Eksperyment z semantyką denotacyjną
(odwracając tradycyjną kolej rzeczy)

Prezentacja i książka (PL i GB) do pobrania na
http://www.moznainaczej.com.pl/inzynieria-denotacyjna
http://www.moznainaczej.com.pl/denotational-engineering

Andrzej Blikle
przy współpracy z Piotrem Chrząstowskim-Wachtlem

8 marca 2019

Idea i filozofia metody

Co staram się zrobić?

Zaproponować możliwy sposób podniesienia jakości systemów softwareowych (programów).

JAKOŚĆ PROGRAMU (SYSTEMU):

1. zgodność specyfikacji z oczekiwaniami użytkownika
2. zgodność programu ze specyfikacją

Na początek dla Pascalo-podobnych
języków programowania sekwencyjnego.
Tym zajmowałem się w latach 1970-1990.

Dlaczego się na to porwałem?

Stan rzeczy w przemyśle IT

Cytat z umowy licencyjnej *Producent oprogramowania zwraca uwagę, że w oparciu o bieżący stan techniki, nie jest możliwe wyprodukowanie programu komputerowego w taki sposób, aby bezbłędnie pracował we wszystkich możliwych konfiguracjach. Producent gwarantuje w oparciu o dotychczasowych użytkowników, że do dnia zawarcia niniejszej umowy nie zna żadnych błędów w przekazywanym programowaniu.*

Stan rzeczy w nauce IT

The KeY Book; From Theory to Practice (Springer 2016)
Przez wiele lat pojęcie formalnej weryfikacji było niemalże synonimem weryfikacji funkcjonalnej. W ostatnich latach staje się coraz bardziej jasne, że pełna funkcjonalna weryfikacja programu jest dla prawie wszystkich zastosowań celem iluzorycznym. (…) Prawdziwym wąskim gardłem weryfikacji funkcjonalnej jest nie weryfikacja, ale specyfikacja.

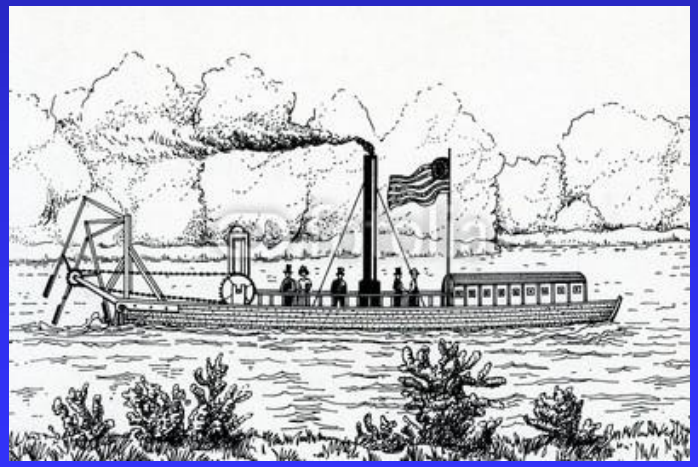
Dlaczego to się nie udawało?

(choć ograniczone próby nadal trwają)

Żeby zbudować logikę programów
trzeba semantykę języka opisać matematycznie.

Dwa historyczne podejścia do matematycznych semantyk języków programowania:

Semantyki operacyjne (VDL)
opisać wirtualną maszynę



Semantyki denotacyjne (VDM)
S : Język → Denotacje
S(P♦Q) = S(P) • S(Q)

Ada i Chill, 1980.

S : AlgSkładni → AlgDenotacji

SEMANTYKA
homomorfizm
algebr wielorodzajowych

Czy dla każdego języka programowania semantyka denotacyjna da się napisać?

Moja hipoteza
Dla znanych mi języków programowania raczej nie da się zbudować (praktycznych) semantyk denotacyjnych.

A z pewnością do tej pory tego nie zrobiono.

