Wybór zagadnień związanych z teorią i konstrukcją obiektowych języków zapytań.

Na podstawie książki (i z wykorzystaniem slajdów do książki) Kazimierza Subiety “Teoria i konstrukcja obiektowych języków zapytań”

Sebastian Kaczanowski

Rzeczy, o których trzeba powiedzieć

- Każdy nawet najdrobniejszy problem semantyczny, jest ogromnym problemem
- Im mniej, różnych co do semantyki, elementów w języku tym lepiej (n^2)
- ortogonalność (czytaj niezależność) – język zapytań, trwałość
- pragmatyzm (informatyka dziedzina inżynierska a nie część matematyki)

  Niedłączną stroną dowolnej teorii jest praktyka. Brak przełożeń praktycznych oznacza, że dana “teoria” jest utopią. Utopia, nawet opisana za pomocą zaawansowanej matematyki i wspomagana matematycznymi dowodami, nie jest teorią, lecz nonsensem technicznym.

- skromny tytuł

Cel książki

- Pokazanie semantyki operacyjnej, która jest w stanie opisać obiektowy język, który jest jednocześnie językiem zapytań. (Stos środowisk)
- Pokazanie modeli składu danych, które mogą być przetwarzane przez powyższy język. (Modele M0, M1, M2, M3)
- Zaprezentowanie samego języka SBQL (Stack-Based Query Language)
- i polemiki

Modele składu danych

- M0: obejmuje dowolnie powiązane hierarchiczne struktury danych; nie obejmuje klas, dziedziczenia, interfejsu i hermetyzacji. Model M0 pozwala wyjaśnić semantykę relacyjnych języków zapytań (szczególnie SQL), przykrywa koncepcję zagnieżdżonych relacji, struktury implikowane przez XML i dane określane jako pół-strukturalne.
- M1: uzupełnia M0 o pojęcia klasy, dziedziczenia i wielodziedziczenia w najczęściej spotykanym rozumieniu; nie obejmuje hermetyzacji i interfejsu.
- M2: uzupełnia model M1 oraz nieco go modyfikuje wprowadzając dziedziczenie pomiędzy obiektami oraz dynamiczne role. Można go również uważać jako model odwzorowujący koncepcję wielu interfejsów do obiektu.

Modele składu danych

- M3: uzupełnia model M1 lub M2 o pojęcie hermetyzacji - podział własności klas i obiektów na publiczne i prywatne.
- Podana rodzina modeli nie zamyka tematu.

Podstawowe pojęcia

- I - zbiór identyfikatorów (i, i_1, i_2, ... - oznaczenia identyfikatorów)
- N - zbiór nazw (n, n_1, n_2, ... - oznaczenia nazw)
- V - zbiór wartości atomowych (v, v_1, v_2, ... - oznaczenia wartości)

- **Obiekt atomowy**: trójka <*i, n, v*>.
- **Obiekt pointerowy**: trójka <i_1, *n*, i_2>. Obiekt jest identyfikowany przez i_1, natomiast i_2 jest pointerem (referencją) do innego obiektu.
- **Obiekt złożony**: trójka <*i, n, T*>, gdzie *T* jest zbiorem dowolnych obiektów. Powyższa reguła umożliwia rekurencyjne tworzenie obiektów o nieograniczonej złożoności i o nieograniczonej liczbie poziomów hierarchii.

Podstawowe pojęcia

- **Skład obiektów** jest zdefiniowany jako para <*S*, *R*>, gdzie *S* jest zbiorem obiektów, zaś *R* jest zbiorem identyfikatorów "startowych”.
  - Zbiór *R* wyznacza punkty wejściowe do składu obiektów, tj. obiekty "korzeniowe" (*root objects*), które mogą być początkiem wyszukiwania w zbiorze przechowywanych obiektów.

S - **Obiekty**:

< i_1 , Prac , {< i_2, Nazwisko, ”Nowak” >, < i_3, Zar, 2500 >,
< i_4, PracujeW, i_{17} > } >,

< i_5 , Prac , {< i_6, Nazwisko, ”Kowalski” >, < i_7, Zar, 2000 >,
< i_8, PracujeW, i_{22} > } >,

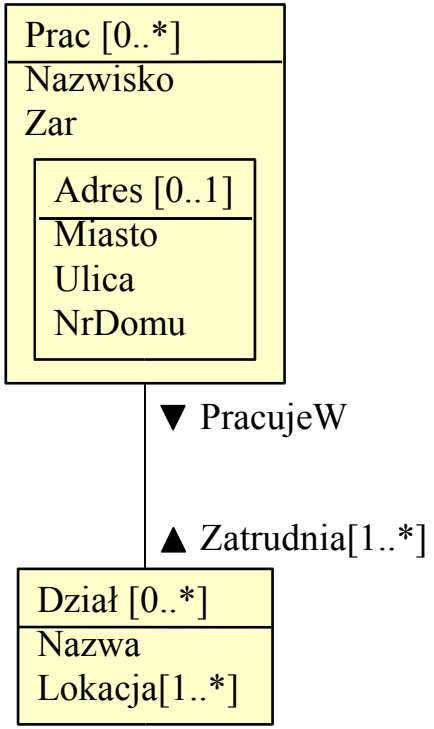
< i_9 , Prac , {< i_{10}, Nazwisko, ”Barski” >, < i_{11}, Zar, 900 >,
< i_{12}, Adres, {< i_{13}, Miasto, ”Radom” >,
< i_{14}, Ulica, ”Wolska” >,
< i_{15}, NrDomu, 12 > } >,
< i_{16}, PracujeW, i_{22} > } >,

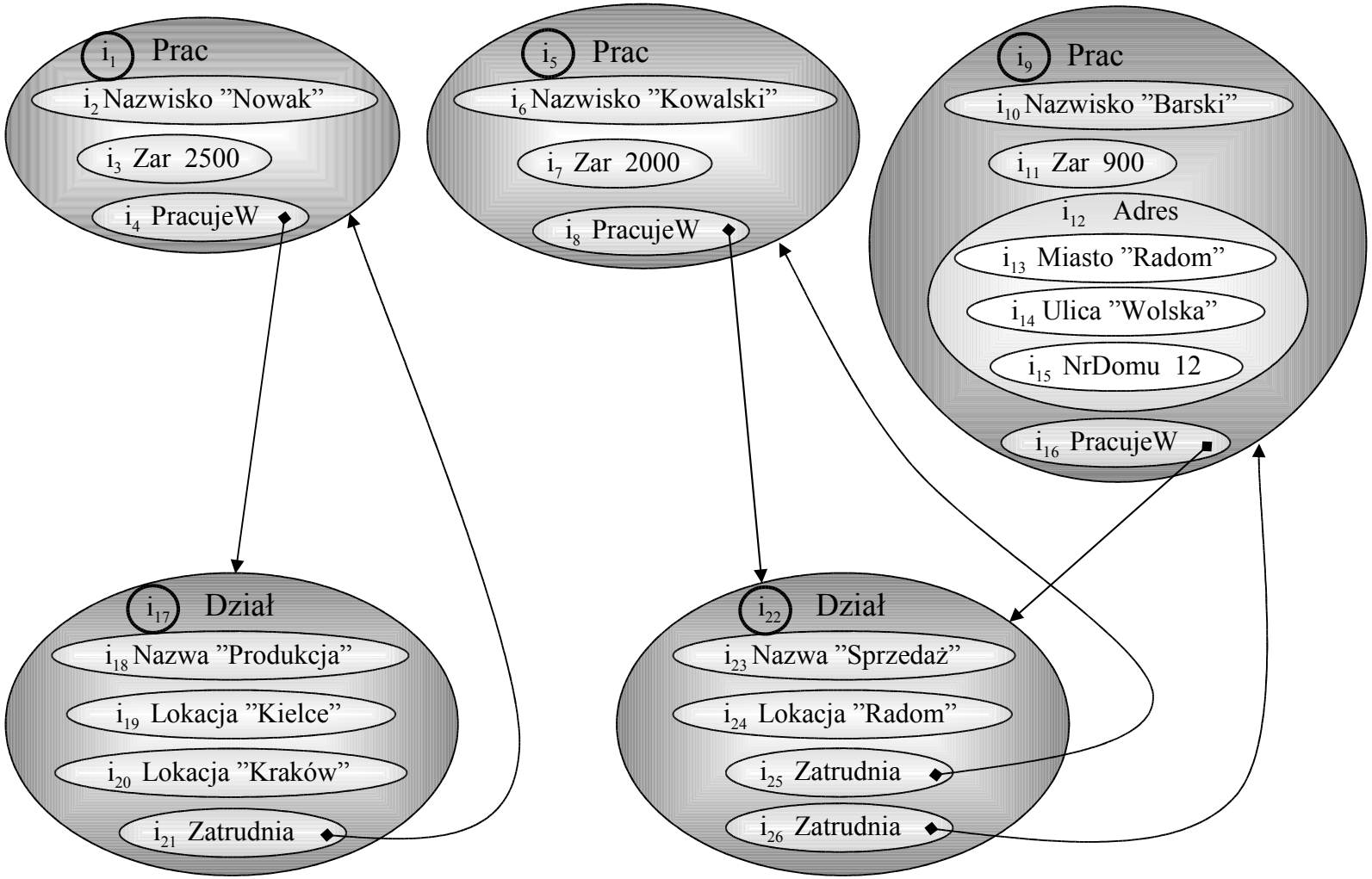
< i_{17} ,Dział, {<i_{18}, Nazwa, ”Produkcja” >,
< i_{19}, Lokacja, ”Kielce” >, < i_{20}, Lokacja, ”Kraków” >,
< i_{21}, Zatrudnia, i_1 > } >,

