włączając programy zapisane w ROM".

7. W dążeniu do swoich interesów producenci oprogramowania - niezależnie od powoływania się na prawa autorskie - poszukują ochrony prawnej w innych sferach uregulowań. Jedną z nich jest sfera regulacji dotyczących zarejestrowanych znaków handlowych. Dokonanie przez producenta rejestracji znaku handlowego w odniesieniu do określonej klasy wyrobów chroni go w ten sposób, że uznane zostają za bezprawne:

- wykorzystywanie tych znaków przez inne osoby dla oznaczania ich wyrobów wprowadzanych do obrotu,

- usuwanie, zamazywanie lub zniekształcanie znaków umieszczanych przez uprawnionego producenta na swoich wyrobach.

W USA nie ma wątpliwości, co do możliwości powoływania się na ochronę prawną znaków handlowych stosowanych do oznaczania oprogramowania komputerowego.

Wątpliwości w innych krajach, hajogólniej mówiąc, związane były z kwalifikacją oprogramowania w odniesieniu do klasyfikacji wyrobów, które oznaczone są takimi znakami. W większości krajów europejskich możliwość stosowania prawnie skutecznych znaków handlowych w przypadku oprogramowania związana jest z możliwością stosowania ich do oznaczania usług. W niektórych krajach bowiem możliwość rejestracji znaku handlowego obejmuje jedynie wyroby będące rzeczami /mające materialny charakter/.

Stanowisko, że dostarczanie produktów softwareowych i udzielanie prawa ich użycia jest świadczeniem usług, wyraził parę lat temu brytyjski urząd znaków handlowych /Trade Mark Registry/.

Obecnie jednak urząd ten akceptuje stosowanie znaków handlowych w przypadku programów komputerowych zakodowanych na taśmach, kartach lub dyskach, a także w przypadku "programów komputerowych jako takich".

Warto wspomnieć, że zgodnie z obowiązującą od 1 lipca 1985 r. nową klasyfikacją wyrobów oznaczanych znakami towarowymi Urząd Patentowy PRL dopuszcza /w klasie 42/ rejestrację znaków towarowych dla oznaczania usług informatycznych.

8. Niezależnie od ochrony znaków handlowych w większości krajów europejskich ukształtował się system prawny ochrony przed tzw. "nieuczciwą konkurencją". Pojęcie to, najogólniej mówiąc, obejmuje wykorzystywanie przez określoną osobę pozycji innego producenta na rynku w celu promocji produktów konkurencyjnych.

Może ona polegać na naruszaniu nawet nie zarejestrowanych znaków handlowych /jeśli utrwaliły się w świadomości odbiorców/ kopiowaniu produktów lub ich wyglądu zewnętrznego, a nawet na określonym sposobie prowadzenia reklamy.

We Francji nieuczciwa konkurencja sankcjonowana jest art. 1382 Francuskiego Kodeksu Cywilnego.

W RFN sądy w szeregu przypadkach uznały, że rzucająca się w oczy intuicja produktu innego wytwórcy stanowi akt nieuczciwej konkurencji, jeżeli imitator zbiera w ten sposób korzyści należne producentowi oryginału.

W Wielkiej Brytanii nie ma ustawowej ochrony przed nieuczciwą konkurencją. Można jednak dochodzić ochrony na gruncie prawa "common low" w przypadkach, w których określona osoba sprzedaje swoje towary stwarzając mylne przekonanie, że pochodzą one od innego producenta - działając tym samym na jego szkodę.

W Polsce obowiązuje formalnie do chwili obecnej ustawa z 1926 r. o zwalczaniu nieuczciwej konkurencji.

9. Dodatkowe możliwości ochrony interesów producentów oprogramowania stwarza system ochrony tajemnicy przemysłowej /handlowej/. System ten w pewnym stopniu wiąże się z systemem ochrony przed nieuczciwą konkurencją, której jednym z przejawów może być wykorzystywanie, uzyskanych w sposób sprzeczny z "dobrymi obyczajami", tajemnic produkcyjnych czy handlowych określonego producenta. Warunkiem możliwości powołania się na fakt naruszenia tajemnicy jest wykazanie, że dołożono należytej staranności w ochronie tej tajemnicy /zarówno przez stworzenie ochrony faktycznej, jak i przez odpowiednie klauzule w umowach o udostępnienie jej innej osobie/ oraz, że jest to w istocie tajemnica, a nie element wiedzy publicznej.

Możliwość powołania się na fakt bezprawnego naruszenia tajemnicy przemysłowej /handlowej/ mają producenci oprogramowania we wszystkich omawianych w niniejszym referacie krajach.

Ustawa o zwalczaniu nieuczciwej konkurencji, obowiązująca w Polsce do dziś, zawiera przepis dający podstawę do pozywania osób, które w sposób sprzeczny z dobrymi obyczajami weszły w posiadanie tajemnicy powoda.

Należy w tym miejscu zaznaczyć, że na przepisy o ochronie przed nieuczciwą konkurencją czy na przepisy o ochronie tajemnicy przemysłowej może powołać się jedynie ten, czyje dobro zostało naruszone.

Jedynie więc firma MicroSoft mogłaby /i to skutecznie/ pozwać z tytułu nieuczciwej konkurencji polskie firmy o podobnych nazwach, gdyby uznała, że firmy te zagrażają w związku z tym jej pozycji na rynku polskim.

10. Jednym z powszechnie stosowanych na świecie środków prawnych ochrony interesu producenta jest odwoływanie się do postanowień umów, na podstawie których producenci rozpowszechniają wytwarzany przez siebie software. Umowy takie, które w praktyce ukształtowały się jako umowy o charakterze licencyjnym, muszą być oczywiście interpretowane w ramach obowiązującego porządku prawnego, na który składają się - obok prawa cywilnego, również uregulowania m.in. w obszarach, o których była mowa wcześniej w niniejszym wystąpieniu. Należy pamiętać, że stan faktyczny - w przypadku sporu - jest zawsze oceniany przede wszystkim w oparciu o postanowienia zawartej umowy. Przedstawiając w zarysach praktykę w zakresie zawierania umów licencyjnych mamy na myśli umowy zawierane przez producenta z użytkownikiem - pomijając bogactwo układów handlowych spotykanych w obrocie.

Umowy licencyjne, zawierane w krajach, o których mowa w niniejszym wystąpieniu zawierają między innymi postanowienia dotyczące:

a/ definicji postaci oprogramowania będącego przedmiotem licencji,

b/ zakresu dozwolonego użycia oprogramowania,

c/ działań licencjobiorcy zakazywanych przez licencję,

niezależnie od postanowień dotyczących opłat, terminów oraz odpowiedzialności.

ad a/ Umowy licencyjne zawierane z użytkownikami dotyczą zwykle software'u w postaci wynikowej /wykonawczej/ i tę postać określają jako przedmiot. Unika się, co jest oczywiste dla każdego producenta, udostępniania

postaci źródłowej, która czyni znacznie łatwiejszym dostęp do rozwiązań zawartych w oprogramowaniu i ułatwia tym samym sytuację osobom zamierzającym naruszać jego interes. Często użytkownik nalega, aby udostępnić mu postać źródłową, zamierzając rozwijać software we własnym zakresie w obawie, że producent nie będzie właściwie go pielęgnował lub, że może np. zbankrutować.

W niektórych przypadkach udostępniano dawniej postać źródłową, zastrzegając w umowie ochronę jej przed udostępnieniem osobom niepowołanym oraz użycie jedynie w określonym celu i w określonych sytuacjach. Obecnie, W Wielkiej Brytanii, wykształciła się instytucja depozytariuszy, którzy przechowują wersje źródłowe na mocy odpowiednich umów trójstronnych. Umowy takie określają sytuacje, w których wersje te mogą być udostępnione użytkownikowi.

ad b/ Postanowienia w tej grupie zastrzegają najczęściej, że oprogramowanie będące przedmiotem licencji wykorzystywane będzie przez licencjobiorcę wyłącznie do własnego użytku oraz określają jednostkę, na której będzie ona eksploatowana. Zwykle określa się ściśle określoną jednostkę, zezwalając na użytkowanie oprogramowania na innej jedynie w sytuacjach awarii tej, dla której było przeznaczone.

Inne postanowienia w tej grupie mogą np. uprawniać licencjobiorcę do prowadzenia prac nad rozwojem dostarczonego oprogramowania.

Oprogramowanie powstałe w wyniku takich prac poddawane jest zwykle temu samemu reżimowi któremu podlega oprogramowanie będące przedmiotem licencji.

ad c/ Działania zakazywane przez umowę licencyjną obejmują najczęściej:

- używania oprogramowania na innym sprzęcie niż ten, dla którego było licencjonowane,
- sublicencjonowanie oprogramowania na rzecz osób trzecich,
- kopiowanie /zwielokrotnienie egzemplarzy/ oprogramowania, z ewentualnym wyjątkiem kopiowania w celach archiwalnych,
- dekompilowanie, disasemblację, opracowywanie adaptacji na inny język lub komputer,
- dokonywanie zmian w oprogramowaniu,
- zmienianie, usuwanie lub zamazywanie znaków handlowych, znaków prawa autorskiego oraz innych oznaczeń własności, umieszczonych w programach lub dokumentacji,
- używanie oprogramowania w sieci lub w komputerze z wielodostępem,
- używanie oprogramowania w celu świadczenia usług informatycznych.

Należy zwrócić uwagę na to, że postanowienia zawieranych umów licencyjnych mogą być trudne do wyegzekwowania w krajach, w których nie istnieje jednoznaczny stan prawny w zakresie stosowalności prawa autorskiego w odniesieniu do oprogramowania. Dotyczy to zwłaszcza sytuacji, w których - w wyniku powszechnego swobodnego kopiowania programów - rozwiązania zawarte w oprogramowaniu przeszły do tzw. "domeny publicznej", stały się faktycznie elementem stanu publicznej wiedzy.

11. W związku z upowszechnieniem mikrokomputerów oraz wynikającym stąd upowszechnieniem obrotu oprogramowaniem pojawiły się w wielu krajach /w Polsce też/ problem praktyczny, dotyczący sposobu licencjonowania oprogramowania dostarczanego masowo, przez "sprzedawców sklepowych przy głównych ulicach". Rozwiązania brytyjskie stosowane w takich przypadkach, zwane licencjami "shrink - wrapped" lub "tear me open" polegają na tym, że producent umieszcza np. dyskietkę w kopercie, której otwarcie - zgodnie z umieszczonym na niej zastrzeżeniem - rozumiane jest jako oświadczenie woli przyjęcia warunków licencji. Producent ponadto stwarza sytuację, w której otrzymać może formalne potwierdzenie zawarcia umowy, poprzez umieszczenie w kopercie z dyskietką tekstu licencji z adresem zwrotnym i poinformowanie, że jej odesłanie spowoduje dostarczenie nabywcy dodatkowych materiałów informacyjnych na temat nabytego oprogramowania.

Jacek Irlik

Jesienna Szkoła PTI
Mragowo, grudzień 1988

PRZYSPIESZANIE OBLICZEŃ PRZEZ ICH ROZPRASZANIE

dr inż. Wacław Iszkowski
Instytut Informatyki
Politechnika Warszawska
00-655 Warszawa, Nowowiejska 15/19

1. Wstęp

Podstawowym celem informatyki było i jest budowanie coraz szybszych komputerów, a jeżeli staje się to niemożliwe lub zbyt kosztowne ,to wtedy podejmuje się prace nad poszukiwaniem innych metod przyśpieszania obliczeń. Rozwiązania te mogą z natury dotyczyć tylko jeszcze oprogramowania, w projekcie którego możliwe jest lepsze lub gorsze wykorzystanie cech funkcjonalnych sprzętu oraz poszukanie nowego algorytmu dającego inną - mniejszą złożoność obliczeniową dla tego samego problemu. Zagadnienia te mają szczególne znaczenie w przypadku konieczności znacznego przyśpieszenia wykonania pojedynczego programu, zamiast zwiększania przepustowości całego systemu cyfrowego

Opierając się na doświadczeniach z życia codziennego poszukuje się metod podziału problemu obliczeniowego na mniejsze podproblemy, które następnie mogą być wykonywane współbieżnie. Możliwość rozproszenia obliczenia problemu musi uwzględniać cechy wykorzystywanego sprzętu(jego praktyczną równoległość działania) oraz własności oprogramowania systemowego. Równocześnie konieczne jest dostarczenie odpowiedniego narzędzia - języka i translatora - za pomocą którego zapis i podział problemu będzie stosunkowo łatwy. To proste z pozoru rozwiązanie w

praktycznych zastosowaniach staje sie bardzo złożone i co gorsze nie przynoszące często oczekiwanych efektów - w myśl przysłowia "Gdzie kucharek sześć tam nie ma co jeść".

Przedmiotem wykładu jest przegląd podstawowych rozwiązań sprzętowych oraz programowych, których celem jest szybsze, w porównaniu z typowym obliczeniem sekwencyjnym, obliczenie danego problemu. Przegląd rozpoczyna sie od najtańszych,dostępnych nawet u nas w kraju, rozwiązań wykorzystujących standardowy sprzęt mikrokomputerowy oraz odpowiednie oprogramowanie. Do tej klasy zaliczamy typowy system jednoprocesorowy ,ewentualnie z koprocesorami, a także systemy wieloprocesorowe oraz proste sieci lokalne. W drugiej części przeglądu zajmiemy się strukturami sprzętu działającego z szybkościami rzędu setek MFLOPs, którego najpopularniejszym reprezentatem, znanym nam niestety ze słyszenia, jest superkomputer CRAY.

W podsumowaniu wykładu porządkowane są pojęcia związane z konstruowaniem programów obliczeń przewidzianych do rozpraszania w sposób świadomy, uświadamiany lub nieświadomy przez programującego. Wykład jest przygotowany na poziomie podstawowym łącząc w sobie prezentacje architektury sprzętu oraz oprogramowania. W wielu punktach łączy się on z tematami prezentowanymi już w ramach poprzednich Szkół Jesiennych PTI.

2. Rozpraszanie proste

Pojęcia podstawowe

W terminologii związanej z naszym tematem, pomijając problem tłumaczenia na język P, nie wszystkie oryginalne pojęcia są definiowane i używane jednakowo przez różnych autorów. Tutaj przyjmuję , dopuszczając ich dyskusję, następujące nieformalne definicje podstawowych pojęć:

Obliczeniem (ang.computation)jest wykonywanie kodu pewnego programu dla określonego zestawu danych w celu otrzymania wyników (rys.1).

Zadaniem (ang.task) jest całość pracy potrzebnej dla realizacji obliczenia.

Procesem (ang.process) jest wykonywanie pojedynczego zadania, przy czym do jego wykonania może być powołanych jeden lub więcej procesów oraz użytych jeden lub więcej procesorów.

Wątkiem (ang.thread) jest wykonywanie sekwencji instrukcji w ramach określonego procesu, przy czym wątków takich w procesie może być określonych więcej niż jeden.

Dane współdzielone (ang.shared data) są danymi dostępnymi dla wielu procesów.

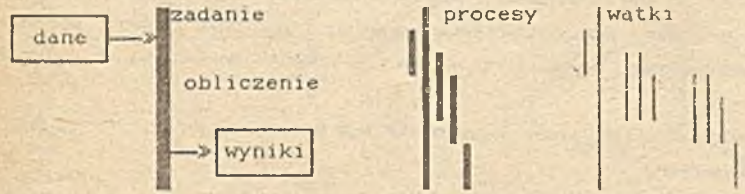
Zamek (ang.lock) jest operacją do synchronizacji współdziałania wątków w oparciu o dane lokalne procesu.

Semafor (ang.semaphore) jest obiektem zlokalizowanym w obszarze danych współdzielonych, będącym zmienną operacji synchronizacji współdziałania procesów.

Wiadomość (ang. messages) jest daną operacji przesyłania informacji pomiędzy procesami.

Równoległość (ang.parallel) jest odnoszona do obiektów statycznych - algorytmów, programów,maszyn, w.których występują elementy mogące być używane równocześnie.

Współbieżność (ang.concurrent) jest odnoszona do obiektów dynamicznych - obliczeń,procesów, wątków, wykonywanych równocześnie.



Rys.1. program => obliczenie => zadanie => procesy => wątki

Pojęcie przyśpieszania będzie tutaj oznaczać poszukiwanie metody szybszego wykonania danego obliczenia po wyczerpaniu podstawowych metod optymalizacji czasowej programu oraz po odpowiednim przygotowaniu systemu cyfrowego (czyli sprzętu i oprogramowania systemowego). Dla porządku wymienimy te - z pewnością znane Czytelnikom (może nie zawsze sobie uświadamiane) od dawna - metody.

1. Zacznijmy od stwierdzenia trywialnego - z systemu usuwamy wszystkie inne programy , aby nawet ich przypadkowe uruchomienie nie wstrzymywało aktywowania naszego gotowego programu.

2. Do zapisu tekstu programu wybieramy taki język programowania i taki

kompilator, aby przy maksimum wygody programisty możliwe było uzyskanie optymalnego czasowo kodu wynikowego . Zakładamy ewentualną konieczność napisania pewnych, wielokrotnie wykorzystywanych i o istotnym czasie wykonywania, procedur w asemblerze danego komputera. Na razie wybór takiego języka nie jest obarczony innymi niż podano wymaganiami.

3. Wykorzystujemy maksymalnie pamięć operacyjną, modyfikując algorytm programu w myśl zasady odwrotnej proporcjonalności czasu wykonywania do wielkości zajętości obszaru danych wewnętrznych. Oczywiście wyszukanie najlepszego pod względem złożoności czasowej algorytmu mamy już za sobą.

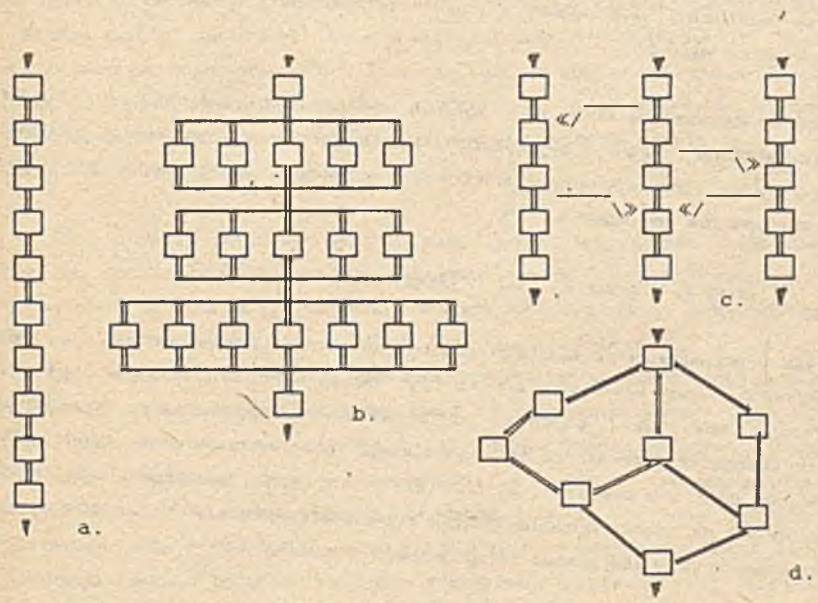
Dalej przyjmujemy, że interesuje nas przyśpieszenie tylko jednego obliczenia, a nie zwiększenie przepustowości systemu dla wykonywania wsadu programów czy sesji interakcyjnych. Po wyczerpaniu podanych wyżej możliwości pozostaje nam tylko rozproszenie tego obliczenia, czyli podzielenie koniecznych do wykonania operacji pomiędzy wiele współbieżnie działających jednostek wykonawczych. Rozpraszanie to może się odbywać na kilku poziomach struktury systemu cyfrowego:

- przez nakładanie lub równoczesne wykonywanie cykli kolejnych instrukcji,
- przez zwiększenie przepustowości jednostki przetwarzającej poprzez podział jednostkowej operacji obliczenia na fazy wykonywane w trybie potokowym,
- przez zwielokrotnienie liczby jednakowych lub specjalizowanych jednostek przetwarzających,
- przez zwielokrotnienie liczby procesorów,
- przez zwielokrotnienie liczby komputerów.

Kryterium rozpraszania jest uwarunkowane założonym stopniem granulacji rozproszenia programu oraz istniejącym w jego algorytmie potencjalnym stopniu równoległości[p. 6]. Stopniem granulacji nazywamy założoną, wynikającą z zastosowanego sprzętu i oprogramowania, rodzajem zrównoleglenia obliczeń programu. Grubą granulacją jest podział treści programu na zadania, z których każde jest złożoną czasową procedurą wymagającą realizacji osobnego algorytmu. Małą granulacją jest zrównoleglenie na poziomie pojedynczych instrukcji czy nawet operacji.

Stopniem równoległości algorytmu jest potencjalna możliwość podziału na obliczenia wykonywane współbieżnie. Stopień ten, wyrażany liczbą możliwych do równoczesnego wykonania obliczeń, jest przeważnie zmienny w ramach całego

programu oraz co ważniejsze jest zależny od przyjętego dla danej metody wykonania obliczenia stopnia granulacji. Graficznym obrazem stopnia równoległości programu jest graf przepływu sterowania (grafy te w zależności od potrzeb i autorów są różnie nazywane - np CFG- Control Flow Graph, PTG-Program Task Graph,TFG-Task Flow Graph, DFG-Data Flow Graph, DAG-Directed Acyclic Graph,etc.). Przykłady charakterystycznych grafów przepływu sterowania dla algorytmów o odmiennie wyznaczanych stopniach równoległości, pokazano na rys.2.



Rys.2. Przykłady grafów przepływu dla algorytmów:
a. szeregowego (strumieniowego),
b. równoległego (wektorowego, macierzowego),
c. równoległego (strumienia słabo związanego),
d. ogólnie równoległego.

Rozpraszanie tanie

Zacznijmy poszukiwanie metod rozpraszania obliczenia od rozwiązań najprostszych , możliwych do natychmiastowego zastosowania (również pewnie znanych większości Czytelnikom) . Analiza prawie każdego programu pod kątem możliwości przyśpieszenia jego wykonania, szybko prowadzi do wniosku, że największe straty powstają w momencie oczekiwania procedury na

wprowadzenie kolejnych wartości oraz procedury wyprowadzania wyników. Oczywiście zupełnie nas nie interesuje fakt, że nieco lepszy system operacyjny realizuje operacje we-wy asynchronicznie - wykorzystując wolny procesor na rzecz innego programu. Możemy jednak na tej podstawie podać najprostsze , wręcz trywialne ,metody przyśpieszania obliczenia:

1. Do wprowadzania danych i wyprowadzania wyników wykorzystujemy najszybsze urządzenie czyli pamięć dyskowa, a niekiedy dysk wirtualny zlokalizowany w pamięci operacyjnej. Zapełnianie plików danymi, czy przesyłanie ich na dysk wirtualny wykonujemy innym programem. Podobnie realizujemy drukowanie wyników.

2. Program dzielimy na mniejsze fragmenty - programy realizujące kolejne wyodrębnione kroki algorytmu. . Następnie wykorzystując systemowe mechanizmy przekazywania wartości pomiędzy programami powołujemy przetwarzanie potokowe :

   progr.A !! progr.B !! ... !! progr.Z

   Zysk z przyśpieszenia obliczeń zanotujemy dopiero wtedy, jeżeli to samo obliczenie mamy wykonać dla kilku czy lepiej kilkuset zbiorów danych. Trzeba też tutaj pamiętać o dodatkowych opóźnieniach spowodowanych koniecznością przekazania do następnego programu wyników czesciowych. Należy przy tym pamiętać, że cały czas mamy jeden procesor. Taki podział programu na części pozwala jedynie zrównoważyć obciążenie zasobów systemu cyfrowego przy obliczaniu tylko jednego problemu.

3. Poszukujemy możliwości przerzucenia części obliczeń na inny procesor. Wybrane fragmenty obliczeń musza być łatwo wyróżnialne - możliwe do zsynchronizowania - logicznie lub nawet sprzętowo. Wprowadzenie bowiem dodatkowych mechanizmów synchronizacji na pewno zwiększy czas obliczeń (do tego problemu powrócimy w rozdziale 3 - na razie pozostańmy przy prostszym rozwiązaniu).

Środki do rozpraszania

Spróbujemy teraz poszukać praktycznej odpowiedzi na jakim sprzecie i z jakim oprogramowaniem możemy zrealizować podane postulaty. Zakładamy, że

mamy do dyspozycji dowolny sprzet i oprogramowanie, pod warunkiem że będzie ono w "kolorze czarnym" czyli oryginały i klony IBM PC i PS/2 z systemem MS-DOS, OS/2, ???X, itp. wraz z dostepnym wyposażeniem sprzętowym i programowym.

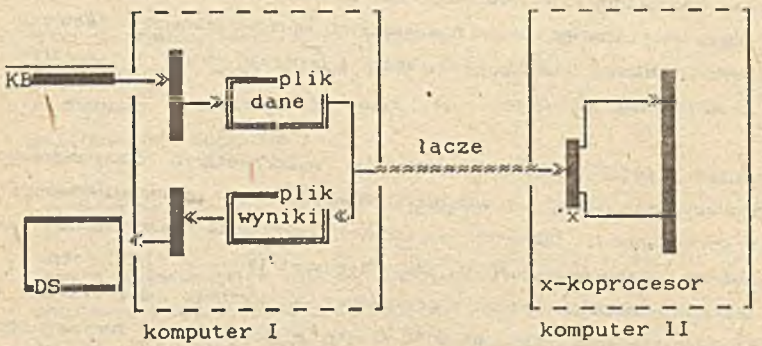
Pkt.1 możemy już zrealizować na każdym komputerze osobistym wykorzystując system operacyjny MS-DOS. Przełączenie standardowych we-wy programu na współpracę z plikami jest łatwe. Pliki te mogą być umieszczone na dysku wirtualnym - co pozwala między innymi wykorzystać pamięć ponad 1 MB (której inaczej system MS-DOS nie widzi).Możemy też wykorzystać pracujący w tle program drukowania Print. Istnieją też bardziej wyrafinowane programy przechwytywania wydruków do plików pomocniczych wyprowadzanych następnie w miarę szybkości działania drukarki. Oczywiście i program Print i te inne obciążają czas procesora.

Dla realizacji pkt.2 konieczne jest już wykorzystanie odpowiedniego systemu operacyjnego o cechach wieloprocesowości(lub wielozadaniowości - tutaj jest to synonim). Spośród dostępnych systemów najżyczliwiej jest oceniany systemu klasy Unix - XENIX, QNX, Microport Unix V, IPIX, itp. Niestety dla naszych celów trzeba zdecydowanie go odrzucić ,gdyż system ten - zresztą bardzo elegancki, chociaż dość trudny w opanowaniu - zużywa sporo czasu procesora i pamięci operacyjnej na tzw. "miłość własną", czyli na kontrole środowiska oraz wykonywanych w trybie wieloprogramowym , wieloużytkowym i wielodostępnym programów. Wada ta nie dotyczy tylko mikrokomputerów - również dla maszyny VAX 11/780 chęć szybkiego wykonania złożonego czasowo programu powoduje zastąpienie systemu UNIX systemem VMS lub nawet poczciwym RSX-11M.

Drugim kandydatem jest jednoużytkowy, wieloaplikacyjny, wielozadaniowy, wieloprocesowy i wielowątkowy system OS/2 firmy Microsoft Corp. opracowany specjalnie dla komputerów rodziny PS/2 (od modelu 50 wzwyż oraz PC/AT). System ten nie jest jeszcze rozpowszechniony - różne też krążą głosy na jego temat, ale w moim odczuciu powinien on się przebić. W każdym razie do naszych celów nadaje się gdyż dostarcza odpowiednich mechanizmów współbieżnego wykonywania procesów z wątkami oraz ich wzajemnej synchronizacji czy wymiany informacji w postaci potoku lub kolejek wiadomości. Pozytywną cechą tego systemu jest też możliwość wykorzystania całej pamięci operacyjnej (do 16 MB).

Zgodnie z postulatem pkt.3 musimy znaleźć inny procesor do wykonania przynajmniej niektórych obliczeń. Możliwe są tutaj następujące rozwiązania:

1. Wykorzystujemy sieć lokalną (rys.3) - zrealizowaną sprzętowo - przenosząc wprowadzanie danych i wyprowadzanie wyników, łącznie z ich drukowaniem, na inny komputer działający w sieci. W tym rozwiązaniu komunikacja z drugim komputerem odbywa się z wykorzystaniem drugiego procesora we-wy, przy odciążeniu procesora głównego. W ten sposób uzyskujemy praktycznie cała moc procesora tylko na same właściwe obliczenia. Inne zastosowania sieci lokalnych nas tutaj nie interesują.



Rys.3. Sieć lokalna

2. Uzupełniamy sprzęt dodatkowymi koprocesorami specjalizowanymi - do obliczeń zmiennoprzecinkowych (np 80287, 80387), operacji we-wy oraz przetwarzań graficznych (jeżeli nasz program potrzebuje takich operacji). Trzeba tutaj zaznaczyć, że w ten sposób nie uzyskujemy komputera wieloprocesorowego. Różnica polega na tym, że wykonywanie operacji przez koprocesor wstrzymuje działanie procesora głównego. Przyśpieszenie obliczeń uzyskuje się poprzez znacznie szybsze wykonanie danej operacji. Oczywiście kod wynikowy programu musi zostać dopasowany do nowej konfiguracji maszyny. Z reguły odbywa się to poprzez wymianę generowanego kodu po podaniu odpowiedniej opcji translatorowi języka. Programista ma tutaj niewiele do roboty.

3. Uzupełniamy sprzęt dodatkowymi specjalizowanymi:
   - kartami pamięci masowych typu RAM o krótkim czasie dostępu ,
   - procesorami sterowania wyświetlaniem w trybie tekstowym oraz graficznym z autonomicznymi operacjami przetwarzania graficznego,

- autonomicznymi kartami buforowania (np. 256 KB) i wyprowadzania wydruków na drukarkę ,
- autonomicznymi procesorami z wbudowanymi programami - np. translatora języka Ada,
- procesorami wektorowymi, macierzowymi czy nawet automatu komórkowego (patrz dalej) - jak np. CAM-6 o siatce 256*256 4-bitowych procesorów (za 1500 $!).

Podane tutaj przykłady są dostępne dla komputerów typu IBM PC czy PS/2. Nie są one też zbyt kosztowne.