Tradycyjne podejście do budowania semantyki denotacyjnej

Najpierw składnia:
jak będziemy wyrażać treści

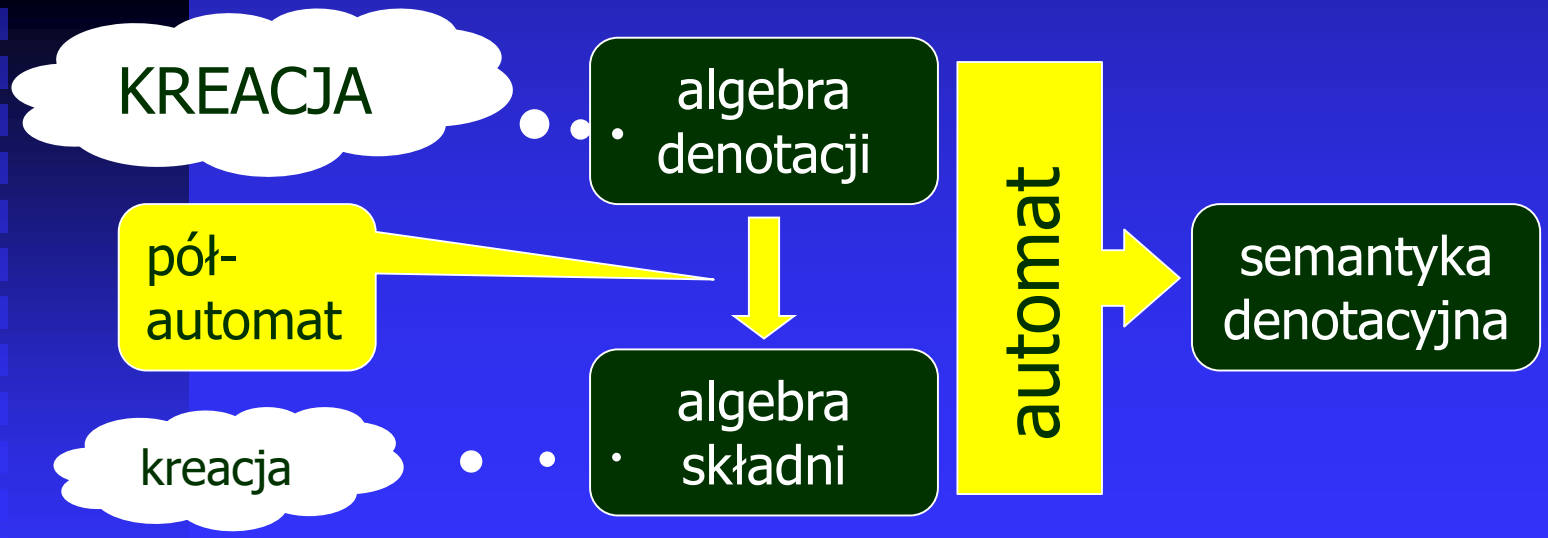
Później denotacje:
jakie treści będziemy wyrażać

Ta kolejność wynikała z przyczyn historycznych: gdy zaczęto myśleć o semantyce, języki już istniały.

Odwróćmy kolejność myślenia i działania

Najpierw opiszmy w metajęzyku świat, który mają reprezentować składowe języka programowania: algebrę denotacji obejmującą programy, instrukcje, deklaracje, wyrażenia, typy, obiekty,...

Później zbudujmy formalną składnię dla języka programowania opisującego ten świat: algebrę składni



Gdy mamy już język z semantyką denotacyjną, można pomyśleć o dowodzeniu poprawności programów.

Czy jednak dowodzenie poprawności programów to dobra droga?

Dwa problemy:

1. Dowód twierdzenia jest zwykle dłuższy od twierdzenia.
2. Programy zwykle nie są poprawne.

Ponownie odwróćmy kolejność myślenia i działania

Matematyk
Najpierw twierdzenie (hipoteza) później dowód.

Inżynier
Najpierw projekt (dowód) później produkt, np. most.