< i_{22} , Dział,{< i_{23}, Nazwa, ”Sprzedaż” >,
< i_{24}, Lokacja, ”Radom” >,
< i_{25}, Zatrudnia, i_5 >, < i_{26}, Zatrudnia, i_9 > } >

R - **Identyfikatory startowe**:
i_1 , i_5 , i_9 , i_{17} , i_{22}

Diagram klas




i1 Prac
i2 Nazwisko ”Nowak”
i3 Zar 2500
i4 PracujeW
i5 Prac
i6 Nazwisko ”Kowalski”
i7 Zar 2000
i8 PracujeW
i9 Prac
i10 Nazwisko ”Barski”
i11 Zar 900
i12 Adres
i13 Miasto ”Radom”
i14 Ulica ”Wolska”
i15 NrDomu 12
i16 PracujeW
i17 Dział
i18 Nazwa ”Produkcja”
i19 Lokacja ”Kielce”
i20 Lokacja ”Kraków”
i21 Zatrudnia
i22 Dział
i23 Nazwa ”Sprzedaż”
i24 Lokacja ”Radom”
i25 Zatrudnia
i26 Zatrudnia

Założenia

- Nie będziemy robić rozróżnienia pomiędzy pojęciem języków programowania określanym jako „zmienna” oraz wprowadzonymi przez nas obiektami.
  - Np. zmienna *x* z języka programowania posiadająca aktualną wartość 5 będzie w naszych modelach składu reprezentowana jako trójka <*i*, *x*, 5>, gdzie *i* jest pewnym identyfikatorem wewnętrznym tej zmiennej, np. jej adresem w pamięci operacyjnej.
- Nie będziemy również rozróżniać pojęcia zmiennej i pojęcia obiektu na zasadzie: *obiekt posiada swoją klasę, zaś zmienna nie posiada klasy*.
  - Wszystkie obiekty posiadają klasę, ale dla niektórych obiektów są one puste (nie zawierają żadnych inwariantów), wobec czego są pomijane.
  - Model M0 nie ma klas, ale wprowadza obiekty (inaczej „zmienne”).

Założenia

- Nie zajmujemy się również cechą trwałości obiektów, tj. czy obiekty są przechowywane na dysku czy też w pamięci operacyjnej.
  - Cecha trwałości nie ma znaczenia dla definiowania semantyki języka zapytań.
- Nie będziemy przywiązywać wagi do podziału obiektów na proste i złożone, a także nie wprowadzamy specjalnej terminologii i pojęć dla obiektów złożonych.
- Tego rodzaju relatywizm obiektów ma zasadnicze znaczenie dla uproszczenia definiowanych języków, znacznie upraszcza metamodel i operacje na metamodelu, zwiększa uniwersalność języka i ma zasadnicze znaczenia dla prostoty oraz klarowności semantyki i pragmatyki.

Założenia

- W naszych modelach składu kolekcja nie występuje jako pojedynczy, identyfikowalny byt programistyczny.
  - Nie można utworzyć pojedynczej referencji prowadzącej do kolekcji.
  - Można zbudować zbiór referencji do elementów danej kolekcji, w szczególności, do *wszystkich* elementów danej kolekcji.
- Można oczywiście utworzyć obiekt, który w środku będzie miał tak samo nazwane podobiekty, i wtedy można zbudować referencję do takiego obiektu.

Możliwości

- M0 umożliwia formalizację wartości zerowych i wariantów (unii). Np. atrybut (złożony) *Adres* występuje tylko dla jednego obiektu *Prac*.
  - Model niczym nie zobowiązuje nas do tego, aby obiekty posiadające te same nazwy posiadały pod-obiekty o tych samych nazwach.
  - Model nie wprowadza (jak dotąd) ograniczeń typologicznych, co daje pełną swobodę w zakresie budowy poszczególnych obiektów.
  - Będziemy uważać, że ewentualny dyskryminator wariantu jest takim samym pod-obiektem jak pozostałe.

Możliwości

- M0 przykrywa nieregularności w danych, określane ostatnio jako dane pół-strukturalne (*semistructured*).
  - Oznacza to, że budowana przez nas semantyka będzie adekwatna do budowy języków zapytań i języków programowania obsługujących takie struktury.
  - W szczególności, semantyka będzie nadawała się do struktur rozważanych w technologiach opartych na XML.

Możliwości

- Model M0 włącza struktury danych zakładane przez model relacyjny jako szczególny przypadek. Semantykę relacyjnego języka zapytań (w szczególności SQL) można będzie zdefiniować jako szczególny przypadek definiowanej przez nas semantyki.
- Model M0 przykrywa również model zagnieżdżonych relacji (NF^2) jako szczególny przypadek.
- Również struktury danych implikowane przez inne modele, określane przez ich autorów jako funkcjonalne, obiektowe, logiczne, semantyczne, itd. dadzą się sformalizować w terminach podanego prostego modelu.

