BRULION

Projekt Loglan: wczoraj, dziś, jutro

Andrzej Salwicki

27 stycznia 2013

E pur si muove.

Wstęp

Zamierzam opisać nowy projekt badawczy – LEM'12. Zapewniam, że projekt jest do wykonania. Może wnieść nowe jakości do programowania. By przekonać Czytelnika o słuszności tych stwierdzeń, przypomnę historię projektu Loglan'82. Następnie, wskażę, na te osiągnięcia Loglanu'82, które są trwałe, lecz pozostają nieznane szerszej publiczności. Warto podkreślić, że Loglan'82 nie mógłby powstać bez "wsadu" prac naukowych. W ich wyniku powstało wiele doktoratów i prac opublikowanych w czasopismach. Większość rozwiązań znanych dziś tylko w Loglanie mogłaby być z pożytkiem włączona do popularnych dzisiaj języków jak: C++, Java, etc.

Na koniec przedstawię problemy badawcze i zadania programistyczne jakie sobie stawiamy w nowym projekcie.

Spis treści

Rozdział 1

Wczoraj

Nieco historii. Jak to się zaczęło?

1.1 Początki

W roku 1977, w Instytucie Maszyn Matematycznych MERA opracowaliśmy język Loglan'77.[SBOM77]
Wynik był na tyle interesujacy, że w r. 1978 podpisano umowę o dzieło: *Zaprojektowanie języka programowania Loglan*[1] *i realizację kompilatora tego języka na maszyny MERA 400.* Dzisiaj umowę taką nazwalibyśmy grantem. Zleceniodawcą było Zjednoczenie MERA - producent minikomputerów. Należy podkreślić zaangażowanie dyrektora zjednoczenia prof. Andrzeja Janickiego i jego wiarę w nasz sukces. Był on promotorem naszych poczynań.
Zleceniobiorcą został Instytut Informatyki UW, a konkretnie Zakład Teorii Obliczeń, którym wtedy kierowałem.

1.2 Zespoły

Byliśmy trochę przerażeni wielkością zadania. Udało się namówic do pracy prawie wszystkich kolegów w Zakładzie. Naszym największym sukcesem było namówienie profesora Antoniego Kreczmara by pokierował pracami nad kompilatorem.
W nieformalny sposób podzieliliśmy się na kilka zespołów:

- Definicja języka Loglan – {A. Salwicki, A. Kreczmar, T. Müldner, W.M. Bartol, H. Oktaba, A. Litwiniuk }

[1] dopiero parę lat później dowiedzieliśmy się, że dr C. Brown obmyślił i ogłosił własny esperanto-podobny język Loglan

- Kompilator – {A. Kreczmar, D. Szczepańska, A. Litwiniuk, M. Lao, W. Nykowski }
- Zespół rozbudowujący środowisko na Merze – {P. Gburzyński, P. Findeisen }
- Zespół wsparcia i programowania w Loglanie –{ T. Müldner, G. Mirkowska, L. Banachowski, A. Szałas, A. Salwicki, U. Petermann, ...}

1.3 Jak się rozliczyliśmy z umowy formalnej?

1. Opublikowaliśmy Raport języka Loglan [B+84]
2. Oddaliśmy kompilator wraz z pełną dokumentacją: Podręcznik użytkownika, pliki źródłowe (1,5MB), etc.,
3. Stworzyliśmy dla komputerów MERA 400: system plików, edytor, assembler, i inne narzędzia.
4. Działaliśmy na rzecz wdrożenia jezyka:
   - zwołaliśmy dwie konferencje międzynarodowe poświęcone Loglanowi (Zaborów 1983 [K+83], Radziejowice 1984),
   - zorganizowaliśmy konferencję szkoleniową PTI, Serock, 1985 [Sal85],
   - wdrażanie w dydaktyce:
     - na politechnikach w Poznaniu i Białymstoku, Loglan był używany i wykładany do mniej więcej 2000 r.,
     - na UW trwało to krócej.