Prezentując rozwiązania umożliwiające przez rozproszenie przyśpieszanie obliczeń nie chciałbym w żadnym przypadku pozostawić złudzeń, że możliwe jest pokonanie problemu mocy obliczeniowej w oparciu nawet o PS/2 Model 80. Te trywialne metody mogą dać jedynie skutek chwilowy - często nie do pogardzenia - ale jeżeli już mamy taki problem to znaczy, że rozwiązania połowiczne wkrótce nic nam już nie dadzą. Konieczne jest więc rozejrzenie się przede wszystkim za szybszym komputerem - niestety już znacznie droższym, bo profesjonalnym oraz znacznie trudniejszym do zdobycia ze względu na embargo.

Wierzę jednak, że w najbliższej przyszłości będzie możliwe przynajmniej pozyskanie sprzętu nazywanego ogólnie stacjami roboczymi, którego najpopularniejszym przedstawicielem jest stacja SUN-3 [20].

3. Rozpraszanie w makroskali

Maszyny MIMD

Po wyczerpaniu prostych możliwości rozpraszania obliczenia konieczne jest poszukanie nowych rozwiązań sprzętowych, umożliwiających współbieżność obliczeń. Oczywiście natychmiast nasuwa się pomysł wykorzystania na rzecz jednego programu wielu procesorów. Tym samym następuje zwielokrotnienie par - strumień instrukcji ze strumieniem danych , a więc zrównoleglenie obliczeń.Na tej zasadzie już od szeregu lat powstają maszyny wieloprocesorowe ściśle lub luźno związane[14].

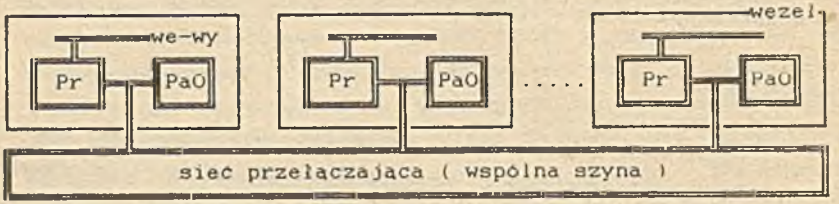
Wieloprocesorem luźno związanym jest sieć kompletnych komputerów połączonych ze sobą podobnie jak sieci lokalne. Istotna różnica polega jedynie na filozofii funkcjonowania takiego układu, zawartej przede

wszystkim w zintegrowanym systemie operacyjnym globalnie nadzorującym współbieżne wykonywanie obliczeń. W sieci lokalnej każda ze stacji ma lokalnie funkcjonujący system operacyjny całkowicie niezależny od działania pozostałych - jego dodatkowym zadaniem jest tylko realizowanie poleceń otrzymywanych z łącza szeregowego - łącznie z pakietem danych ,oraz przekazywanie poleceń.

Przykładem takiego systemu jest 16 mikrokomputerów typu IBM PC połączonych ze sobą w sieć z protokołem dostępu CSMA (Carrier Sense Multiple Access)i sterowanym przez rozproszony system operacyjny DRAGON SLAYER. Idea tego systemu ,realizowanego dla US Army [w 1], jest oparta na Teorii Systemów Interakcyjnych z algorytmem szeregowania zasobów z unikaniem zakleszczeń - opracowanym przez J.Winkowskiego [28]. System ten jest przeznaczony do wykonywania programów zapisanych w języku CSP.

Na podobnej zasadzie w oparciu o kilka mikrokomputerów PC/XT/AT połączonych w sieci Ethernet opracowano rozproszony system przetwarzania wykorzystujący język programowania Martlet - bazujący na języku Ada [24].

Wieloprocesorem ściśle związanym jest zestaw kilku, kilkunastu lub wiecej typowych procesorów, komunikujacych sie poprzez wspólnie dostepna pamieć operacyjną. W systemach tych wspólna pamieć może być wyróżnionym osobnym blokiem lub też być rozproszona przy poszczególnych procesorach,sprzężonych z blokiem lokalnej pamieci operacyjnej.Dostep do pamieci jest realizowany poprzez wspólną szynę lub przezróżnego typu sieć połaczeń procesorów z blokami pamięci (rys.4).



Rys.4. Struktura wieloprocesora

Po zrealizowaniu klasycznych już rozwiązań wieloprocesorów ściśle związanych (C.mmp, Cm*,Cambridge Ring, itp.) złożenie takiej maszyny nie przedstawia już obecnie wiekszych trudności. Możliwe jest na przykład wykorzystanie katalogowych układów serii Intel - mikroprocesora 8086, iSBC 86/12A, iSBC 032,itp. (Multi-Maren [11], Elwro 800[4]).

Obecnie jednak najżywiej analizowanym jest wieloprocesor BBN Butterfly Parallel Processor [2]. Może on być skonfigurowany z od 1 do 256 węzłów. Każdy węzeł (Processor Node) jest osobną kartą zawierającą 8 MHz mikroprocesor MC68000 (lub MC68020 z koprocesorem MC68881), do 1 MB pamięci operacyjnej (plus 3 MB na płycie dodatkowej), koprocesor kontrolera węzła (PNC- Processor Node Controller)zbudowany z układów bit-slice AM2901(Processor Node Controller), szyne We-Wy oraz łącze do sieci przełączającej. Siecią tą może być Butterfly Switch (umożliwiająca równoczesne skonfigurowanie dwóch ścieżek połączeń pomiędzy węzłami, z szybkością transferu do 32 Mb/s)lub sieć typu Omega. Wykorzystując przełącznik - każdy procesor oprócz dostępu do własnej lokalnej pamięci(w czasie 2 µs) ma dostęp do pamięci każdego innego węzła(w czasie około 6 µs). Możliwe jset również inne skonfigurowanie systemu przez dołączenie osobnych bloków pamięci tworzących wspólną pamięć dla procesorów. Procesor BBN jest poprzez sieć Ethernet podłączany do systemów SUN lub VAX. Oprogramowaniem wieloprocesora jest system UNIXo-podobny , Chrysalis oraz język C.

Oprogramowanie wieloprocesora

Dla wieloprocesora typowym językiem programowania jest jeden ze znanych języków współbieżnych : Concurrent Pascal, Edison, Modula-2, CSP, Ada, CHILL, itp [10]. Znając zasady programowania w tych językach, naturalnym jest przypisanie każdemu z procesorów jednego lub więcej procesów (zadań - w Adzie), przy wykorzystaniu mechanizmu komunikacji lub monitorów zlokalizowanych w pamięci wspólnej do wymiany informacji. Tak też jest w rzeczywistości, przy czym już nawet pobieżna analiza tych języków ujawnia wiele niedogodności z punktu widzenia systemu wieloprocesorowego. Brakuje na przykład możliwości zapisu w samym języku sposobu przydziału statycznego lub dynamicznego procesu (zadania) do procesora. Również trudne jest zaimplementowanie mechanizmu synchronizacji pomiędzy procesami (a tym samym procesorami). Kontynuowane są więc prace nad opracowaniem nowych języków tego typu , ze zmodernizowanymi mechanizmami komunikacji, z możliwościami zapisu rozproszenia procesow oraz na przykład dodatkowo zorientowanych obiektowo.

Pomimo wielu już lat rozwoju metodyki programowania równoległego oraz strukturalizacji programu poprzez składnię i semantykę nowopowstających języków, brak jest łatwych metod skonstruowania i weryfikacji poprawności programu równoległego. Co więcej po latach wykrywane są błędy w podstawowych ,klasycznych algorytmach implementacji podstawowych operacji czy rozwiązań klasycznych zadań. Istnieją też duże trudności w uruchomieniu i przetestowaniu tego rodzaju , niederministycznych z założenia,programów.

Efektywność działania

W analizie szybkości obliczeń w systemie wieloprocesorowym badany jest przede wszystkim współczynnik przyśpieszenia obliczeń S(n) , (ang.speedup) związany ze zwielokrotnieniem liczby procesorów.

$$S(n)=Tn/T1$$

T1 - średni czas obliczenia dla 1 procesora,

Tn - średni czas obliczenia przy wykorzystaniu n procesorów.

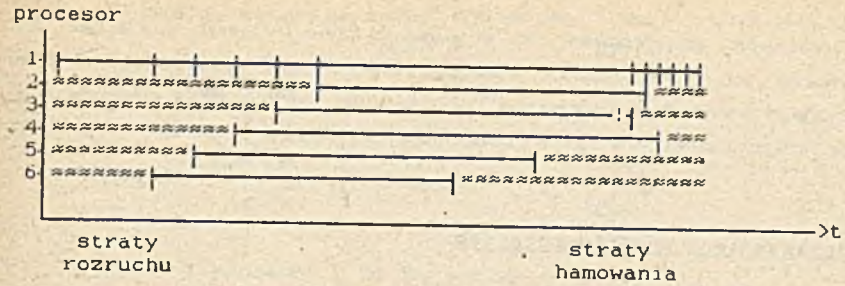
Teoretycznie współczynnik ten powinien być równy liczbie wykorzystywanych w obliczeniu procesorów. W praktyce jednak w wyniku powstawania nasycenia w dostepie do wspólnych obszarów pamięci na poziomie wspólnej szyny systemu oraz na poziomie oprogramowania współczynnik ten maleje. Poprawienie jego wartości jest na przykład możliwe przez umieszczanie w pamięci lokalnej danego węzła całego kodu wszystkich procesów, które mogą być wykonywane w danym procesorze. W ten sposób kosztem zwiększenia zajętości pamięci uzyskuje sie znaczne przyśpieszenie obliczeń.

Do nieuniknionych strat należą też straty wynikające z istnienia na poziomie oprogramowania nasycenia programowego związanego z czasowym blokowaniem dostepu do monitora lub określonego portu dostepu w pamieci lokalnej. Również dokładniejsze analizy przebiegu obliczeń rozproszonych ujawniają powstawanie strat związanych z (rys.5):

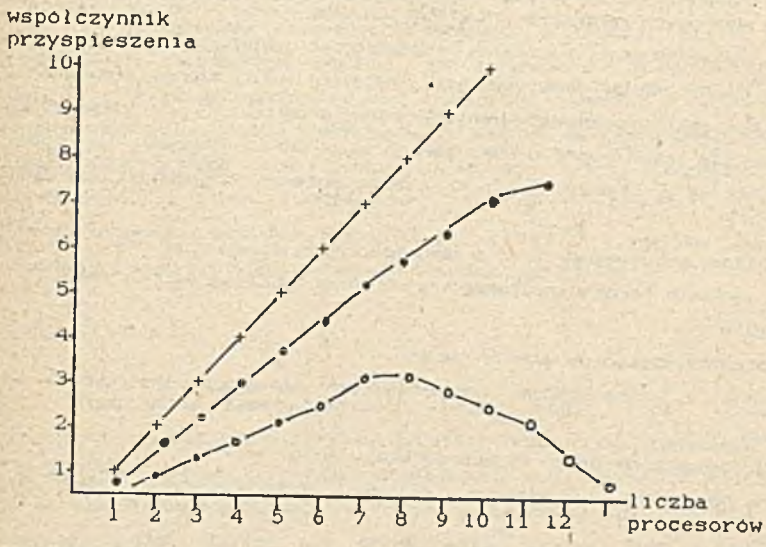
a) początkową bezczynnością procesorów ,powodowaną koniecznością rozdzielenia dla nich podproblemów do obliczania(straty rozruchu),

b) opóźnieniem po znalezieniu wyniku obliczeń przez jeden z procesów, powodowanym koniecznością zatrzymania równolegle biegnących obliczeń (straty hamowania).

Suma tych strat jest przyczyna znacznego spadku efektywności system wieloprocesorowego, mierzonego współczynnikiem przyśpieszenia. Na podstawie wielu analiz możemy z grubsza podać jego zależność od liczby procesorów i sposobu ich powiązań [1]. Przedstawiona na rys.6 zależność wartości współczynnika przyśpieszenia od liczby wykorzystywanych procesorów są charakterystyczne dla większości przeprowadzanych eksperymentów. Inne są tylko wartości optymalnej liczby użytych procesorów.



Rys.5. Straty w wykorzystaniu 6-procesora



Rys.6. Wykres efektywności działania wieloprocesora

Oczywiście zmniejszanie przyśpieszenia ze wzrostem liczby procesorów oznacza całkowitą nieefektywność stosowania wieloprocesora. Przyczyny takiego wyniku ,oprocz już oczywistych, podanych wyżej, mogą być ukryte w samej strukturze algorytmu (por.rys.1), a także w zbyt wolno działajacej sieci przełączającej (wspólnej szynie), czy mało efektywnym oprogramowaniu systemowym wykorzystującym mechanizm synchronizacji poprzez komunikaty, zamiast akurat lepszego dla danego typu obliczeń mechanizmu synchronizacji we wspólnej pamieci.

Reasumując, wieloprocesory są stosunkowo tanie i łatwo dostępne. Dla efektywnego ich wykorzystania wymagają one jednak specjalnego systemu operacyjnego oraz dużych umiejętności w zakresie opracowania algorytmu - odpowiednio rozpisanego na "głosy" obliczeń dla każdego procesora.

4. Rozpraszanie w superkomputerach

Podrasowane maszyny SISD

Już od pierwszych realizacji uniwersalnych komputerów , ich projektanci starali się wykorzystywać naturalną równoległość działania pojedyńczych układów, czy całych bloków funkcjonalnych. Pomijając tutaj szereg rozwiązań , koncepcja maksymalnego podrasowania klasycznej architektury von Neumanna, najpełniej została zrealizowana w komputerach serii CRAY. Maszyny te obok systemów Cyber są praktycznie jedynymi superkomputerami dostępnymi handlowo.

W projekcie architektury tego i jemu podobnych superkomputerów starano się pokonać wszelkie bariery ograniczające szybkość obliczeń. W tym celu ([1,2,12,23,itp]):

- zwielokrotniono dostęp do banków pamięci,
- zbuforowano z wcześniejszym rozpoznawaniem sekwencje instrukcji do wykonania,
- radykalnie zwiększono liczbę rejestrów skalarnych,
- dołączono bufory umożliwiające czasowe przechowywanie zawartości wektorów,
- zwielokrotniono liczbę specjalizowanych, równocześnie działających jednostek wykonawczych, relizujących operacje w trybie potokowym,
- łańcuchowanie operacji wektorowych polegające na rozpoczęciu obliczania następnej operacji natychmiast po uzyskaniu wyników z poprzedniej dla jednego elementu wektora (oczywiście jeżeli ma to sens).

W rezultacie dla tego typu maszyn uzyskano szybkości działania rzędu kilkuset MFLOPS (Million Floating-Point Operations Per Second - mln operacji zmiennoprzecinkowych na sekundę). Osiągi te są możliwe jedynie po specjalnych zabiegach związanych z odprowadzaniem ciepła oraz z minimalizowaniem długości połączeń i wreszcie przy zastosowaniu GaAs (arsenku Galu).

Przykładowo dla maszyn serii CRAY możliwe jest uzyskanie szybkości rzędu:

		czas cyklu	MFLOPS	
CRAY-1		12.5 ns	160	
CRAY X-MP	116M	9.5 ns	210	
	22 M		420	
	48 M		840	
	64? M		13440	! w budowie
CRAY-2	12 G	4.1 ns	500	
	42 G	5 ns	2000	
CRAY-3		1 ns	1000	

116M - 1 procesor i 16 Mega słów 64-bitowych.
42 G - 4 procesory i 2 Giga słów 64-bitowych.

Warto zauważyć, że po dopracowaniu danego modelu maszyny , podwojenie jej mocy następowało przez dodanie drugiego procesora. Potem rozpoczynano opracowanie następnego modelu, już z założenia znacznie szybszego - uzyskując z jednego procesora moc większą niż uprzednio z dwóch. Niestety w klasycznej architekturze komputerów, ze względu na technologię trudno będzie przekroczyć granice czasu cyklu rozkazowego 1 ns. Konieczne jest więc poszukiwanie innych architektur, głównie opartych na znacznym zwielokrotnieniu liczby procesorów - prostych elementów obliczających. Dokonajmy więc krótkiego przeglądu istniejących rozwiązań.

Maszyny SIMD

Podstawowa zasada działania maszyn klasy SIMD zwanych również macierzowymi jest istnienie wielu prostych procesorów sterowanych tym samym strumieniem instrukcji , ale działających na odrębnych całkowicie lub częściowo strumieniach danych. Procesory te mogą się ze sobą komunikować poprzez pamięć wspólną lub bezpośrednio poprzez sieć połączeń. W obu przypadkach rozważane są (i porównywane) różne struktury tych sieci - sieć wielopoziomowa synchroniczna z przełączaniem pakietów - kratowa, kwadratowa, bananowa, kostkowa, tasowana, itp. Problem doboru struktury sieci oraz jej złożoności, wprowadzanego przez nią opóźnienia w przekazywaniu informacji, a także odporności na uszkodzenia jest obecnie najczęściej analizowanym zagadnieniem.

Spory rozgłos zyskał transputer IMS [2] - kostka zawierająca prosty 8,16 lub 32-bitowy procesor (10 MIPS) z wbudowaną lokalną wieloportową pamięcią o pojemności 2KB z szybkością transferu 80 MB/s oraz z czterema dwukierunkowymi szeregowymi łączami o szybkości transferu do 20 Mb/s. Wykorzystując ten element możliwe jest skonfigurowanie sieci złożonej nawet z 16384 węzłów uzyskując moc 196 GFLOPSów (!). Wykorzystując układ transputera opracowano na przykład procesor równoległy FPS T Series

Parallel Processor złożony z 8 węzłów tworzacych kostkę. Dla oprogramowania wykorzystuje się język Occam (opracowany dla transputera) lub VAX/VMS Fortran V.4.2.

Nie sposób przedstawić tutaj nawet najważniejszych rozwiązań tego rodzaju architektur. Na szczęście są one przeważnie dość dobrze opisane w dostepnej literaturze. Znacznie istotniejsze jest pytanie jakie typy architektur są aktualnie badane (czytaj preferowane po analizie uzyskanych wyników badań ich efektywności działania) oraz do jakiego typu architektur, maszyn będzie się dążyć w przyszłości.

Na to ostatnie można odpowiedzieć stosunkowo prosto - wszystkie dotychczas opracowane najszybsze elementy maszyn zostaną włożone do jednego supernadkomputera będącego wielopotokową siecią obliczania funkcji złożonych wektorowo (Multiple Networking for Fast Evaluation of Vector Compound). Maszyna ta będzie (a może już jest ?) złożona z wielu potokowych zespołów funkcjonalnych połaczonych przez buforowana sieć przełaczająca, również potokową. Struktura połączeń w sieci ma być rekonfigurowana do danego problemu obliczeniowego na podstawie jego grafu przepływu danych.

Automaty komórkowe

Kontynuując krótki przegląd najnowszych osiągnięć w zakresie sprzętowego rozpraszania obliczeń zatrzymajmy się jeszcze nad specjalizowanymi automatami komórkowymi przeznaczonymi do symulacji złożonych zjawisk fizycznych. Co ciekawsze, podane niżej opisy zostały zaczerpnięte z czasopisma Byte [27]!

Interesującym zagadnieniem do rozproszenia jest problem modelowania przepływów - np. przepływu turbulencji wokół skrzydła samolotu. Dotychczas przeprowadzano próby praktyczne w tunelach. Czasami próbowano też problem policzyć - niestety ze względu na liczbę obliczeń - było to niewykonalne. Do symulacji modelu opracowano automat komórkowy odwzorowujący ruch cząstek w dużej heksagonalnej kracie. Sąsiednie czastki oddziaływują na siebie lokalnie według łatwo przeliczalnych praw - powodując wzajemne kolizje ze spełnianiem prawa Newtona zachowania momentu. Istnieje 64 (zredukowanych przez rotacje i symetrie do 14)różnych możliwości wzajemnego oddziaływania na siebie cząstek. Pozostawmy jednak fizykom problem oddziaływania cząstek. Dla nas wystarczające jest stwierdzenie ,że pojedyncze lokalne obliczenie jest trywialne.

W Princetown University [27] zaprojektowano i zbudowano specjalizowany komputer ze specjalnie wykonanych układów kostek VLSI . Jedna kostka zawiera 37 procesorow ułożonych w sześciokacie z 84 zewnetrznymi końcówkami dla połaczenia z innymi. Jeden procesor połaczony jest więc 6 liniami wejściowymi i 6 wyjściowymi z 6 sąsiadami. Dla przyspieszenia komunikacji pomiędzy sąsiednimi procesorami wykorzystano rejestr przesuwający. Trwają też próby ze zwielokrotnieniem rejestrów tak aby można było kolejne generacje cząstek przetwarzać potokowo. W aktualnej wersji każda kostka może przetwarzać do 20 milionów komórek na sekunde. Przygotowaniem obliczeń - dostarczaniem danych i odbieraniem wyników - zajmuje się stacja robocza SUN 3 ; ktora wprowadza ograniczenia przetwarzania do 2/3 mln komórek/sek przez jedną kostkę.

Na nieco innej zasadzie do rozwiązania tego problemu wykorzystuje się Connection Machine(CM). CM jest złożona z 65536 procesorów,z których każdy może być połączony z 12 sąsiadami tworząc hipersześcian. Pojedynczy procesor jest prostym bitowym komputerkiem z własną pamięcią 8 KB oraz ze specjalizowaną własną jednostką zmienno-przecinkową. Każdy cykl działania CM składa się z fazy komunikacji z sąsiadem i fazy obliczenia. Pracę CM nadzoruje zewnetrzny komputer. Przygotowuje i przesyła program do każdego z procesorów równoległych, dostarcza dane oraz odbiera wyniki. Może on też dokonać rekonfiguracji CM na przykład jako planarnego komputera macierzowego (każdy procesor ma 4 sąsiadów). Możliwe jest też wpisanie do każdego z procesorów programu wieloprocesorowego tworząc na przykład 200 wirtualnych procesorów równoległych w oparciu o jeden procesor rzeczywisty. Autor [ ] podaje ,że CM może w ten sposób symulować pracę 550 mln procesorów równoległych, a więc uaktualniać stan 10^9 komórek na sek. (50 full speed Princetown kostek !).

Aha, wg [26] Connection Machine produkowana przez firmę Thinking Machine (czy już jesteśmy blisko czasow produkcji nadkomputerów przez myślące komputery ?) ma 15 instalacji i kosztuje 1..7 mln dolarow. Dla porównania CRAY 2 ma już 11 instalacji (CRAY X-MP 123) i kosztuje 20 mln dolarów. I chyba dalej dowiadując się z dostepnych przecieków o rozwoju tego rynku należy szeroko rozpropagować ostatnie zdanie W.M.Turskiego z [26] - "...(Jako gorzką ciekawostkę zauważmy, że sumaryczna liczba tych tylko superkomputerów, ktore obejmuje przytoczona tablica przykładów, dwukrotnie przekracza liczbę podstawowych komputerów zainstalowanych w Polsce.)".

Efektywność obliczeń

Powróćmy jeszcze z przyszłości do problemów standardowych superkomputerów. Dla przykładu popatrzmy na wyniki testowania szybkości wykonania (w MFLOPSach)pewnego programu z zakresu obliczeń inżynierskich (zaczerpnięte z [23]) na różnego typu komputerach. Dla naszych rozważań najbardziej interesujące jest skonstatowanie faktu, że dopiero rozproszenie na potoki obliczeń wektorow ujawnia istotną różnicę w klasie maszyn typu

CRAY w stosunku do pozostałych.

	pojedyncza precyzja	podwójna precyzja
VAX 11/780	0.11	0.06
IBM 3033	1.50	1.30
IBM 370/195	2.60	2.60
	kod skalarny	kod wektorowy
CRAY-1S	3.30	18.70
CRAY X-MP/2	4.20	31.90

Oprogramowanie superkomputerów

Tutaj najważniejszym językiem programowania jest język Fortran 77, a potem w niewielkim stopniu Pascal, C i Occam.

Popularność języka Fortran wynika z przyzwyczajeń w kręgach naukowo-technicznych. Już w latach 60-tych projektowano i implementowano translatory tego języka, zawierające elementy wyszukiwania możliwości zrównoleglania obliczeń w oparciu o zależności pomiędzy zmiennymi wykorzystywanymi w kolejnych instrukcjach. Analiza ta była szczególnie efektywna dla instrukcji cyklu. Metody te, wykorzystujące między innymi grafy przepływu danych, były stale doskonalone. Wada rozwiązań z tamtych lat była przede wszystkim mała efektywność działania takiego translatora.

Okazuje się, że najefektywniejszą metodą zrównoleglania obliczeń jest przekształcanie instrukcji cyklu, w tym również zagnieżdżonych. Oprócz klasycznej instrukcji cyklu z języka Fortran

```
      DO  etyk  I=....
      ..........
etyk  CONTINUE
```

wyróżniono podobnie skonstruowane instrukcje cyklu zawierające dodatkowo informacje o możliwym sposobie wykonywania instrukcji w nich zawartych

DOSPREAD - cykl rozprzestrzeniający kolejne iteracje obliczeń na inne istniejące i wolne procesory ze statycznym (w trakcie kompilacji) lub dynamicznym (w trakcie wykonywania programu) ich przydziałem do obliczeń,

DOALL - cykl nie zawierający żadnych zależności iteracyjnych, a więc

wszystkie kroki cyklu mogą być wykonane współbieżnie,

DOACROSS - cykl zawierający zależności, ale możliwe jest jego wykonanie w trybie wielopotokowym.

Obecnie krystalizują się dwie drogi postępowania. Pierwsza z nich polega na opracowywaniu półautomatycznych narzędzi wspomagających programistę w określaniu możliwości zrównoleglenia obliczeń. Programy te - jak np. PTOOL [1] konwersacyjnie analizują kolejne fragmenty programu wskazując zależności lub ich brak pomiędzy zmiennymi występującymi w bloku instrukcji cyklu.

```
      DO 100 I=1,N                    DO  100  I=1,N
        A(I)=...                        A(I)=...
        ....=A(I)                       ...=A(I-1)
100 CONTINUE                    100 CONTINUE
```

Druga metoda polega na implementacji preprocesora języka Fortran, którego zadaniem jest wykrywanie w programie możliwości zrównoleglenia obliczeń, a następnie zastąpienia znalezionego fragmentu kodu sekwencją wywołań podprogramów realizujących to obliczenie równolegle, łącznie z koniecznymi synchronizacjami. Preprocesory te, mogą dotyczą tego samego języka, różnią się w zależności od rodzaju maszyny docelowej, na której ma działać tak zmodyfikowany program.

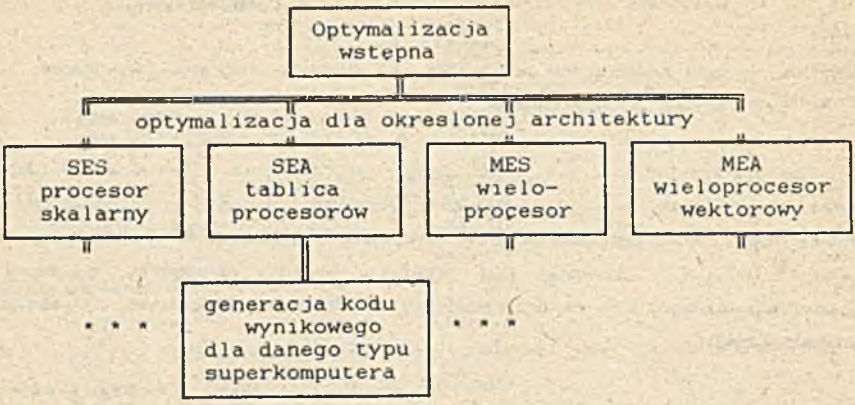
Przykładem takiego zestawu preprocesorów jest zestaw produktów oferowanych przez grupę Kuck and Associates Inc., a przeznaczony dla:

procesora tablicowego ST-100	KAP/ST-100
superkomputera Mark IIa, S-1	KAP/S-1
superkomputera Cyber 205	KAP/205

Na tej idei oparty jest też kompilator Parafrase języka Fortran, którego jedną z pierwszych faz po ogólnej optymalizacji programu jest optymalizacja ukierunkowana na metodę realizacji obliczeń w maszynie docelowej. Optymalizacja ta polegana przekształceniu struktury instrukcji cyklu DO..... w odpowiednią instrukcję cyklu lub w zbiór równoległych zadań - zależnie od typu docelowej maszyny. Przewidziano cztery typy takich maszyn:

- SES - Single Execution Scalar - typowy jednoprocesor,
- SEA - Single Execution Array - tablica procesorów lub przetwarzanie potokowe
- MES - Multiple Execution Scalar - wieloprocesor
- MEA - Multiple Execution Array - wieloprocesor z procesorami wektorowymi,

Strukturę tego kompilatora przedstawia rys.7



Rys.7. Struktura translatora Parafrase

Zaawansowane są już prace na standaryzacja rozszerzeń Fortranu 77, które mają być*zawarte w Fortranie 8x (88 ?). Rozszerzenia te mają umożliwić jawne specyfikowanie operacji wektorowych i tablicowych. Do najważniejszych z nich należą:

- wyrażenia wektorowe i tablicowe łącznie z operatorem przypisania, np.:

```
X = Y
X = X + 0.5
X = X/X(2)
A(1:20,1) = B(J,101:120)
```

- identyfikacja mapowanych fragmentów tablic, np.:

```
IDENTIFY /1:M/ D(I)=C(I,1+1)

D = A(1:M,J)                    ( C(1,2)=A(1,J)...)
```

- warunkowe wyrażenie maskujące elementy tablicy, np.:

```
WHERE( A.GE.0.5) A = A + B
```

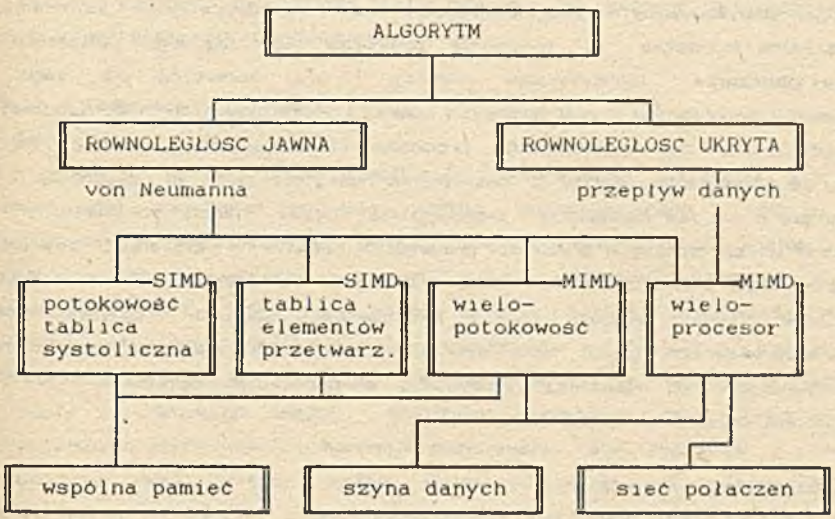
Podsumowując ten temat warto zwrócić uwagę na dążenie do ujednolicenia postaci zapisu tekstu źródłowego programu - przenosząc na preprocesory lub kompilatory problem dopasowania kodu wynikowego do architektury docelowego komputera.