Reguły logiki programów należy zastąpić regułami budowania programów poprawnych

Programowanie walidujące

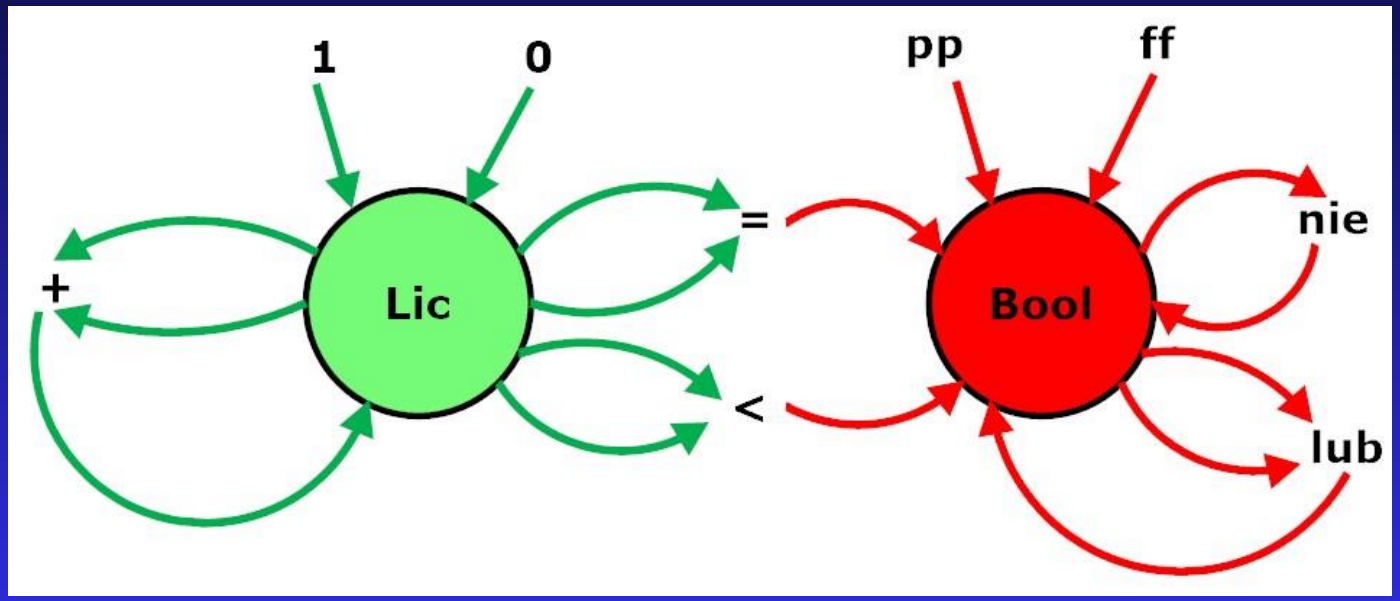
Ogólna koncepcja modelu denotacyjnego

Idee opisane w moich pracach w latach 1971 – 1989
(niektóre z Antonim Mazurkiewiczem i Andrzejem Tarleckim)

NARZĘDZIA MATEMATYCZNE

- teoria punktu stałego w zbiorach częściowo uporządkowanych,
- teoriomnogościowe równania dziedzinowe (bez D.Scott'a),
- trójwartościowy rachunek predykatów,
- algebry wielorodzajowe,
- błędy abstrakcyjne jako mechanizm obsługi błędów