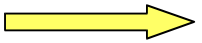
Możliwości

Schemat relacyjny:
Prac(Nazwisko, Zarobek, PracujeW)

Krotki relacji jako obiekty złożone

Model relacyjny - Relacja Prac

Nazwisko	**Zarobek**	**PracujeW**
Nowak	2500	Produkcja
Kowalski	2000	Sprzedaż
Barski	2000	Sprzedaż



Model składu obiektów M0:

S - Obiekty:

< i_1 , Prac, { < i_2, Nazwisko, ”Nowak” >,
< i_3, Zarobek, 2500 >,
< i_4, PracujeW, ”Produkcja” > } >,

< i_5 , Prac, { < i_6, Nazwisko, ”Kowalski” >,
< i_7, Zarobek, 2000 >,
< i_8, PracujeW, ”Sprzedaż” > } >,

< i_9 , Prac, { < i_{10}, Nazwisko, ”Barski” >,
< i_{11}, Zarobek, 2000 >,
< i_{12}, PracujeW, ”Sprzedaż” > } >

R - Identyfikatory startowe:
i_1 , i_5 , i_9

M1

S - Obiekty i klasy:

< i_1 , Osoba , { < i_2, Nazwisko, ”Wilski” >, < i_3, RokUr, 1950 > } >,

< i_4 , Prac , { < i_5, Nazwisko, ”Nowak” >, < i_6, RokUr, 1944 >,
< i_7, Zar, 2500 >, < i_8, PracujeW, i_{127} > } >,

< i_9 , Prac , { < i_{10}, Nazwisko, ”Kowalski” >, < i_{11}, RokUr, 1940 >,
< i_{12}, Zar, 2000 >, < i_{13}, PracujeW, i_{128} > } >,

< i_{40} , KlasaOsoba , { < i_{41}, Wiek, (..kod metody Wiek..) >,
inwariant: Nazwa obiektów = "Osoba",
..pozostałe inwarianty klasy KlasaOsoba ..}>,

< i_{50} , KlasaPrac , { < i_{51}, ZmieńZar, (..kod metody ZmieńZar..) >,
< i_{52}, ZarNetto, (...kod metody ZarNetto..) >,
inwariant: Nazwa obiektów = "Prac";
..pozostałe inwarianty klasy KlasaPrac .. }>,

R - Identyfikatory startowe:

i_1 , i_4 , i_9

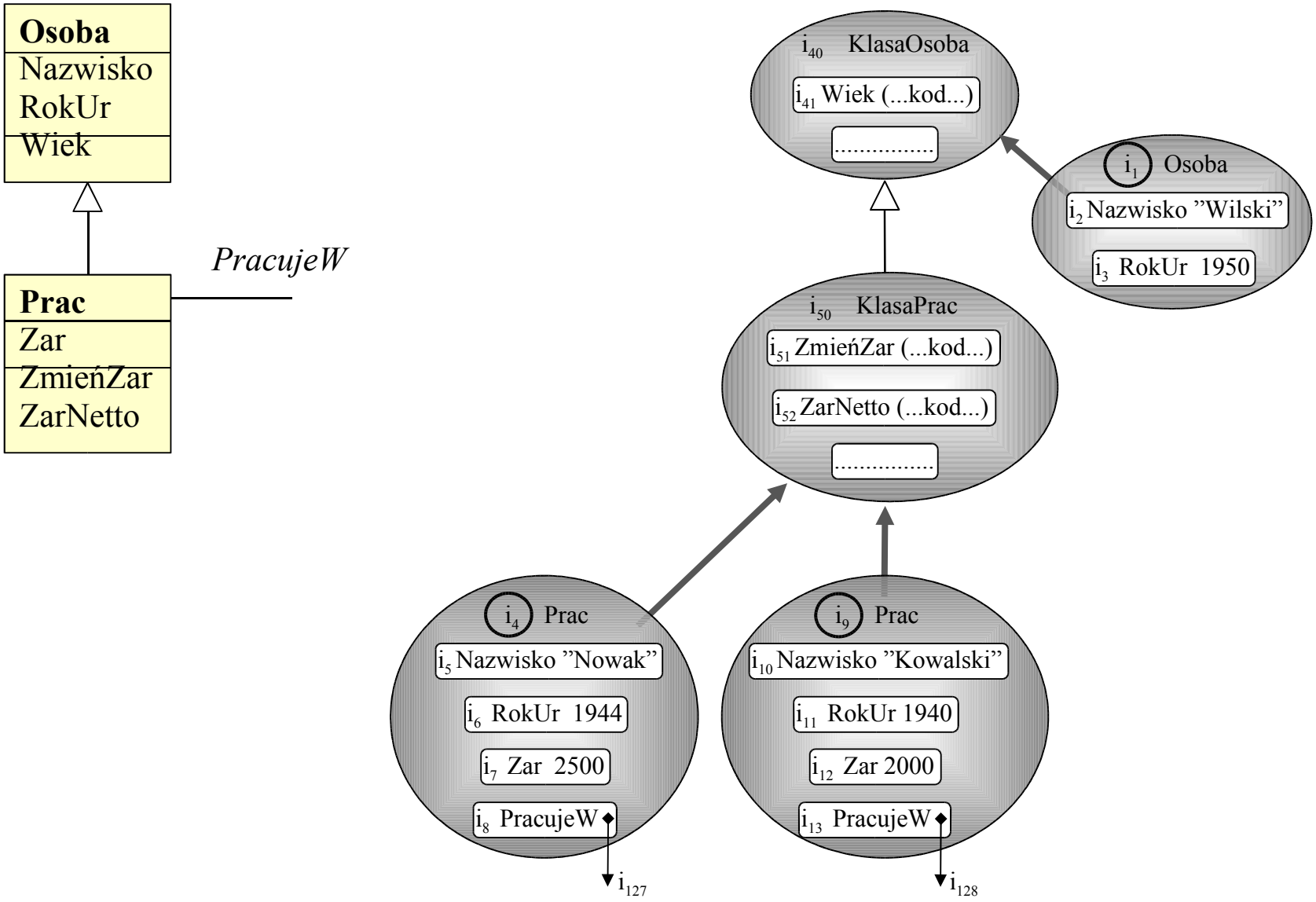
KK - Związki dziedziczenia między klasami:

< i_{50} , i_{40} >

OK - Związki dziedziczenia między obiektami i klasami:

< i_1 , i_{40} >, < i_4 , i_{50} >, < i_9 , i_{50} >

M1


Osoba
Nazwisko
RokUr
Wiek
PracujeW
Prac
Zar
ZmieńZar
ZarNetto
i40 KlasaOsoba
i41 Wiek (...kod...)
.....
i1 Osoba
i2 Nazwisko "Wilski"
i3 RokUr 1950
i50 KlasaPrac
i51 ZmieńZar (...kod...)
i52 ZarNetto (...kod...)
.....
i4 Prac
i5 Nazwisko "Nowak"
i6 RokUr 1944
i7 Zar 2500
i8 PracujeW
i127
i9 Prac
i10 Nazwisko "Kowalski"
i11 RokUr 1940
i12 Zar 2000
i13 PracujeW
i128

M2

S - Obiekty i klasy:

$< i_1$, Osoba , $\{ < i_2$, Nazwisko, "Wilski" >, $< i_3$, RokUr, 1950 > $\}$ >,
$< i_4$, Osoba , $\{ < i_5$, Nazwisko, "Nowak" >, $< i_6$, RokUr, 1944 >$\}$ >,
$< i_7$, Osoba , $\{ < i_8$, Nazwisko, "Kowalski" >, $< i_9$, RokUr, 1940 >$\}$ >,
$< i_{13}$, Prac , $\{ < i_{14}$, Zar, 2500 >, $< i_{15}$, PracujeW, i_{127} > $\}$ >,
$< i_{16}$, Prac , $\{ < i_{17}$, Zar, 2000 >, $< i_{18}$, PracujeW, i_{128} > $\}$ >,
$< i_{19}$, Student , $\{ < i_{20}$, NrIndeksu, 76943 >, $< i_{21}$, Wydział, "fizyka" >$\}$ >,

$< i_{40}$, KlasaOsoba , $\{ < i_{41}$, Wiek, (...kod metody Wiek...) >,
...pozostałe inwarianty klasy KlasaOsoba ...}>,

$< i_{50}$, KlasaPracownik , $\{ < i_{51}$, ZmieńZar, (...kod metody ZmieńZar...) >,
$< i_{52}$, ZarNetto, (...kod metody ZarNetto...) >,
...pozostałe inwarianty klasy KlasaPrac ... }>,

$< i_{60}$, KlasaStudent , $\{ < i_{61}$, ŚredniaOcen, (...kod metody ŚredniaOcen...) >,
...pozostałe inwarianty klasy KlasaStudent ... }>,