1.4 Kontynuacja w latach 1986-90

W roku 1985 wicedyrektor ds. naukowych Instytutu Informatyki prof. Ludwik Czaja zlecił nam przygotowanie wniosku o resortowy program badawczy. Przygotowaliśmy taki wniosek wciągając do współpracy 26 zespołów z 8 uczelni. Prawie jedna trzecia tematów związana była z Loglanem. W pierwszym roku otrzymaliśmy 47 mln zł. w ostatnim było to już ponad 4 mld (inflacja była olbrzymia, ale i tak finansowa nagroda za wyniki wzrosła wielokrotnie). Ciekawe wyniki przyniosły prace kolegów z Uniwersytetu Śląskiego (symulacja ruchu pociągów w węźle górnośląskim, kompilator z L-kodu do języka C, i in.), Politechniki Poznańskiej(warto wspomnieć, że w

Politechnice wykorzystywano Loglan w dydaktyce przez ponad 15 lat). Najważniejsze, że bardzo wielu uczestników tego programu uzyskało kolejne stopnie naukowe i tytuły naukowe, co było zresztą założeniem prof Czaji i moim.

1.5 Współpraca międzynarodowa

- Institut für Informatik, Universität zu Kiel,
  Wiele zawdzięczamy profesorowi Hansowi Langmaackowi:
  - wspólna praca nad semantyką dostępu do wielkosci nielokalnych, zakończona znalezieniem algorytmu obliczającego taką permutację wskaźników w wektorze Display, która zapewnia poprawny statyczny dostęp do wielkości nielokalnych przy zmianie poziomu dziedziczenia. [KKLS84, KLKW86],
  - przeniesienie Loglanu na mainframe'y IBM (m.c. Siemens) i Riad, 1985
  - wyznaczanie bezpośrednich superklas w Javie (2010-2011) z wykorzystaniem wyników prac nad Loglanem, 2005 – 2011. biblio
- IASI CNR Roma,
  Przeniesienie Loglanu na komputery VAX/VMS, 1986. Bardzo dobre i częste kontakty z dr Gianną Cioni.
- Universita "La Sapienza" Roma,
  zastosowanie Loglanu w algebrze komputerowej,
  wykłady Loglanu.
  Wieloletnia współpraca z prof. A. Miolą, prof. M. Temperini i in.
- i in. (uniwersytety w Tübingen, Bordeaux, Caen, ...)

Dzisiaj 8 osób z zespołu Loglanu to profesorowie w kraju (5) i zagranicą (3), dwie osoby pracują w firmach software'owych w USA, jedna w administracji państwowej. Prof. Antoni Kreczmar zmarł w 1996 – jego wyniki są jeszcze niedoceniane – zrobił bardzo wiele – gdyby żył, mógłby zdziałać jeszcze więcej.

1.6 Wkład doktorantów

1. O. Świda: VLP - wieloprocesorowy, rozproszony, wirtualny komputer loglanowski i środowisko,

2. A. Szałas: komunikacja procesów przez przerwania, ta koncepcja weszła do Loglanu'88 1983,
3. D. Szczepańska: system zgłaszania wyjątków i ich obsługi, 1982(wdrożenie), 1990 doktorat,
4. P. Gburzyński: System automatycznego dowodzenia twierdzeń zaprogramowany w Loglanie, realizacja koncepcji prof. M. Bibela, 1982 (Bibel miał swoją realizację 2 lata później)
5. H. Oktaba: formalizacja systemu zarządzania pamięcią 1982,
6. W.M. Bartol: opis systemu współprogramów, 1983
7. U. Petermann: komunikacja procesów przez przerwania 1987
8. A. Litwiniuk - autor generatora kodu w kompilatorze, 1988
   wraz z P. Gburzyńskim przenieśli Loglan na mainframe'y, później na maszyny VAX/VMS