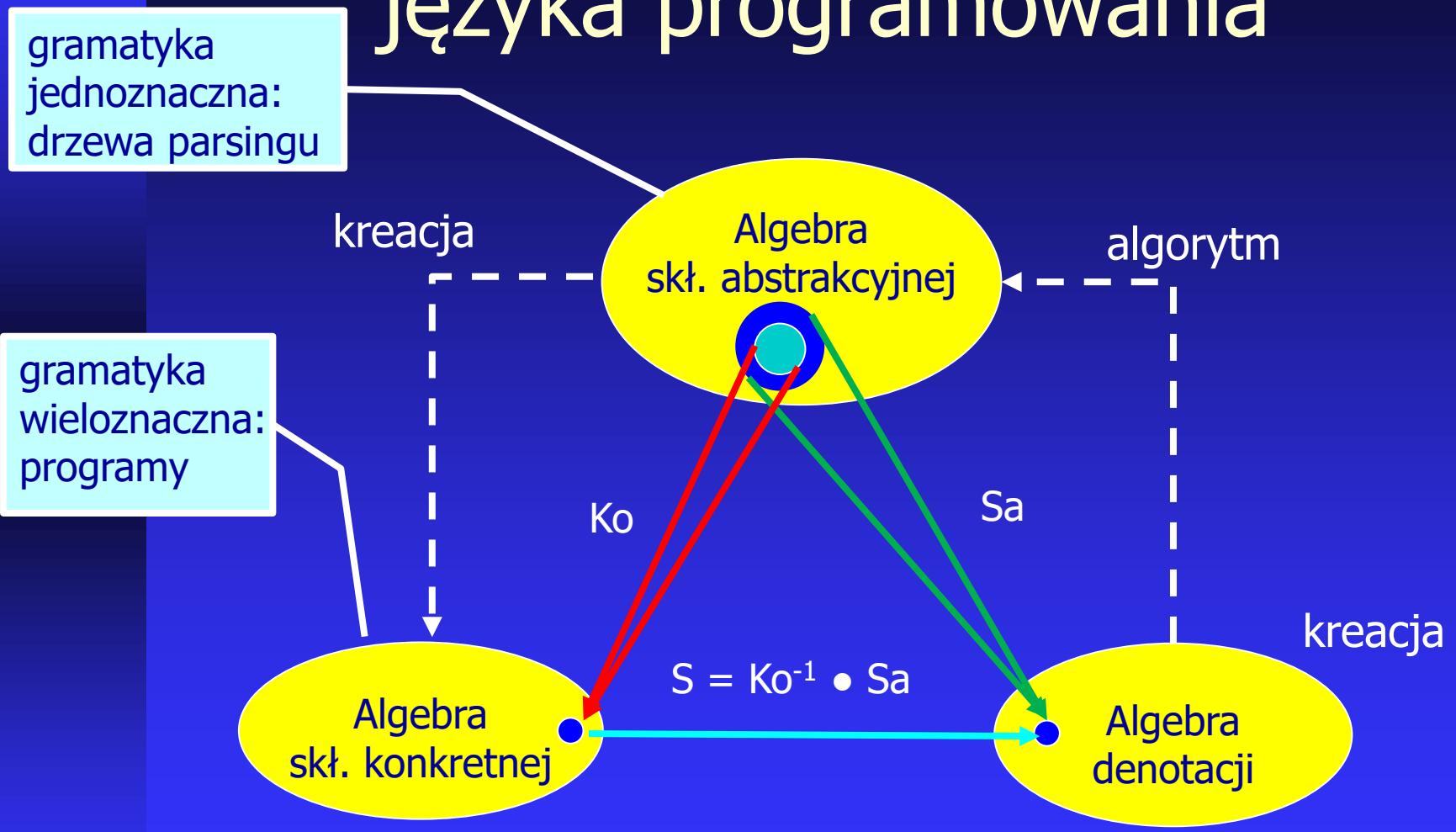
Przykład algebry wielorodzajowej



DWA RODZAJE ELEMENTÓW ALGEBRY:

- liczby, np. rzeczywiste
- wartości boolowskie

Denotacyjny model języka programowania



Jeżeli Ko skleja nie więcej niż Sa, to (jedyny) homomorfizm S istnieje.