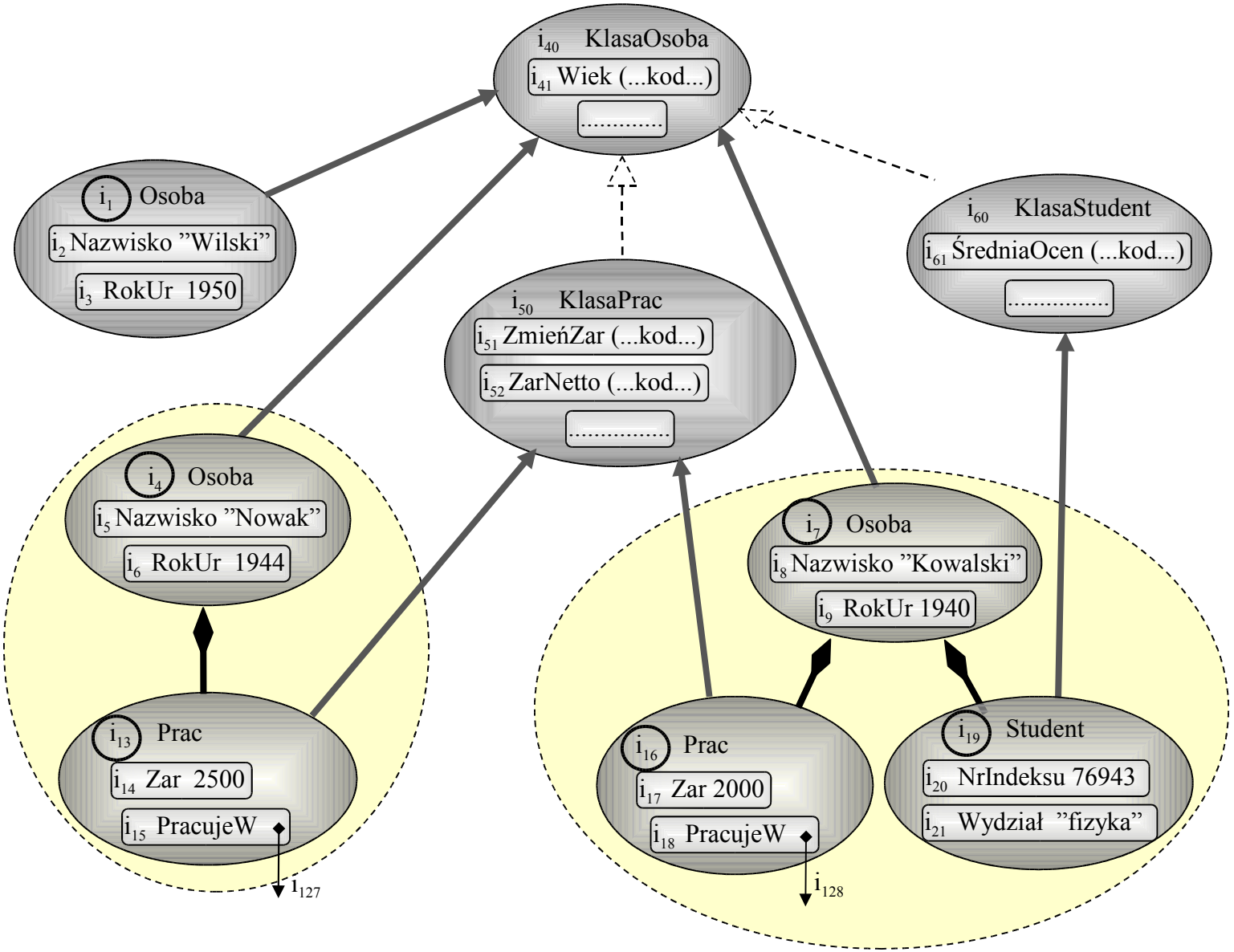
R - Identyfikatory startowe: i_1 , i_4 , i_7 , i_{13} , i_{16} , i_{19}

KK - Związki dziedziczenia między klasami: Zbiór pusty

OK - Związki dziedziczenia między obiektami i klasami:
$< i_1$, $i_{40} >$, $< i_4$, $i_{40} >$, $< i_7$, $i_{40} >$, $< i_{13}$, $i_{50} >$, $< i_{16}$, $i_{50} >$, $< i_{19}$, $i_{60} >$,

OO - Związki dziedziczenia między obiektami i obiektami:
$< i_{13}$, $i_4 >$, $< i_{16}$, $i_7 >$, $< i_{19}$, $i_7 >$

M2


i40 KlasaOsoba
i41 Wiek (...kod...)
.....
i1 Osoba
i2 Nazwisko ”Wilski”
i3 RokUr 1950
i60 KlasaStudent
i61 ŚredniaOcen (...kod...)
.....
i50 KlasaPrac
i51 ZmieńZar (...kod...)
i52 ZarNetto (...kod...)
.....
i4 Osoba
i5 Nazwisko ”Nowak”
i6 RokUr 1944
i7 Osoba
i8 Nazwisko ”Kowalski”
i9 RokUr 1940
i13 Prac
i14 Zar 2500
i15 PracujeW
i127
i16 Prac
i17 Zar 2000
i18 PracujeW
i128
i19 Student
i20 NrIndeksu 76943
i21 Wydział ”fizyka”

M2 - role

- **Wielokrotne dziedziczenie:** Ponieważ role są hermetyzowane, nie może wystąpić konflikt nazw nawet wtedy, gdy różne role (czyli specjalizacje obiektu) posiadają własności o tych samych nazwach.
- **Powtarzalne dziedziczenie**: Jest normalne, że obiekt może mieć dwie lub więcej ról o tej samej nazwie. Np. Kowalski może być dwa razy studentem, w różnych szkołach. Ten przypadek nie jest objęty klasycznym modelem dziedziczenia lub wielokrotnego dziedziczenia.
- **Przechowywanie obiektów historycznych**: Role idealnie nadają się do przechowywania obiektów historycznych nie powodując przy tym anomalii z unikalnością identyfikatorów obiektów. Np. można łatwo zapisać fakt, że Kowalski był już kiedyś dwa razy studentem.

M2 - role

- **Wielo-aspektowe dziedziczenie**: Klasa może być specjalizowana wg wielu aspektów, np. według stosunku do zatrudnienia lub stosunku do wykształcenia. UML przykrywa tę cechę, ale jest ona nieobecna w narzędziach obiektowych, co prowadzi m.in. do efektu "eksplozji klas". Role automatycznie mają cechę wielo-aspektowego dziedziczenia.
- **Migracja obiektów**: Role mogą pojawiać się i znikać dynamicznie, co w terminach klasycznych modeli obiektowych oznacza, że obiekt zmienia klasę (czyli "migruje") bez zmiany tożsamości. Dla klasycznych modeli jest to duży problem. W przypadku ról problem ten nie istnieje.
- **Spójność referencyjna**: W przypadku ról związki mogą prowadzić do ról, a nie do całych obiektów. Np. jeżeli nawigujemy od obiektu *Szkoła* do obiektu Kowalskiego poprzez jego rolę *Student*, wówczas niedostępny jest atrybut *Zar* i metoda *ZarNetto*. Jest to znaczne uściślenie hermetyzacji.

M2 - role

- **Dynamiczne dziedziczenie**: *KlasaPrac* nie dziedziczy statycznie z *KlasaOsoba*. Zamiast tego, rola *Prac* dziedziczy dynamicznie z roli *Osoba* wszystkie cechy, włączając metody zawarte w klasie *KlasaOsoba.* Stwarza to nową sytuację dla przesłaniania i polimorfizmu.
- **Heterogeniczne, przecinające się kolekcje.** W klasycznych modelach, np. w ODMG, jeżeli obiekt był elementem kolekcji, to nie mógł być jednocześnie elementem innej kolekcji. Jest to ograniczenie modelowania pojęciowego. Dynamiczne role posiadają naturalną zdolność modelowania heterogenicznych, przecinających się kolekcji.
  - Np. można utworzyć rolę *Pacjent,* i tą rolę objąć ludzi i zwierzęta, oraz inną rolę *ObiektyDzisiajAktualizowane* obejmującą obiekty dowolnego typu. Kolekcje *Pacjent* i *ObiektyDzisiajAktualizowane* są heterogeniczne i zachodzą na siebie.