1.7 Wkład studentów

- kompilator: parser – W. Nykowski 1981
- koncepcja i realizacja obcego wołania metod – B. Ciesielski 1988
- debugger – T. Przytycka 1984
- przeniesienie kompilatora
  - na IBM PC/AT – M. Benke i G. Grudziński 1985
  - do systemu Unix – P. Susicki 1989
  - na komputer Atari – Sebastien Bernard, Université de Pau 1992
  - z 16 bitowego DOS na 32 system Windows95 – F. Pataud 1994
- środowisko Lotek – grupa studentów UW 1983
- klasy dla grafiki i obsługi myszki
  - XIIUWGRaph studenci z Universite de Pau, J. Larrieu, P. Becourt

  - IIUWGraph dla 32 bitowych PC – F. Pataud,

- wtyczka Loglanowska do Eclipse - A. Chwedoruk 2005
- kompilacja na Linuxa - A. Adamski, 2011
- wersja na platformę Windows: 7, XP - T.Reda 2011

Ta lista nie jest pełna.

Rozdział 2

Dziś

Dzisiaj możemy się pochwalić kilkoma osiągnięciami, jakich pomimo upływu lat, nie powtórzono dotąd w żadnym innym języku programowania obiektowego.

2.1 Pięć ważniejszych osiągnięć

Język Loglan powstał w wyniku dyskusji i analiz. Rozwiązaliśmy kilka problemów dotyczących semantyki. W efekcie otrzymano język o wielu ciekawych cechach umożliwiających tworzenie niebanalnych algorytmów i struktur danych. Poniżej wyliczam te cechy, które do dziś pozytywnie odróżniają Loglan'82 od powstałych później języków programowania obiektowego C++, Javy, ...

- System bezpiecznego zarządzania pamięcią obiektów.
- Protokół "*obcego wołania metod*" (ang. alien call) w procesach.
- Poprawny system współprogramów
- Klasy wewnętrzne i dziedziczenie ukośne.
- Łączenie maszyn wirtualnych w wirtualny wieloprocesorowy komputer.

Ponadto: można przekazywać procedury i funkcje - wymóg specyfikacji, mozna przekazywać typy czyli nazwy klas,

Połączenie zagnieżdzania klas i dziedziczenia stwarza wiele możliwości: klasa moze opisywać strukturę wielosortową np. struktura elementów i stosów, geometria (punkty , linie, okręgi)

Wzorce znane dzisiaj znaliśmy w latach 80.

2.2 Bezpieczne usuwanie obiektów

Prof. Antoni Kreczmar (1945-1996) zaprojektował i zrealizował kompletny i bezpieczny system zarządzania pamięcia obiektów, czyli stertą (ang. heap), zob. [CK84].
Znane dziś języki programowania cechują się albo wysokim ryzykiem związanym z usuwaniem obiektów. [1] albo niemożnością bezpośredniego usuwania wskazanych obiektów [Java].

2.2.1 Aksjomat pamięci obiektowej

W każdym języku programowania obiektowego jego maszyna wirtualna (lub inaczej: running system) musi zapewniać prawdziwość następującego twierdzenia:

Twierdzenie 1. *Jeśli zmienna x jest zadeklarowana jako zmienna typu T to wartością tej zmiennej jest albo obiekt podklasy klasy T albo none. Co można wysłowić tak:*

$$type(x) = T \Rightarrow (x \ \texttt{in}\ T \vee x = none)$$

lub w terminach Javy:

$$type(x) = T \Rightarrow (x \ \texttt{instanceof}\ T \vee x = null)$$

Przypomnijmy definicję binarnej relacji być podklasą w zbiorze klas danego programu:

Definicja 1. *Relacja klasa T* jest podklasą *klasy Q, jest najmniejsza relacją taką że:*

a) *Klasa T jest* podklasą *klasy T.*

b) Jeśli *klasa Q rozszerza (`extends`) klasę R,* oraz *R jest podklasą klasy T,* to *Q jest podklasą klasy T.*

oraz definicje pary relacji **is** oraz **in** jakie zachodzą pomiędzy obiektami klas i klasami:

Definicja 2. *Niech T oraz Q będą klasami, a o obiektem,*

is) *new T(params)* ***is*** *T,*

in_1) jeśli *o* ***is*** *T,* to *o* ***in*** *T,*

in_2) jeśli *o* ***in*** *T* oraz *Q jest podklasą klasy T* to *o* ***in*** *Q.*

[1] por. niebezpieczeństwo wystapienia zjawiska wiszących referencji (ang. dangling reference problem)[C++, Pascal, ...]