Nośniki

Algebra denotacji

Ide = {x, y, z,…}

DenWyr = Stan → Liczba

DenIns = Stan → Stan

Stan = Ide ⟹ Liczba

Konstruktory

zmienna	: Ide		⟼ DenWyr
plus	: DenWyr	x DenWyr	⟼ DenWyr
razy	: DenWyr	x DenWyr	⟼ DenWyr
przypisz	: Ide	x DenWyr	⟼ DenIns
sekwencja	: DenIns	x DenIns	⟼ DenIns

Bardzo uproszczony przykład cz. 1

Oznaczenia:

A → B; f. częściowe

A ⟼ B; f. całkowite

A ⟹ B; f. skończone

AUTOMAT

Algebra (gramatyka) składni abstrakcyjnej

Ide = {x, y, z,…}

Wyr = zm(Ide) | plus(Wyr, Wyr) | razy(Wyr, Wyr)

Ins = przypisz(Ide, Wyr) | złóż(Ins, Ins)

AUTOMAT

Semantyka składni abstrakcyjnej (Sa)

Sid : Ide ⟼ Ide (identycznościowo)

Swy : Wyr ⟼ DenWyr

Sin : Ins ⟼ DenIns

Bardzo uproszczony przykład cz. 2

Algebra (gramatyka) składni abstrakcyjnej

Ide = {x, y, z}
Wyr = zm(Ide) | plus(Wyr, Wyr) | razy(Wyr, Wyr)
Ins = przypisz(Ide, Wyr) | złóż(Ins, Ins)

KREACJA
wspomagana

Algebra (gramatyka) składni konkretnej

Ide = {x, y, z}
Wyr = Ide | (Wyr + Wyr) | (Wyr * Wyr)
Ins = Ide := Wyr | Ins ; Ins

dopuszczalna wieloznaczność

Algebra (gramatyka) składni kolokwialnej

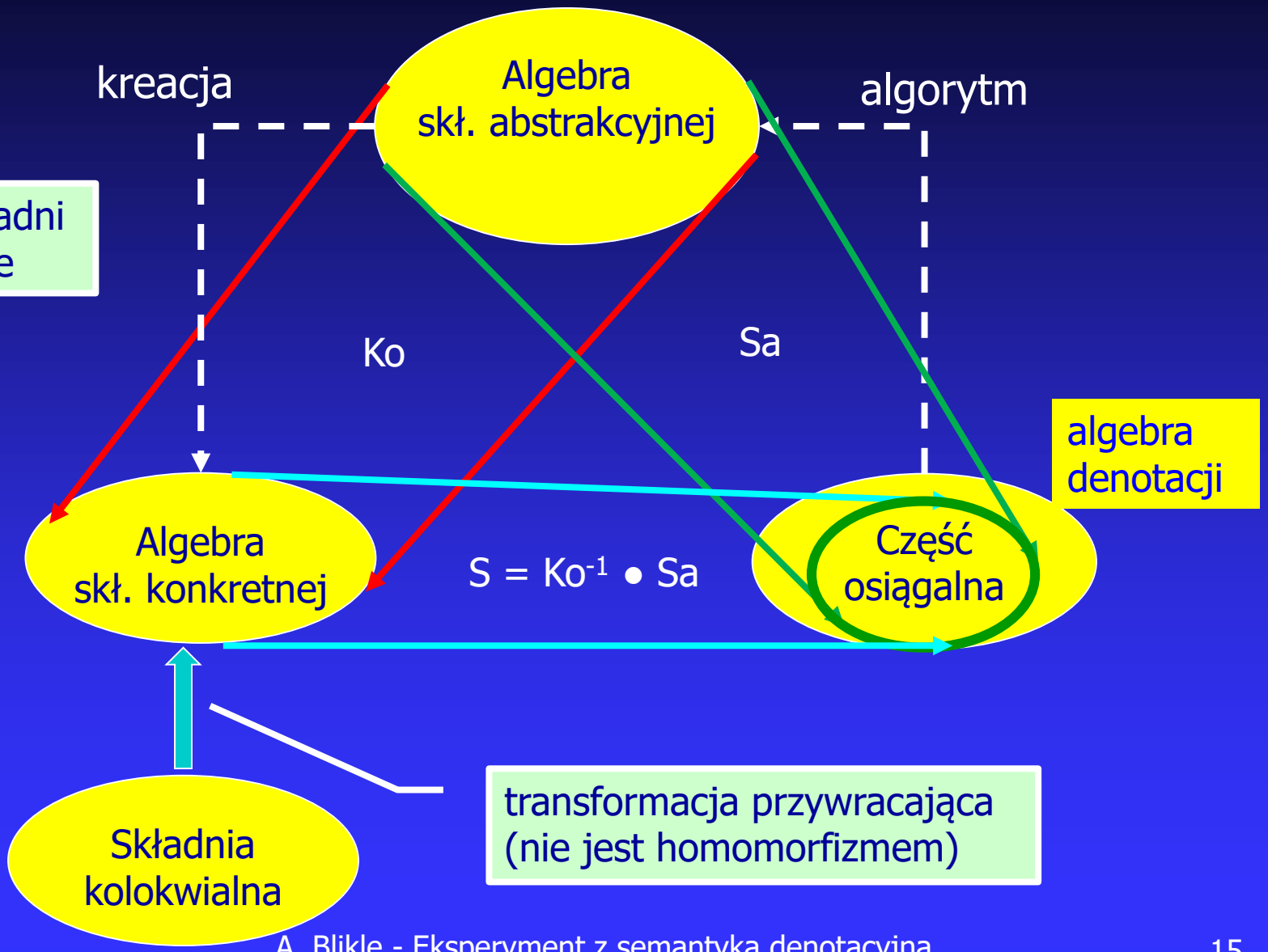
Ide = {x, y, z}
Wyr = Ide | (Wyr + Wyr) | (Wyr * Wyr)
Wyr + Wyr | Wyr * Wyr
Ins = Ide := Wyr | Ins ; Ins