M2 - role

- **Programowanie aspektowe** (*Aspect-Oriented Programming, AOP*) **i rozdzielenie aspektów**. AOP zajmuje się rozdzieleniem przecinających się aspektów (*cross-cutting concerns*) celem umieszczenie każdego aspektu w odrębnym module programu (np. historię zmian, reguły bezpieczeństwa, wizualizację, itd.). Dynamiczne role mają wiele zbieżności z AOP lub mogą być wykorzystane jako techniczne wspomaganie AOP.

M3

- Model M3 uzupełnia model M1 lub M2 w taki sposób, że klasy są wyposażone w dodatkowy inwariant - *listę eksportową*. Jest ona zbiorem nazw własności tej klasy i jej obiektów (metod, atrybutów), które będą widoczne z zewnątrz.

SBQL

- Język SBQL jest sformalizowanym obiektowym językiem zapytań w stylu SQL lub OQL. Posiada semantyczne odpowiedniki podstawowych konstrukcji tych języków. Może być zdefiniowany (uściślony) dla wielu modeli składu, w szczególności dla modeli M0 – M3.
- W odróżnieniu od relacyjnych i obiektowych algebr, rachunków, logik i innych tego rodzaju koncepcji, definicja SBQL bazuje na pojęciu *stanu*, którego składnikami są *skład obiektów* oraz *stos środowisk*.
- SBQL będziemy uważać za wzorzec teoretyczny podobny do algebry relacji. SBQL jest jednak nieporównywalnie bardziej uniwersalny i konsekwentny niż dowolna tego rodzaju algebra, włączając tzw. algebry obiektowe. Będziemy starali się wykazać, że:
- **Języki zapytań mogą zawierać operatory nie-algebraiczne, których odwzorowanie w dowolną algebrę jest niemożliwe bez wprowadzenia poważnych ograniczeń koncepcyjnych.**

SBQL

- Dowolny literal jest zapytaniem; np. 2, 3.14, ”*Kowalski*”
- Dowolny element zbioru *N* jest zapytaniem; np. *Osoba, Student, zarobek, wiek.*
- Zapytania można łączyć w większe zapytania przy pomocy operatorów.
- Operatory będą podzielone na *unarne* i *binarne*, oraz *algebraiczne* i *nie-algebraiczne*.
- Ogólnie wszystko jest zapytaniem, ale warto powiedzieć, że:
  - Jeżeli $n \in N$ jest nazwą procedury, funkcji, lub metody posiadającej k parametrów, zaś $q_1, q_2, \ldots, q_k$ są zapytaniami, wówczas $n(q_1; q_2; \ldots, q_k)$ jest zapytaniem.

Przykładowe zapytania SBQL

2000

"Kowalski"

zarobek

Osoba

2+2

zarobek > 2000

Osoba **where** (*zarobek* > 2000)

(*Osoba* **where** (*wiek*() > 30)) . (*zarobek* + *x* + 2000/y)

((*Osoba* **as** *p*) **join** (*p.pracuje_w.Dział* **as** *d*)) . (*p.nazwisko*, *d.nazwa*)

∀ *Osoba* (*wiek* < 65)

Dział **where** (∃(*zatrudnia.Osoba*) **as** *p* (*p.wiek*() < 17))

(((*Osoba* **as** *p*) **join** (*p.pracuje_w.Dział* **as** *d*)) **where** (*p.Nazwisko* = "Nowak" **and** *d.Nazwa* = "Kontrola")) . (*p.nazwisko*, *d.nazwa*)

Niepoprawne zapytanie

(*Nazwisko* = "Nowak") **where** *Osoba*

Semantyka

- Semantyka SBQL będzie zdefiniowana poprzez procedurę *eval*.
- Argumentem procedury *eval* jest dowolne zapytanie, zaś wynikiem procedury jest rezultat tego zapytania włożony na wierzchołek *QRES*.
- Procedura *eval* będzie korzystać ze składu obiektów, *ENVS* oraz *QRES*.
- Rezultaty zapytań zapisane na QRES są „konsumowane” przez operatory języka, dla których zapytania były parametrami.
- Jeżeli zapytanie jest literalem $l \in L$, to procedura *eval* wkłada odpowiadającą mu wartość atomową $\underline{l} \in V$ na wierzchołek *QRES*.

Semantyka

- Jeżeli zapytanie jest nazwą $n \in N$, to procedura *eval* dokonuje wiązania tej nazwy na *ENVS* (funkcja *bind*), a następnie wynik tego wiązania wkłada na wierzchołek stosu *QRES*.

```
procedure eval( q : zapytanie )
begin
        parse( q );              (* rozbiór gramatyczny *)

        case q jest rozpoznane jako l ∈ L :
        push( QRES, l );

        case q jest rozpoznane jako n ∈ N :
        push( QRES, bind( n ) );

        case .....
        .....
end;
```

Semantyka

- Operatory łączące zapytania będziemy dzielić na *algebraiczne* i *nie-algebraiczne*.
  - Istotą podejścia stosowego są operatory nie-algebraiczne.
- Fundamentalna różnica pomiędzy operatorami algebraicznymi i nie-algebraicznymi polega na ich stosunku do stosu środowisk *ENVS*.
- Operatory algebraiczne nie używają *ENVS*: działanie takiego operatora dotyczy wyłącznie stosu *QRES* (z reguły jednego lub dwóch wierzchołkowych elementów).
- Operatory nie-algebraiczne, oprócz działań na *QRES*, bezpośrednio odwołują się do konstrukcji i operacji zachodzących na *ENVS*.

Operatory algebraiczne

- Cechą dowolnej algebry jest m.in. to, że w wyrażeniu $x_1 \Delta x_2$ (gdzie Δ jest operatorem algebry) kolejność ewaluacji argumentów x_1 oraz x_2 tego operatora nie ma znaczenia. Jest to zasadnicza różnica w stosunku do operatorów nie-algebraicznych.
- Operatory algebraiczne:
- $<, \leq, >, \geq, +, -, *, /, \neq, =$
- tangens, podłoga, pierwiastek
- operatory i funkcje działające na stringach, datach
- Funkcje arytmetyczne zagregowane: *sum* (suma liczb), *avg* (średnia), *min* (liczba minimalna), *max* (liczba maksymalna)
- count(Pracownik), distinct(Pracownik.zawód), exists(Pracownik where zawód = ”analityk”), dereferencja itd.
- Definicja nazwy (operator **as**)

eval dla operatorów algebraicznych

```
procedure eval( q : zapytanie )
begin
  .....
    case q jest rozpoznane jako Δ( q_1 ) lub Δ q_1:
    begin
        wynik_q1: Rezultat;    (* lokalna zmienna typu Rezultat *)
        eval( q_1 );      (* rezultat q_1 na wierzchołku stosu QRES *)
        wynik_q1 := top( QRES );  pop( QRES );
        push( QRES, Δ ( wynik_q1 );
    end;

    case q jest rozpoznane jako q_1 Δ q_2 :
    begin
        wynik_q1, wynik_q2: Rezultat;   (* lokalne zmienne *)
        eval( q_1 );     (* rezultat q_1 na wierzchołku stosu QRES *)
        eval( q_2 );     (* rezultat q_2 na wierzchołku stosu QRES *)
        wynik_q2 := top( QRES );  pop( QRES );
        wynik_q1 := top( QRES );  pop( QRES );
        push( QRES, wynik_q1 Δ  wynik_q2 );
    end;

    case .....
  .....
end;
```