5. Świadomość rozpraszania

Wygoda użytkownika

Omawiając temat przyspieszania obliczeń przez ich rozpraszanie patrzyliśmy na ten problem raczej z punktu widzenia projektanta takiego systemu cyfrowego. W mniejszym stopniu zajmowaliśmy się problemami z jakimi spotka się w tym przypadku użytkownik takiego systemu. Rodzi się bowiem pytanie, na ile musi on znać i rozumieć cechy funkcjonalne sprzętu i oprogramowania tak aby móc uzyskać jak najlepsze przyśpieszenie obliczeń?

Rysują się tutaj trzy stopnie takiego uświadomienia - nieświadomy, uświadamiany i świadomy.



Rys.8. Taksonomia architektur równoległych

Zanim jednak bliżej omówię tę klasyfikację - spróbuję w skrócie podsumować uzyskane dotychczas wiadomości. Opierając się na jednej z taksonomii algorytmów i maszyn równoległych (rys. 8) widzimy, że w przypadku komputerów bazujących na klasycznej architekturze von Neumanna , równoległość algorytmu dla maszyny musi być zadawana jawnie. Istotne jest tylko na ile i w jaki sposób użytkownik musi tę równoległość wyrazić. Im bardziej zaś równoległość algorytmu będzie niespójna z równoległością

wykorzystywanej do jego obliczenia maszyny tym gorsze uzyskamy wyniki - czasem nawet gorsze niż dla obliczenia realizowanego na komputerze uniwersalnym.

Przy okazji zaprezentowanego rysunku muszę wspomnieć o koncepcji realizacji obliczeń równoległych w komputerach sterowanych przepływem danych. W ich przypadku określenie równoległości obliczeń wynika w naturalny sposób z zasady działania tego typu komputera. Niestety rozwiązanie to oprócz instalacji badawczych nie znalazło wiekszego zastosowania. Inna sprawa jest konieczność zmiany sposobu programowania dla tych komputerów co na pewno nie spotka sie z entuzjazmem u wielu dotychczasowych użytkowników komputerów klasycznych.

Z drugiej jednak strony program sterowany przepływem danych może być efektywnie wykonywany w wieloprocesorach. Każdy z procesorów jest wtedy uniwersalna jednostką do wykonania prostego lub złożonego obliczenia. Rodzaj obliczenia (identyfikator operacji) oraz konieczne dla niego argumenty sapobierane z puli gotowych operacji zlokalizowanych we wspólnej pamięci. Cechą charakterystyczną procesów realizujących obliczenia jest fakt, że odwiedzając monitor w poszukiwaniu nowej operacji do wykonania - nigdy sie w nim nie zawieszają. Zwielokrotnie liczby takich monitorów - punktów dostarczających prace dla procesorów pozwala na unikanie nasycenia programowego. Brak kosztownych czasowo operacji zawieszania i odwieszania procesów również zmiejsza straty efektywności działania systemu. Na podstawie własnych badań taki system "Flying Processes" jest znacznie efektywniejszy od realizacji klasycznej z centralnym monitorem i z operacjami delay i continue [11].

Brak świadomości rozpraszania

Najprostszym dla użytkownika jest brak konieczności uświadamiania sobie istnienia jakichś wyrafinowanych metod rozpraszania obliczeń, Znając jednak dotychczasowe rozwiązania sprzętowe i programowe wiemy, że jest to jeszcze postulat nierealny. W niewielkim bowiem stopniu udało sie w pełni zautomatyzować podział programu na obliczenia równoległe. Istniejące już w tej dziedzinie dokonania wymagają jednak współudziału programisty, gdyż przecież to on najlepiej zna opracowany algorytm i może wskazać fragmenty równoległe.

Często też gwarancją znacznego przyśpieszenia obliczeń jest opracowanienie odpowiedniego algorytmu w zależności od stopnia i sposobu zrównoleglenia sprzętu. Poszukiwanie takich algorytmów jest obecnie bardzo popularne. Trudno jednak sobie wyobrazić, aby ten proces graniczący ze sztuką mógł być szybko zautomatyzowany bez radykalnych postępów w rozwiązywaniu problemów sztucznej inteligencji.

Rozpraszanie uświadamiane

Wydaje się, że świadomość użytkownika dotyczaca cech narzędzi do rozpraszania obliczenia powinna wystarczyć. Pozwala to przy wykorzystaniu programów wspomagających przekształcić, z udziałem użytkownika, jego program w postać zrównoleglonych obliczeń. Ułatwione jest też zadanie wyjaśnienia istnienia ograniczeń w sprzęcie. Należy sądzić, że jeszcze długo ten stopień znajomości problemu rozpraszania będzie wymagany od każdego programisty i użytkownika systemów równoległych.

Pełna świadomość rozpraszania

Pełna świadomość istnienia możliwości rozpraszania sprowadza się do konieczności dokładnego poznania bardzo skomplikowanego sprzętu i oprogramowania narzędziowego. Można tutaj odnieść się do analogii z programowaniem w asemblerze, a czasem nawet z mikroprogramowaniem maszyn matematycznych. Z drugiej jednak strony tylko w takim przypadku można mówić o rzeczywistym pełnym wykorzystaniu podrasowanego sprzętu oraz specjalnego oprogramowania. Wydaje się więc, że podobnie jak w przeszłych latach istniały w ośrodkach zespoły analityków systemu oraz programistów pomagających użytkownikom w sprawnym posługiwaniu się systemem - podobne rozwiązanie musi istnieć obecnie. Nawet bowiem w już w komercjalnie działających ośrodkach z superkomputerami, sprawność wykonywanych tam programów nie jest zbyt duża w porównaniu z możliwościami.

Podsumowaniem tych rozważań niech będzie przedstawienie przewidywanych kierunków rozwoju oprogramowania dla systemów równoległych:

1. Rozszerzanie istniejących języków programowania - zapewne dalej Fortranu 8x oraz Pascala (i C ?), umożliwiające programiście zapisanie współbieżności i synchronizacji zadań wykonywanych z wykorzystaniem potokowego przetwarzania wektorowego.

2. Rozbudowanie istniejących kompilatorów o wykrywanie i przekształcanie treści programu w zrównoleglony kod wynikowy, czyli dążenie do maksymalnego automatycznego zrównoleglania obliczeń dla danego typu maszyny równoległej.

3. Wprowadzania metajęzyków umożliwiających na bazie istniejących języków tworzenie wielozadaniowych zsynchronizowanych zapisów złożonych obliczeń.

4. Zaprojektowania całkowicie nowych ,zorientowanych obiektowo i wieloprocesowo języków ukierunkowanych na prosty zapis nowych rozwiązań równoległych algorytmów obliczeń oraz na najnowsze rozwiązania sprzętowe jak na przykład wielopotokowe wektorowe sieci obliczeniowe.

6. Podsumowanie

W podsumowaniu spróbuje odpowiedzieć na dwa podstawowe pytania.

Kiedy stosować przyśpieszanie ?

Najprostsza odpowiedzią jest - zawsze kiedy tylko jest to możliwe. Trudno jest się jednak zgodzić z takim poglądem. Oczywiście stosując w miarę posiadanych środków odpowiednie zabiegi jesteśmy w stanie często nawet znacznie przyśpieszyć obliczenie. Musimy jednak pamiętać, że wymaga to od nas - projektanta algorytmu, programisty, operatora systemu znacznie większej pracy niż zwykle. Również translator naszego programu działa znacznie dłużej niż normalne. Co więcej, często dopiero po kilku (oby !) próbach różnych modyfikacji algorytmu obliczeń uzyskujemy lepsze osiągi. Wykrycie poważniejszych błędów logicznych prowadzi czasem do powtórnego poszukiwania dobrego, w sensie efektywności, rozwiązania.

Tak więc w wielu przypadkach nie ma sensu "gonić" za wątpliwym przyśpieszeniem obliczeń (chyba, żeby zdobyć doświadczenie).Z drugiej strony istnieje wiele problemów z różnych dziedzin wiedzy, których ilościowe rozwiązanie jest możliwe dopiero obecnie dzięki istnieniu bardzo szybkich komputerów. I trzeba stwierdzić, że takich problemów w samej informatyce jest niewiele, a więc na to pytanie odpowiedzi powinni w zasadzie udzielić fizycy, chemicy, meteorolodzy,... wskazując na znane im potrzeby.

Jaki wybrać komputer ?

Prawie na pewno mogą paść tutaj pytania o ile szybciej wykona się program bez i z wykorzystaniem koprocesora zmienno- przecinkowego czy wykorzystującego inną metodę trywialnego rozpraszania. Podobne pytania mogą być formułowane po poznaniu różnego typu architektur superkomputerów, wieloprocesorów, komputerów macierzowych i komórkowych.

Oczywiście takie badania dla typowych mieszanek programów istnieją, a uzyskane wyniki można znaleźć, bądź w reklamach firm bądź w poważnych analizach porównawczych. Niestety, podawane tam czasy wykonania mieszanek programów niwelacyjnych są dla zwykłego użytkownika mało przydatne. Pomijając bowiem wykonywanie typowych już obliczeń,w oparciu o programy biblioteczne, w pozostałych przypadkach trudno jest wyrokować, który z dostępnych komputerów będzie najlepszy dla danego typu problemu. Możliwe jest jedynie pewne oszacowanie w oparciu o analizę struktury algorytmu i sposobu obliczeń najczęściej wykonywanych w nim operacji. Główny wysiłek musi zostać skierowany na zaprojektowanie "na miarę" dla danego komputera odpowiedniego programu. Zdajemy już sobie sprawę, że nie jest to zadanie trywialne, nawet korzystając ze specjalnych translatorów i narzędzi wspomagających. Co więcej programy te są oparte o algorytmy, nam informatykom, mało zrozumiałe, a z kolei obecny stan wiedzy o efektywnym wykorzystaniu superkomputera jest jeszcze dziedziną znaną tylko niewielu.

Znacznie prostsza sytuacja występuje u nas w kraju. Po pierwsze bowiem nie mamy armat. Bolejąc nad tym powinniśmy sobie jednak uświadomić następujące fakty.

1. W Polsce brak jest w ogóle jakichkolwiek doświadczeń w użytkowaniu tego rodzaju maszyn, a więc też nie ma możliwości nauczenia, przynajmniej podstaw, następnych pokoleń.

2. Boom mikrokomputerowy, rozwijający się w dużym stopniu w oparciu o domowe oprogramowanie oraz brak możliwości zakupu przynajmniej VAXa, powoduje wytworzenie się przekonania ,że tysiącami PC/Z zastąpi się jednego CRAYa.

3. Możliwe jest jednak zakupienie, bądź zrealizowanie siłami krajowymi prostych systemów wieloprocesowych ,opartych na standardowych układach mikroprocesorowych lub zintegrowanych programowo sieciach

mikrokomputerowych - żałować należy, że podejmowane już w poprzednich latach próby zakończyły się typowym fiaskiem.

4. Konieczne jest stałe propagowanie problematyki przyśpieszania obliczeń, nawet z wykorzystaniem trywialnych rozwiązań, i zdobywanie wiedzy o najnowszych trendach w tej dziedzinie, choćbyśmy mieli czekać na CRAYa w kraju nawet kilka... lat.

BIBLIOGRAFIA

Większość treści referatu została oparta na materiałach zamieszczonych w :

1. Proceedings of the 1986 International Conference on Parallel Processing, August 19-22,1986, Penn State University. stron 1050.

Pozostałe pozycje są wybranymi z licznej bibliografii dotyczącej tego tematu.

2. Babb II R.G.ed. ; Programming Parallel Processors , Addison-Wessley Publishing Company, Inc. 1988.
3. Bucher I.Y.,Simmons M.L.; Performance .Assessment of Supercomputers, Report Los Alamos National Laboratory LA-UR-85-1505.
4. Cellary W. ; Systemy wielomikroprocesorowe ,Szkoła Jesienna PTI ,Rydzyna 9-13 października 1984. , Informatyka 11-12/86,1/87.
5. Deminet J. ; Cm* - Przykład komputera wieloprocesorowego o strukturze hierarchicznej,Druga Jesienna Szkoła PTI, 1985.
6. Fernbach S. ed. ; Supercomputers Class VI Systems, Hardware and Software , North-Holland 1986.
7. Friedlander C.B., Wedde H.F.; Distributed Processing under the DRAGON SLAYER Operating System , Proceedings of the 1986 International Conference on Parallel Processing, August 19-22,1986, Penn State University. stron 1050.
8. Gajski Tutorial
9. Hockney R.W., Jesshope C.R. ; Parallel Computers , Adam Hilger Ltd. Bristol. 1981.
10. Iszkowski W., Maniecki M. ; Programowanie współbieżne , WNT 1982.
11. Iszkowski W. ; Performance Evaluation for a Dependency Graph Computations on a MultiProcessor System , Research Reports , No 56, Instytut Informatyki 1983 .
12. Iszkowski W. ; Superkomputer CRAY, Informatyka 5/85.
13. Iszkowski W. ; Implementation of Monitor Operations ; MICROSYSTEM '85 , Tabor 1985 ; streszczenie w Microprocessing and Microprogramming, The Euromicro Journal , Vol.17, No 4, April 1986.
14. Iszkowski W. ; Architektura systemów n-mikroprocesorowych , Informatyka 7-8/86 .
15. Iszkowski W. ; System Operacyjny OS/2 , IV Szkoła Mikrokomputerowa PTI, 1987.
16. Kugelmass S. , Squier R. , Steiglitz K. ; Performance of VLSI Engines for Lattice Computations , "Proceedings of the 1987 International Conference of Parallel Processing. Penn State University. PA: Pensylwania.
17. Madey J.; Ockham, Edison, Joyce i inni, czyli o programowaniu współbieżnym,Czwarta Jesienna Szkoła PTI, 1987.

18. Owczarczyk J., Stolarski M., Woźniak E. ; Architektura współbieżnych systemów przetwarzania obrazów , Informatyka 7-8/86.
19. Owczarczyk J., Stolarski M., Woźniak E. ; Procesory tablicowe - współbieżne systemy przetwarzania obrazów , Informatyka 2/87.
20. Rybnik J., Solak J. ; Rodzina komputerów SUN-3 ,IV Szkoła Mikrokomputerowa PTI, 1987.
21. Randy A.,Kennedy K. ; Automatic Translation of FORTRAN Programs to Vector Form , ACM Transactions of Programming Language and Systems , VOL.9, No.4, Oct 1987.
22. Bysio M.M. ; Maszyny i algorytmy równoległe , Druga Jesienna Szkoła PTI, 1985.
23. Szczerbiński Z. ; Nowa Generacja Superkomputerów CRAY ,Informatyka 12/88.
24. Shoja G.C., Gurr R.G. ; Parallel Processing in Local Networks,Proceedings of the 1986 International Conference on Parallel Processing, August 19-22,1986, Penn State University. stron 1050.
25. Turski L.A. ; Komputery w fizyce i fizyka w komputerach ,Czwarta Jesienna Szkoła PTI, 1987.
26. Turski W.M. ; Superkomputery szturmują rynek , Biuletyn PTI, Nr 7, lipiec 1988.
27. Wayner P. ; Modeling Chaos , BYTE , May 1988.
28. Winkowski J.; Protocols of Accessing Overlapping Sets of Resources , Information Processing Letters, Vol.12,No. 5 (1981); North Holland.

Zastrzeżone znaki towarowe - Trademarks

Alliant, Concentrix, FX/8, FX/Fortran	Alliant Computer Systems Corporation.
Butterfly, Chrysalis, Uniform System	Bolt Beranek and Newmann Inc.
CRAY-1, CRAY X-MP	Cray Research Inc.
Ethernet	Xerox Corp.
iBSC,MULTIBUS	Intel Corp.
Occam,IMS	Inmos Group of Companies
SUN Workstation	SUN Microsystems
UNIX	AT&T Bell Laboratories
VAX, VMS, RSX-11	Digital Equipment Corp.
MS-DOS,OS/2,XENIX	Microsoft Corp.
IBM PC/XT/AT, PS/2	International Business Machines Corp.

Jesienna Szkola PTI
Mragowo, grudzien 1988

SPRAWNA PRACA MALYCH ZESPOLOW

Andrzej Klopotowski
Konsultant Zarzadzania
i Programow Produktywnosci
Bukowa 10, 02-708 Warszawa
tel. 190415 (484848)

1. Wstep

Wspolczesna praca ludzka to glownie praca zespolowa. O sprawnosci pracy instytucji przesądzaja zespoly ludzi a w znacznie mniejszej mierze jednostki. Powodzenie kompleksowych przedsiewziec - do nich naleza takze budowy systemow informatycznych - rowniez uzaleznione jest od sprawnosci i harmonii wysilku zespołowego. W Polsce zdajemy sobie sprawe z wagi sprawnego wysiłku zespołowego. Niestety jednoczesnie pogodnie, z rezygnacja konstatujemy ze "Polacy to indywidualisci" i nie robimy nic, by stan taki zmienic. Czy rzeczywiscie nie mozna temu zaradzic? Owszem sa odpowiednie sposoby. W ostatnich dwu dekadach w krajach wysoko rozwiniętych, a takze i niektorych rozwijajacych sie, podjeto usystematyzowanie i zakrojone na szeroka skale działania racjonalizyjace prace zespolow ludzkich, podnoszac ich skutecznosc oraz ozywiajac zbiorowy wysilek a bardziej obrazowo tchnac ducha w organizacje jako calosc. Podejscie "rozwoj organizacyjny" [Organizational Development lub krotko OD] oraz "kola jakosci - produktywnosci" [Quality Control Circles] to najszerzej w swiecie rozpowszechnione z takich kompleksowych

programów. W ramach filozofii w nich przyjętej, zakładającej upodmiotowienie możliwie wszystkich uczestników organizacji, wyłoniła się także sylwetka doradcy nowego typu - tak zwanego konsultanta lub katalizatora procesu, którego sposób operowania wyraźnie różni się od znanego doradcy-eksperta.

2. Niepowodzenia pracy zespołowej

Co jest przyczyną niepowodzeń pracy zespołowej? Odpowiadając na to pytanie zespoły kierowników, specjalistów i pracowników-robotników przygotowujące się do wprowadzenia programów produktywności w Polsce, najczęściej wymieniają:

Złe, na przykład zbyt wąskie formułowanie problemów i stąd rozwiązania wycinkowe, niespójne lub sprzeczne; niejasno określone cele, maksymalizowanie celów [wspinanie się od razu na szklaną górę a przecież mały postęp to też postęp], brak analizy przyczyn powstawania problemu, niestaranne gromadzenie i weryfikowanie informacji [uzasadnianie], słabe przekazywanie informacji, stąd błędne diagnozy.

Złe inicjowanie pracy - niewystrzegające kompetencje osób [niepełna wiedza, znajomość tematyki, doświadczenie] rozwiązujących problem [włączonych bezpośrednio], zbyt wąskie grono osób włączonych w rozwiązywanie problemu, pominięcie niektórych z tych, którzy mają istotny wpływ na jego rozwiązanie; słaba współpraca z innymi związanymi z problemem; brak [nieskuteczne pobudzenie] zainteresowania, brak wiary w powodzenie [celowość przedsięwzięcia], niepełne [dlaczego?] zaangażowanie ["niemoc" wewnętrzna, powoływanie się na czynniki zewnętrzne], obawa przed nowym ["sprawdzone lepsze"]; niewłaściwa wycena wynagrodzenia.

Nieprzemyślany [nieusystematyzowany] sposób rozwiązywania problemu; nie zasięganie opinii specjalistów, a te zbyt często bywają nietrafne, brak obiektywizmu; brak planu działania, niewyznaczenie osób odpowiedzialnych

[niezdefiniowanie ich roli], nierealny lub niesprecyzowany termin realizacji, nieliczenie sie z mozliwosciami [niewlasciwa ocena pracochlonnosci], chaos [niemoc] organizacyjna, nieznajomosc technik rozwiazywania problemow.

Brak konsekwencji w realizacji, zniechecenie lub zaniechanie dzialan przy pierwszych niepowodzeniach; zajmowanie sie drobiazgami [bo priorytety nie zostaly okreslone]; dlugi czas realizacji, okresy bezczynnosci a potem nerwowa presja na przyspieszenie realizacji; dominacja "ruchow pozornych" nad rzeczywistymi; brak solidnego zakanczania i oceny [podsumowania] pracy.

Nadmierne uzaleznienie od zwierzchnika w procesie rozwiazywanie problemow [wtracanie sie zakresy swoich podwladnych, narzucanie rozwiazan, calkowite scentralizowanie decyzji]; rownoczesnie "podrzucanie" rozwiazywania problemow na wyzsze szczeble.

Utrudnienia zewnetrzne - slabosc strony wykonawczej [zaopatrzenie, brak wykonawcow, niekompletna baza wyposazenie, niedorozwoj uslug zewnetrznych], nakladanie sie zbyt wielu spraw-problemow [nadmiar inicjatyw w czasie]; biurokracja; niechec ludzi odpowiedzialnych, brak inicjatywy, unikanie ryzyka ale i brak prawa do ryzyka [bledu]; zawisc; partykularyzm grup zawodowych, rozbieznosc interesow.

Podobne determinanty zlej pracy zespolow podkreslano takze przy wielu innych okazjach. Z pewnoscia praca zespolowa i jej rezultaty ulegly by znacznej poprawie z chwila wyeliminowania powyzszych slabosci, jasno uswiadomionych przez samych zainteresowanych. Wylacznie zdrowo-rozsadkowe podejscie nie wydaje sie jednak byc wystarczajace. Przede wszystkiem listy powyzsze nie sa kompletne i niektore wazne aspekty pracy zespolowej bywaja, przez mocno zajetych praktykow, w ogole nieuswiadomione lub rzadko dostrzegane. Zdarzaja sie tez blednie zalozenia i generalizacje jakie kierownicy, zbyt pochopnie, bezkrytycznie

wyprowadzają ze swojej praktyki.

Wiele nieustrukturalizowanych grup treningowych na przykład rozpoczyna prace od mianiowania kierownika swego zespołu, wyraźnie starając sie zbliżyc do wzorca sprezystej, opartej na rozkazach czy wręcz drylu, organizacji wojskowej. Wzorzec taki przyjmują często jako synonim porządku i sprawnej zbiorowej, opartej na silnym liderze "maszynerii", odpowiedni we wszystkich okolicznościach i w każdym czasie.

Jakże często pierwsze z brzegu zadowalające rozwiązanie analizowanego problemu bywa akceptowane i przyjmowanie niezwłocznie do realizacji. Na rozwiązanie optymalne, moglibyśmy wpasc zapewne w dalszej kolejności, z reguły jednak nie generujemy dostatecznej liczby jego opcji i nie odwlekamy ich oceniania do chwili gdy wyraźnie wzrośnie ich liczba i oryginalność.

Nie przestrzegamy też zwykle w praktyce, stosowanego powszechnie w naukach wojennych czy w grze w szachy, wymogu staranniejszego przewidywania problemów [identyfkowanie problemów potencjalnych] i nie czekanie dotąd aż wystąpią w pełnym natężeniu. Lepiej jest przecież unikać problemów niż je potem mozolnie rozwiązywać.

Wreszcie w dywagacjach na temat pracy zespołowej, podobnie zresztą jak i w praktycznym działaniu zespołów w organizacjach-instytucjach, cała uwaga skierowana jest na treść rozpatrywanych zagadnień, samą istotę, "substancje" problemu. Totez i napływające od praktyków propozycje usprawnień dotyczą prawie wyłącznie tej właśnie warstwy, którą nazywamy warstwą zadaniową [funkcjami zadaniowymi]. Niedostrzegane lub pomijane bywają z reguły aspekty związane z relacjami uczestników grupy ich zachowaniem, odnoszeniem sie do siebie, pomocą lub rywalizacją, a także problemy przywództwa [jak wskazano wyżej, widziane są zwykle bardzo jednostronnie]. Funkcje tego typu nazywamy "funkcjami procesu" lub "funkcjami podtrzymującymi [konserwującymi]" zespół jako spójną, zdolną do sprawnego

działania, w dłuższym przedziale czasu, całość. One właśnie decydują o "stanie zdrowia" zespołu a przez to i jego zdolności do efektywnego, długofalowego działania.

W racjonalizowaniu pracy zespołowej staje się konieczne, jak widzimy, bazowanie zarówno na zdrowym rozsądku i praktycznych doświadczeniach uczestników poszczególnych grup, jak i na zdobyczach metodologii sprawnej pracy zespołowej [zresztą wyrosłej także ze starannie zweryfikowanej empirii].

3. Stan docelowy sprawności i zdrowia zespołu

Warto, by na początku doskonalenia jego uczestnicy wyobrazili sobie jak ich zespół mógłby i jak powinien funkcjonować. Wybraliśmy [Załącznik 1] kilka przykładowych kategorii ze stosowanego przez nas do tych celów materiału pomocniczego [Kłopotowski]. Krok ten równocześnie podnosi "gotowość" zespołu do zmian swego sposobu funkcjonowania. Gdy wizja docelowej sprawności stanie się jasna, wybrać możemy jedną z kilku możliwych dróg postępowania.

4. Spotykane sposoby doskonalenie pracy zespołowej

Najbardziej naturalna to droga "konsultowania procesu" [Schein]. Wysokokwalifikowany konsultant-trener obserwuje prace zespołu, na przykład narady naczelnego kierownictwa, i na zakończenie [lub w odpowiednim czasie] koncentruje uwagę uczestników na procesach i incydentach, które tu i teraz miały miejsce. Pomaga następnie grupie w przeprowadzeniu analizy i poszukaniu innych, lepszych sposobów zachowania się, działania i podejmowania decyzji. Zaletą jest to, że grupa rozpoczyna wprowadzanie zmiany tam, gdzie jej gotowość do zmian już powstała a nie tam, gdzie dyktowałyby to wskazówki "teorii" sprawnej pracy

zespolowej lub pospieszne konstatacje konsultanta-trenera. Krotkie mini-wykłady i odpowiednie cwiczenia sa wowczas chetnie przyjmowane. Jest to jednak proces dosc wolny i pracochłonny a więc kosztowny.

Powyzszy system racjonalizacji bywa wiec upraszczany przez rezygnacje z konsultanta zewnetrznego. Wowczas funkcje obserwatora pracy zespolu i inspiratora usprawnien powierza sie jednemu z uczestnikow zespołu, bądz tez kolejno roznym członkom zespołu. Jest oczywistym, ze zarowno taki obserwator jak i kierownik zespołu powinni przejsc wstepne przeszkolenie.

Inne podejscie zakłada tradycyjną sekwencję: naucz sie - rób. Tu tylko kierownik i kandydat na konsultanta wewnetrznego pracy zespolowej, a czasem i wybrani uczestnicy zespolu [lub ochotnicy] przechodza przeszkolenie w warunkach "laboratoryjnych". Po powrocie do wlasnych zespolow juz oni szkola w praktycznym działaniu swych kolegow.

5. Zorientowane na rezultaty doskonalenie pracy zespolowej

Stosowane przez nas podejscie łączy elementy omowionych programow. Jest ono jednak wpisane w szerszy program podnoszenia produktywnosci-skutecznosci organizacji jako całości i posiada kilka dodatkowych elementow. W podejsciu naszym doskonaleniu podlega zarowno praca zespolow złożonych z kierownikow i specjalistow-pracownikow [praca komorek a takze zespolwo zadaniowych] jak i praca ogolu pracownikow i robotnikow z programem czastkowym "kół jakosci-produktywnosci".

Podstawowe usprawnienia koncentrują sie na rajconalizacji procesow rozwiazywania problemow. Obejmuje ona wyodrebnienie dwu przedziałów w jakich w rzeczywistosci problemy sa identyfikowane i atakowane: przedział ogolno-organizacyjny, będący domeną kierownctwa wyzszego szczebla oraz przedział rozwiazywanego problemu, bedacy domena zespolu realizujacego w rzeczywistosci usprawnienie [Schein]. Zwykle okazuje sie tu konieczne wprowad-

zenie odpowiedniej modyfikacji rol tych dwu grup wspol-realizatorow. Na przykład pozostawienie przez naczelne kierownictwo nizszym szczeblom "meki przemysliwania" ale i satysfakcji ze znajdowania rozwiazan problemow. Eliminujemy przez to demotywujace wyreczanie rzeczywistych wykonawcow, ktorym tradycyjnie przekazywano juz tylko do wdrozenia drobiazgowo sporzadzone harmonogramy i ujete w nich dzialania czastkowe. Ta powszechnie spotykana wyreczycielska rutyna zuboza prace podwladnych i niszczy szanse glebszego pobudzenia ich zainteresowan i talentow. Ulatwiamy zastapienie takiego postepowania operacja przekazania-przejecia problemu w zrealizowana w sposob ktory upodmiotawia rzeczywistych wykonawcow usprawnienia a takze i tych, ktorzy miec beda wpływ na jego wdrożenie. Rola kierownictwa zostaje ograniczona do zapewnienia wspolnie poprawnej definicji problemu, wskazania jego hierarchii, celow i wymagan [kryteriow "dobroci"] jakie spelniac powinno rozwiazanie. Pozniej kierownictwo wysłuchuje prezentacji proponowanego rozwiazania, a gdy zostanie ono zaakceptowanie - udziela pomocy w jego wdrozeniu. Z kolei zespol pracujacy nad usprawnieniem koncentruje uwagę na pełnej analizie problemu, jego przyczyn i skutkow. Staranne zdefiniowanie problemu ulatwia poszukiwanie rozwiazan wzmocnione metodami innowatyki. Zespol juz sam staranne planuje i przemysliwuje dzialania usprawniajace. Prowadzi takze samokontrole postepu prac oraz ocene rezultatow koncowych. Zapewnia tez ze nowe rozwiazanie zostanie utrzymane. Zespol opanowuje jednoczesnie kilka najwazniejszych technik m.in. analize przyczyn i skutow, hierarchizowanie przy pomocy metody ABC [Pareto], obróbkę statystyczną danych, nominalne grupowanie [burze mozgow]. Dla zespolow bardziej ambitnych, w miare jak ich apetyty rozwojowe wzrastaja przygotowane sa bardziej zaawansowane narzedzia i programy treningowe. Na przykład zespoł uczy sie podejmowac decyzje w grupie, rozwazywac konflikty wewnetrzne a takze poprawiac współdziałanie z innymi zespołami [organizacjami].