KREACJA
wspomagana

Alg. skł. kolokw. nie jest podobna do alg. denot.
Nie istnieje dla niej semantyka denotacyjna!

niedopuszczalna wieloznaczność

Model ze składnią kolokwialną



Lingua – przykładowy j. wirtualny
służący do wyjaśniania wybranych zastosowań modelu

- wartości boolowskie, liczby, słowa, listy, tablice, rekordy i dowolne ich kombinacje, a także SQL-owe bazy danych,
- 3-wartościowy rachunek predykatów dla wyrażeń boolowskich,
- błędy abstrakcyjne w algebrach denotacji,
- typy strukturalne definiowane przez użytkownika,
- podstawowe konstruktory programów (:=, if-then-else-fi, while-do-od)
- procedury z rekursją i multirekursją,
- zdrowe reguły budowania programów poprawnych oparte na logice Hoare'a z czystą terminację (3-wart. rachunek zdań).

pominięte w prezentacji

Dane i ich równania dziedzinowe

ide : Identyfikator = …

OGÓLNE

boo : Boolowska = {pp, ff}
lic : Liczba = …
sło : Słowo = {'} Alfabet* {'}
lis : Lista = Dana^{c*}
tab : Tablica = Liczba ⟹ Dana
rek : Rekord = Identyfikator ⟹ Dana
dan : Dana = Boolowska | Liczba | Słowo | Lista | Tablica | Rekord

Definiujemy nadzbiory przyszłych dziedzin osiągalnych.

Tu jeszcze nie ma błędów abstrakcyjnych!

SQL

dap : DanProsta = Boolowska | Liczba | Słowo | Data | Czas | … | {Θ}
wie : Wiersz = Identyfikator ⟹ DanProsta
tab : Tabela = Wiersz^{c*}

puste pole tablicy

Korpusy i kompozyty

wspólne korpusy elementów listy/tablicy

OGÓLNE

kor : Korpus = {(‘boolowska’)} | {(‘liczba’)} | {(‘słowo’)} |
{‘L’} x Korpus | (korpusy list)
{‘T’} x Korpus | (*korpusy tablic*)
{‘R’} x (Identyfikator ⟹ Korpus) (*korpusy rekordów*)

SQL

kor : KorProsty = {(‘boolowska’), (‘liczba’), (‘słowo’), (‘data’), (‘czas’),…
kor : KorWiersz = {‘Wq’} x (Identyfikator ⟹ KorProsty)
kor : KorTabela = {‘Tq’} x Wiersz x (Identyfikator ⟹ KorProsty)