2.2.2 Zaśmiecanie pamięci

Z *zaśmiecaniem pamięci* (lepiej znanym pod nazwą, *wyciek pamięci*) mamy do czynienia gdy podczas wykonywania programu rośnie liczba niepotrzebnych już obiektów. Zjawisko zaśmiecania pamięci to błąd, czasem jego konsekwencje są bardzo poważne.
Co robić?

A) Jedne języki programowania zezwalają na usuwanie obiektu x (por. *free(x)* w Pascalu, *delete(x)* w C++, ...), jednak wiąże się z tym ryzyko pojawienia się wiszących referencji (*ang.* dangling references).

B) Inne języki (Java) nie dopuszczają takich instrukcji, tłumacząc, że odśmiecacz *gc*() (ang. garbage collector) usunie śmieci.

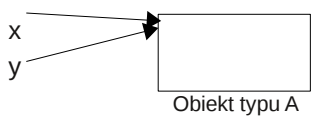
C) Język Loglan'82 ma

- odśmiecanie i
- usuwanie obiektów,
- a co najważniejsze, usuwanie obiektów jest bezpieczne.

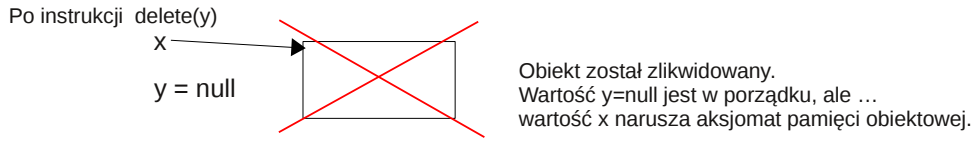
Błąd twórców Javy polega na tym, że zakładają, że obiekty niedostępne czyli *śmieci* i obiekty niepotrzebne to to samo, ale to nie jest prawda.
W związku z tym nie zawsze usuwanie obiektów niedostępnych zapobiega wyciekowi pamięci.
Natomiast programista może wiedzieć, o pewnym obiekcie, że choć jest dostępny (tj. nie jest śmieciem), to nie będzie używany w dalszym ciagu obliczeń (czyli jest niepotrzebny). Warto by się go pozbyć.
W tej sytuacji – w Javie – programista **musi sam przekształcić niepotrzebny obiekt** *o* **w śmieć** usuwając wszystkie referencje do niego. A to nie jest ani łatwe, ani wolne od ryzyka.

Przy okazji: Odśmiecacz (ang. garbage collector) działa w czasie proporcjonalnym do rozmiaru pamięci. Ten rozmiar wzrósł ponad 1000 razy od czasu wprowadzenia Javy! w 1995.

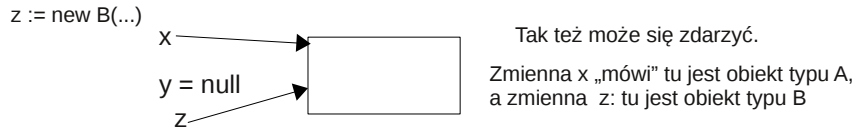
2.2.3 Wiszące referencje



Częsta sytuacja



Pogwałcenie aksjomatu pamięci obiektowej! Zmienna x jest wiszącą referencją do nieistniejącego już obiektu!



Sprzeczność!

Przypomnijmy podstawowy fakt:
Tw. Nie istnieje algorytm wykrywania wiszących referencji!