Jednoczesnie postepuje doskonalenie współdziałania członkow zespolu i utrzymywanie "zdrowia" zespołu jako całości. Punktem wyjscia do tego było, jak wspomniano w rozdziale 3, okreslenie stanów docelowej sprawnosci-zdrowia zespolu, tworzacych swoisty

kodeks zachowania sie zespolu. Majac go przed oczyma [warto wywieszac go w miejscu w ktorym zespol pracuje, przeznacza sie ostatnich 15 minut kazdego spotkania na wychwycenie przypadkow zachowan uczestnikow i niewłasciwych procesow współdzialania, ktore właśnie miały miejsce [tu i teraz], ich przeanalizowanie i znajdowaanie innych, właściwszych. Jesli zespol przelamie tylko poczatkowa niechec do zajmowania sie tym jak pracuje i jak ludzie czuja i postepuja, to szybko uzyska imponujacy wzrost sprawnosci.

BIBLIOGRAFIA

1. Beckhard R., Organization Development: Strategies and Models, Addison-Wesley, Reading, Mass., 1969.

2. Durbin A. J., Praktyczna psychologia zarzadzania, Panstwowe Wydawnictwo Naukowe, Warszawa 1979.

3. Kłopotowski A.: 4. Sprawnosc pracy i normy efektywnego zachowania w zespole. Zbior materialow pomocniczych dla programu produktywnosci [maszynopis], Warszawa 1987.

4. Schein E. H., Process Consultation: Its Role in Organization Development, Addison-Wesley, Reding, Mass., 1969.

Andrzej Kłopotowski - *konsultant*

Załącznik 1

4. Sprawność pracy i normy efektywnego zachowania w zespole

3. Od NIE W PEŁNI ŚWIADOMEGO [SPONTANICZNEGO]
do ŚWIADOMIONEGO I EFEKTYWNEGO PROCESU
ROZWIĄZYWANIA PROBLEMÓW I PODEJMOWANIA DECYZJI W ZESPOLE

[1] Atakowane są symptomy a nie problemy; przeskakuje się do działań zaradczych zanim dokona się diagnozy i ustali przyczyny; natychmiast ocenia się pojawiające się rozwiązanie, stąd brak usprawnień alternatywnych; potrzebne decyzje nie są podejmowane; inne podejmowane tylko przez część zespołu, przy braku zaangażowania pozostałych

[5] Gdy pojawi się problem, sytuacja jest starannie przeanalizowana [symptomy, problemy i przyczyny]; działania usprawniające atakują najważniejsze a nie trywialne przyczyny; wystarczająca liczba wariantów rozwiązań jest wygenerowana, przy zachęcie dla rozwiązań oryginalnych, zanim podda się je ocenie i porównaniom; różnice poglądów są szanowane i użyte dla udoskonalenia rozwiązań-decyzji; zmierza się do zgodnej aprobaty i weryfikuje się czy takowa nastąpiła; podjęte decyzje są w pełni popierane; wszyscy troszczą się zarówno o sprawny przebieg procesu pracy jak i o konserwację "stanu zdrowia" zespołu [Uzupełnienie 2]

1 2 3 4 5

Jakie normy [opisać konkretnie] zespół nasz przyjąć powinien za docelowe i jakie związane z nimi zmiany procesów [pracy i "konserwacji"] zesołu oraz indywidualnych zachowań uczestników należy wprowadzić?

.....
.....
.....
.....

4. Od NIEWIELKIEGO do PEŁNEGO WYKORZYSTYWANIA
ZASOBÓW INTELEKTUALNYCH ZESPOŁU

[1] Niewielu dominuje, część jest pasywna; inni nie są wysłuchiwani, przerywa się im wypowiedzi; kilku mówi jednocześnie

[5] Wszyscy są włączeni w stopniu jaki zakreślają ich rzeczywiste kompetencje w rozpatrywanym zagadnieniu; zachęca się i ułatwia wzajemnie wypowiadanie; wszystkich wysłuchuje się z uwagą a gdy trzeba powtórzeniem-pytaniem weryfikuje się poprawność odbioru

1 2 3 4 5

ACTIVE DATABASES: THEORY AND APPLICATION

Laurent Siklóssy
Department of Mathematics and Computer Science
Free University
Postbus 7161
1007 MC Amsterdam, Netherlands

ABSTRACT

This article contains two papers:

1. L. Siklóssy, "Artificial Intelligence and Databases," in: Marciano, J. P. (ed.) Conférence Tutoriale, Economics and Artificial Intelligence, Aix-en-Provence, France, 101-104, 1986.

2. E. Tulp and L. Siklóssy, "TRAINS. An Active Time-table Searcher." (with Tulp, E.) ECAI'88 (European Conference on Artificial Intelligence), Munich, W. Germany, 170-175, 1988.

The first paper describes mostly the theory of Active Databases, and gives some examples of applications. The second paper describes some new algorithms in dynamic networks, and an implemented active system which finds optimal train connections on the entire Dutch railway network.

ARTIFICIAL INTELLIGENCE AND DATABASES

Laurent Siklóssy
Department of Mathematics and Computer Science
Free University
Postbus 7161
1007 MC Amsterdam, Netherlands

ABSTRACT

Artificial intelligence techniques can enhance the utility of databases. We describe several such enhancements, including:
1. a user's limited knowledge of the database can be overcome, and information of potential relevance brought to his attention, information which had not been specifically requested.
2. a user's assumptions, which may lead to incorrect conclusions, can be clarified.
3. a user's normal conversational expectations can be fulfilled.
A user dependent topology of neighborhoods in the database guides these enhancements.

1. INTRODUCTION

Data bases are becoming essential components of modern organizations. They can be accessed by specialized programs, by queries written in a query language, or (within some limitations) by queries expressed in a natural language. Several investigators have pointed out that systems which only answer a query are unfriendly, unsatisfactory and much less helpful than they could be.

A goal of our research is the development and mastery of techniques which will permit the rapid and efficient implementation of certain types of knowledgeable, friendly and helpful assistants to decision makers. These assistants will make use of data bases in an active, collaborative way: they are Active Collaborative Systems (ACS). By contrast, a data base system which only answers queries can be viewed as passive: it waits for a question, and has finished its task after providing the answer.

In section 2, we review three different ways in which passive question-answering systems have been judged unsatisfactory. The solutions that have been suggested to render the systems more satisfactory have lacked generality. In section 3, we propose a general framework which may remove all the inadequacies mentioned. This framework is based on a descriptive topology, both quantitative and qualitative, of the space of topics that is stored in the database. Section 4 outlines an extension of ACS to what we have called Active Data Bases. An Active Data Base provides a user with the sort of information that he would have requested, had he taken the time to request it. (But in fact, for a variety of reasons -lack of time, lack of computer-related skills, etc.- the user did not formulate the queries.) Thus, we see that an active data base broadcasts information to the user. It can be seen as responding to a set of latent questions that describe a user's interests. (Sometimes, we include active data bases under the term ACS, since the two types of systems have the same goal: inform and help the user.)

User models are needed by both of the above active systems: the systems' reactions must be adapted to the user's goals. Section 5 discusses user models. Section 6 is centered around the need for automatic restructuring of the storage structures of existing data bases to increase the efficiency with which the topological neighborhoods can be accessed from one particular topic.

2. INADEQUACIES OF PASSIVE QUESTION-ANSWERING SYSTEMS

We call passive those question-answering systems which simply answer a question (see Siklóssy [10].) Wahlster [12, 13] discusses the inadequacies of such systems, namely:

a) the system may fail to inform the user that a modified question, closely related to his initial question, has a "significantly better" answer, which would help the user better to achieve his goals.

b) the system may fail to inform the user that his question includes presuppositions which are not actually valid, so that the answer given may be misleading or meaningless.

c) the system may fail to provide the user with additional information which the user would normally expect, and which he must now request explicitly.

To these categories, we add a fourth one:

d) the system may fail to inform the user of the additional, related topics that the system is ready to pursue at the user's request.

We shall now give short, illustrative examples for each of these inadequacies.

a) A company asks its decision support system about the cost of receiving a shipment of 800 widgets on Monday. The system answers, but fails to point out that an interesting quantity discount is available for orders of 1000 or more widgets, and that delivery on Wednesday or later would avoid the heavy "rush" surcharge.

Here is another similar example:

A company asks its decision support system about the cost

of manufacturing 10000 widgets in its plant at location A. The system answers, but fails to point out that plant A has a considerable backlog and therefore the order may be much delayed. Another possibility (in addition, or separately from the above) is the failure of the system to mention that plant B could have the order ready sooner and/or at less cost, or that the order could be subcontracted to Company XYB at advantageous terms.

Siklóssy [8,9,11] was the first to discuss the above type of inadequacy. He developed a theory, based on discontinuities, which tells a question-answering system when it should provide similar additional information. The theory has been implemented in such diverse areas as: air travel and airline tariffs, shipment of goods by truck or air, manpower assignment to projects, selection of a restaurant, selection of commercial real-estate for investment, distribution of tools and machines to various jobs and sites in a large corporation, vacationing, ordering spare parts for machines, etc.

b) Kaplan [4] and Webber [14] give examples of user presuppositions and misconceptions which are not usually detected by question-answering systems. Let us illustrate their points by considering the two short question/answer sequences:

- How many women teach in the Department of Information?
- Zero.

- Is John at the meeting?
- No.

The first query assumes that there is a Department of Information, and that it has teachers. If there is no Department, or if there is one but no teaching within it, then the answer "zero" is correct but misleading. The second query assumes that there is a John, that there is a particular meeting and that John could

come to the meeting. If any of these suppositions is false, the response is again misleading.

Here is another small dialogue. The first answer is somewhat misleading, while the second one is not.

- How many boxes of computer cards do we have in stock?
- Zero.
- Zero. We no longer use computer cards here.

c) Wahlster [12, 13] describes a system which gives answers that are typical of those expected by human users when the minimal answer would be yes or no. To the question:

- Is there someone in room S5.23?

the minimal answer would be : "Yes." or "No." A more acceptable answer would be:

- Yes, George and Martha both entered S5.23 at 10.32 am.
or:
- No. George and Martha left the room at 6.45 am.

A similar example is:
- Have you started on Project XBY?
- No.
- No. I dont even know anything about Project XBY.

We could expect a good ACS to be even more helpful, as for instance in the following:

- Have you started on Project XBY?
- No.
- No. Our department does not handle project XBY. The project is handled by the Archetypes department.
or:

- No. Our department does not handle project XBY. We handle design problems. Project XBY is a security project; these projects are usually handled by the Confidentiality Department. Shall I check whether they are indeed working on this project?

d) One problem in giving additional, not specifically requested information is: how much information should be given? A user model will help, but sometimes it is important to let the user decide so as not to annoy or bore her. Also, it is important to bring to the user's attention additional information, pertaining to the user's question, which the user may wish to explore. This information would not always interest the user, and therefore it should not be broadcast every time. Instead, it makes the user aware of other directions that she may wish to explore.

We shall not try to provide a complete bibliography of work touching on this area. Additional works in natural language processing which can diagnose presuppositions, and respond accordingly, include Lesmo [5,6]. The references mentioned in this section contain pointers to other relevant works. Of particular relevance are the developments of plan based theories of dialogue, which try to understand a user's goals, and which will be taken up briefly in section 5, User Models.

3. A TOPOLOGY OF TOPICS FOR ACTIVE QUESTION-ANSWERING SYSTEMS.

The four types of inadequacies in responses that were mentioned in the preceding section can all be removed by the introduction of a topology of the topics that are known to the system. The topology describes the multi-dimensional structure of the topics and their neighborhoods. We shall outline the form of the topology for

each of the examples of the preceding section.

a) The topic here is a shipment of goods. The description of a shipment includes such dimensions as: what is shipped, quantity, dates for pick-up and delivery, route followed, carrier, itemized costs, likelihood of delays, etc. Given a particular topic, i.e. a particular shipment, the system searches for discontinuities that are favorable to the user, i.e. where a small change in the topic along some dimensions can result in large, favorable changes along some other dimensions. Thus, we notice that the topology includes metric information. As mentioned, this technique of discontinuities has been implemented in a variety of domains (see section 2.a.)

b) In the first example of section 2.b, the topic concerns departments (of a university, say). Information about departments includes their name, composition, leadership, location, function, budget, etc. Certain aspects are dependent, and therefore may presuppose others. For example, an "empty" department, i.e. one with a name but no budget or staff, has no leader. If a department has no teaching function, there are no teachers teaching in it, etc. Neighboring topics include departments with similar names (Department of Information Science), similar functions, some or considerable overlap of staff, etc. Therefore, a possible answer to the question:

- How many women teach in the Department of Information?

could have been:

- There is no such Department. The Department of Information Science has 3 women teachers out of 11 teachers.

or:

- There is no teaching within that Department. Many of the Department members teach in the Department of Information Science.

In the second example, the topic is a meeting. The description of a meeting includes place, time, expected and invited attendees, whether the meeting is open and to whom, etc. Neighborhoods include other relevant meetings (say over the same topic, both in the past and future), meetings scheduled for the expected attendees, etc. Therefore, possible answers to the question:

- Is John at the meeting?

could be:

- No, it is a closed meeting, and he was not invited.
- No, John is away. He is expected at the following meeting of the "Widget advertisement group" on July 8th.

c) in the example of the query: "Is there someone in room S5.23?," the topic is rooms, or perhaps, more generally, locations. (Actually, there is a second topic: the activities of John.) The dimensions of the topic include identification of the room, uses (with dynamic values), contents, etc. Neighboring topics include similar rooms (with similar functions, or occupied by similar occupants -persons or objects-), rooms which are geographically close by, etc. Descriptions of the dimensions can be bundled (preferably in a hierarchical way, which would be user dependent) so that responses which touch upon one dimension within a bundle will trigger further explanations along the other dimensions of the bundle. In our example, a request about occupancy of a room would search automatically (if the occupancy is not zero) for number and identification of the occupants, and for some information

about times of arrival. So an answer could have been:

- The room is occupied by members of the "Widget advertisement group." They have occupied the room since 10 am.

Here is another example in a similar vein:
- Do you have Smid's book on Computer Networks?
- No.

A helpful ACS might say:
- No. In fact, we have no books by Smid.

And an even more helpful ACS would elaborate:
- No. We have no books by Smid on Computers. We have books on Computer Networks by A. S. Smit, and one book on Computer Communication by B. T. Smith. Here are the references:...

d) The mechanisms required for informing the user of additional information that the system may provide, if the user wishes to have it, are similar to those required for part c) above. For example, one of the previous answers could have been:

- The room is occupied by members of the "Widget advertisement group." I can further tell you about:

A. when they started occupying the room.
B. until what time they have reserved the room.
C. which members of the group are present.
D. which members of the group are absent. etc.

Here is another example where the ACS indicates where it could be helpful, if the user so chose:
- Did Smid finish his program for the Widget Project?
- No.

An ACS could do better:
- No. Smid left the company ten months ago, before the beginning of the Widget Project. Would you like to know where he now is?
- Yes.
- He works for Outstanding Software Inc. I can tell you more about that company if you wish.

Therefore, it suffices to indicate, in the description of the topic, which dimensions or related neighborhoods will be specifically searched, with their values printed, and which dimensions or related neighborhoods will be simply mentioned as areas that the system could further explore. Again, these indications can be user dependent.

The topology provides a methodology for describing active question-answering systems and active data bases (see section 4). The topology provides a systematic approach to the design of these systems. We envision a simple man-machine interface which will allow the user to describe the topics of his domain, their dimensions, metrics, neighborhoods, etc. We hope that, with the use of this methodology, it will be possible to build, relatively rapidly and easily, high quality active systems.

4. ACTIVE DATA BASES: INFORMATION WITHOUT QUESTIONING.

Some persons find that they do not have the time or the desire to access a question-answering system. They may not wish to learn about the use of computers and terminals, or they find that their time is better spent in other pursuits. Such potential users of question- answering systems, for example high level managers, medical doctors, etc., could profit greatly from information in data bases. It may even appear that those who

would profit most from the use of databases, due to the importance of their decisions, are the ones least likely to use them!

Active Data Bases may be the solution. An Active Data Base (Siklóssy [10]) broadcasts information to a user without requiring the user to actually ask for information. In practice, from a user model (section 5) we generate a set of temporary or permanent latent questions that are used as queries to the data base. Ideally, these questions would be precisely those that the user would have generated. They could be implemented as questions, filters or demons (equivalently, triggers.) Active Data Bases would inform the user of new, relevant developments on a timely basis, in the areas of interest to the user, and at the level of detail that the user would want. (Active Data Bases could have other uses too, such as occasional reviews of activities in a field of interest.)

5. USER MODELS AND GOALS.

The inadequacies of question-answering systems that we have discussed can be understood as follows: users have goals and expectations. Their goals are often not well formed. They ask questions to be able to gather information to achieve their goals. They have expectations about the level and amount of detail of an answer, the inclusion of relevant information, etc. They want to be informed of relevant new developments. The active systems that we have described attempt to make it easier for the user to achieve his goals.

Active systems will require user models. We intend to develop frameworks which will permit the description of user models. Much work has been done recently in the area of models of user plans, in particular to understand dialogue: Schmidt [7], Cohen [1], Hayes-Roth [2], Jackson [3], etc. For us, it is particularly important to have user models which can interact

smoothly with the topics topology that was described in section 3.

6. AUTOMATIC RESTRUCTURING OF DATA TO SUPPORT THE TOPOLOGY.

Active systems must not only find answers to a particular query, but must also search the neighborhoods of the query. The techniques developed in data base research have emphasized the development of techniques of storing data and subsequently retrieving such data given specific, single queries. To support efficiently the search within neighborhoods, we must look into novel techniques for data storage. Present storage techniques, left unchanged, cannot provide an adequate support to the development of active systems.

The topology that we have described can be viewed as describing certain types of relations among data items. We can view this topology as an extension of the conceptual schema of a data base. The topology introduces additional considerations not found in current conceptual schema: relationships among neighboring elements in a data base. Some of these relationships are quantitative and are based on metrics; others are qualitative. Data base technology does not provide facilities for easy and efficient passage from one datum to a neighboring datum, especially when the neighborhoods can be along several dimensions. (There are some exceptions: for example, neighborhoods defined by 'next' or 'close by' keys in relational data bases.)

7. CONCLUSIONS.

The Active Systems that we have described fall into two categories: Active Collaborative Systems and Active Data Bases. These two types of systems are complementary. Active

Collaborative Systems will provide pertinent and, in a certain sense, practically complete answers to questions. Active Data Bases will keep users informed of developments in a changing environment. These systems will be built with the help of a single, uniform tool: a descriptive topology of the topics known to the database. The use of a single tool is a far cry from the diverse, often ad-hoc techniques that have been used to remove the inadequacies of passive data base systems. The topology is user dependent, and user models play an important role. One of the goals of this proposal is the development of practical, efficient framework for the description of users. Finally, the practicality of active systems requires new developments in data storage techniques to support the descriptive topology.

8. REFERENCES.

[1] Cohen, P. R. and C. R. Perrault.: Elements of a Plan-Based Theory of Speech Acts (Cognitive Science, 3, 1979, pp. 177-212).

[2] Hayes-Roth, B. and Hayes-Roth, F.: A Cognitive Model of Planning (Cognitive Science 3, 1979, pp. 275-310).

[3] Jackson, P. and Lefrere, P.: On the Application of Rule-Based Techniques to the Design of Advice-Giving Systems (Int. J. Man-Machine Studies, 20, 1984, pp. 63-86).

[4] Kaplan, S. J.: On the Difference between Natural Language and High Level Query Languages (Proc. Association for Computing Machinery 1978 Annual Conference, 1978, pp. 27-38).

[5] Lesmo, L., Siklóssy, L. and Torasso, P.: A Two-Level Net for Integrating Selectional Restrictions and Semantic Knowledge. (Proc. IEEE Int. Conf. on System, Man and Cybernetics, India, 1983, pp. 14-18).

[6] Lesmo, L., Siklóssy, L. and Torasso, P.: Semantic and

Pragmatic Processing in FIDO; A Flexible Interface for Database Operations. (Information Systems, 1984, to appear).

[7] Schmidt, C. F., Sridharan, N. S. and Goodson, J. L.: The Plan Recognition Problem (Artificial Intelligence, 11, 1978, pp. 45-83).

[8] Siklóssy, L.: Question-Asking -·Question-Answering Systems (Proc. Int. Seminar on Intelligent Question-Answering and Database Systems, I.R.I.A., Rocquencourt, France, 1977).

[9] Siklóssy, L.: Impertinent Question-Answering Systems: Justification and Theory (Proc. Association for Computing Machinery 1978 Annual Conference, 1978, pp. 39-44).

[10]Siklóssy, L.: Passive vs. Active Question-Answering (Proc. First Int. Symposium on Policy Analysis and Information Systems, Durham, NC, USA, 1979, pp. 271-276).

[11]Siklóssy, L.: Experiments with a Query Adjusting Knowledge Based System (Proc. Fifth Int. Congress on Cybernetics and Systems, World Organization of General Systems and Cybernetics, Mexico City, 1981).

[12]Wahlster, W., Marburger, H., Jameson, A. and Busemann, S.: Over-Answering Yes-No Questions: Extended Responses in a NL Interface to a Vision System (Proc. 1983 Int. Joint Conference on Artificial Intelligence, Karlsruhe, 1983, pp. 643-646).

[13]Wahlster, W. Cooperative Access Systems. Future Generation Computer Systems, 1, 2, 1984, pp. 103-111.

[14]Webber, B. L. and Mays, E.: Varieties of User Misconceptions: Detection and Correction (Proc. 1983 Int. Joint Conference on Artificial Intelligence, Karlsruhe, 1983, pp. 650-652).

TRAINS, An Active Time-table Searcher

Eduard Tulp
*TOA/AM**
Centrum Voor Informatieverwerking
Croeselaan 22
NL - 3521 CB Utrecht

Laurent Siklóssy
Dept. of Math. and Computer Sc.
Vrije Universiteit
De Boelelaan 1081
NL - 1081 HV Amsterdam

ABSTRACT

TRAINS is an implemented system which helps users find the fastest or most convenient train connections to travel from any one station to any other station within the Netherlands. The entire Dutch railway (NS) network and all its trains are known to TRAINS, including special trains, connection times, restrictions, etc. TRAINS consists of a network editor and a network searching program and features a user friendly user-machine interface. The searching program makes use of AI techniques to limit search and has an active component. This component may suggest trains departing somewhat before the (user-) desired time of departure, or departing after the next available train, if these alternatives are attractive (fewer changes of trains; shorter time traveled, etc). Longer journeys (in kilometers) are also suggested if they end up being shorter (in time). NS plans to use the system.

1. Introduction

The system presented in this paper, TRAINS, is a development system for an electronic railway-guide. TRAINS includes a program for searching a railway network. TRAINS is also a development system, and therefore includes a set of tools (a network editor) for creating and modifying a railway network. The primary goal of TRAINS is to provide the user who queries the system, with a question answering service which advises on train journeys. The user tells TRAINS the departure and end railway station of the planned journey, the date, and either at what time she wishes to depart or at what time she wishes to arrive. The system then searches for the quickest (in time) journey satisfying the user's wishes. Note that the quickest journey (the journey that takes the least time) does not necessarily imply the shortest journey (the journey that covers the least distance). This was done since NS (Nederlandse Spoorwegen; Dutch National Railroad Company) allows a detour for the same minimal fare if that detour makes the journey quicker or more comfortable. So, TRAINS has no tariff interest. The system was designed for use by NS and currently (May 1988) works on the entire 1988 - 1989 Dutch railway schedule. Figure 1 shows a partial map of the network. TRAINS is currently being tested at three NS information centers across The Netherlands and is being developed further at CVI, the software company of NS.

Practice shows that people often have great difficulty finding a quick route from one city to another using a conventional railway-guide. Such a guide is usually divided into time-tables (100 for the Dutch network). A table may include multiple trains and thus connections. Manual search is usually time-consuming. When the distance between two cities is large and the journey consists of several stages, finding a solution requires searching multiple tables in parallel. A conventional guide then appears

*Work partially carried out at the Dept. of Math. and Computer Science, Vrije Universiteit.

unsatisfactory. The temptation is great to use few tables, even though multiple tables may lead to a better solution. Sometimes, although it may appear that a single table contains all the required information, this is not the case. Furthermore, once a route is found, no attempts are made to improve it or to find a quicker one.

Searching a network is one of the classic computer tasks and usually the computer is able to do this task far quicker than any human being. Many algorithms for searching networks exist. TRAINS uses an augmented version of Dijkstra's algorithm [Di, 1959] and some AI and programming techniques to improve its efficiency. Dijkstra's algorithm cannot be used unaltered: it does not cope with the time needed to change trains (which translates into a loss of time within a node instead of loosing time traveling from one node to another). In addition Dijkstra's algorithm is too "eager" to leave a station with the first available train in the right direction, and often finds solutions with too many unnecessary train changes.

The efficiency of the implemented algorithm and the speed of the computer allow for a search for quicker or equivalent alternate journeys within acceptable response times (typically 3 - 5 seconds per request on an ATARI ST). The results of this search for alternate journeys are used to implement an *active* component: the program gives answers to questions the user did not specifically ask, but which the user might have asked if she had had much more information about the domain. It is easily seen that an active component is necessary in an application like TRAINS. A small change in the question, for example a desired departure 5 minutes earlier, might result in a shorter journey due to better connections or faster trains, or perhaps a journey with fewer train changes. For a discussion of active systems see [Si 1978, 1979, 1984]. A good review of active systems can be found in [Wa, 1985].

Bearing in mind that we wanted a high performance system running on a personal computer, it appeared improbable that such

a system could be built within an expert system shell. So we used standard programming tools. We would be curious to know if an equally high performance, together with guaranteed best solutions, can be achieved using an expert system shell (this can be viewed as a challenge to builders and users of expert system shells!).

2. Data Representation

In TRAINS, the railway network is represented by a graph*. Each city is represented by a node, a train connection is represented by an arc. The railway network graph is sparse: many cities lie between exactly two other cities. In addition there are relatively few different routes between two given cities. And when an upper bound is placed on the length of alternate routes, even fewer survive.

We differentiate between two levels in the structure of a railway network: a static and a dynamic level. The static part represents the stations (nodes) and rail distances (in kilometers) between adjacent stations (arcs). The static network has almost no parallel arcs. The dynamic level describes the actual trains that run between a city and its neighbors. The length of time it takes to reach a neighbor is the length of the arc. This time is not fixed, it is time-of-day (and possibly special day, e.g. Sunday or Christmas) dependent. In general, many parallel arcs, each representing a train, will join two nodes.

The static network is significantly smaller and simpler than the dynamic network. Actual calculations must use the dynamic network (which contains the actual trains). The static network is used as an abstraction (and simplification) of the dynamic network to limit search. Using abstract search spaces is a standard technique in Artificial Intelligence (see, for example, [Sa, 1974]).

---

*All our comments about railway networks apply to the Dutch networks. We expect that they would also apply to other railway networks.

The abstraction is used as follows: TRAINS first cuts out that part of the network that contains all static routes between two cities (using only the static network). Actually, an upper bound is placed on the length of the routes, to avoid generating enormous detours, a heuristic limitation which has not resulted in any loss of optimal solutions in our experience. TRAINS then searches for trains on the part we just cut out, using information from the dynamic network. The cutting and searching works efficiently with respect to the active component. Cutting the network once and then searching for trains on a part of the network several times (trying to construct alternate, better routes) is far more efficient than searching the entire network several times.

Earlier we made the observation that many cities have degree two, i.e. they lie between exactly two cities. We do not have to store each of these cities as a node (thereby saving space). A city must be a node only because we can change trains there. It is not much use to change trains in a city that lies between exactly two cities. If we did, the new train we would take, would go to the very same city to which we were going in our original train. The change will only make our journey longer since the train we change to, must leave later than the train we came with. And if the train we change to does not go to that same city, then it must go to the city we came from. Clearly, that is useless. The cities that are not stored as nodes (node cities) are called offset cities. For an offset city we only have to store between which two node cities it lies, how far away it is from these two node cities and the time separating the offset city and its node cities for the different types of trains.

3. Search Algorithm

Our starting point for a search algorithm has been Dijkstra's algorithm [Di, 1959] as described in [Ev, 1979]. Briefly, in Dijkstra's algorithm each node in the network is labeled with its distance from the source node A along the best path that is known

at the time of labeling. When the label of a node is made permanent we have found the shortest route from A to that node. The crux of the algorithm is that when we have found a shortest route from the source A to a node B, then if a shortest route from A to C passes through B, then the shortest route from A to B is part of a shortest route from A to C. So, for each node only the best known path needs to be remembered.

However, in a train network this need not be true since time is needed to change trains. For instance, assume that a change of trains at B requires 4 minutes and consider the following timetable:

	10	20	30
A	9:32	9:32	
B	11:00	11:02	12:02
C		12:46	13:46

It is more efficient to travel with train 20 between A and C (although train 20 is slower than train 10 between A and B). Traveling with train 10, one must wait for the connecting train 30 (we cannot catch train 20 because we do not have the 4 minutes required to change trains at B). However, for node B, Dijkstra's algorithm would only remember the best known path from A to B: train 10. And so, it will give as solution trains 10 and 30! We loose time within the node because of train changes. Conceptually, a node may add length to a path depending on its entrance and exit arcs. The algorithm was modified to take these considerations into account.

If we search forward from a desired time of departure DTD, and we modify Dijkstra's algorithm to include the time cost of train changes, then we will find the best solution which leaves as soon as possible after this desired time DTD. However, it may very well be that a later departure will make us arrive at the same time, therefore offering a better solution. (Similar comments apply if we had searched backwards from the DTA, the desired

time of arrival: the user wants to arrive at or before this DTA.)

In a railway network, Dijkstra's algorithm is most eager to leave a node soon as possible. If we have to wait at a node, we could have caught perhaps a later train from a previous node. (This eagerness to leave a node also results in unnecessary train changes: a later train may be more direct.) Therefore, we modified the algorithm: a search backwards "along" the initial solution will find such alternate later trains. Indeed, it can be shown (see [Tu, 1987]) that the solution thus found is the best among the trains leaving our origin between the DTD and the departure of the train of our solution. We call this our candidate solution. (Similar arguments and proofs hold for searches for trains where a DTA is given.)