Operatory niealgebraiczne

- Do nich należą operator *where*, operator kropki, kwantyfikatory, zależne złączenie *join*, sortowanie (*order by*), i inne. Wszystkie są binarne.
- Mimo podobieństwa do operatorów algebraicznych, semantyka operatorów niealgebraicznych nie da się prosto sprowadzić do algebry.
  - To zdanie może wydawać się niejasne. W modelu relacyjnym operatory selekcji (operator *where*), projekcji (operator kropki) oraz złączenia są przecież traktowane jako operatory algebraiczne algebry relacji.
  - Tu właśnie jest nieporozumienie. Takie traktowanie jest możliwe wyłącznie przy **ograniczonej funkcjonalności**, oraz po przyjęciu **triku formalnego**.
  - Trik polega na tym, że część semantyki jest przenoszona na poziom metajęzykowy. Operatory te są dodatkowo kwalifikowane **wyrażeniem metajęzykowym**. Np. w wyrażeniu algebry relacyjnej:

$$\sigma_{Zar>1000}(\ Prac\ )$$

operator selekcji σ jest kwalifikowany wyrażeniem metajęzykowym *Zar* >1000. Operator selekcji **nie jest pojedynczym operatorem**, lecz nieskończoną rodziną zawierającą tyle operatorów, ile jest warunków selekcji.

Operatory niealgebraiczne

- Powyższy trik można uważać za uzasadniony w przypadku, gdy wyrażenie metajęzykowe parametryzujące operator jest proste, a jego semantyka jest oczywista. Nie zawsze to jest prawda. Operator selekcji może mieć bardziej złożony warunek selekcji, np.

$$\sigma_{ZarobekNetto(\ Zar\ +\ 100\ )\ >1000}(\ Prac\ )$$

  - Warunek selekcji zawiera operator + oraz wywołuje pewną funkcję *ZarobekNetto*. Wyrażenie metajęzykowe posiada nietrywialną wewnętrzną semantykę, **która jest nieformalna**. Jest to po prostu nieformalny komentarz.
- **Jeżeli jakikolwiek składnik języka nie ma formalnej semantyki, to cały język również nie ma formalnej semantyki.**
  - Mimo podobieństwa wizualnego, w powyższych wyrażeniach nazwy *Prac* oraz *Zar* są ulokowane w dwóch różnych światach.
  - Pierwsza nazwa należy do języka algebry relacji, jest „pierwszej kategorii”, podlega formalnemu traktowaniu.
  - Natomiast druga nazwa należy do metajęzyka algebry relacji, jest „drugiej kategorii”, nie podlega formalnemu traktowaniu.

Operatory niealgebraiczne

- W ten sposób złamana została zasada relatywizmu, zgodnie z którą nazwy nie mogą posiadać fundamentalnie różnej semantyki w zależności od tego, jakiego rodzaju dane oznaczają.
- Ta zasada staje się szczególnie istotna dla języków obiektowych, gdyż obiekty mogą mieć strukturę hierarchiczną, zaś nazwy mogą oznaczać dane na dowolnym poziomie hierarchii obiektów.
- Podobny problem dotyczy operatorów. Operator selekcji σ jest elementem języka tej algebry, należy do „pierwszej kategorii", podczas gdy operator < występuje w metajęzyku, jest „drugiej kategorii".
- Powyższa językowo-meta-językowa schizofrenia podważa poprawność definicji semantyki.
- Podane argumenty są wystarczające aby twierdzić, że tzw. **algebry obiektowe są pseudo-matematyczną bzdurą** (włączając algebry dla XML).

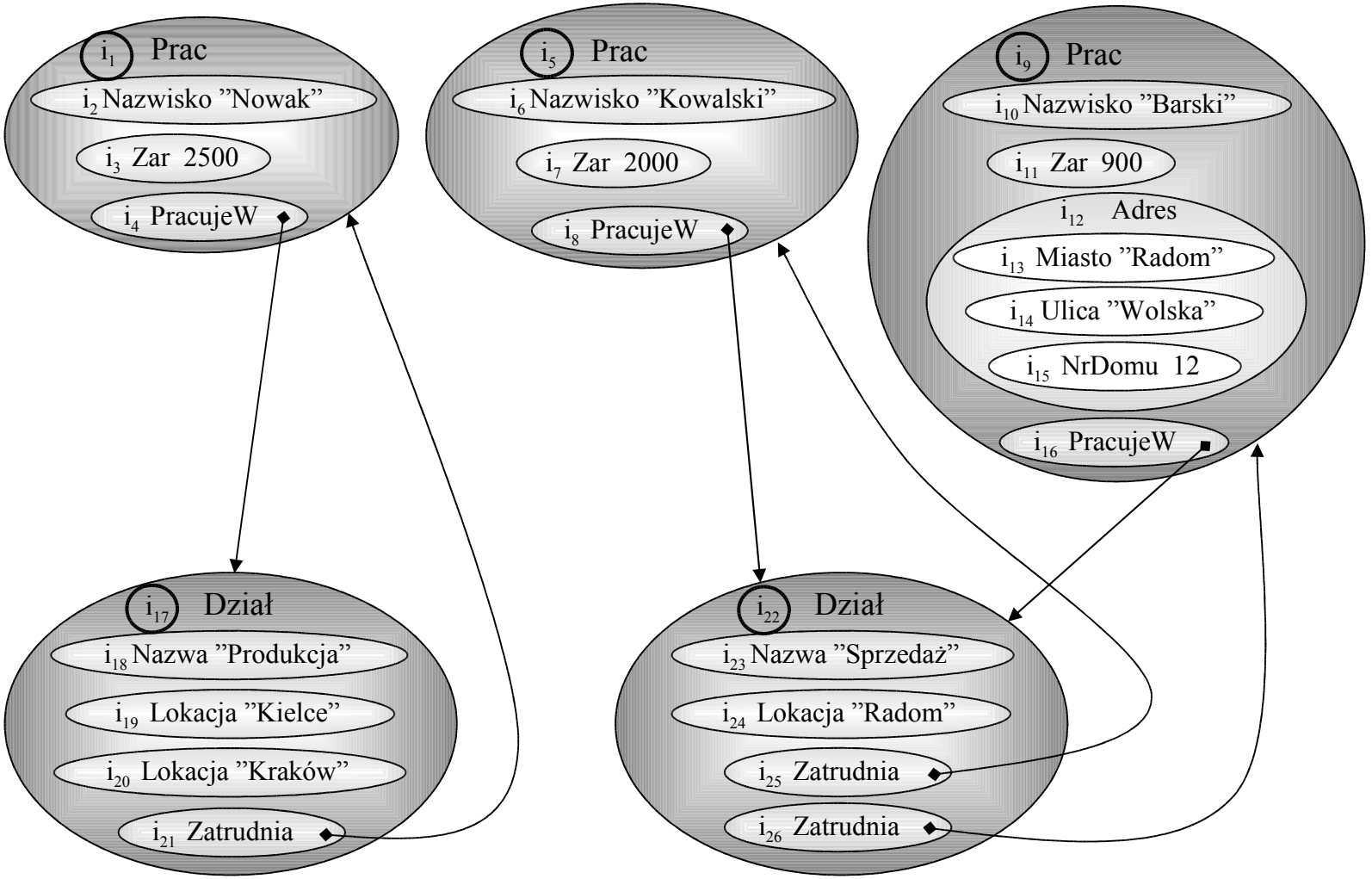
Operatory niealgebraiczne

- W podejściu stosowym dowolne operatory nie są indeksowane wyrażeniami meta-językowymi.
- Koncepcja operatorów niealgebraicznych jest bardzo prosta oraz ma dobrze ugruntowane korzenie w semantyce języków programowania.
- Definicja operatorów niealgebraicznych będzie się nieco różniła w zależności od tego, który modelu składu (M0 - M3) będzie rozpatrywany. Wszystkie te definicje będą bazowały na podanej niżej podstawowej definicji dla modelu M0.