2.2.4 Bezpieczeństwo

W systemie Kreczmara, zob. [MS87] pp. 328-341, zapewniono prawdziwość następującej formuły {schematu formuł}:

Niezmiennik systemu Loglan. *Maszyna wirtualna Loglanu (inaczej mówiąc running system) zapewnia prawdziwość każdej formuły o nastepującym schemacie:*

$$\boxed{(x_1 = x_2 = \cdots = x_n \neq none) \Rightarrow [kill(x_i)](x_1 = x_2 = \cdots = x_n = none).}$$

Formuła ta tłumaczy się w ten sposób: jeśli jakiś obiekt o jest wartością n zmiennych $x_1, x_2, \ldots, x_n$ to po wykonaniu instrukcji $kill(x_i)$, gdzie $1 \leq i \leq n$, wszystkie te zmienne przyjmują wartość *none* (ponadto, sam obiekt o zostaje zniszczony, a obszar pamięci zajmowany dotąd przez ten obiekt zostaje zwolniony i można go użyć do innych celów). [2]
Zauważ! Koszt tej operacji kill jest **stały**, niezależny od liczby n zmiennych wskazujących na usuwany obiekt.

Podsystem zarządzania obiektami w running systemie – maszynie wirtualnej Loglanu'82 jest kompletny i obejmuje:

- tworzenie obiektu i wirtualnej referencji do niego,

[2] Warto pamiętać, że w takiej sytuacji próba wykorzystania metody m tego obiektu (np. call x_2.m) lub bezpośredniego dostępu do pewnego atrybutu tego obiektu (np. x_3.z) zostanie wykryta i zakończy się podniesieniem sygnału błędu.

- instrukcja przypisania ...
- sprawdzanie czy wartością zmiennej jest none i ew. sygnalizacja błędu "`reference to none`"
- usuwanie obiektu – instrukcja kill,
- klonowanie obiektu – instrukcja copy,
- zarządzanie pamięcią:
  a) szukanie miejsca na nowy obiekt:
     a1) na liście zwolnionych miejsc o rozmiarze k,
     a2) w polu wolnej pamięci ,
  b) kompresowanie pamięci obiektowej – sterty, gdy nie ma wolnego miejsca,
  c) uruchamianie odśmiecacza – ang. *garbage collector*, gdy kompresja nie dała pożądanego wyniku.

zrób rysunek diagram systemu

O ile wiem to w żadnym innym języku programowania nie osiągnięto taniej, bezpiecznej dealokacji obiektów uznanych za niepotrzebne. W języku C++ i innych można usunąć obiekt, ale mogą przy tym powstać *wiszące referencje* – błąd groźny i bardzo trudny do zlokalizowania (por. wprowadzenia do Moduli 3[CDG+89] i do Javy[GM95]). W Javie instrukcja usuwania obiektów jest niedozwolona, autorzy języka twierdzą, że odśmiecacz wykona zadanie usuwania śmieci. Zgoda, jak jednak zmienić statut obiektu o z "obiekt niepotrzebny' na 'śmieć' czyli obiekt niedostępny? Jedyny sposób to nadać wszystkim zmiennym wskazującym na juz niepotrzebny obiekt o wartość null. Programista musi więc mieć w pamięci wszystkie zmienne $z_1, z_2, ..., z_n$ które wskazują na obiekt i wykonać po kolei instrukcje $z_1 = null$; $z_2 = null$; ... $z_n := null$. Jeśli niektóre wskaźniki do obiektu znajdują się w strukturach danych typu lista, drzewo etc. to sprawa jest beznadziejna. Prawie na pewno dojdzie do tzw. 'wycieku pamięci', tj. do sytuacji w której przybywa niepotrzebnych już obiektów i zaczyna brakować miejsca na nowe obiekty. Kolejna uwaga: operacja odśmiecania jest wykonywana w czasie proporcjonalnym do rozmiaru pamięci, a ten wzrósł w ciągu kilkunastu lat od powstania Javy tysiąckrotnie.

2.3 Klasy wewnętrzne i dziedziczenie ukośne

W języku SIMULA67 – matce wszystkich języków obiektowych – klasy mogą być zagnieżdżane. Ale dziedziczyć możemy tylko z klasy, która jest bratem danej

klasy(to jest pewne uproszczenie), nie można dziedziczyć z klasy która jest wujem klasy dziedziczącej. Dziedziczenie w SIMULI67 jest więc *poziome*. W konsekwencji, język SIMULA nie dopuszcza do stworzenia biblioteki klas [DMN70].
W Loglanie'82 zgodziliśmy się na dziedziczenie z klasy, która jest widoczna, np. jest zadeklarowana na wyższym poziomie zagnieżdżania klas. W związku z tym trzeba było rozwiązać kilka problemów:

- czy można zachować mechanizm dostępu do wielkości nielokalnych znany jako Display Vector?