It may still happen that a yet later departure will get us to our destination at the same time (*not* earlier) along a different route! We have to make a full backward search to find such alternate routes. However, the candidate solution just found greatly limits the search: backward before the arrival of the candidate solution (candidate found forward), forward after its departure (candidate found backward). This additional search now guarantees a shortest journey within the time interval of the candidate solution.

Furthermore, the algorithm so modified and enhanced can be extended to offset cities, maintaining its optimality properties. The algorithm only needs to consider the two neighboring nodes of the offset city.

4. The Active Component

The theory of active systems was first discussed by [Si 1978, 1979, 1984]. He developed a theory, based on discontinuities, which tells a question answering system when it should provide additional information, information the user had not specifically asked. Hence the term "active system" or "impertinent system". Simply stated, a question answering system should be impertinent if some small change along the question results in some large,

beneficial change in the answer.

In TRAINS, the active component tries to find quicker routes by searching for trains that depart/arrive a bit earlier or later. There is one small difference with the theory of impertinence, however. In this theory, alternatives are only shown to the user if they are strictly better. TRAINS also shows alternatives if they are as good as the initial solution or in special cases even if an alternative is slightly worse. Indeed, the authors suspect that users often overspecify their question: An exact time of departure or arrival is requested because the system requires a specific time. But the user might very well be interested in a train that takes exactly the same time but leaves, say 15 minutes later. That train may be closer to the time the user *really* wants to leave. Another case arises when the solution found by the system is not really close to the user's wishes. For example, a user wants to arrive in Amsterdam at 10:00 AM. The system searches for the best solution with an arrival time before 10:00 AM: let us say 9:45 AM, with a journey taking 30 minutes. Now suppose that an alternate journey is possible with a time of arrival of 10:05 AM and a duration of 35 minutes (so in terms of travel-time a worse solution). TRAINS will also provide this possibility for it is "closer" to the question at little extra cost in traveling time. We have assumed, that the user does not know exactly when she wants to leave or arrive until she has seen the actual possibilities, which the system must provide.

5. Examples of Results

The Dutch railway network has a combined length of 3000 kilometers and consists of 364 railway stations. Of the 364 stations, 153 had to be stored as node cities and 211 could be stored as offset cities. From these 153 node cities, approximately 22000 trains depart. By using various encoding techniques, the database could be limited to 180 Kilobytes.

TRAINS was demonstrated several times at NS headquarters

in Utrecht. At these demos, it became clear that indeed no one could search as fast and as well as TRAINS did. Surprisingly, TRAINS managed to find routes that NS apparently never thought of. For an example, consider a trip from Vlissingen {29}* to Zwolle {33}. In their time-table NS have included a special table (table 60) which includes direct trains from Vlissingen to Zwolle via the route Vlissingen - Roosendaal {21} - Tilburg {25} - Nijmegen {18} - Arnhem {5} - Zutphen {32} - Zwolle. However, TRAINS will never suggest such a route: *at all times* the route Vlissingen - Roosendaal - Dordrecht {8} - Rotterdam {22} - Gouda {11} - Utrecht {27} - Amersfoort {3} - Zwolle (which can be found by using tables 10 and 80) is faster. The former route has a travel-time (tt) of 3:47 at best and 3:56 at worst; by the latter route the tt is 3:18 minutes at best and 3:32 minutes at worst. For a departure around 12:00, then by the former route one departs at 12:14 and the tt is 3:56; by the latter route one departs at 11:56 the tt is 3:18.

Similarly, from Hoorn {15} to Den Haag HS {36}. Table 41 gives a route via Alkmaar {1} - Haarlem {13} - Leiden {17}. The tt is 1:36 (with one train change). If one uses tables 40 and 10 which give a route via Zaandam {31} - Sloterdijk {24} - Haarlem {13} - Leiden {17}, the tt is 1:20 (also one train change).

Another example of a good alternate route used by TRAINS is the trip from Beverwijk {6} to Purmerend {19}. The shortest route (32 km) is via Uitgeest {26} - Zaandam {31}. However, the route via Haarlem {13} - Sloterdijk {24} - Zaandam {31} (47 km), is always faster. The savings can be up to 30 mn; these savings are considerable, since the best journey (found by TRAINS) takes typically 0:48 via Sloterdijk.

An excellent example of TRAINS' active component was unintentionally provided by an NS employee during one of the

---

*Cities with numbers in {} are drawn in figure 1.

demos. The previous day, he had to travel from Voorburg {30} to Amsterdam RAI {35} and he had to arrive around 10. He, an experienced user of the time-tables, departed at 8:53 and arrived at 10:05. Changing trains three times. TRAINS found the same answer, but also pointed out that if he had left 4 mn earlier, then he would have arrived 15 mn earlier with only one train change! TRAINS also suggested leaving at 9:19 and arriving at 10:20.

For examples of the performance of TRAINS see [Tu, 1987]. Here are two of them.

For a short trip, consider from Heemskerk {14} to Amsterdam CS {34} arriving around 8:00 AM. Within 4 s, TRAINS gives an initial solution via Haarlem {13}, 7:00 to 7:41 (tt 0:41, no changes). Within 5 more s, three alternatives are generated: two routes via Uitgeest {26}: 6:57 to 7:34 and 7:28 to 8:05 (both changing at Uitgeest, a 5 mn wait) and tt 37 minutes, and one route via Haarlem 7:31 to 8:11 (tt 0:40).

For a long trip, consider Chèvremont {7} to Visvliet {28}, departing around 8:00. Within 5 s TRAINS gives an initial via Sittard {23} - Roermond {20} - Eindhoven {9} - Utrecht {27} - Amersfoort {3} - Zwolle {33} - Groningen {12}, 8:38 to 13:38 (tt 5:00). Within 14 more s, two alternatives are proposed: one via the same route 7:37 to 12:38 (tt 5:01); the other via Zwolle {33} - Leeuwarden {16}, 7:01 to 11:51 (tt 4:50).

6. Conclusions

TRAINS can search the Dutch railroad network efficiently. It is a complete system, taking into account times to change trains, special trains, etc. It provides not only an optimal answer to the user's request, but searches actively for other good solutions in the neighborhood of the user's initial question. Just like most users, TRAINS prefers journeys with fewer train changes. Unlike most railroad information employees, it will suggest better trains which depart a little before the user's suggested time of departure, or arrive a little after the user's suggested time of arrival.

7. References

[Di, 1959] Dijkstra, E.W., "A Note on Two Problems in Connection with Graphs", *Num. Math.*, **1**, 269 - 271.

[Ev, 1979] Even, S., *Graph Algorithms.* Potomac, Md.: Computer Science Press.

[Sa, 1974] Sacerdoti, E. D.,"Planning in a hierarchy of abstraction spaces", *Art. Int.*, **4**, 145 - 180.

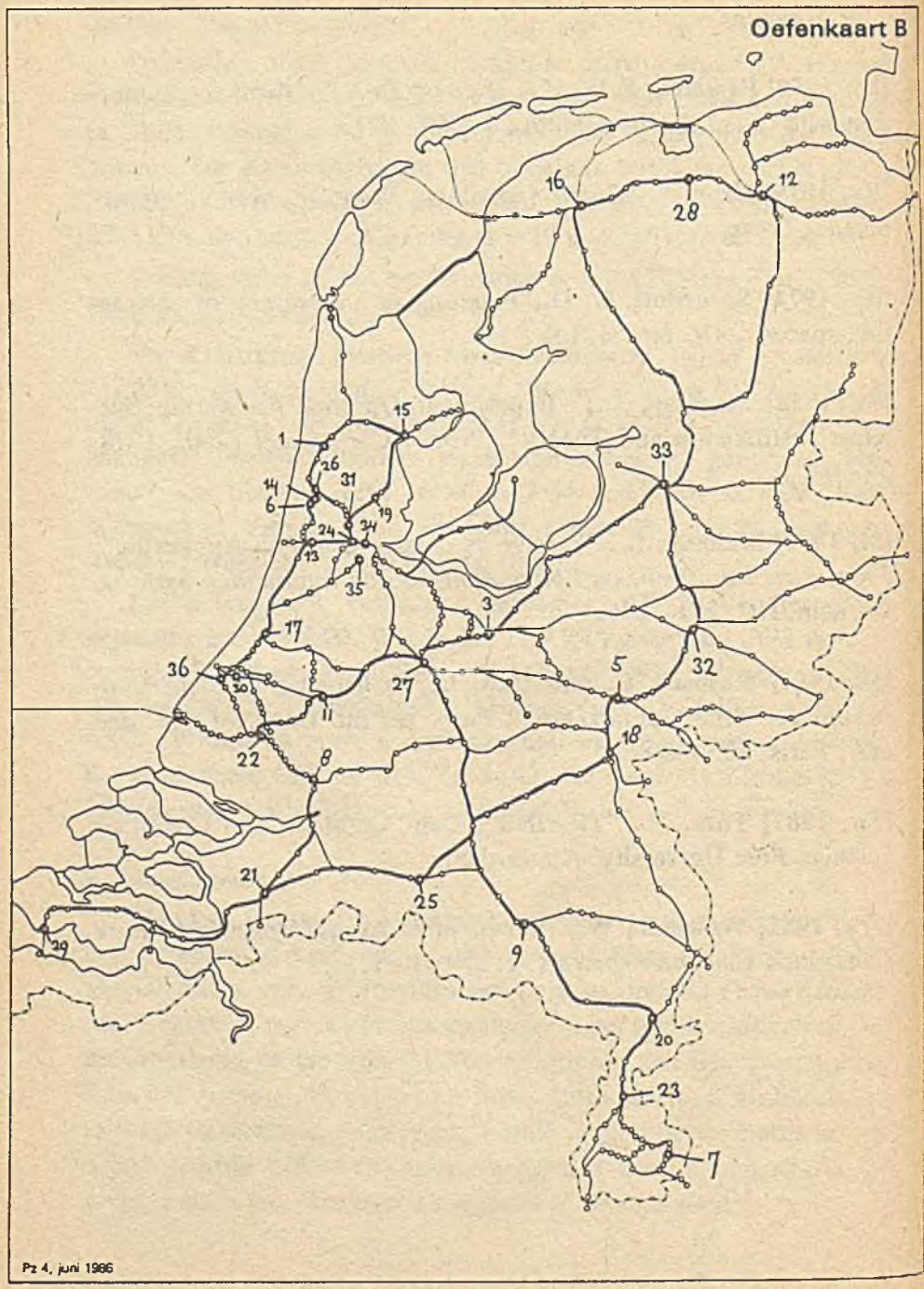
[Si, 1978] Siklóssy, L., "Impertinent Question-Answering Systems: Justification and Theory", *Proc. ACM Annual Conf.*, 1978, 39 - 44.

[Si, 1979] Siklóssy, L., "Passive vs. Active Question Answering", *Proc. First Int. Symp. on Policy Analysis and Information Systems*, Durham, NC, 271 - 276.

[Si, 1984] Siklóssy, L. and Cao, D., "Changing Problem Constraints for True Optimization", *Proc. 6th Int. Cong. of Cyb. and Sys.*, Paris, 423 - 428.

[Tu, 1987] Tulp, E., "TRAINS", Dept. of Math. and Computer Science, Free University, Amsterdam.

[Wa, 1985] Wahlster, W., "Cooperative Access Systems", *Future Generation Computer Systems*, **1**, 2, 103 - 111.


Oefenkaart B
16
28
12
15
1
26
31
14
6
19
24
34
13
35
33
3
17
36
30
27
5
32
11
22
8
18
21
25
29
9
20
23
7

Jesienna Szkoła PTI

Mrągowo, grudzień 1988

LOGIKI NIEKLASYCZNE

(dla informatyka pracującego)

Władysław M. Turski

Instytut Informatyki

Uniwersytetu Warszawskiego

PKiN pok. 850, 00-901 Warszawa

1. Co to jest logika?

Niekiedy wystarczy zadać jedno pytanie i już mamy temat do dowolnie długiej dyskusji. Tak właśnie jest z pytaniem, które przewrotnie wybrałem na tytuł pierwszego punktu niniejszego wykładu. Ulubionych poglądów na tę sprawę jest bowiem co najmniej tyle, ilu słuchaczy. Ja sam mam ich zresztą kilka.

Istnieje tradycja, która w logice każe upatrywać sztukę myślenia; jest także jej skromniejsza wersja, wiążąca logikę z umiejętnością rozumowania. Chcąc wyrazić dezaprobatę względem pewnego sądu, postanowienia, przepisu a nawet czynu lub postępku, dość powszechnie przypisujemy im brak logiki. Wykrzykniki 'to nielogiczne' i 'to nie ma sensu' nie różnią się zbytnio w potocznym języku. I przeciwnie, przypisując czemuś walor logiczności, wyposażamy to coś w atrybuty prawdziwości i poprawności (niezbyt ściśle zresztą rozgraniczane), a w każdym razie - wyrażamy aprobatę.

Związek logiki z prawdziwością wypowiedzi - przynajmniej w potocznej mowie - jest zupełnie powszechny, choć trudno byłoby znaleźć uzasadnienie takich poglądów. Na pewno nie dostarczają go sami logicy, którzy od tysiącleci podkreślają różnice między poprawnością i prawdziwością wypowiedzi, tę pierwszą rozdzielając jeszcze na poprawność składniową i dowodliwość, czyli zgodność przyjętymi regułami wnioskowania. Co do tego zaś, jak logicy traktują prawdziwość wypowiedzi, posłużymy się cytatem liczącym bez mała dwa tysiące lat: *O każdym bowiem zdaniu rozstrzyga się, że jest prawdziwe, albo że jest fałszywe ze względu na jego odniesienie do tej rzeczy, o której zostało orzeczone. Jeżeli bowiem okazuje się ono zgodne z rzeczą, o której zostało orzeczone, wydaje się prawdziwe, jeżeli niezgodne - fałszywe.* [1]

To stanowisko zyskało poparcie dziesiątków pokoleń logików, jak byśmy dziś powiedzieli: zawodowych. Jest ono też powszechne np. wśród prawników, którzy starannie rozdzielają prawdę procesową (logiczną) od rzeczywistej, do której nie ma się nieraz jak odnieść, skoro w argumentacji procesowej brak odpowiednich instrumentów ('punktów zaczepienia').

Nie jest więc logika gwarantem *prawdziwości* wypowiedzi lecz jedynie jej *poprawności*, pojmowanej w dosyć wąskim sensie, jako zgodność z uprzednio przyjętymi regułami gramatyki i wywodu.

Podobnie, nie ma żadnych podstaw by utożsamiać logikę, uprawianą jako dyscyplina naukowa z procesem myślenia (rozumowania). Nikt chyba nigdy nie przyłapał się na świadomym stosowaniu reguł wnioskowania na własny użytek.

Jeśli pada deszcz to idę do kina.

Właśnie pada deszcz.

Przeto idę do kina.

- to dobry przykład *wypowiedzi* uzasadniającej dlaczego idę do kina, lecz nader kiepski protokół procesu myślowego, który doprowadził do podjęcia decyzji o pójściu do kina. Rzeczywisty proces myślowy nie ograniczał się zapewne do

wymienionych dwu przesłanek; jest wielce prawdopodobne że wpłynęło nań wiele innych czynników, takich choćby, jak bieżący program telewizyjny, pora dnia, stan odcisków na stopach itd., które śmiało mogliśmy wyeliminować z wypowiedzi, ale których niesposób pozbyć się z myśli. Wielu czynników wpływających na rzeczywisty proces myślowy możemy nie uwzględniać w wypowiedzi, gdyż w przytoczonej, uproszczonej postaci jest ona już wystarczająco przekonywająca i nikt, kto ją przyswoił, widząc, jak w potokach deszczu udajemy się do kina, nie może nam zarzucić 'nielogiczności' postępowania.

Innymi słowy, nawet gdyby nasze rozumowanie było rzeczywiście tak proste i uwzględniało tak ubogi zestaw przesłanek, zewnętrzny obserwator nie dostrzegłby rozbieżności między naszym postępowaniem i jego uzasadnieniem. Zauważmy jednak, że przyjmując przytoczoną wypowiedź za dobrą monetę, powinniśmy iść do kina również wtedy gdy deszcz pada, a kino jest zamknięte, albo gdy do najbliższego kinoteatru jest kilkadziesiąt kilometrów. Naturalnie, gdyby któraś z tych okoliczności rzeczywiście występowała i mimo padającego deszczu nie poszlibyśmy do kina, wypowiedź uzasadniająca nasze postępowanie musiałaby być inna.

Rozsądna i oszczędna wypowiedź uwzględnia te tylko czynniki, które są potrzebne i wystarczające do przekonania rozmówcy; myślenie obejmuje także - co najmniej! - ocenę istotności innych czynników. Jak pamiętamy, wielki Holmes rozwiązał zagadkę kryminalną zauważywszy, że pies nie szczekał; to natomiast, że w tym samym czasie prababka panny Krysi zgubiła łyżwy, wcale go nie interesowało!

Logika nie jest więc ani gwarantem prawdy, ani zwierciadłem rozumowania. Czym jednak jest i dlaczego warto nauczyć się nią posługiwać?

Logika jest *rachunkiem*, systemem formalnym pozwalającym mechanicznie (tj. bez natchnienia i twórczości) wykonywać działania na pewnych obiektach, przy czym zarówno same te obiekty, jak i wyniki wykonanych działań w stosunkowo

naturalny sposób można systematycznie interpretować jako wypowiedzi o innych obiektach.

Nie ma przy tym jakiejś jednej, uniwersalnie obowiązującej logiki. Znamy wiele rachunków, pozwalających działać na wypowiedziach; można wymyślić bardzo wiele innych. Użyteczność *każdego* takiego rachunku polega na tym, że zamiast poddawać krytycznej analizie treściowej wszystkie, nieraz bardzo długie i skomplikowane wypowiedzi, badamy zasady rachunku (reguły wnioskowania) i poprawność ich zastosowania w konkretnym wnioskowaniu (prowadzącym do interesującej nas wypowiedzi). Jeśli wnioskowanie jest poprawne, a jego reguły uprzednio zaakceptowaliśmy, to nie kwestionując przesłanek wnioskowania (prostych wypowiedzi, których prawdziwość nie podlega logicznej analizie) musimy przyjąć jego konkluzje.

Inaczej powiedziawszy, jeśli nie podoba się nam konkluzja, to musimy albo zakwestionować przesłanki, albo odrzucić daną logikę. Rzecz całą można też ująć pozytywnie: jeśli zaakceptowaliśmy logikę i pewien zestaw przesłanek, zgadzamy się akceptować wszystkie konkluzje, jakie można uzyskać drogą poprawnego stosowania jej reguł wnioskowania. Ewentualne wątpliwości i zastrzeżenia odnosimy przeto nie do poszczególnych wnioskowań lecz do pierwotnych przesłanek (zwanych *aksjomatami pozalogicznymi*) i do użytej logiki. Ponieważ jednak logiki nie stosuje się jednorazowo, opłaca się zainwestować nawet sporo wysiłku w jej analizę, żeby potem móc stosunkowo 'tanio' i bezpiecznie z niej korzystać. (Jak wcześniej wspomnieliśmy, prawdziwość aksjomatów pozalogicznych z reguły nie poddaje się sprawdzeniu drogą spekulacji myślowych; wyjąwszy pewne niezbyt interesujące nas tutaj przypadki, wymaga ona weryfikacji doświadczalnej.)

Krótko powiedziawszy, każda *logika to pewien zestaw uznanych reguł działania na wypowiedziach.*

Dlaczego nie posługujemy się jedną logiką? Po prostu dlatego, że nie znamy takiego zestawu reguł, który pozwoliłby wyrazić wszystkie argumentacje, jakie

skłonni jesteśmy uznać za przekonywające. W różnych okolicznościach i w różnych celach posługujemy się różnymi sposobami przekonywania rozmówców. Nawet jeśli ograniczyć się do językowych tylko sposobów przekonywania (a ludzie stale posługują się i innymi), bez trudu spostrzeżemy pokaźną rozmaitość używanych schematów argumentowania. Jednymi posługujemy się częściej, innymi - rzadziej, jeszcze innymi - tak rzadko, że kiedy wypada się do nich uciec, brak wprawy powoduje, iż trochę się nam one mylą. Niekiedy zawodzą też schematy, którymi posługujemy się z całą ufnością.

Każde zdanie twierdzące jest albo prawdziwe albo fałszywe, słynna zasada wyłączonego środka, przez scholastyków nazwana *tertium non datur*, zawodzi znacznie częściej niż by się można po tak szacownym schemacie spodziewać. Istotnie, zawodzi ona nie tylko w odniesieniu do zdań nie poddających się empirycznej weryfikacji ('Jednorożce mają szczerozłote kopyta'), co w końcu można by jakoś przeboleć albo zatuszować odpowiednią konwencją (ogłaszając np. że wszystkie zdania z natury rzeczy niemożliwe do zweryfikowania uznajemy za prawdziwe), lecz także w odniesieniu do zdań, które łatwo można by zweryfikować, gdyby nie to, że historia świata stworzyła inny kontekst. Zdanie 'przestałem grać na mandolinie', wypowiedziane przez kogoś, kto grywał na tym instrumencie istotnie jest albo prawdziwe, albo fałszywe; natomiast jeśli wypowiem je ja (którym nigdy nie miał mandoliny w ręku!), to nie będzie ani takie, ani takie, a w przypadku zadekretowania prawdziwości (albo fałszywości, obojętnie) doprowadzi do sprzeczności z innymi, całkowicie weryfikowalnymi zdaniami.

Podobnie, stosowanie zupełnie naturalnych schematów wnioskowań, powszechnie uważanych za niezawodne i jak najbardziej logiczne, może doprowadzić do ewidentnie nonsensownych konkluzji. Schemat wnioskowania

Jeśli A to B.

A.

Przeto B.

zwany regułą odrywania albo *modus ponens*, we wszystkich podręcznikach logiki szkolnej uznawany jest za niezawodny. Tymczasem wnioskowanie

Jeśli Cezar nie zostanie zabity przez spiskowców

to (Cezar) wygłosi przemówienie w Senacie. [B --> C]

Jeśli Cezar pozostanie w domu

to (Cezar) nie zostanie zabity przez spiskowców. [A --> B]

Cezar pozostanie w domu. [A]

Przeto (Cezar) wygłosi przemówienie w Senacie. [C]

będące dwukrotnym zastosowaniem tego schematu, (((A,A-->B)-->B,B-->C)-->C), jest ewidentnie wadliwe. Co zawiodło? Schemat czy jego zastosowanie? W gruncie rzeczy, odpowiedź jest obojętna, bo zupełnie niezależnie od niej widzimy, że aby wnioskowania dotyczące podobnych sytuacji miały moc przekonywającą należy posługiwać się inną logiką.

Nawiasem powiedziawszy, to, że bez trudu dostrzegamy wadliwość przytoczonego wnioskowania, że razi nas absurdalność konkluzji wyprowadzonej z całkowicie dopuszczalnych przesłanek, dobitnie świadczy o odmienności istoty rozumowania i wnioskowania logicznego.

2. Logika klasyczna

Logika klasyczna, jaka jest, każdy wie.

3. Kołmogorowa rachunek zadań, czyli logika intuicjonistyczna

Przegląd logik nieklasycznych zaczniemy od jednej z najdawniejszych - logiki intuicjonistycznej - w bardzo atrakcyjnej postaci nadanej jej przez A.A. Kołmogorowa w roku 1931 [2]. Logika intuistycjonistyczna jest bodajże najważniejsza z wszystkich logik nieklasycznych; cała wielka szkoła matematyczna konstruktywistów, nie uznająca logiki klasycznej (a ściśle powiedziawszy, zasady wyłączonego środka), posługuje się różnymi wariantami logiki intuicjonistycznej. Wersja,

która tu naszkicujemy, powinna być szczególnie przydatna dla osób zajmujących sie metodyka zastosowań informatyki.

Przyjmijmy, że wiemy, co to znaczy 'zadanie'. Dla przykładu, oto pięć zadań:

1. Znaleźć cztery liczby całkowite x,y,z,n, które spełniają związki

$$x^n + y^n = z^n, \quad n > 2$$

2. Udowodnić fałszywość tzw. wielkiego twierdzenia Fermata.

3. Przeprowadzić okrąg przez trzy zadane punkty p,q,r, nie posługując sie innymi narzędziami niż cyrklem i linijką.

4. Zakładając, że znany jest jeden pierwiastek równania

$$ax^2 + bx + c = 0$$

znaleźć drugi pierwiastek tego równania.

5. Zakładając, że liczba *pi* jest wymierna, *pi* = n/m, znaleźć podobne wyrażenie dla liczby e.

Zauważmy, że pierwsze dwa zadania są niekoniecznie równoważne: wystarczy co prawda rozwiązać zadanie 1, by mieć rozwiązanie zadania 2, ale być może można pomyślnie rozwiązać zadanie 2 (tj. udowodnić fałszywość twierdzenia Fermata) nie podając przy tym konkretnego kontrprzykładu. Zauważmy także, iż przy pozornie pełnym podobieństwie ostatnich dwu zadań, zadanie 4 jest rozsądne, podczas gdy zadanie 5 jest bezsensowne, gdyż założenie o wymierności liczby *pi* jest niemożliwe do spełnienia. Wykazanie, że zadanie jest bezsensowne będziemy uważać za rozwiązanie tego zadania.

Zadania będziemy oznaczać małymi literami a, b, c, ... Jeśli a oraz b są zadaniami, to przez a ∧ b będziemy rozumieć zadanie polegające na rozwiązaniu obydwu zadań (a tudzież b); natomiast przez a ∨ b - zadanie polegające na rozwiązaniu przynajmniej jednego z nich (a lub b). Zapis a => b będzie oznaczać tyle co: 'zakładając, że dysponujemy rozwiązaniem zadania a, rozwiązać zadanie b', albo (równoważnie): 'sprowadzić rozwiązanie zadania b do rozwiązania zadania a'.

Nie zakładamy, że każde zadanie można rozwiązać. Gdyby np. twierdzenie Fermata było prawdziwe, rozwiązanie zadania 1 byłoby sprzeczne z tym twierdzeniem. Wprowadzimy przeto zapis ¬a oznaczający zadanie: 'zakładając, że dane jest rozwiązanie zadania a, znaleźć kontrprzykład'.

Każda formuła, w której występują zadania a,b,c,..., znaki ∧,∨,¬ i => oraz, być może, nawiasy, zgodnie z powyższymi zasadami jest traktowana jako zadanie. Dla 'typowej' formuły będziemy używać oznaczenia p(a,b,c,...). Jeśli a,b,c,... są nie tyle konkretnymi zadaniami, ile zmiennymi mogącymi przyjmować wartości zadań, formuła p(a,b,c,...) staje się *funkcją zadaniową*; po podstawieniu konkretnych wartości (tj. zadań) za a,b,c,... otrzymuje się wartość tej funkcji (w pewnym 'punkcie'), czyli - konkretne zadanie.

Niech x będzie zmienną (dowolnego typu), zaś a(x) - zadaniem zależnym od tej zmiennej. Przez Ax:a(x) będziemy rozumieć zadanie: 'znaleźć *ogólną metodę* rozwiązywania zadania a(x)', czyli 'podać sposób rozwiązywania wszystkich zadań, które powstają z a(x), gdy x przyjmuje wszystkie możliwe znaczenia'. Jeśli mamy funkcję zadaniową p(a,b,c,...) z wolnymi zmiennymi a,b,c,... to zamiast AaAbAc...: p(a,b,c,...) będziemy pisać [p(a,b,c,...)]; ten ostatni zapis oznacza więc 'zadanie polegające na określeniu ogólnej metody rozwiązywania zadania p(a,b,c,...) przy dowolnym wyborze zadań a,b,c,...

Jesteśmy już wyposażeni we wszystkie środki notacyjne do zaprezentowania elementarnej teorii rozwiązywania zadań.

Przyjmiemy, że potrafimy rozwiązywać następujące zadania:

(3.1) [a => a ∧ a]

(3.2) [a ∧ b => b ∧ a]

(3.3) [a ∧ b => (a ∧ c => b ∧ c)]

(3.4) [(a => b) ∧ (b => c) => (a => c)]

(3.5) [b => (a => b)]

(3.6) [a ∧ (a => b) => b]

(3.7) [a => a ∨ b]

(3.8) [a ∨ b => b ∨ a]

(3.9) [(a => c) ∧ (b => c) => (a ∨ b => c]

(3.10) [¬a => (a => b)]

(3.11) [(a => b) ∧ (a => ¬b) => ¬a]

Założenie, że umiemy rozwiązywać zadania (3.1) - (3.11) nie jest wygórowane. Istotnie, przyjrzyjmy się np. zadaniu (3.3). Czy rzeczywiście potrafimy je rozwiązywać? Jeśli umiemy rozwiązywać zadanie a ∧ b to znaczy, że umiemy rozwiązywać zadanie a i zadanię b, wobec czego istotnie nie sprawi nam trudności sprowadzenie rozwiązania zadania b ∧ c do rozwiązania zadania a ∧ c; inaczej powiedziawszy, z umiejętności rozwiązywania zadania a ∧ b istotnie wynika umiejętność redukowania rozwiązania zadania b ∧ c do zadania a ∧ c. Formuła (3.3) to właśnie wyraża. Podobnie można sobie 'wytłumaczyć' pozostałe formuły (3.1) - (3.11).

Zwróćmy jeszcze uwagę na (3.10). Jeśli potrafimy rozwiązać zadanie ¬a, to wiemy, że rozwiązanie a jest niemożliwe, wobec czego zadanie a => b nie ma sensu, tj. - zgodnie z umową - mamy jego rozwiązanie; innymi słowy, rzeczywiście każde zadanie typu a => b potrafimy rozwiązać przez redukcję do zadania ¬a.

Załóżmy dodatkowo, że dla dowolnych funkcji zadaniowych p,q,r,s,... potrafimy rozwiązywać następujące zadania

(3.12) Dysponując rozwiązaniem zadania [p ∧ q], rozwiązać zadanie [p]

(3.13) Dysponując rozwiązaniem zadań [p] i [p => q], rozwiązać zadanie [q]

(3.14) Dysponując rozwiązaniem zadania [p(a,b,c,...)], rozwiązać zadanie [p(q,r,s,....)]