- Aby dokonać ewaluacji zapytania q_1 θ q_2 wykonaj następujące czynności:
  - Dokonaj ewaluacji zapytania q_1. Zapytanie to zwraca wielozbiór elementów.
  - Dla każdego elementu *e* należącego do wyniku q_1 wykonaj następujące czynności:
    - Oblicz wartość funkcji *nested*(*e*). Wynik jest zbiorem binderów.
    - Włóż obliczony zbiór binderów jako nową sekcje na wierzchołek stosu ENVS.
    - Dokonaj ewaluacji zapytania q_2 w tym nowym środowisku.
    - Oblicz wynik cząstkowy dla danego elementu *e* poprzez połączenie *e* z wynikiem zwróconym przez q_2. Funkcja łącząca zależy od operatora θ.
    - Usuń nowo wstawioną górną sekcję ze stosu ENVS.
  - Zsumuj wszystkie wyniki cząstkowe w wynik końcowy. Sposób sumowania *sumuj* (∪) zależy od rodzaju operatora θ.
- Stan stosu środowisk ENVS po zakończeniu ewaluacji jest taki sam, jak przez rozpoczęciem ewaluacji.

```
procedure eval( q : zapytanie )
begin
    .....
    case q jest rozpoznane jako q1 θ q2 :   (* q1 , q2 są zapytaniami, θ jest operatorem niealgebraicznym *)
    begin
        wyniki_pośr: bag of Rezultat;          (* lokalna kolekcja wyników pośrednich *)
        wynik_pośredni: Rezultat;              (* lokalna zmienna na wynik pośredni *)
        wynik_końcowy: Rezultat;               (* lokalna zmienna na wynik końcowy *)
        e: Rezultat;          (* lokalna zmienna na element kolekcji zwracanej przez q1 *)
        wyniki_pośr := ϕ;                      (* zerowanie kolekcji wyników pośrednich *)
        eval( q1 );    (*q1 zwraca kolekcję elementów; wynik q1 na czubku stosu QRES *)
        for each e in top( QRES ) do (* iteracja po wszystkich elementach wyniku q1 *)
        begin
            push( ENVS, nested( e ) );              (* nowa sekcja na stosie środowisk *)
            eval( q2 );                             (* wynik q2 na czubku stosu QRES *)
            wynik_pośredni := połączθ( e, top( QRES ) );  (* połączenie e z wynikiem q2; zależne od θ *)
            wyniki_pośr := wyniki_pośr ∪ { wynik_pośredni };  (* akumulacja wyniku pośredniego *)
            pop( QRES );                            (* usuniecie z QRES wyniku q2 *)
            pop( ENVS );                            (* usuniecie z ENVS nowej sekcji *)
        end;
        wynik_końcowy := sumujθ( wyniki_pośr );  (* zsumowanie wyników pośrednich; zależne od θ *)
        pop( QRES );                                (* usuniecie z QRES wyniku q1 *)
        push( QRES, wynik_końcowy );       (* włożenie na QRES końcowego wyniku *)
    end;
    .....
end;
```

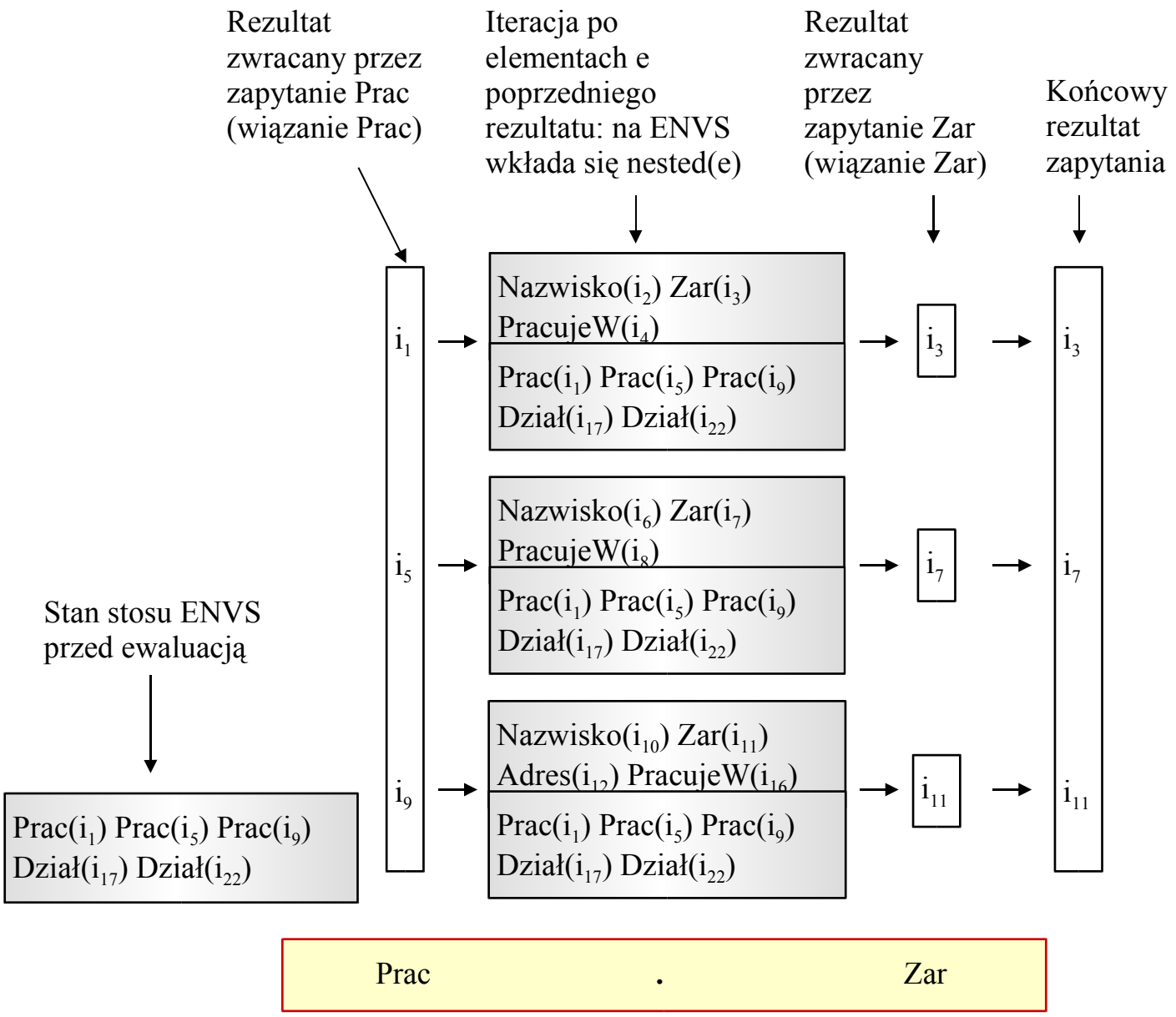
Operator kropka



Operator kropki

- Składnia: q_1 **.** q_2
- Semantyka
  - Dla każdego elementu *e* zwróconego przez q_1, ENVS jest podwyższany o *nested*(*e*)
  - Następnie ewaluowane jest q_2
  - Po ewaluacji q_2 stos ENVS wraca do poprzedniego stanu
  - Ostateczny wynik jest sumą mnogościową wyników q_2
- Objaśnienie funkcji *eval*
  - Funkcja *połącz*: ignoruje *e*; zwraca wynik podzapytania q_2.
  - Funkcja *sumuj*: sumuje (mnogościowo) wszystkie wyniki pośrednie.
- Przykład: *Prac* **.** *Zar*
- Operator kropki przykrywa tzw. wyrażenia ścieżkowe (*path expressions*) w najbardziej uniwersalnej postaci, pozwalając je jednocześnie dowolnie kombinować z innymi operatorami.