- w jaki sposób wyznaczać klasę z której dana klasa ma dziedziczyć? Zauważ, że teraz w programie może wystąpić wiele klas o tej samej nazwie.

Problemy te udało sie rozwiazać zob. [BKLO83, KKLS84, KLKW86].
W Javie drugi z tych problemów jest dalece niebanalny – opublikowaliśmy poprawny i kompletny algorytm rozwiązujący ten problem.[LSW09, LSW08]

2.4 Współprogramy, ang coroutines

Ten mechanizm jest znany od prawie 50 lat. Pierwotnie termin coroutine oznaczał tylko nieco więcej niz subroutine znane z Fortranu. W tym rozumieniu został rozpropagowany przez Donalda Knutha [**?**].

Nowe znaczenie i nowe możliwości współprogramów zostały zrealizowane w SIMULI67 i w Loglanie'82. PROBLEMY... Został nieco zapomniany i dopiero od niedawna zyskuje na popularności. Obliczenia z współprogramami możemy określić jako obliczenia quasi-współbieżne.

Współprogramy w SIMULI oceniono bardzo wysoko, m. in. D. Knuth [KR03]. Ale ich opis nie był całkiem klarowny, a A. Wang wykazał, że zawiera on sprzeczność [Wan82]. Wynika to z faktu, że w Simuli67 system współprogramów w niezbyt precyzyjny sposób powiązano z blokiem prefiksowanym.

W Loglanie uproszczono definicję współprogramu. Obiekt współprogramu zawiera quasi-wątek, lub inaczej włókno (ang. fiber). W systemie quasi-współbieżnym conajwyżej jeden quasi-wątek jest aktywny. Instrukcje $attach(x)$ przenoszą sterowanie do łańcucha dynamicznego rekordów aktywacji metod zaczynającego się w obiekcie współprogramu x. Zauważmy, że dzisiaj w wielu językach programowania stosowana jest bezparametrowa instrukcja yield(),

w tych językach nie wiadomo, który współprogram zostanie wznowiony, decyzja należy do systemu, nie do programisty.

Pożytki płynące z wykorzystywania współprogramów zilustrujemy przykładem (jednym z wielu, w których współprogramy okazują się pożyteczne).

2.4.1 Łączenie drzew binarnych poszukiwań

Zadanie polega na połączeniu zawartości pewnych drzew binarnych poszukiwań w ciąg niemalejący.
Pierwszy pomysł jaki się nasuwa to powtarzanie

```
dopóki choć jedno drzewo jest niepuste
powtarzaj
    dla każdego drzewa
        znajdź element najmniejszy w tym drzewie
            wydrukuj najmniejszy z tych elementów
        usuń ten element z jego drzewa
    koniec dla
zakończ powtarzanie
```

Koszt takiego algorytmu jest za duży. Każdy element tych drzew będzie oglądany wielokrotnie.
Węzły drzewa BST są obiektami klasy node:

```
unit node: class(value: integer);
    var lewy, prawy: node;
    unit insert: procedure(val: integer) ... end insert;
    unit ismember: function(val: integer): Boolean; ... end ismember;
    unit delete: procedure(val:integer); ... end delete;
end node;
```

Dodajmy procedurę traverse

```
unit traverse: procedure(n: node);
begin
    if n =/= none then
        call traverse(n.lewy); drukuj(n.value); call traverse(n.prawy)
    endif
end traverse
```