Możemy obecnie przystąpić do (nieskończonej) rozbudowy listy zadań, które potrafimy rozwiązać. Zaczynamy od wykazu (3.1) - (3.11). Gdy na naszej liście znajdziemy [p ∧ q] (p i q są dowolnymi funkcjami zadaniowymi!) to na mocy

założenia (3.12) dopisujemy do niej [p]. Kiedy na liście znajdziemy obydwa zadania [p] i [p ∧ q], to na mocy (3.13) dopisujemy do niej [q]. Jeśli wreszcie znajdziemy na naszej liście [p(a,b,c,...)], zaś r,s,t,... są dowolnymi funkcjami zadaniowymi, to na mocy (3.14) dopisujemy do listy [p(q,r,s,...)]. Proces ten możemy kontynuować dowolnie długo, uzyskując rozwiązania coraz to nowych zadań. (Naturalnie, w praktyce interesować nas będzie postępowanie odwrotne: mając konkretne zadanie, szukamy jego rozkładu na zadania, które już umiemy rozwiązywać!).

Gdyby lista podstawowych formuł zawierała

(3.0) [a ∨ ¬a]

to opisany rachunek logiczny byłby całkowicie równoważny logice klasycznej (rachunkowi pierwszego rzędu). Jednakże w przyjętej interpretacji formuła (3.0) oznaczałaby, że dysponujemy ogólną metodą rozwiązywania *dowolnego* zadania lub - przy założeniu istnienia takiego rozwiązania! - znajdowania kontrprzykładu. Oczywiście nie pretendując do wszechwiedzy, nie możemy na to przystać.

W związku z wykluczeniem formuły (3.0), w omawianej logice nie obowiązuje zasada

(3.*) [¬¬a => a]

(Nie twierdzimy, że dysponujemy ogólną metodą rozwiązywania zadania [¬¬a=>a]). Zasada (3.*) jest przy tym zupełnie naturalna w logice klasycznej ('podwójne przeczenie jest twierdzeniem').

W rachunku zadań można jednak wykazać, że potrafimy rozwiązywać zadania postaci

(3.**) [¬¬(a ∨ ¬a)]

W logice klasycznej z formuły (3.**) po zastosowaniu (3.*) natychmiast wynika zasada wyłączonego środka; ponieważ jednak w logice intuicjonistycznej

nie dysponujemy formułą (3.*), reguły *tertium non datur* nie da się wprowadzić 'kuchennymi schodami'.

Warto zapewne uzmysłowić sobie 'co tu jest grane'. Budujemy aparat formalny, pozwalający manipulować wypowiedziami na temat powiązań między zadaniami, a w szczególności, pozwalający wyliczać pewne (niezawodne) schematy redukcji jednych zadań do innych. Jak w każdym rachunku logicznym, musimy przyjąć pewną liczbę aksjomatów. Postanawiamy nie przyjmować aksjomatu (3.0), gdyż w przyjętej przez nas konwencji, *która w odniesieniu do innych aksjomatów, (3.1) - (3.14), nam się podoba*, akceptacja aksjomatu (3.0) oznaczałaby uznanie za pewnik naszej wszechwiedzy (umiejętności rozwiązania wszystkich zadań), co pominąwszy nawet kwestię skromności, byłoby sprzeczne z zamiarem zbudowania rachunku zadań.

Decyzja, by nie uznać aksjomatu (3.0), powoduje, że musimy uważać, aby nie pojawił się on jako następstwo innych aksjomatów. I dlatego skoro tylko uświadomimy sobie, że z (3.1) - (3.14) wynika (3.**), czyli formuła, która mały tylko kroczek dzieli od zasady wyłączonego środka, nie możemy nawet marzyć o tym by przyjąć pozornie niewinny i naturalny aksjomat (3.*); na szczęście on sam nie wynika z (3.1) - (3.14). Chwila zastanowienia wystarcza jednak by uświadomić sobie, że chęć na (3.*) jest wtedy tylko naturalna, gdy przyjmujemy zasadę *tertium non datur* i przeciwnie, jeśli przyjmiemy za naturalne (3.*), to nie możemy się bronić przed przyjęciem zasady wyłączonego środka.

Na zakończenie sekcji poświęconej logice intuicjonistycznej warto podkreślić, że podana przez Kołmogorowa interpretacja wyprzedza informatykę o dobre ćwierć wieku, a mimo to ma zupełnie bezpośrednie związki z aktualną tematyką badawczą, np. z tzw. programowaniem w logice i problemem interpretacji negacji w bazach wiedzy.

4. Logika domniemań

O ile logika intuicjonistyczna, przedstawiona w poprzednim punkcie w najciekawszej bodajże postaci - rachunku zadań, jest ściśle związana z logiką klasyczną, gdyż różni się od niej brakiem jednej tylko zasady, (3.0), o tyle logika, którą chcę przedstawić w tym punkcie, jest zupełnie inna.

Kolekcje wypowiedzi uzyskiwane z przyjętych aksjomatów przy pomocy reguł manipulacyjnych obydwu wspomnianych dotychczas logik, klasycznej i intuicjonistycznej, cechuje pewien konserwatyzm. Wypowiedzi zasadnie uznane za twierdzenia pozostają twierdzeniami na zawsze; żadne nowe twierdzenie nie ma wpływu na poprawność uprzednio skonstruowanych dowodów. Wielu ludzi uważa, że taki konserwatyzm rachunku logicznego ogranicza jego stosowalność, rozmaitość dziedzin, w których można skutecznie modelować teorie budowane przy jego użyciu.

W ostatnich latach, zwłaszcza pod presją badań z dziedziny tzw. sztucznej inteligencji, w których za wszelką cenę dąży się do aproksymacji rozumowania regułami formalnego rachunku, powstało wiele interesujących systemów *logiki niemonotonicznej*, pozbawionej wspomnianego konserwatyzmu. Odznaczają się one tym, że w miarę uznawania nowych wypowiedzi następuje rewizja całokształtu wypowiedzi i niektóre z nich, dotychczas uznawane, mogą być odrzucone. Krótko powiedziawszy, w logikach niemonotonicznych, nowe stwierdzenia mogą obalać stwierdzenia uprzednie nie prowadząc do sprzeczności systemu.

Typową logiką niemonotoniczną jest *logika domniemań*, [3], której matematycznie systematyczną postać nadał W.Łukaszewicz [4], [5].

Cechą charakterystyczną logik niemonotonicznych jest dopuszczenie bardzo liberalnych reguł wnioskowania o postaci:

Jeśli zdania R1,...,Rn nie są uznane, uznaj zdanie Q

a więc np.

Jeśli nie zostało udowodnione, że Sokrates jest śmiertelny,

przyjmij, że Sokrates jest bogiem.

Wprowadźmy operator M(.), który będziemy interpretować tak: 'uznanie zdania (.) nie prowadzi do sprzeczności'. Operator ten można ściśle określić przy pomocy związku

(4.0) $\underline{M}(P) = \neg(L(\neg P))$

w którym $\neg$ jest operatorem klasycznej negacji, zaś L(.) - operatorem interpretowanym jako: 'zostało udowodnione, iż (.)'.

Tak więc M(P) możemy traktować jako skrót wypowiedzi: 'nie zostało udowodnione, że nie P'.

Podstawową regułą domniemywania jest formuła

(4.1) $A(x) \wedge \underline{M}(P(x)) \Rightarrow W(x)$

głosząca, że jeśli uznajemy A(x), czyli *warunek wstępny*, oraz jeśli uznanie *warunku niesprzeczności*, P(x), nie prowadzi do sprzeczności, to należy uznać *następstwo* W(x).

Zamiast postaci (4.1), przypominającej formuły modalne, o których będzie mowa w następnych sekcjach, regułę domniemywania przedstawia się w literaturze często w postaci

$$(4.1') \quad \frac{A(x) : \underline{M}(P(x))}{W(x)}$$

Jeśli ani warunek wstępny, ani warunek niesprzeczności, ani następstwo nie zawierają zmiennych wolnych, domniemanie (4.1) nazywamy zamkniętym. Podstawiając konkretne wartości za wolne zmienne otwartego domniemania, otrzymujemy domniemanie zamknięte. Naturalnie, systemy wnioskowań zawierające domniemania otwarte są, znacznie bogatsze w konsekwencje (wynikające z licznych konkretyzacji) od systemów z samymi domniemaniami zamkniętymi.

Szczególnie często stosowany wariant domniemań nosi nazwę *domniemania normalnego*

(4.2) $A(x) \wedge \Box(P(x)) \Rightarrow P(x)$

Przykładem domniemania normalnego może być reguła

Jeśli nadlatuje samolot i nie można wykluczyć, że jest to samolot wroga, to uznajemy, że jest to samolot wroga.

W logice domniemań, jak w każdej logice, oprócz gramatyki i reguł wnioskowania, interesują nas *teorie*, czyli maksymalne zbiory następstw, jakie daje się wyprowadzić z ustalonych, początkowych układów aksjomatów pozalogicznych.

W logice klasycznej (i intucjonistycznej też), pojęcie teorii jest stosunkowo proste i jego realizacja nie nastręcza większych kłopotów. Bierzemy zadany układ aksjomatów pozalogicznych, uznajemy go za początkowy zbiór twierdzeń i stosujemy do jego elementów wszystkie możliwe reguły wnioskowania; uzyskane w ten sposób wyniki dołączamy do zbioru twierdzeń i znowu stosujemy wszystkie możliwe reguły wnioskowania itd. Wiadomo, że jeśli ten proces wykonać nieskończenie wiele razy, to uzyskamy wszystkie możliwe następstwa, czyli właśnie teorię. Dzięki monotoniczności logiki, tak uzyskana teoria nie zawiera żadnych niepotrzebnych wypowiedzi: są to wszystkie konsekwencje i tylko konsekwencje danego zbioru aksjomatów. Bez trudu można wykazać, że zarówno w przypadku klasycznym, jak i intuicjonistycznym, z zadanego (początkowego) zbioru twierdzeń (aksjomatów) wynika tylko jedna teoria, choć, naturalnie, z różnych zbiorów aksjomatów mogą wynikać różne teorie.

W przypadku logik niemonotonicznych opisane postępowanie niestety zawodzi. Rozszerzając jeden i ten sam zbiór aksjomatów pozalogicznych, często zwanych *faktami*, zależnie od tego, którymi domniemaniami się posłużymy i w jakiej to uczynimy kolejności, otrzymujemy różne zbiory konsekwencji. Zbiory te są w ogólności ortogonalne, czyli sprzeczne między sobą. Niesprzeczne wnioskowanie

jest co prawda nadal możliwe, ale tylko w obrębie jednego niesprzecznego rozszerzenia początkowego zbioru faktów.

W praktyce wymaga to utrzymywania wielu wzajemnie sprzecznych rozszerzeń i - przy każdorazowym stosowaniu reguły domniemania - potencjalnie zupełnej rewizji wszystkich niemonotonicznych wnioskowań wychodzących z tego samego zbioru faktów. Prowadzi to do bajońskich kosztów obliczeniowych systemów niemonotonicznych.

Niekiedy, oczywiście za dodatkową cenę niebagatelnych obliczeń, utrzymuje się także maksymalne rozszerzenie niesprzeczne z wszystkimi zastosowanymi domniemaniami. Jeśli jednak chcemy, by nie sprowadzało się ono wyłącznie do początkowego zbioru faktów, należy ograniczyć swobodę stosowania domnieman, a to wymaga - przed każdorazowym zastosowaniem reguły domniemywania - obliczania konsekwencji i ich wpływu na całą zbudowaną dotąd część teorii .

5. Logiki modalne.

W znakomitej *Logice z Port Royale* [6], Arnauld i Nicole piszą: '*...filozofowie zwrócili szczególną uwagę na zdania zwane przez nich* modalnymi. *Są to zdania, w których twierdzenie lub przeczenie uległo modyfikacji wskutek użycia jednego z czterech trybów*: możliwy, faktyczny, niemożliwy, konieczny.' Problematyka przekonywającej argumentacji uwzględniającej stosunek autora do wypowiadanego sądu jest bardzo stara; można ją odnaleźć w dziełach takich klasyków, jak Arystoteles i Diogenes Laertios. W złotym wieku scholastyki wiele inkaustu zużyto na egzegezę różnic między *prawdą niezbędną* i *prawdą uwarunkowaną*. Na przykład, nie podając w wątpliwość prawdziwości zdania 'Bóg istnieje', można długo dyskutować, czy mogło by być inaczej (i wtedy byłaby to prawda uwarunkowana), czy też nie mogło by być inaczej (i wtedy byłaby to prawda niezbędna).

Rachunkową postać logikom modalnym nadał C.I.Lewis tuż przed I Wojną Światową.

W logice klasycznej uznajemy, m.in., dwie formuły

$$P \Rightarrow (Q \Rightarrow P) \text{ i } (\neg P) \Rightarrow (P \Rightarrow Q)$$

z których łatwo można otrzymać

$$(5.1) \quad (P \Rightarrow Q) \vee (Q \Rightarrow P)$$

Formuła (5.1) jest dość niezwykła; zobowiązuje nas ona do uznania, że dla dowolnych dwu zdań, P i Q, albo Q wynika z P, albo Q wynika z P, albo nawet każde z tych dwu zdań wynika z drugiego. Kłopot polega oczywiście na tym, że interpretacja $\Rightarrow$ jako 'operatora wynikania' (lub 'operatora pociągania za sobą') niezbyt tu zgadza się odczuciami językowymi. Lewis wprowadził przeto 'operator ścisłej implikacji', który będziemy oznaczać przez $\Rightarrow$, definiując go przy pomocy dodatkowego pierwotnego *operatora możliwości*, $\underline{P}$, $\underline{P}(A)$ będziemy interpretować jako 'jest możliwe, że A'. Zgodnie z Lewisem mamy

$$(5.2) \quad A \Rightarrow B =_{def} \neg(\underline{P}(A \wedge \neg B))$$

Przy pomocy operatora $\underline{P}$ definiujemy *operator niezbędności* (albo *pewności*)

$$(5.3) \quad \underline{N}(A) =_{def} \neg\underline{P}(A)$$

Wzbogacając (5.2) i (5.3) o kilka aksjomatów i reguł wnioskowania, uzyskuje się różne logiki modalne. Znany jest np. ciąg systemów logicznych S1, S2, S3, S4, S5, odznaczający się tym, że każdy kolejny system zawiera system poprzedni. (Mówimy, że system T zawiera system S jeśli każde twierdzenie, które można wyprowadzić w systemie S można też wyprowadzić w systemie T.) Przykładowo, w systemie S4 występują aksjomaty

$$\underline{N}(P) \Rightarrow P$$

$$\underline{N}(P \Rightarrow Q) \Rightarrow (\underline{N}(P) \Rightarrow \underline{N}(Q))$$

$$\underline{N}(P) \Rightarrow \underline{N}(\underline{N}(P))$$

zaś w systemie S5 - aksjomaty

$$\underline{N}(P) \Rightarrow P$$

$$\underline{N}(P \Rightarrow Q) \Rightarrow (\underline{N}(P) \Rightarrow \underline{N}(Q))$$

$$\underline{P}(P) \Rightarrow \underline{N}(\underline{P}(P))$$

W obydwu tych systemach obowiązują też aksjomaty logiki klasycznej, jej reguły wnioskowania oraz, dodatkowo, reguła

(5.4) jeśli A jest twierdzeniem, to $\underline{N}(A)$ też jest twierdzeniem

Jak widać, w obydwu tych systemach stoimy na gruncie prawd niezbędnych.

Podkładając pod operatory $\underline{P}$ i $\underline{N}$ różne interpretacje słowne (i w niczym nie zmieniając rachunkowych własności odpowiednich systemów) możemy otrzymać różne inne ciekawe interpretacje logiki modalnej. Tytułem przykładu, interpretując $\underline{N}(A)$ jako: 'koniecznym jest aby A', zaś $\underline{P}(A)$ jako: 'przyzwala się aby A', otrzymujemy rachunek deontologiczny.

W bardzo popularnej swego czasu książce [7], J. Hintikka wprowadził dwie pary operatorów modalnych:

$\underline{K}_a(P)$: a wie, że P

$\underline{A}_a(P)$: P jest niesprzeczne z tym, co wie a

$\underline{B}_a(P)$: a wierzy, że P

$\underline{C}_a(P)$: P jest niesprzeczne z tym, w co wierzy a

Operatory te można powiązać parami ze sobą, tak jak (5.3) wiąże operatory $\underline{N}$ i $\underline{P}$

(5.5) $\underline{K}_a(P) = \neg(\underline{A}_a(\neg P))$

(5.6) $\underline{B}_a(P) = \neg(\underline{C}_a(\neg P))$

Niestety, mimo werbalnej atrakcyjności słownej interpretacji formuł modalnych, wnioski do jakich dochodzimy stosując odpowiednie rachunki nie zawsze nam odpowiadają. Tytułem przykładu, w logice Hintikki można wywnioskować, że

'a wie, że P^1 = 'P daje się wyprowadzić z tego co wie a'

Twierdzenie to dość wyraźnie kłóci się z powszechnie akceptowanymi pojęciami wiedzy i dedukcji. Po chwili zastanowienia nad tym, co tu zawodzi, podejrzenie nasze kieruje się w stronę reguły (5.4), która w przypadku logiki Hintikki ma postać

jeśli P jest twierdzeniem, to $K_a(P)$ też jest twierdzeniem

tj. - inaczej powiedziawszy - 'a wie o wszystkich twierdzeniach'.

Ponieważ jednak bez takiego (lub bardzo podobnego) przejścia od twierdzeń materialnych do modalnych nie ma rachunkowej logiki modalnej, wydaje się, że pozostając przy naszych podejrzeniach, musimy zaakceptować ten rachunek 'z dobrodziejstwem inwentarza'.

6. Logiki topologiczne

W logikach modalnych próbuje się wyrazić odcienie stosunku autora do wypowiadanych zdań. Natomiast logiki topologiczne (z ich szczególnie ważnym wariantem - logikami temporalnymi) są rachunkami wypowiedzi, co do których nie zakłada się milcząco, że obowiązują zawsze i wszędzie.

Logika klasyczna nie rozważa zależności wypowiedzi od jej czasu i miejsca. Prawdziwość (lub nie) zdania 'Warszawa leży nad Wisłą' w rzeczy samej nie zależy od tego czy się ją wygłasza w Polsce, czy w Zanzibarze, w 1988, czy 1688 roku. Jednakże o zdaniu 'Właśnie pada deszcz' trudno to powiedzieć!

Wprowadzamy przeto nowy operator B_a, interpretowany tak:

$B_a(P)$ = P jest spełnione w punkcie a

przy czym przez 'punkt a' rozumiemy punkt czasoprzestrzeni.

Standardowymi aksjomatami logiki topologicznej (od *topos* - miejsce) są

$$\underline{R}_a(\neg P) = \neg \underline{R}_a(P)$$

$$\underline{R}_a(P \wedge Q) = \underline{R}_a(P) \wedge \underline{R}_a(Q)$$

. . .

(6.1)

$$(Ax: \underline{R}_a(P(x))) = \underline{R}_a(Ax: P(x))$$

$$(Ax: \underline{R}_a(P)) \Rightarrow P$$

$$\underline{R}_a(\underline{R}_b(P)) = \underline{R}_b(P)$$

(Na szczególną uwagę zasługują dwa ostatnie aksjomaty; warto nad nimi chwilę pomyśleć. Pouczające jest też przeanalizowanie różnicy ról, jakie odgrywają zmienne związane x w obydwu aksjomatach z kwantyfikatorami.)

Do reguł wnioskowania logiki klasycznej, w logice topologicznej dodajemy

jeśli P jest twierdzeniem, to $\underline{R}_a(P)$ też jest twierdzeniem

czyli regułę głoszącą, że twierdzenia niezależne od punktu, w którym są wygłaszane, są twierdzeniami w każdym (dowolnym) punkcie czasoprzestrzeni.

Logikę modalną można powiązać z logiką topologiczną, np. przy pomocy definicji

(6.2) $\underline{N}(P) =_{df} Ax: \underline{R}_a(P)$

(6.3) $\underline{P}(P) =_{df} Ex: \underline{R}_a(P)$

Jeśli uznamy, że nie wszystkie punkty (zwane niekiedy, zwłaszcza przez ludzi skłonnych do przesady, *światami*) są jednakowo łatwo dostępne i wprowadzimy *relację dostępności* AC(w,w1), spełnioną wtedy i tylko wtedy gdy punkt w1 jest dostępny z punktu w, możemy zdefinować 'konieczność spełnienia zdania P w punkcie w':

(6.4) $\underline{R}_w(\underline{N}(P)) =_{df} Aw1: (AC(w,w1) \Rightarrow \underline{R}_{w1}(P))$

czyli zdanie P jest niezbędnie spełnione w punkcie w jeśli P jest spełnione we wszystkich punktach dostępnych z w (z których punkt w jest dostępny).

Z taką logiką możemy już zaatakować paradoksalne wnioskowanie o Cezarze, który jeśli nie opuści domostwa, to wygłosi mowę na Kapitolu. Istotnie *świat, w którym Cezar pozostaje w domu* i wobec tego *nie może być zamordowany* (w każdym razie przez tych spiskowców, którzy oczekują na niego u stóp Kapitolu) i *świat, w którym* - opuściwszy dom! - *Cezar może ujść zamachowcom*, lub nie, i w zależności od tego wygłosi, lub nie, mowę w Senacie, są na tyle różne i mają tak odmienne następstwa, że wnioskowanie nie uwzględniające tego rozwarstwienia można uznać za nieuzasadnione. Formalnie zaś, nie jest prawdą by we wszystkich możliwych światach wszystkie ogniwa łańcucha wnioskowań były spełnione; co więcej, nie ma żadnego świata, w którym wszystkie trzy ogniwa byłyby jednocześnie spełnione. Wydaje się więc, że logika topologiczna, w takiej jak ją podaliśmy postaci, jest tym, czego szukamy.

Sprawa nie jest jednak taka prosta, i to z dwu względów.

Po pierwsze, trochę szkoda uznać za wadliwe wszystkie takie wnioskowania, w których występują ogniwa niekoniecznie spełnione we wszystkich punktach czasoprzestrzeni. Przecież wśród możliwych światów są i takie, które nas nie bardzo obchodzą, albo przynajmniej obchodzą znacznie mniej niż inne. Jeśli więc jakieś ogniwo wnioskowania jest niekoniecznie spełnione w takich mało interesujących światach, to czy rzeczywiście musimy uznać całe wnioskowanie za nieuchronnie wadliwe?

Po drugie, czy nie można by spróbować nadać jakiejś wyrazistszej struktury zbiorom punktów (światów), które nas obchodzą; wydaje się bowiem, że formuły takie jak (6.4), w których występuje pojęcie *wszystkich punktów dostępnych* są niekiedy zbyt prostackie jak na nasze wyrafinowane potrzeby.

Rozwój logik topologicznych poszedł w dwu kierunkach. Jeden z nich uwzględnia różne, czasem bardzo subtelne struktury zbiorów punktów czasoprzestrzeni, drugi - ogranicza się do współrzędnej czasowej; omówieniu tego ostatniego,

logikom temporalnym, poświęcamy następną sekcję, tu więc, na zakończenie, kilka słów o kierunku pierwszym.

D.Lewis [8] wprowadza trójczłonową relację *sąsiedztwa* CL(x,y,z), spełnioną wtedy i tylko wtedy, gdy punkt y znajduje się nie dalej od punktu x niż punkt z. Relację sąsiedztwa można związać z relacją dostępności za pomocą takiej, lub podobnej, formuły

(6.5) $AC(x,y) \wedge AC(x,z) \Rightarrow CL(x,y,z) \vee CL(x,z,y)$

a od samej relacji sąsiedztwa można wymagać pewnych rozsądnych własności, np.

(6.6) $CL(w,x,y) \wedge CL(w,y,z) \Rightarrow CL(w,x,z)$

Ustaliwszy punkt x można teraz określić jego najbliższe sąsiedztwo, zbiór najbliższych sąsiadów spełniających te czy inne warunki, itp.

Dysponując relacją sąsiedztwa, możemy uściślić interpretację operatora $\Rightarrow$, a dokładniej, wprowadzić nowy operator 'lokalnego wynikania', $\succ\Rightarrow$, zdefiniowany tak, że $A \succ\Rightarrow B$ jest spełnione w punkcie x jeśli

(a) istnieje taki punkt y, że zdania A i B są w nim spełnione, nie ma natomiast takiego punktu z, w którym zdanie A byłoby spełnione, a zdanie B - nie, sam zaś punkt z spełniałby relację CL(x,z,y)

lub

(b) zdanie A nie jest spełnione w żadnym punkcie.

Innymi słowy, $A \succ\Rightarrow B$ jest spełnione w punkcie x jeśli w pewnym niepustym sąsiedztwie tego punktu (zawierającego co najmniej punkt y) nie ma punktów, w których spełnione by było zdanie A bez spełnienia zdania B.

Łańcuchowe wnioskowanie

(6.7) $$\frac{A \succ\Rightarrow B,\ B \succ\Rightarrow C}{A \succ\Rightarrow C}$$

uznamy za poprawne w punkcie x wtedy i tylko wtedy, gdy nie ma takiego punktu z, w którym zdania A i B byłyby spełnione, zaś zdanie C - niespełnione, a jednocześnie zachodziłoby CL(x,z,y), gdzie y jest najbliższym sąsiadem punktu x, w którym są spełnione zdania A i C. Inaczej powiedziawszy, wnioskowanie (6.7) uznajemy za zasadne w punkcie x jeśli istnieje niepuste sąsiedztwo tego punktu, w którym zdania A i C są spełnione a jednocześnie sąsiedztwo to nie zawiera punktów, w których przy spełnieniu zdań A i B nie byłoby spełnione zdanie C.

Jak widać, wnioskowania w logice topologicznej Lewisa wymaga by oprócz znajomości wartości logicznej (*prawda*, *fałsz*) zdań w punkcie, dla którego prowadzimy wnioskowanie, znane było wartościowanie zdań w pewnym otoczeniu tego punktu.

7. Logiki temporalne

Ograniczając się do współrzędnej czasowej punktów czasoprzestrzeni (światów), z logiki topologicznej otrzymujemy temporalną. Kwantyfikacja 'we wszystkich punktach' staje się kwantyfikacja 'zawsze', kwantyfikacja 'w niektórych punktach' - kwantyfikacja 'niekiedy'. Czas jednak zwykliśmy dzielić nie tylko na 'zawsze' ('nigdy') i 'niekiedy', lecz także na 'teraz', 'w przyszłości' i 'w przeszłości'. Uwzględniając oba te podziały, otrzymujemy takie np. zwroty: 'zawsze w przyszłości' czy 'nigdy dotąd'. Traktując je jako tryby ('mody') wypowiedzi P, możemy wprowadzić cztery *modalne operatory czasu*

$\underline{F}(P)$ - zdarzyło się, że zdanie P było spełnione

$\underline{F}(P)$ - zdarzy się, że zdanie P będzie spełnione

$\underline{G}(P)$ - zawsze odtąd zdanie P będzie spełnione

$\underline{H}(P)$ - zawsze dotąd zdanie P było spełnione

które można, rzecz prosta, powiązać z operatorami $\underline{N}$ i $\underline{P}$, np.

$$\begin{aligned} &\underline{N}(P) = (P \wedge \underline{G}(P)) \\ &\underline{P}(P) = (P \wedge \underline{F}(P)) \end{aligned} \tag{7.1}$$

albo

$$\begin{aligned} &\underline{N}(P) = (P \wedge \underline{G}(P) \wedge \underline{H}(P)) \\ &\underline{P}(P) = (P \wedge \underline{F}(P) \wedge \underline{E}(P)) \end{aligned} \tag{7.2}$$

(Wzory (7.1) przypisywane są Diodorowi Kronosowi, natomiast wzory (7.2) - samemu Arystotelesowi, rzecz nie jest więc tak bardzo świeża!)

Jeśli chcemy by we wnioskowaniach mogły figurować *chwile czasowe*, musimy przyzwolić by operator R mógł być indeksowany wskaźnikiem chwili:

$\underline{R}_t(P)$ interpretujemy jak: w chwili t zdanie P jest spełnione

Dla skutecznego rachowania potrzebne nam będą ogólne aksjomaty topologiczne w nowej postaci, np.

$$\begin{aligned} &\underline{R}_t(\neg P) = \neg\underline{R}_t(P) \\ &\underline{R}_t(P \wedge Q) = \underline{R}_t(P) \wedge \underline{R}_t(Q) \\ &\ldots \\ &\underline{R}_{t'}(At: P) = At: \underline{R}_{t'}(P) \\ &\underline{R}_{t'}(\underline{R}_t(P)) = \underline{R}_t(P) \end{aligned} \tag{7.3}$$

(w których z łatwością rozpoznajemy naszych dobrych znajomych, por. (6.1)), a także specyficzne aksjomaty temporalne w których występuje pojęcie *uprzywilejowanej chwili* - 'teraz', oznaczanej literką n. Za takie typowo temporalne aksjomaty możemy przyjąć

$$\begin{aligned} &\underline{R}_t(n = t1) = (t = t1) \\ &\underline{R}_t(t1 = t2) = (t1 = t2) \\ &(At: P) \Rightarrow P(t/n) \end{aligned} \tag{7.4}$$

gdzie przez P(t/n) oznaczamy zdanie P, w którym wszystkie wystąpienia zmiennej

t zostały zastąpione przez n, zaś zdanie P, występujące we wszystkich aksjomatach (7.3) i (7.4), nie zawiera zmiennej n we wskaźnikach).

Nie obejdzie się też bez specjalnej reguły wnioskowania, np. takiej

jeśli P jest twierdzeniem to At: $R_t(P)$ też jest twierdzeniem

(której nieco bardziej poetycka wersja brzmi; wszystkie prawdy są wieczne).

Wprowadzając relację *poprzedzania w czasie*, BF(t,t1) = t poprzedza t1, spełniającą

$$\underline{R}_t(BF(n,t1)) = BF(t,t1)$$

$$\underline{R}_t(BF(t1,n)) = BF(t1,t)$$

$$\underline{R}_t(BF(t1,t2)) = BF(t1,t2) \text{ dla } \neg(t1 = n \wedge t2 = n)$$

możemy wyrazić operatory $\underline{F}$, $\underline{P}$, $\underline{G}$, $\underline{H}$ w następujący sposób

$$\underline{F}(P) = Et: (BF(t,n) \wedge \underline{R}_t(P))$$

$$\underline{P}(P) = Et: (BF(n,t) \wedge \underline{R}_t(P))$$

$$\underline{G}(P) = At: (BF(n,t) \Rightarrow \underline{R}_t(P))$$

$$\underline{H}(P) = At: (BF(t,n) \Rightarrow \underline{R}_t(P))$$

skąd wynika m.in. $\underline{G}(P) = \neg\underline{F}(\neg P)$ i $\underline{H}(P) = \neg\underline{P}(\neg P)$.