Operator kropki



Inne operatory niealgebraiczne

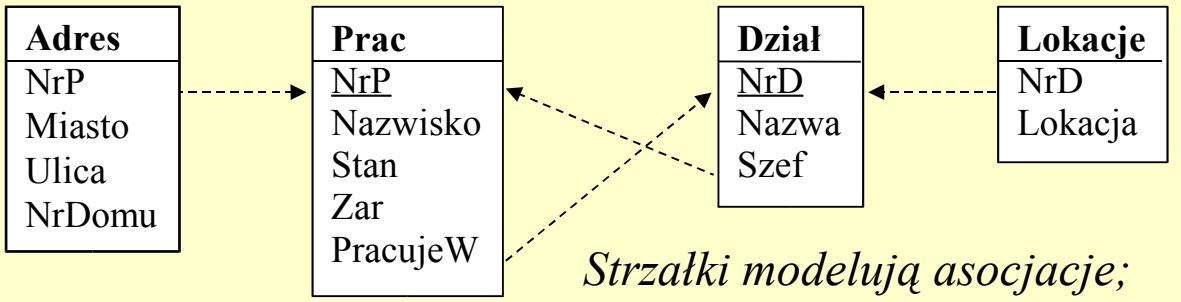
- q1 **join** q2
- q1 **where** q2
- q1 **order** by q2

Prac order by ((PracujeW.Dział.Nazwa), Zarobek)

- $\exists\, q_1 (q_2)$ lub $q_1 \exists\, q_2$
- $\forall\, q_1 (q_2)$ lub $q_1 \forall\, q_2$

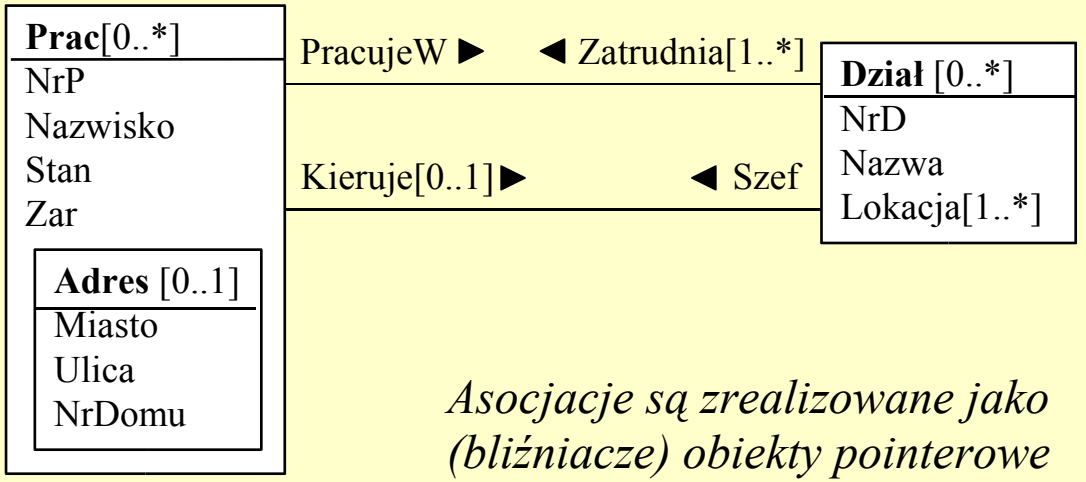
Schematy bazy do przykładów

Schemat relacyjny



Strzałki modelują asocjacje; prowadzą od klucza obcego do głównego

Schemat obiektowy (diagram klas)



Asocjacje są zrealizowane jako (bliźniacze) obiekty pointerowe

Przykłady zapytań

Podaj pełną informację o pracownikach:

Prac

Jest to odpowiednik zapytania SQL: **select** * **from** *Prac.* Wbrew popularnym opiniom, lukier **select** ... **from** ... będziemy uważać za szkodliwy.

Różnice semantyczne: zapytanie SQL zwraca tabelę *Prac*, podczas gdy *Prac* zwraca referencje do obiektów *Prac*. **Zapytania SBQL nigdy nie zwracają obiektów**.

Podaj nazwiska wszystkich pracowników:

Prac **.** *Nazwisko*

Zapytanie jest odpowiednikiem zapytania SQL: **select** *Nazwisko* **from** *Prac.* Zapytanie SQL zwraca jedno-kolumnową tablicę stringów będących nazwiskami, natomiast zapytanie SBQL zwraca tablicę referencji do pod-obiektów *Nazwisko* w obiektach *Prac*. Do tej tablicy można oczywiście zastosować operator dereferencji, który referencje na stringi, ale automatyczna dereferencja prowadzi do straty informacji. Referencje są bardziej uniwersalne niż stringi, gdyż. np. mogą być użyte po lewej stronie operacji podstawienia.

Przykłady zapytań

Podaj średnią liczbę pracowników w działach.

Dla schematu obiektowego:

avg(*Dział* **.** *count*(*Zatrudnia*))

Dla schematu relacyjnego:

avg(*Dział* **.** *count*(*Prac* **where** *NrD* = *PracujeW*))

Analogiczne zdanie w standardzie SQL-89 nie istnieje; zapytanie można wyrazić z pomocą dodatkowej perspektywy.

W standardzie SQL-92 zdanie to można sformułować przy pomocy opcji *group by*. Opcja ta prowadzi do znanej rafy semantycznej, polegającej na tym, że jeżeli pewien dział nie będzie miał ani jednego pracownika, wówczas nie zostanie uwzględniony przy obliczaniu średniej. W SBQL ta rafa nie występuje.

Przykłady zapytań

Dla pracowników zarabiających więcej niż 2000 i pracujących w budynku A podaj nazwisko, stanowisko, nazwę działu i nazwisko szefa działu.

((*Prac* **where** *Zar* > 2000) **join** (*PracujeW* **.** (*Dział* **where** *"budynek A"* ∈ *Lokacja*))) **.** (*Nazwisko*, *Stan*, *Nazwa*, (*Szef*.*Prac*.*Nazwisko*))

Wynikiem będzie kolekcja struktur struct{$i_{Nazwisko1}$, i_{Stan}, i_{Nazwa}, $i_{Nazwisko2}$ }, gdzie każda struktura zawiera cztery referencje.

Czy w każdym dziale jest pracownik zarabiający więcej od swojego szefa?

∀ *Dział* (∃ *Zatrudnia*.*Prac* (*Zar* > *Szef*.*Prac*.*Zar*))

Wynikiem zapytania jest wartość boolowska *prawda* lub *fałsz*. Kwantyfikatory są operatorami niealgebraicznymi, wobec czego (jak w całym SBQL), użycie nazw pomocniczych (czyli „zmiennych związanych kwantyfikatorami") nie jest konieczne. Jeżeli zachodziłaby potrzeba, wówczas takie „zmienne" można byłoby powołać w postaci pomocniczych nazw:

∀ *Dział* **as** *x* (∃ *x*.*Zatrudnia*.*Prac* **as** *y* (*y*.*Zar* > *x*.*Szef*.*Prac*.*Zar*))

Zmuszanie użytkowników do obowiązku stosowania pomocniczych nazw, jak w OQL, jest konsekwencją pseudo-matematycznych koncepcji semantyki.

Materiały

Kazimierz Subieta, “Teoria i konstrukcja obiektowych języków zapytań”

Slajdy:
http://www.ipipan.waw.pl/~subieta/wyklady/Jezyki i srodowiska programowania baz danych JPS 02005/