KLAN-ko : Korpus ⟼ Zb.Dana
KLAN-ko.('liczba') = Liczba
KLAN-ko.('L', ('liczba')) = Liczba^{c*}

kom : Kompozyt = {(dan, kor) | dan : KLAN-Kr.kor }
kom : KompozytBoo = {(boo, (‘boolowska’)) | boo : {pp, ff}}
kom : KompozytE = Kompozyt | Błąd
kom : KompozytBooE = KompozytBoo | Błąd

błędy są słowami (komunikatami)

Transfery i jarzma

tra : Transfer = KompozytB ⟼ KompozytB
jar : Jarzmo = KompozytB ⟼ KompozytBooB

Jarzma opisują własności kompozytów

PRZYKŁADY WYRAŻEŃ REKORDOWYCH

record.salary - jeżeli argument niesie rekord z atrybutem 'salary', to dana tego atrybutu i jej korpus, a w przeciwnym przypadku komunikat błędu, np. 'oczekiwany-rekord'
record.salary + **record**.bonus

row.pensja + **row**.premia < 7000	- jarzmo rekordów
all-list (**row**.pensja + **row**.premia < 7000)	- jarzmo list rekordów
SMALLINT	- jarzmo liczb
DECIMAL(p,s)	- jarzmo liczb

W Lingua-SQL jarzma opisują więzy integralności poza relacjami podporządkowania dla tabel, które są opisywane w inny sposób.

Typy i wartości

```
typ   : Typ = Korpus x Jarzmo
typ   : TypB = Typ | Błąd

war   : Wartość  = {(dan, (kor, jar)) |
                        dan : KLAN-ko.kor &
                        jar.(dan, kor) = (tt, ('boolowska')) }
```

wartości	są przypisywane identyfikatorom w stanach
typy	są przypisywane identyfikatorom w stanach i są wynikami wykonywania wyrażeń typologicznych
kompozyty	są wynikami wykonywania wyrażeń danologicznych

Typy są składowalne w stanach, ale korpusy i transfery nie są. To decyzja inżynierska, a nie matematyczna konieczność.

Stany i denotacje

STATES

sta : Stan = Środ x Skład - stan
skł : Skład = Wart x (Błąd | {'OK'}) - skład
wrt : Wart = Identyfikator ⟹ Wartość - wartościowanie
śro : Środ = ŚroTyp x ŚroPro - środowisko
srt : ŚroTyp = Identyfikator ⟹ Typ - środowisko typów
srp : ŚroPro = Identyfikator ⟹ Procedura - środowisko proc.
pro : Procedura = ProImp | ProFun - procedura
pro : ProImp = ParAkt x ParAkt ⟼ Skład → Stan - proc imperatywna
prf : FunPro = ParAkt ⟼ Stan → Kompozyt | Błąd - proc. funkcyjna
pak: ParAkt = Identyfikator^{c*} - lista parametrów aktualnych

dla uniknięcia samoaplikacji

DENOTACJE

dwd : DenWyrDan = Stan → KompozytB
dwt : DenWyrTyp = Stan ⟼ TypB
tra : DenWyrTra = Transfer
ddz : DenDekZmi = Stan ⟼ Stan
ddt : DenDefTyp = Stan ⟼ Stan
din : DenIns = Stan → Stan
ddp : DenDekPro = Stan ⟼ Stan

wybrane nośniki algebry denotacji

Konstruktory denotacji

(wybrane przykłady)

zmienna-dan	: Identyfikator	⟼ DenWyrDan
stała-typ	: Identyfikator	⟼ DenWyrTyp
dan-plus	: DenWyrDan	⟼ DenWyrDan
typ-plus	: DenWyrTyp	⟼ DenWyrTyp
wołaj-pro-fun	: Identyfikator x ParAkt	⟼ DenWyrDan
przypisz	: Identyfikator x DenWyrDan	⟼ DenIns
jeżeli	: DenWyrDan x DenIns	⟼ DenIns