Podobnie jak w ogólniejszej logice topologicznej sprawą kluczową była struktura relacji dostępności, w logice temporalnej wiele zależy od struktury jaką nadamy relacji BF. Jeśli np. będzie to relacja porządku częściowego, tj. dopuścimy by bywały takie chwile t1 i t2, że ani BF(t1,t2), ani BF(t2,t1), mamy do czynienia z tzw. *czasem rozwidlającym się*, albowiem nie dopuszczamy raczej myśli o różnych możliwych przeszłościach (nawet jeśli przystajemy na wariantową przyszłość).

Aksjomaty

(7.5) AxAyAz: (BF(x,y) ∧ BF(y,z) => BF(x,y) ∨ BF(y,z))

(7.6) AxAyAz: (BF(x,y) ∧ BF(y,z) => BF(x,z))

zapewniają, odpowiednio, 'wsteczną liniowość' i przechodniość czasu (por. (6.5) i (6.6)). Odpowiednie związki między operatorami modalnymi przyjmują postać

(7.7) $\underline{H}(P \vee Q) \wedge (\underline{H}(P \vee \underline{H}(Q)) \wedge (\underline{H}(\underline{H}(P) \vee Q) \Rightarrow \underline{H}(P) \wedge \underline{H}(Q)$

(7.8) $\underline{G}(P) \Rightarrow \underline{G}(\underline{G}(P))$

(7.9) $\underline{H}(P) \Rightarrow \underline{H}(\underline{H}(P))$

Związki (7.7) - (7.9), mimo swej oczywistości, nie dają się wyprowadzić z aksjomatów temporalnych podanych dotychczas; zazwyczaj postuluje się je równolegle z aksjomatami (7.5) i (7.6).

Warto w tym miejscu podkreślić, że z faktu, iż w opisywanej logice występuje rozwidlenie czasu, wcale nie należy wyciągać wniosku, że tylko Wszechświat z rzeczywiście rozwidlonym czasem (jeden z ulubionych pomysłów autorów SF) może być modelem teoryj na takiej logice się zasadzających. (Słów 'model' i 'teoria' używam tu w matematycznym sensie, który wyjaśniałem w swym wykładzie na Jesiennej Szkole PTI w ubiegłym roku.) Fizyczny (lub astronomiczny) czas może sobie płynąć liniowo, natomiast opisy niedeterministycznych systemów w nim występujących może być łatwiej wyrazić w języku takiej właśnie logiki z rozwidlonym czasem.

I przeciwnie, postulując (obustronną: wprzód i wstecz) liniowość czasu, przez przyjęcie aksjomatu przechodniości (7.6) i aksjomatu spójności

(7.10) $At1At2: (BF(t1,t2) \vee t1 = t2 \vee BF(t2,t1)$

uzyskujemy co prawda jednowymiarowe continuum czasu, ale wcale nie opowiadamy się co do natury *rzeczywistego czasu*; równie dobrze może to być np. wykluczenie niedeterminizmu wydarzeń.

Dodanie aksjomatu (7.10) o strukturze czasu wymaga dodania odpowiedniego aksjomatu o operatorach modalnych

(7.11) $\underline{F}(\underline{F}(P)) \Rightarrow \underline{F}(P) \vee P \vee \underline{F}(P)$

a jeśli chcemy zapewnić 'gęstość' continuum czasowego, dodajemy jeszcze aksjomaty

(7.12) $\underline{F}(P) \Rightarrow \underline{F}(\underline{F}(P))$

(7.13) $\underline{E}(P) \Rightarrow \underline{E}(\underline{E}(P))$

Jak widać, postulując różne aksjomaty możemy postulować różne własności biegu czasu logicznego, czyli relacji następowania w systemach, które zamierzamy opisywać. Jeśli obiekty, o których chcemy się wypowiadać (i z raz wypowiedzianych zdań rachunkowo wyprowadzać inne), są automatami, lub dają się dobrze przedstawić jako automaty, dla których - jak wiemy - rzeczą charakterystyczną jest przechodzenie od stanu do stanu, nie jesteśmy zapewne zbytnio zainteresowani ciągłym czasem logicznym. Za Kripkem [9] przyjmuje się wtedy czas dyskretny, składający się nieskończonego ciągu ponumerowanych chwil, odpowiadających kolejnym stanom systemu. Interwały czasu astronomicznego między kolejnymi chwilami są nieistotne, liczą się tylko numery chwil. (W języku polskim słowo chwila ma niestety dwa znaczenia: jedno, w którym chwila trwa dłużej lub krócej, drugie, w którym nie trwa, tj. odpowiada 'punktowi' czasowemu. W tym referacie używam tego słowa wyłącznie w drugim znaczeniu: chwila nie trwa!) Odpowiada temu *dyskretna logika temporalna* opisana przez wielu autorów, np. przez Kroegera [10].

Formalnie, rozważamy nieskończony ciąg $\mathfrak{K}$ odwzorowań

n_i : Z --> *zbiór wartości logicznych*

gdzie Z jest zbiorem zdań elementarnych, zaś *zbiór wartości logicznych* obejmuje np. wartości *prawda* i *fałsz*

Ciąg $\mathfrak{K}$ nazywamy *strukturą temporalną*, a każde odwzorowanie (wartościowanie) n_i - *stanem*. Oznaczmy przez $\mathfrak{K}_i(f)$ wartość logiczną formuły f w stanie n_i, tj.

$\mathfrak{K}_i(z) = n_i(z)$ dla $z \in Z$

$\mathbb{K}_i(\neg P) = \neg\mathbb{K}_i(P)$

$\mathbb{K}_i(P \wedge Q) = \mathbb{K}_i(P) \wedge \mathbb{K}_i(Q)$

itd.

oraz wprowadźmy nowy operator an (= at next), którego intuicyjne znaczenie można wyrazić tak: P an Q = zdanie P jest spełnione w najbliższym stanie, w którym zdanie Q jest spełnione. Dokładniej powiedziawszy

$\mathbb{K}_i(P \text{ an } Q)$ = *prawda* wtedy i tylko wtedy gdy

$A j>i: \neg(\mathbb{K}_j(Q) = \textit{fałsz})$

lub

$\mathbb{K}_k(P)$ = *prawda* dla takiego najmniejszego k,

$k > i$, przy którym $\mathbb{K}_k(Q)$ = *prawda*

O zdaniu P powiemy, że jest *spełnione w strukturze temporalnej* $\mathbb{K}$ jeśli $\mathbb{K}_k(P)$ = *prawda* dla wszystkich k. Zdanie P jest *spełnione* jeśli jest spełnione we wszystkich strukturach temporalnych.

Oznaczając przez T zdanie zawsze prawdziwe, a przez F - zawsze fałszywe, możemy zapisać kilka ciekawych spostrzeżeń:

P an T = zdanie P jest spełnione w następnym stanie

$P \wedge (F \text{ an } \neg P)$ = zdanie P jest spełnione w bieżącym i wszystkich następnych stanach

$P \vee \neg(F \text{ an } P)$ = zdanie P jest spełnione w bieżącym lub jednym z następnych stanów

W dyskretnej logice temporalnej wprowadza się zazwyczaj specjalne operatory, odgrywające rolę swego rodzaju skrótów stenograficznych

$\circ P =_{def} P \text{ an } T$ [następnym razem P]

$\Box P =_{def} P \wedge (F \text{ an } \neg P)$ [zawsze P]

$\Diamond P =_{def} P \vee \neg(F \text{ an } P)$ [niekiedy P]

Korzystając z licznych tożsamości wiążących te operatory, a jest wśród

nich (tj. tożsamości) wiele nader użytecznych, np.

$\neg\circ P = \circ\neg P$ $\quad \Box P \Rightarrow P$ $\quad \Diamond\Diamond P = \Diamond P$ $\quad \circ(P \wedge Q) = \circ P \wedge \circ Q$

$\neg\Box P = \Diamond\neg P$ $\quad P \Rightarrow \Diamond P$ $\quad \Box\circ P = \circ\Box P$ $\quad \Box(P \wedge Q) = \Box P \wedge \Box Q$

$\neg\Diamond P = \Box\neg P$ $\quad \Box\Box P = \Box P$ $\quad \Diamond\circ P = \circ\Diamond P$ $\quad \Diamond(P \wedge Q) = \Diamond P \wedge \Diamond Q$

$\Box P = P \wedge \circ\Box P$ $\quad \Diamond P = P \vee \circ\Diamond P$

$P \underline{\text{an}} Q = \circ(Q \Rightarrow P) \wedge \circ(\neg Q \Rightarrow (P \underline{\text{an}} Q))$

możemy sprawnie prowadzić rachunkowe wnioskowanie w logice temporalnej.

8. Logiki wielowartościowe

Logiki wielowartościowe wprowadził do literatury Jan Łukasiewicz [11] w roku 1920. W jego logice trójwartościowej, wartością logiczną zdania może być *prawda*, *fałsz* i *wartość nieokreślona*. Przyjmując, że prawdzie odpowiada 1, fałszowi - 0, zaś wartości nieokreślonej - I, mamy następujące tabliczki wartości formuł

P	¬P
1	0
I	I
0	1

	P∧Q			P∨Q			P⇒Q			P⇔Q		
Q	1	I	0	1	I	0	1	I	0	1	I	0
1	1	I	0	1	1	1	1	I	0	1	I	0
I	I	I	0	1	I	I	1	1	1	I	1	I
0	0	0	0	1	I	0	1	1	1	0	I	1

Nie wszystkie wnioski, jakie da się wyliczyć przy pomocy tych tabliczek są intuicyjnie do przyjęcia. Dla przykładu, typowym zdaniem o nieokreślonej wartości logicznej jest 'Jutro będzie deszcz'. Jego zaprzeczeniem jest zdanie 'Jutro nie będzie deszczu', na mocy pierwszej tabelki też mające wartość I. Przyjmując pierwsze z tych zdań za P, drugie zaś za Q, z ostatniej rubryki drugiej tabelki odczytujemy, że zdanie P ⇔ Q, czyli 'Jutro będzie deszcz wtedy i tylko wtedy, gdy jutro nie będzie deszczu', ma wartość 1, czyli jest prawdziwe.

Trzecią wartość logiczną interpretowano nie tylko jako 'nieokreśloność', lecz także jako 'paradoksalność', 'bezsensowność' i 'brak danych'. Dość charakterystycznym przykładem zastosowania logiki trójwartościowej może być formalny rachunek poprawności programów, gdzie nierozstrzygalność wielu problemów (by wspomnieć tylko o problemie stopu) podsuwa myśl, że zdanie 'Ten program nie petli się' być może rzeczywiście powinno być wartościowane w trójelementowym zbiorze.

Naturalnie, skoro raz pozbyliśmy się zahamowań i uznaliśmy, że można zdaniom przypisywać wartości z bogatszego niż dwuelementowy zbioru, nic nie stoi na przeszkodzie by pójść dalej. Można np. przyjąć, że wartością logiczną zdania jest prawdopodobieństwo tego, że jest ono prawdziwe.

Logiki wielowartościowe można też łączyć z modalnymi, topologicznymi i temporalnymi. Osiąga sią to w ten sposób, że przy zachowaniu struktury wynikającej np. z modalności, poszerza się zbiór wartości, jakie mogą przyjmować zdania. Wiele bardzo dziwnych logik zaproponowano jako 'lepsze przybliżenie' tych czy innych aspektów rozumowania. Jak dotąd nie zaproponowano żadnej, która nie pozwalałaby wyprowadzić antyintuicyjnych wniosków z rozsądnych przesłanek. Wydaje się, że nie należy o to mieć pretensji do logiki, ani do logików. Po prostu, logika nie jest aproksymacją rozumowania i można tylko współczuć tym, których to martwi tak bardzo, że wciąż obiecują sobie i innym już jutro odwrócić naturalny porządek rzeczy.

SPIS CYTOWANEJ LITERATURY

[1] SEKSTUS EMIPRYK: *Przeciw logikom*. Państwowe Wydawnictwo Naukowe, 1970.

[2] KOŁMOGOROW A.A.: *Zur Deutung der intuitionistischen Logik*. Mathematische Zeitschrift 35 (1931), ss. 58 - 65.

[3] REITER R.: *A logic for default reasoning*. Art. Intel. 13 (1980), pp. 81 - 132.

[4] ŁUKASZEWICZ W.: *General approach to non-monotonic logics.* Proc. 8th Int. Conf. A.I. (1983), pp. 352 - 354.

[5] ŁUKASZEWICZ W.: *Considerations on default logic.* AAAI Workshop on non-monotonic reasoning, New Paltz, 1984, pp. 165 - 193.

[6] ARNAULD A. i NICOLE P.: *Logika.* Państwowe Wydawnictwo Naukowe, 1958.

[7] HINTIKKA J.: *Knowlege and Belief: An Introduction to the Logic of the Two Notions.* Cornell University Press, 1962.

[8] LEWIS D.: *Conterfactuals.* Harvard University Press, 1973.

[9] KRIPKE S.A.: *Semantical analysis of modal logic I.* Z. math Logik u. Grundl. Math. 9 (1963), ss. 67 - 96.

[10] KROEGER F.: *Temporal Logic of Programs.* Tech.Univ. Muenchen, TUM-I8521 , 1985

[11] ŁUKASIEWICZ J.: *O logice trójwartościowej.* Ruch Filozoficzny 5 (1920).

Jesienna Szkola PTI
Mragowo, grudzien 1988

PROTOKOLY KOMUNIKACYJNE

Piotr Dembiński
Brzozowa 6/8 m. 10
00-286 Warszawa

1. Wstęp

Rozwój techniki obliczeniowej można dzielić na rózne okresy oraz wskazywać na rózne wydarzenia i okoliczności, które decydowały (lub współdecydowały) o takich lub innych trendach lub przyjmowanych rozwiązaniach. Z punktu widzenia problematyki, która nas tutaj interesuje, warto chyba przypomnieć i uwypuklić zmiany zachodzące w stosunkach użytkownik-komputer.

Przez większość czasu dzielącego nas od pierwszych lat, w których komputery rozpoczęły odgrywać jakąś rolę w naszym życiu, pozostawały one dużymi, skomplikowanymi i drogimi maszynami zainstalowanymi w izolowanych pomieszczeniach, do których dostęp był zarezerwowany dla tajemniczych (i najogół brodatych) ekspertow. W latach 50-tych użytkownicy (w tym również eksperci) pracowali w systemie wsadowym, w którym zadne oddziaływanie na programy w trakcie ich wykonywania nie było możliwe. Bezpośrednia komunikacja użytkownik-komputer nie istniała.

Lata 60-te to pierwszy przełom w tej dziedzinie spowodowany wprowadzeniem systemów wielodostępnych z podziałem czasu (time-sharing), w których potężne komputery mogły jednocześnie obsługiwać wielu uzytkowników komunikujacych sie z nimi - w sposób interakcyjny - za pomocą podłączonych urządzeń końcowych (interactive terminals). Użytkownicy otrzymali więc własny dostęp do komputera, na którym mogli jednocześnie pracować, każdy wykonując swoje własne zadanie. Centralnie zgromadzone w komputerze dane mogły być wspólnie wykorzystywane. Pojawiają się pierwsze rozwiązania komunikacyjne związane z połączeniem terminal-komputer. Gwałtownie wzrosła liczba zastosowań i użytkowników. Ciągle jednak przeważają eksperci a conajmniej użytkownicy o pewnym profilu zawodowym.

Przełom następuje w latach 70-tych wraz z wprowdzeniem minikomputerów i komputerów osobistych. Użytkownikiem staje się każdy. Biura, urzędy i przedsiębiorstwa zapełniają się małymi urządzeniami liczącymi, których moc obliczeniowa często jest porównywalna z dużymi jednostkami centralnymi z niedalekiej przeszłości. Komputer przeniesiony w miejsce pracy każdego

użytkownika, staje się jednym z jego podstawowych narzędzi działania. Dostęp do niego jest łatwy a nawet zachęcający. Staje się zupełnie oczywiste, że efektywne wykorzystanie tej masy urządzeń, oprogramowania i zgromadzonych w różnych miejscach informacji, wymaga połączenia ich siecią komunikacyjną. W ten sposób, ogromna liczba użytkowników wykorzystywać może potencjał obliczeniowy rozproszony w różnych miejscach tego samego budynku, w różnych filiach czy wydziałach tego samego przedsiębiorstwa lub uniwersytetu, w wielu miastach czy wreszcie krajach. W niepamięć odeszły rojenia o komputerze-gigancie, który pomieści "całą" dostępną informację i obsługiwać będzie miliony użytkowników. Nadeszła (a może raczej nadchodzi) era rozproszonego przetwarzania i przechowywania informacji, era, w której miliony użytkowników mają dostęp do tej informacji i rozproszonych mocy obliczeniowych za pomocą szybkich, niezawodnych i łatwo dostępnych środków komunikacji. Zapewnienie takich środków komunikacyjnych pozwala na elastyczne, dostosowane do potrzeb, rozbudowywanie i zmienianie konfiguracji sieci.

W latach 70-tych powstaje bardzo wiele tzw. sieci lokalnych, tzn. sieci o ograniczonym - np. do jednego budynku czy kompleksu budynków - zasięgu połączeń. Najbardziej może powszechnie znaną obecnie siecią lokalną jest ETHERNET [1] firmy Digital Equipment Corporation (DEC). W tym też czasie zbierane są doświadczenia z użytkowania pierwszej publicznej sieci ARPANET [2] łączącej najpierw wybrane uniwersytety amerykańskie. Powstają inne sieci publicznego użytku, jak IBM-owska SNA (System Network Architecture) czy DECNET firmy DEC. Powstają też węzły komunikacyjne zapewniające możliwość włączania sieci lokalnych do sieci o większym zasięgu wykorzystujących łącza telefoniczne, satelitarne czy radiowe. Badane i mierzone są efekty pracy węzłów komunikacyjnych w różnych konfiguracjach i przy różnych metodach dostępu do nich. Następuje rozwój burzliwy, ale też najczęściej nieskoordynowany. Producenci zazdrośnie strzegą swoich rozwiązań w nadziei, że zdołają uzyskać wyłączność swoich produktów. Problem to nie nowy, ale nie są to już lata 60-te, w których niepodzielnie panował "rynek producentów", tj. taki, w którym użytkownik przyjmował z pokorą skutki walk konkurencyjnych kilku gigantów komputerowych. Wytwarza się wyraźna presja na wprowadzenie rozwiązań ujednoliconych, znormalizowanych. Mówiąc krótko, użytkownik ma dość sytuacji, w której zainstalowany sprzęt wymaga kosztownego oprogramowania dostosowawczego za każdym razem, gdy chce się go włączyć do współpracy z już istniejącym, np. innej firmy. Co gorsze, coraz częściej komputery wyprodukowane przez tę samą firmę nie umieją współpracować ze sobą bez takich specjalnych zabiegów adaptacyjnych. W tej sytuacji dojrzewa koncepcja ustalenia norm światowych. W rezultacie, na przełomie lat 70-tych i 80-tych, powstaje tzw. "Podstawowy Model Referencyjny ISO OSI" (International Standards Organization - Open Systems Interconnection - Basic Reference Model) [3], który powoli i nie bez oporów, staje się punktem odniesienia w projektowaniu sieci i protokołów komunikacyjnych.

W tym miejscu należy zwrócić uwagę na to, że wprowadzenie norm w tak szybko i dynamicznie rozwijającej się dziedzinie wymaga niemałego wysiłku i pokonania wielu trudności. Jeszcze trudniej jest przekonać zainteresowanych, żeby zechcieli istniejących norm

przestrzegać. Instytucje normalizacyjne mogą jedynie proponować swoje rozwiązania, ale nikt nie jest zobowiązany do ich stosowania. Wspomniane tutaj i zrozumiałe interesy producentow, koordynacja pomiędzy instytucjami państwowymi różnych krajów, a także dostosowanie sie do już istniejących standardów, np. w telekomunikacji, to tylko niektóre przeszkody, które należy pokonać. Trzeba tez wspomnieć o pewnej konkurencyjności międzynarodowych organizacji normalizacyjnych. Z jednej bowiem strony ISO z drugiej CCITT (Comite Consultatif Internationale Telegraphique et Telephonique) pretendują, każdy nie bez racji, do kreowania norm w tej dziedzinie. Mimo dobrej woli obu organizacji, powoduje to nieuniknione konflikty, dublowanie wysiłków - w sumie mnożenie trudności. Mimo tego, a może właśnie dlatego, wydaje się, że osiągnięto w tym przypadku sukces. Znormalizowana architektura sieci staje się (prawie) faktem.

2. Warstwowa architektura sieci

W projektowaniu sieci od poczatku stosowano zasadę ustalania hierarchii zadań, które prowadziły do podziału architectury każdego węzła sieci na tzw. warstwy (layers). Warstwa niżej usytuowana w hierarhii oferuje usługi warstwie znajdujacej sie bezpośrednio powyżej niej. Ta ostatnia korzysta z oferowanych usług dla zrealizowania swojego zadania.

Spróbujmy scharakteryzowac taką warstwową organizację przez analogię z "przykładem z życia wziętym" (chociaż może z życia nieco wydumanego):

> Dowódcy jednostek (warstwa 5) wymieniaja pomiędzy sobą b. ważne i b. tajne depesze. Każdy z nich korzysta z usług Biura Szyfrow (BS: warstwa 4), w którym to Biurze tekst dostarczanej depeszy zostaje zaszyfrowany. Dobór szyfru zależy nie tylko od BS danej jednostki, ale także od zgody adresata (adresatów) depeszy i, być może, innych wysoce tajnych rozporządzeń. Cały ten kłopot z ustaleniem aktualnie obowiązujacego szyfru BS rozwiązuje w nawiązaniu kontaktu z odpowiedniemi BS-ami innych jednostek, nie zaprzątając głowy dowódcom, którzy (jak to dowódcy) przygotowują inne b. ważne i b. tajne depesze. Dla tych swoich własnych celów i dla ostatecznego przekazania treści depeszy, BS dołącza swe własne informacje i przekazuje tak uzupełnioną wiadomość do Biura Transportu (BT: warstwa 3). Biuro Transportu ustala aktualne adresy odbiorców (dowódca przecież nie pamięta wszystkich adresów - zna tylko nazwy jednostek - a szyfranci znać ich nie powinni) i dorzuca je do otrzymanego z BS tekstu. BT ma też do swojej dyspozycji ścisłe instrukcje, które ustalają jakie węzły komunikacyjne należy omijać, jako podejrzane o wrogi sabotaż. Poza tym, zdecydowano (ku rozpaczy pracowników Biura), że BT ma wprowadzić własny system ochrony przed błędami takimi, jak: nieprawidłowa kolejność przesyłania depesz, gubienie ich po drodze, lub dostarczanie dwa i wiecej razy tej samej wiadomości. Pomimo bowiem ciągłych zapewnień Biura Łączności (BŁ: warstwa 2) o 100% niezawodności jego usług, z których korzysta BT,

zdarzają się tego typu błędy bardzo denerwujące dowodców. Tak więc, BT opatruje tekst otrzymany z BS całą masą dodatkowych informacji kontrolnych i dopiero taki pakiet przekazuje do BL, w którego gestii jest ustalenie dokładnej trasy przekazu i zabezpieczenie przed w/w i innymi błędami transmisji. Znowu więc, informacja przekazana przez BT zaopatrzona zostaje w dodatkowe elementy kontrolne BL i dopiero wtedy otrzymuje ją Departament Telekomunikacji (DT : warstwa 1), który zgodnie z instrukcjami dokonuje transmisji wybranym przez siebie kanałem komunikacyjnym.

Przy odbiorze, DT otrzymane dane przekazuje do BL, które sprawdza czy jest to ta wiadomość, np. co do kolejności, na którą oczekuje i czy można ją uznać za wolną od błędów. Jeżeli tak, to BL usuwa nagłówki kontrolne dotyczące komunikacji pomiędzy BL-ami i następnie pozostałą część komunikatu przekazuje do BT wysyłając jednocześnie potwierdzenie odbioru do BL jednostki, która depeszę nadała. BT z kolei analizuje i usuwa nagłówki kontrolne swoich kolegów z nadającego BT. Jeżeli stwierdzi błąd (np., że nie do niego adresowana jest wiadomość) to wysyła (poprzez swoje BL) odpowiedź odmowy przyjęcia depeszy (np. z dopiskiem "adresat nieznany"). Jeżeli błędów nie ma, to uznaje, że tekst należy przekazać do BS do odszyfrowania. Wreszcie odszyfrowana depesza trafia na biurko dowódcy.

W Centralnym Biurze Rozliczeń rozumiano dobrze konieczność wymiany b. ważnych i b. tajnych depesz, ale wielokrotnie zwracano uwagę na olbrzymie koszty komunikacji. Problem był trudny i delikatny ze względu na drażliwość dowodców, którzy mogliby źle przyjąć administracyjnie zarządzone ograniczenia. Problem doczekał się jednak zadowalającego rozwiązania dzięki pomysłowi skromnego szeregowca St.B. z jednego z Biur Łączności. Zauważył on mianowicie, że dowódcy mają tendencje do wysyłania depesz bardzo często (od tego są przecież dowódcami), ale bardzo krótkich (to cecha dobrych dowódców). Ponieważ najwięcej kosztuje samo połączenie, szeregowiec St.B. wpadł na pomysł by nie wysyłać każdej depeszy z osobna ale czekać jakiś czas aż uzbierze się ich więcej, łączyć je w zestawy o tym samym adresie i dopiero takie całe zestawy przesyłać. Otrzymane zestawy depesz będą rozdzielane ponownie w BL adresata. Ten prosty sposób poprawił znacznie wyniki eonomiczne Centrali a dowódcy, a także ich BS i BT, nie zauważyli nawet dokonanej modyfikacji.

Ten nieco swobodny przykład miał na celu wskazanie na główne cechy i elementy charakteryzujące warstwowy model sieci. Omawiamy je poniżej.

(A) Warstwy i ich zadania

Architektura każdej jednostki w sieci składa się z takiej samej liczby warstw i każda warstwa ma określone zadania do wykonania. Zadaniem, np. Biura Szyfrów (BS) jest szyfrowanie depesz dowódcy własnej jednostki i odszyfrowywanie depesz nadchodzących z innych jednostek. Jednakże, nie ogranicza się ono

do tego. BS ma także za zadanie uzgadnianie z BS każdego adresata i być może wraz z każdą depeszą, odpowiedniego szyfru oraz, na przykład, kontrolę jakości szyfrów, która może spowodować odrzucenie przez BS otrzymanej depeszy zawierającej błędy szyfrowania.

(B) Protokół warstwy

Zadanie każdej warstwy wymaga jej komunikacji i dialogu z odpowiadającą jej warstwą innej jednostki. Ten dialog dotyczy spraw interesujących tę właśnie warstwę i jest niezależny od komunikatu, który dialog ten zainicjował. Taki komunikat inicjujący dialog między warstwami tego samego poziomu, pochodzi (oprócz warstwy najwyższej) od warstwy bezpośrednio powyżej w hierarchii warstw, gdy jednostka pełni role nadajnika, lub od warstwy bezpośrednio poniżej (oprócz warstwy najniższej), gdy jednostka pełni rolę odbiornika. Ten właśnie bezpośredni dialog między tymi samymi warstwami jednostek składowych sieci stanowi protokół danej warstwy. Odbywa sie on poprzez wymianę wiadomości oznaczanych X.PDU (Protocol Data Unit), gdzie X jest nazwą warstwy.

PDU każdej warstwy X zawiera dane (ewentualnie przekodowane) warstwy bezpośrednio z nią sąsiadującej oraz nagłówek (header), w którym zawarte są informacje właściwe dla protokołu warstwy X. Tak więc, opis protokołu pomiędzy, np. Biurami Transportu w naszym przykładzie, pomijałby całkowicie rolę Biur Łączności i Departamentów Telekomunikacji a opis (części dotyczącej nadawania) protokołu wymiany depesz pomiędzy dowódcami jednostek (warstwa najwyższa) mógłby brzmieć następująco:

1. wywołaj nazwę jednostki, do której nadana będzie depesza;

2. poczekaj X sekund na zgłoszenie się wywołanej jednostki i jeżeli nie otrzymasz zgłoszenia, to przejdź do 1 lub 5;

3. nadaj treść depeszy;

4. zakończ nadawanie i przejdź do 6;

5. anuluj nadawanie;

6. poczekaj y sekund na potwierdzenie przyjęcia depeszy lub anulowania nadawania i, jeżeli nie nadejdzie ono w tym czasie, to przejdź do 3 lub 5 lub 7;

7. anuluj lub zakończ nadawanie nie czekając na potwierdzenie;

itd.

Opis podany w powyższych terminach stwarza wrażenie bezpośredniej komunikacji pomiędzy dowódcami jednostek. Dowódca wie, że ma przekazać b. ważną i b. tajną wiadomość, lecz nie musiał nawet słyszeć o tym, że szyfrowanie służy utajnianiu przekazu, nie mówiąc już o rodzajach i zasadach działania technicznych środków komunikacji w posiadaniu Departamentu Telekomunikacji, czy sposobach używanych przez Biura Łączności i

Transportu by zapewnić bezbłędny przekaz, uporać się z problemem krzyżujacych się wiadomości (colisions), albo z tym, że jedne jednostki działają o wiele sprawniej i szybciej niż inne i te ostatnie zawalone są nadmiarem danych, których nie mogą przetworzyć (backpressure problem). Co by to zresztą było, gdyby dowódca dowiedział się, że drogie urzadzenia telekomunikacyjne psują się często i wtedy jego Departament Telekomunikacji poprostu posługuje się zwykłym posłańcem?!

Tak więc, wymiana wiadomości typu PDU jest pewną "horyzontalną abstrakcją" rzeczywistości, mającą podkreślić fakt, że opis protokołu komunikacji dla danej warstwy powinien być niezależny od jakiejkolwiek jego implementacji.

(C) Usługi warstwy i sprzęgi międzywarstwowe

W rzeczywistości, i tak było w naszym przykładzie, komunikacja między tymi samymi warstwami różnych składowych sieci odbywa się (z wyjątkiem warstwy najniższej) korzystając z usług (services), oferowanych przez warstwy znajdujące się bezpośrednio pod niemi. Usługa formułuje zasady wymiany informacji i treść tej informacji, pomiędzy sasiadujacemi warstwami w jednostce składowej sieci. Inaczej mówiąc, usługi warstwy X charakteryzują globalnie działanie warstw 1,...,X z punktu widzenia warstwy X+1, która z tych usług korzysta. Korzystanie z usług warstwy X przez warstwę X+1 polega na przekazywaniu i odbieraniu przez tę ostatnią wiadomości, oznaczanych przez X.SDU (Service Data Unit), które warstwa X definiuje jako te, które odpowiednio ona sama może otrzymywać i wysyłać.

Jaki jest związek pomiędzy X.PDU i X.SDU ? Oczywiście, X.PDU musi zostać, w jakiejś formie, przekazane niżej, ale X.SDU na ogół zawiera dodatkowe informacje i parametry. Te parametry i informacje mogą, np. dotyczyć jakości usługi QOS (Quality Of Service). Żeby trzymać się naszego przykładu, wyobraźmy sobie, że Biuro Transportu (BT) oferuje dwie możliwości: albo całkowite zabezpieczenie przesyłanych informacji przed dostępem do nich innych jednostek niż te, do których informacja jest skierowana, albo brak takiego zabezpieczenia. Użytkownik, czyli Biuro Szyfrów (BS), może wybrać jedną z nich przez nadanie odpowiedniej wartości pewnwmu parametrowi BS.SDU. Poza tym, BS.SDU może mieć inne parametry oraz zawiera tekst zakodowanej depeszy z dołączonym do niej nagłówkiem BS (czyli BS.PDU). Inny przyklad: Biuro Łaczności (BL), po pewnym okresie działania, zebrało dane statystyczne, na podstawie których ustalono zależność pomiędzy pomiedzy długością przesyłanych informacji a prawdopodobieństwem błędu transmisji. Dla utrzymania pewnej założonej efektywności swoich działań, BL ustaliło nieprzekraczalna długość komunikatu jednorazowo otrzymywanego z BT. W tej sytuacji, wydzielona agenda BT zobowiązana została do dzielenia komunikatow o długosci przekraczajacej ten limit na części tworzące pojedyńcze BT.SDU oraz do składania ich w jedna całość przy odbiorze.

Jak wynka z powyższych przykładów, pomiędzy sasiadującymi warstwami danej jednostki w sieci musi istnieć pewna - zależna od konkretnych rozwiązań i niekiedy skomplikowana - procedura wiążąca te dwie warstwy. Ta procedura nazywana jest sprzęgiem

(interface). Sprzęg umożliwia i zapewnia prawidłowe wykorzystanie usług warstwy X dla realizacji protokołu warstwy X+1.

(D) Niezależnosc warstw

W naszym przykładzie, dodaliśmy na końcu opowiastkę o zmianach w pracy Biura Łączności zaproponowanych przez szeregowca St.B. Ilustruje ona jedną z ważnych cech architektury warstwowej: możliwość dokonywania wymiany lub modyfikacji warstwy bez konieczności dostosowywania czegokolwiek w innych warstwach. Na ogół jednak, takie modyfikacje pociągają konieczność zmian w sprzęgu międzywarstwowym. Inna sprawa, że rozróżnienie pomiędzy protokołem a sprzęgiem, ważne z metodologicznego punktu widzenia, z wielu powodów nie jest tak jednoznaczne w praktyce realizacyjnej. Stąd, niejednokrotnie niezależność warstwowa bywa nieco pozorna.

3. Model ISO

Model warstwowy ISO [3,4,5] posiada siedem warstw:

- warstwę fizyczną (physical level),
- warstwę łącza danych (data link layer),
- warstwę sieciową (network layer),
- warstwę transportową (transport layer),
- warstwę sesji (session layer),
- warstwę prezentacji (presentation layer),
- warstwę zastosowań (application layer).

Dlaczego akurat siedem ? Liczba ta jest wynikiem uzgodnień międzynarodowych ekspertów opierających się na doświadczeniach w projektowaniu istniejących sieci, reprezentujących interesy poszczególnych producentów, itd. Najogólniej rzecz biorąc, przyjęli oni założenie ([4,5]), że każda warstwa powinna odpowiadać dobrze określonemu i w miarę niezależnemu zadaniu, którego wybór uwzględniać będzie istniejące międzynarodowe normy. Liczba warstw powinna być wystarczająco duża by ważne funkcje znalazły się w osobnych warstwach, ale jednocześnie na tyle niewielka by redukowała przepływ informacji przez sprzęgi (a więc, oczywiście 7!).

Nim przejdziemy do krótkiego omówienia poszczególnych warstw, warto w tym miejscu powiedzieć co jest przedmiotem normalizacji.

Norma ISO 7498 [3] dotyczy modelu OSI. Zawiera ona w miarę dokładną definicję pojęcia warstwy i ramową koncepcję architektury warstwowej sieci wraz z ustaleniem odpowiedniej terminologii. Opisuje następnie kolejno siedem w/w warstw, podając dla każdej z nich jej zadanie i funkcje, które musi wykonywać. Normalizuje konieczne usługi, które dana warstwa ma zapewnić warstwie położonej bezpośrednio nad nią w hierarchii. Krótko mówiąc, norma ISO OSI wprowadza ujednolicony podział architektury na warstwy i normalizuje nazewnictwo ich elementów składowych.

Norma ISO OSI nie zawiera opisu protokołów warstw. Każdy taki protokół posiada osobny, już istniejący lub przygotowywany, dokument normalizacyjny, który posługuje się pojęciami i terminami ustalonymi dla danej warstwy w bazowym modelu referencyjnym. Ważnym jest również podkreślenie, że sprzęgi pomiędzy sąsiadującymi warstwami nie są w ogóle normalizowane. Zasadą przyjętą przez ISO jest, że norma powinna definiować jedynie "logikę" systemu pozostawiając zupełną dowolność realizacji tej "logiki". W myśl tej zasady, normy nie powinny sugerować żadnego konkretnego rozwiązania. Zagadnienie czy jest to w ogóle możliwe pozostaje dyskusyjne, ale częściowo tłumaczy dlaczego właśnie protokoły i usługi są normalizowane a sprzęgi nie (nawiasem mówiąc normalizacja usług napotykała i ciągle napotyka na pewne opory). Jak usiłowaliśmy wskazać w przykładzie z poprzedniego rozdziału, protokół jest pewną "abstrakcją" dialogu na ustalonym poziomie, który w rzeczywistości nie odbywa się bezpośrednio. Tak więc, w pewnym sensie, protokół nie odnosi się w sposób prosty do konkretnych rozwiązań dialog ten zapewniających.

Trzeba na koniec powiedzieć, że protokoły trzech pierwszych warstw (warstwy fizycznej, łącza danych i sieciowej) dotyczą sytuacji nieco bardziej złożonej niż komunikacja bezpośrednia pomiędzy odpowiednimi warstwami porozumiewających się komputerów. W rzeczywistości protokoły te opisują połączenia warstwowe: komputer-węzeł komunikacyjny sieci, węzeł-węzeł, oraz węzeł-komputer. Rodzaj złożenia ich wszystkich tworzy właściwy protokół miedzy komputerami. Tak wiec, trzy pierwsze warstwy występują zarówno w komputerach jak i węzłach sieci, podczas gdy pozostałe tylko w komputerach. Dla zrozumiałego uproszczenia nie rozwijamy tego zagadnienia.

(1) Warstwa fizyczna

Warstwa fizyczna zajmuje się przekazywaniem w kanale komunikacyjnym sygnałów reprezentujących bity. Określa ona sposób łączenia i rozłączania fizycznych przewodników sygnałów oraz rodzaj (jednokierunkowy, dwukierunkowy) ich przesyłania podczas połączenia. Stosuje pewne zabezpieczenia (kody korekcyjne, kody detekcyjne) przeciwko błędom powstałym w wyniku zakłóceń (tzw. szumów) fizycznej trasmisji sygnałów. Bardzo ważne jest dostosowanie funkcji tej warstwy do istniejących norm stosowanych w telekomunikacji. Np. norma CCITT ("zalecenie" w nomenklaturze tej organizacji) X.21, która weszła w życie w 1976r., określa sposób w jaki komputer abonenta zestawia i przerywa połączenia z urządzeniem komunikacyjnym przedsiębiorstwa telekomunikacyjnego.

(2) Warstwa łącza danych

Warstwa ta ma zapewnić bezbłędna transmisję, tzn. odpowiedzialna jest za rozwiązanie problemów spowodowanych gubieniem, powielaniem i uszkadzaniem wiadomości podczas przesyłania. Usługa jaką oferuje warstwa łącza danych warstwie sieciowej polega właśnie na gwarantowaniu - w mniejszym lub większym stopniu - niezawodności transmisji. W tym sensie, usługa ta może być oferowana z parametrem jakości (i zależnej od niego ceny) do wyboru użytkownika. Warstwie łącza danych przypada na ogół także funkcja regulująca szybkość przepływu danych w wypadku

nadajnika pracującego w zestawie z o wiele wolniejszym odbiornikiem (te funkcje spełniają zresztą także inne warstwy).

Te, z grubsza powyżej opisane, cele warstwy łącza danych realizuje ona umieszczając dane w "ramkach" (frames) kontrolnych. Tak "opakowane" dane wysyłane są często wielokrotnie dla zapewnienia poprawnosci odbioru. Niemal wszystkie protokoły stosowane w tej warstwie oparte są na podstawowym schemacie protokołu z przesuwnym oknem (sliding window protocol). W najprostszym przypadku (tzn. gdy okno nadajnika i odbiornika redukuje sie do jednej "klatki"), protokół ten polega na : pobraniu wiadomosci, utworzeniu zeń ramki danych (data frame) zawierajacej jej numer identyfikujacy zapamiętany jednocześnie w klatce okna i wysłaniu ramki. Następna wiadomość nie zostanie pobrana do wysyłki dopóki nadajnik nie otrzyma ramki potwierdzajacej (acknowledgment frame). Ewentualnie, po pewnym czasie oczekiwania na potwierdzenie, duplikat tej samej ramki zostaje ponownie wysłany. Protokół, w którym okna mają wiecej klatek (liczba klatek nie jest koniecznie taka sama po stronie nadajnika i odbiornika) umożliwia przesyłanie i potwierdzanie większej liczby ramek naraz, zostawia także większa swobodę co do kolejności ich nadawania i odbierania.

(3) Warstwa sieciowa

Jak powiedzieliśmy wcześniej, opis protokołu stwarza złudzenie jakoby komunikacja odbywała się bezpośrednio pomiędzy odpowiednimi warstwami współpracujących komputerów. W rzeczywistości każdy komputer utrzymuje bezpośrednia łaczność z tzw. węzłem (pod)sieci komunikacyjnej, który z kolei komunikuje sie z innymi węzłami. W ten sposób, zapewniona jest łaczność pomiędzy komputerami. Pierwsze trzy warstwy występują zarówno w komputerach, jak i węzłach. Warstwie sieciowej przypada zadanie zapewnienia połączenia komputer-węzeł. Międzynarodowy standard X.25, zaproponowany przez CCITT, opisuje protokół takiej łaczności. Jest on powszechnie stosowany, poza krajowymi sieciami USA (w komunikacji międzynarodowej USA stosuje X.25). Innym zadaniem warstwy sieciowej jest wybór drogi pomiędzy wezłami oraz zabezpieczenie sieci przed przeciążeniami i blokadami, w taki sposób, by wyższe warstwy nie musiały się kłopotać tymi prblemami. Usługa warstwy sieciowej,- oferowana warstwie transportowej komputera obliczeniowego - może zapewniać lub nie zapewniać połączenia wolnego od błędów typu zagubienia, dublowania pakietów, albo ich nadchodzenia w niespodziewanej kolejności (usługa połączenia wirtualnego lub datagramowego). Norma X.25 umożliwia łaczność zarówno w pierwszym (do pewnego stopnia), jak i drugim przypadku. Warstwa sieciowa w sieci DECNET oferuje jedynie usługę datagramowa, tzn. nie zabezpieczającą przed w/w błędami.

(4) Warstwa transportowa

Warstwa transportowa przyjmuje dane od warstwy sesji i, w razie potrzeby, dzieli je na mniejsze jednostki. Przy odbiorze, z warstwy sieciowej, części danych składa znów w poprawną całość. Ważną funkcją tej warstwy jest, niewidoczne dla warstwy sesji, tzw. zwielokrotnianie (multiplexing), które polega na tym, że warstwa transportowa moze żądać kilku połączeń sieciowych dla

wykonania jednego połączenia transportowego (w ten sposób zwiększa przepustowość) lub, odwrotnie, kilka połączeń transportowych realizuje jednym połączeniem sieciowym (oszczędzając na połączeniach). Warstwa transportowa może przejmować także obsługę błędów, o których mówiliśmy uprzednio, w zależności od usługi oferowanej przez warstwę sieciową, lub też na skutek niewiary w jakość tych usług. Jeszcze innym zadaniem warstwy transportowej jest zarządzanie połączeniami, czyli ich zestawianie i kasowanie. Połączenia te inicjowane i negocjowane są w warstwie sesji, ale adresy użytkownika wymagają przekształcenia na adresy transportowe ujęte w pewien, jednoznaczny system adresowania. Dopiero za pomocą tych adresów, niższe warstwy ustalą konkretną drogę przesyłania w sieci.

Zauważmy, że dialog wymagany do realizacji tych wszystkich zadań odbywa się bezpośrednio pomiędzy warstwami transportowymi komputerów, tzn. nie korzystając z pośrednictwa, jak to było w niższych warstwach, odpowiedniej warstwy węzłów. Inaczej mówiąc, protokół warstwy transportowej opisany jest w terminach wymiany PDU tej warstwy pomiędzy komputerami.

(5) Warstwa sesji

Jak powiedzieliśmy już powyżej, warstwa sesji inicjuje i negocjuje połączenia, tzn. żąda połączenia od warstwy transportowej, oczekuje zgody adresata i ustala z nim pewne parametry sesji, czyli rodzaju wymiany informacji w czasie trwania połączenia. Warstwa sesji sprawuje pewien nadzór prowadzonego dialogu (np. dba o to, by przesyłanie pewnych grup danych dokonywało się w całości) oraz dbać musi o poprawne zakończenie sesji. W wielu istniejących sieciach zadania warstwy sesji przejmowane są przez warstwę transportową, w nowszych realizacjach, na ogół, dba się o to, by te dwie warstwy traktować osobno, zgodnie z normą ISO.

(6) Warstwa prezentacji

Ostateczną formą wiadomości wysyłanej przez komputer jest ciąg sygnałów binarnych (bitów). Odbiorca, ażeby móc zrozumieć przesłaną wiadomość, musi umieć odtworzyć jej strukturę i treść z otrzymanego strumienia bitów i to niezależnie od komputera, którym się posługuje lub terminala, którego używa. Jednym słowem, nadawca z odbiorcą muszą dojść do porozumienia "w jakim języku mowią ze sobą" i jakiego systemu kodującego w tej rozmowie używają. Negocjowanie tego porozumienia to, z grubsza rzecz biorąc, zadanie warstwy prezentacji.

(7) Warstwa zastosowań

W zasadzie, w tej warstwie nie ma ograniczeń na zadania jakie mogą się okazać użyteczne w zastosowaniach. Jednakże, wcześnie zauważono, że istnieją funkcje ogólne lub specyficzne, z których większość zastosowań korzysta. Skupiono się więc na definiowaniu i normalizowaniu tej grupy zadań. W niektórych realizacjach tworzą one coś w rodzaju podwarstwy jak, na przykład, dość powszechnie stosowana podwarstwa CASE (Common Application Service Elements), do której elementów odwołują się poszczególne moduły zastosowań.

Podamy poniżej kilka takich "jednostek usług zastosowaniowych" (Application Service Elements), dla których już istnieją normy międzynarodowe, lub normy te są w przygotowywaniu [6].

- ACSE (Association Control Service Element), który zawiera funkcje bazowe dla dokonania połączenia pomiędzy dwoma procesami niezależnie od ich lokalizacji.

- CCR (Commitment, Concurrency and Recovery) o funkcjach umożliwiających utrzymanie integralności rozproszonej bazy danych.

- RTS (Reliable Transfer Server), którego funkcje zapewniają ciągłość komunikacji we współpracy z warstwą sesji.

- ROS (Remote Operation Service) o funkcjach koniecznych na to by moduły zastosowań mogły komunikowac się z ich użytkownikami.

- MHS (Message Handling System), którego funkcje realizują zadania poczty elektronicznej poza samym przesyłaniem.

- FTAM (File Transfer, Access and Management) o funkcjach koniecznych dla przesyłania plików i przekształcania ich na odległość.

- VT (Virtual Terminal), którego funkcje pozwalają na współpracę dowolnego terminala z dowolnym komputerem sieci.

- TP (Transaction Processing), którego funkcje "porządkują" proces wykonywania operacji na bazach danych rozproszonych i powielanych tak, aby w wyniku tych operacji baza pozostawała w stanie spójnym.

- ODA (Office Document Architecture) o funkcjach pozwalających na kreowanie, formatowanie, zmianę i przesyłanie dokumentów na odległość.

Podsumowując, można zapytać jaki jest stan zaawansowania prac normalizacyjnych w protokółach sieciowych i ich wdrożeń. Cały proces normalizacyjny rozpoczął się na dobre w r. 1978 i można przewidywać, że zakończy się około r. 1998 [6]. Znajdujemy się więc w połowie drogi. Protokoły warstw 1-4 są już dobrze ustabilizowane w istniejących sieciach. Protokoły wyższych warstw są wprowadzane stopniowo. Należy podkreślić, że wdrożenie protokołu i jego stabilizacja trwa od pięciu do dziesięciu lat a nowe protokoły pojawiają się sukcesywnie.

Zaznaczmy na koniec tego rozdziału, że normy protokołów starają się zaspokoić jak największą liczbę wymagań i oferują pełną gamę możliwych usług, ich parametrów i atrybutów w zakresie postawionego zadania. Formułują one jednoczesnie pewien zbiór bazowy tych usług, konieczny w każdej realizacji, pozostałe traktując jako fakultatywne, tzn. takie, z których można zrezygnować w kokretnej implementacji w zależności od przewidywanych potrzeb i ograniczeń. Każdy producent, czy też grupa producentów, definiuje własne tzw. profile (profiles) standardów, czyli normy, w których dokładnie wymienia się zrealizowane elementy opcjonalne. Przykładami takich grup

producentów przygotowujących profile obowiązujące dla danej grupy, są: amerykańska korporacja COS (Corporation for Open Systems), organizmy MAP i TOP (także amerykańskie), z których pierwszy zajmuje się zastosowaniami przemysłowymi a drugi naukowymi i biurotycznymi, japońska organizacja POSI (Japonese Promoting Conference for OSI), czy wreszcie grupa SPAG (Standards Promotion and Application Group) zrzeszająca 12 największych konstruktorów europejskich (ICL, Siemens, Thompson, Philips, Bull, Olivetti, itd.). Prace tych grup mają też ogromny wpływ na ustalanie samych norm.

4. Formalne techniki specyfikacji protokołów

Normy międzynarodowe są formułowane z reguły w języku angielskim (amerykańskim) oraz używają pomocniczych środków takich, jak tablice, diagramy itp. Taki sposób opisu skomplikowanych systemów jakimi są protokoły i ich usługi, prowadzi do wielu niejasności i zostawia miejsce na różne interpretacje. Nie pozwala także na stworzenie i zastosowanie wspomagającego oprogramowania mogącego usprawnić projektowanie i powstawanie opisu protokołu, jego weryfikację i wreszcie implementację. Z tych też powodów, we wczesnych latach 80-tych, rozpoczęto w ISO prace nad opracowaniem formalnych technik (języków), w których opisy protokołów nie miałyby w/w wad opisów w językach naturalnych. W wyniku tych prac, powstały dwa sformalizowane języki specyfikacji (FDTs - Formal Description Techniques): Estelle [7] i Lotos [8]. Oba uzyskały już (1988) status standardu międzynarodowego IS (International Standard). Wcześniej, normą (zaleceniem) CCITT stał się język SDL (Specification and Description Language) [9]. Estelle i SDL są oparte o model kooperujących automatów, a Lotos bazuje na algebraicznym podejściu do opisu współpracy procesów zaproponowanym w [10]. Charakteryzując bardzo pobieżnie te trzy techniki opisu, można powiedzieć, że mocną stroną SDL jest jego sformalizowana reprezentacja graficzna, Estelle daje najwięcej możliwości szybkiego uzyskania prototypowych implementacji, a Lotos wyróżnia sie eleganckimi mechanizmami przekształcania opisów na różne poziomy abstrakcji.

W tych trzech językach powstały dotychczas specyfikacje wielu protokołów (np. warstwy sieciowej, transportowej, sesji, a także protokoły warstwy zastosowań FTAM, VT i inne). Specyfikacje te nie zastępują oficjalnych norm sformułowanych w języku naturalnym, ale ewentualnie je uzupełniają (co też nie przebiega bez oporów).

Bardzo ważnym elementem, o którym wspomnieliśmy wcześniej i który zapewne zadecyduje o powodzeniu w rozpowszechnianiu tych formalnych metod opisu - nie tylko być może dla protokołów - będzie skuteczność i łatwość w użyciu rozlicznych narzędzi programistycznych (edytorów, kompilatorów, symulatorów) ułatwiających projektowanie, weryfikację i implementację powstających w nich opisów systemów. Takie narzędzia powstały już w wielu ośrodkach i są w stadium eksperymentalnych zastosowań [10].

Protokoły warstwy zastosowań mają najczęściej do przekazania złożone informacje (często o bardzo szczególnym charakterze), których zdefiniowanie wymaga pewnej strukturalizacji i standaryzacji. Te złożone informacje przekazywne są do warstwy prezentacji, w której kodowane sa w ciagi bitów. Od pewnego czasu narastało przekonanie, że normy protokołów w warstwie zastosowań muszą posługiwać się jakimś spójnym i jednolitym systemem definiowania (typów) danych i, jednocześnie, odwoływać się do pewnego "uniwersalnego" systemu ich kodowania. Bez takich wspólnych ram, trudno sobie wyobrazić spójnosc opisów w różnych zastosowaniach a także zgodność realizacji opartych o te opisy.

Wynikiem tego przekonania było powstanie dwóch, ściśle ze sobą związanych, standardów ISO/CCITT: ASN.1 (Abstract Syntax Notation One) [11] i Bazowe Reguły Kodowania (Basic Encoding Rules) [12]. Pierwszy z nich definiuje składnię (i do pewnego stopnia semantykę) języka definiowania typów danych oraz metodologię jego używania. Drugi formułuje jednoznaczny przepis kodowania (i pośrednio dekodowania) informacji i jej struktury (typu) zadanej w ASN.1. Oba standardy sa pewnymi, nieznacznie zmodyfikowanymi, wersjami zaleceń zawartych w dokumencie X.409 opracowanym wczesniej przez CCITT. Trzeba odrazu powiedzieć, że obie normy zostały bardzo szybko zaakceptowane przez środowisko i cokolwiek o nich można powiedzieć nie zmienia to faktu, że już są i z cała pewnością będą powszechnie stosowane w definiowaniu protokołów nowej generacji. Szereg firm software'owych opracowało już - lepsze lub gorsze - kompilatory, które tłumaczą struktury ASN.1 na struktury danych wybranego języka programowania (głównie C, ale także Pascala czy Cobolu) i na odwrót oraz produkuja procedury automatycznego kodowania i dekodowania zgodne z Bazowemi Regułami Kodowania.

Można postawić w tym miejscu pytanie, dlaczego w językach takich jak Estelle, Lotos czy SDL nie zastosowano mechanizmów ASN.1 definiowania typów danych. Takie połączenie dawałoby bowiem ujednolicone i całościowe formalizmy specyfikacji. Złożyły się na to przyczyny natury historycznej, organizacyjnej (wewnatrz ISO i CCITT) i, w pewnym stopniu, technicznej (silne ukierunkowanie ASN.1 i specyficzny rodzaj jego interpretacji). Koegzystencja dwoch formalizmów w jednym opisie stwarza pewne problemy, które jednak, dla celów praktycznego korzystania z takich opisow, udaje sie częściowo rozwiązać przy pomocy, na przykład, zabiegów podobnych do tych jakie stosuje się przy rozdzielnej kompilacji (separate compilation).

Implementacja norm protokołów nie może tylko polegać na wierze w zdolności programistów do poprawnego zrealizowania zawartych w tych normach zaleceń. Wymaga dokładnej i wielostronnej weryfikacji. Jakkolwiek zaawansowane stosowlibyśmy metody projektowania i sprawdzania poprawności programów w procesie ich powstawania, nie unikniemy konieczności testowania zgodności (conformance testing) systemu z jego specyfikacją (normą) już po jego uruchomieniu. Pytanie polega na tym, co i jak testować oraz jak zagwarantować jednolitość i spójność przeprowadzanych testów w różnych realizacjach. Pytania to nie łatwe i opracowanie odpowiedzi na nie wymaga znowu wspólnego, międzynarodowego wysiłku. Podjęto go i tym razem, w r. 1984, w ISO w ścisłej

współpracy z CCITT. Punktem wyjściowym rozpoczętych prac było rozróżnienie dwu ich zasadniczych celów: znormalizowania metodologii testowania protokołów i uzyskania norm ustalających zbiory ("gry") testów dla poszczególnych protokołów. Metodologię testowania ujęto w pięciu częściach [13], z czego dwie pierwsze osiągnęły status Wstępnych Propozycji (Draft Proposals) a inne są w stadium opracowywania roboczych dokumentów. Gry testów dla poszczególnych protokołów są przedmiotem intensywnych badań i dyskusji prowadzonych wewnątrz firm komputerowych, w ich zgrupowaniach (COS, POSI, SPAG i inne) czy wreszcie w krajowych organizacjach normalizacyjnych.

Jednym z problemów metodologii testowania był wybór odpowiedniego formalizmu, w którym znormalizowane gry testow byłyby zdefiniowane. Kandydatami były języki Estelle, Lotos i SDL, ale zdecydowano sie raczej (decyzja nie jest ciągle ostateczna) na opracowanie nowej wyspecjalizowanej notacji nazwanej TTCN (Tree and Tabular Combined Notation). W dużym uproszczeniu, jest to język tablic, w których w jednej z kolumn przedstawiona jest dynamika zdarzen testujących (interakcji wejściowych i wyjściowych) w postaci drzewa wystąpień tych zdarzeń. W innych kolumnach, z każdym wystąpieniem zdarzenia w drzewie, stowarzyszone są jego pewne atrybuty, parametry, stałe, a także możliwy werdykt testowania: zawiódł (failed), przeszedł (passed), nieroztrzygnięty (inconclusive). Choć TTCN nie jest jeszcze językiem zatwierdzonym a nawet do końca zdefiniowanym, wiele próbnych specyfikacji systemów testów zostało już w nim zaproponowanych. Podkreślmy na koniec, że nie jest to, w zasadzie, język bezpośrednio implementowalny, ale język przeznaczony do możliwie jednoznacznego opisu zbioru sekwencji testujacych (test suits) dla określonego standardu protokołu OSI, lub jego wybranego profilu.

Istnieje niezliczenie wiele innych propozycji formalizmów służacych specyfikacji i projektownia protokołów komunikacyjnych, nie mowiąc już o jezykach programowania, w których ostatecznie są one implementowane. Skupilismy naszą uwagę w powyższym przeglądzie tylko na tych, które przeszły, lub przechodza, proces normalizacji równolegly do procesu, któremu podlegaja same protokoły. Wydaje się, że takie podejście pozwala spojrzeć na rozległą i skomplikowaną dziedzinę z pespektywy unifikującej często, na pozór, rozbieżne tendencje.

5. Podsumowanie

Przedstawiliśmy wtym krótkim szkicu, bardzo pobieżnie, problematykę związaną z sieciami komputerowymi i protokołami komunikacyjnymi. Ustabilizowanie zasad ich projektowania i uruchamiania umożliwi niezawodną i łatwą wymianę informacji na odległość, a także racjonalne wykorzystanie olbrzymiego a rozproszonego potencjału obliczeniowego.

Staraliśmy się zwrócić uwagę na zasadnicze cechy i główne elementy architektury sieci. Pominęlismy szczegóły, które zainteresowany czytelnik znaleźć może w bogatej literaturze (np. [5]). Uwypukliliśmy rolę znormalizowanego modelu ISO OSI i w ogóle

rolę standardów międzynarodowych w tej dziedzinie ponieważ, wydaje się, że ich powszechne wprowadzenie w życie nie podlega już dyskusji, jest tylko kwestią czasu. Można mieć pretensje do niedoskonałości norm oraz, niekiedy, do pochopności w ich kreowaniu, ale docenić także należy zbiorowy wysiłek podjęty w celu opanowania i wyznaczenia uzgodnionego i pożądanego kierunku rozwoju niezwykle szybko postępującej rzeczywistości.

BIBLIOGRAFIA

[1] The Ethernet - A Local Network, Version 1.0, Digital, Intel, Xerox, 1980

[2] J.M. McQuillan, D.C. Walden, The ARPA Network Design Decisions, Computer Networks, vol.1, 1977

[3] ISO 7498, Information Processing Systems, Open Systems Interconnection, Basic Reference Model, 1984

[4] H. Zimmermann, OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection, IEEE Trans. on Comm., vol COM-28, 1980

[5] A.S. Tanenbaum, Sieci komputerowe, WNT, Warszawa 1988

[6] G. Pujolle, L'etat de normalisation de la couche application OSI, Proc. CFIP'88, Eyrolles, Paris 1988

[7] S. Budkowski, P. Dembiński, An Introduction to Estelle: a specification language for distributed systems, Computer Networks and ISDN Systems, 1988 (takze Materialy Jesiennej Szkoly PTI, Mragowo 1987)

[8] T. Bolognesi, E. Brinksma, Introduction to the ISO Specification Language Lotos, Computer Networks and ISDN Systems, 14, 1987

[9] CCITT SG XI, Recommendation Z.100, 1987

[10] M. Diaz, Ch. Vissers, S. Budkowski, Estelle and Lotos Software Environments for the Design of Distributed Systems, Proc ESPRIT Conference Week, 1987

[11] ISO 8824, Information Processing Systems, Open Systems Interconnection, Specification of Abstract Syntax Notation One, 1987

[12] ISO 8825, Information Processing Systems, Open Systems Interconnection, Specificati

NOTATKI

NOTATKI

NOTATKI

Druk. Wyd. "ALFA" nakł. 300 zam. 1969/83