Przykład definicji konstruktora

```
zmienna-dan.ide.sta =
   jest-błąd.sta  ➔ błąd.sta
   niech
      (śro, (war, 'OK')) = sta
   war.ide = ?    ➔ 'zmienna-niezadeklarowana'
   niech
      ((dan, kor), jar) = war.ide
   dan = Ω        ➔ 'zmienna-niezainicjalizowana'
   true           ➔ (dan, kor)
```

Lingua-SQL

z lotu ptaka

W Lingua już mamy
(dla odświeżenia pamięci)

KORPUSY W SQL
kor : KorProsty = {(‘boolowska’), (‘liczba’), (‘słowo’), (‘data’), (‘czas’),…
kor : KorWiersz = {‘Wq’}x (Identyfikator ⟹ KorProsty)
kor : KorTabela = {‘Tq’} x Wiersz x (Identyfikator ⟹ KorProsty)

JARZMA W SQL
record.salary + **record**.bonus < 10.000
SMALLINT
DECIMAL(p,s)

WARTOŚCI W SQL
WarWie = {(wie, kor), jar) | …}
WarTab = {(tab, kor), jar) | …}

Dodajemy: grafy podrzędności oraz wartości bazodanowe

grp : GrafPod = Identyfikator x Identyfikator x Identyfikator

dziecko kolumna rodzic

rbd : RekBazDan = Identyfikator ⟹ ValTab - rekord bazodanowy
wbd : WarBda = {(rbd, grp) | rbd spełnia grp} - wartość bazodanowa

Wartości bazodanowe (bazy) są przypisywane identyfikatorom w stanach. Jeden stan może przechowywać wiele baz danych.

Aby działać na bazie danych trzeba ją aktywować, co oznacza, że w bieżącym stanie:

- jej tabele zostaną przypisane identyfikatorom,
- jej graf podrzędności zostanie przypisany identyfikatorowi systemowemu 'graf-rp'

Kolokwialna deklaracja zmiennej tablicowej w SQL

```
create table Employees with
  Name      Varchar(20)   NOT NULL,
  Salary    Number(5)     DEFAULT 0,
  Bonus     Number(4)     DEFAULT 0,
  Dep_Id    Number(3)     REFERENCES Departments,
  CHECK (Bonus < Salary)
ed
```

SCHEMAT SKŁADNI KONKRETNEJ

```
create table Employees as
  table-type wyr_dan with wyr_jar ee
ed;
set reference of Employees at Dep_Id to Departments ei
```

wiersz wartości domyślnych

jarzmo

składna konkretna instrukcji definiującej relację podrzędności tabel

Kolejna zmiana przywracająca

```
create table Employees as
  table-type wyr_dan with wyr_jar ee
ed;
```

```
create table Employees as
 table-type
  expand-row
   expand-row
    expand-row
     row Name val empty-word ee
    by Salary val 0 ee
   by Bonus val 0 ee
  by Dep_Id by empty-number ee
with
 all
  varchar(20)(row.Name) and
  …
  row.Bonus < row.Salary
```

wyr_dan

wyr_jar

Przykłady problemów badawczych

Teoria i inżynieria software'owa:

- modele dla języków skryptowych; HTML, TEX,…
- modele dla języków ze współbieżnością,
- pełen system reguł konstruowania programów poprawnych,
- pełne modele dla języków Lingua-podobnych

Narzędzia wspierające budowniczego języków:

- generator składni abstrakcyjnej z opisu algebry denotacji,
- pół-aut. generator składni konkretnej,
- wspomaganie tworzenia składni kolokwialnej i tr. przywracającej,
- wspomaganie przy tworzeniu opisu semantyki.

Narzędzia wspierające programistę:

- implementacje języków Lingua-podobnych
- wspomaganie wykonywania konstruktorów programów

Eksperymentalne zastosowania, np. do mikroprogramów.

DZIĘKUJĘ ZA UWAGĘ