Pozornie nie wnosi to nic nowego
Ole-Johan Dahl i Arne Wang zaproponowali użycie współprogramów. Dla

każdego drzewa d tworzymy współprogram W_d wyposażony w metodę *traverse*, której zadaniem jest odwiedzenie drzewa w porządku inorder.

```
unit W : coroutine(root: node);
    var val: integer;
    unit T: procedure(y : node);
    begin
        if y =/= none then
            call T(y.left); val := y.val; detach; call T(y.right);
        fi
    end T;
begin (* konstruktor dla W jest pusty *)
    return; (* tu rozpoczynamy po uaktywnieniu przez attach() *)
    call T(root); (* a gdyby tu wywołano traverse ...?*)
    val := Maximal;
end W;
```

Patrzmy jak zachowa się układ dwu współprogramów, Współprogramy B i A razem wydrukują kolejne elementy od najmniejszego do największego elementu zawartego w drzewie rt. Kazdy element tego drzewa jest odwiedzany tylko raz. Zostawiamy jako ćwiczenie stworzenie tablicy współprogramów typu CA, po jednym dla każdego drzewa. Po wydrukowaniu elementu najmniejszego spomiędzy dotychczas nieodwiedzonych, należy wznowić przy pomocy instrukcji attach ten współprogram, który dostarczył najmniejszy element i urządzic "dogrywkę" by znaleźć kolejny element najmniejszy.

```
program test;
  var A: CA, B: CB, kolejny, n: integer, rt: node;
  unit CA: coroutine(n: node);
    unit T: procedure(y: node);
    begin
      if y<>none  then
         call T(y.left);
         kolejny := y.val; detach;
        call T(y.right)
      fi
    end T;
  begin return;call T(n);
  end CA;

  unit CB: coroutine;
  begin
     return;
    do
       attach(A);
       write(kolejny);
     od
  end CB

begin
   rt := zbudujBST;  A:= new CA(rt); B :=new CB; attach(B);
end
```

Dzięki użyciu współprogramów można napisać program rozwiązujący nasze zadanie w czasie proporcjonalnym do liczby elementów w drzewach BST, każdy element będzie oglądany tylko raz. Nie wiem czy stosując tylko operacje insert, ismember, delete i traverse można rozwiazać nasze zadanie w czasie liniowym.

2.5 Alien call

Protokół współpracy procesów tj. wątków polega na tym, że jeden proces Y (proces *wzywający*) wzywa drugi proces X (proces *wzywany*) do wykonania jego metody m z parametrami aktualnymi dostarczonymi przez proces wzywający. Proces wzywany może się zgadzać (enable m) lub nie (disable m) na przyjęcie takiego zlecenia. Proces wzywany może oczekiwać na nadejście takiego zlecenia, jeśli wykona instrukcję accept m. Mamy wtedy do czynienia z synchronicznym wykonaniem pary instrukcji

$$\langle \underbrace{\textbf{call}\ X.m(arg_1, ..., arg_n)}_{\text{w procesie } Y}, \underbrace{\textbf{accept}\ m}_{\text{w procesie } X} \rangle.$$

Jeśli proces X wykonał instrukcję enable m, to nadejście instrukcji call X.m(...) ze strony innego procesu jest traktowane jako przerwanie na chwilę czynności procesu X.

Warto przypomnieć, że jest to wynalazek Bolesława Ciesielskiego [Cie88].

2.5.1 Klasy, współprogramy, procesy

W Loglanie mamy trzy rodzaje modułów (ang. unit):

- klasy – **unit** C: **class** ... **end** C;
- wspólprogramy – **unit** W: **coroutine** ... **end** W;
- procesy – **unit** P: **process** ... **end** P;

Operacja **new** powołuje do życia odpowiednio:obiekty klas, obiekty współprogramów i obiekty procesów (inaczej obiekty aktywne). Scenariusze tych obiektów są istotnie różne:
• Obiekt klasy po utworzeniu pozostaje pasywny.
• Obiekt procesu po utworzeniu jest w stanie *pasywny*, lecz można go uaktywnić wykonując polecenie $resume(x)$.

$$(Active = S \& Passive = Q) \Rightarrow [resume(x)](Active = S \cup \{x\} \& Passive = Q \setminus \{x\})$$

• Obiekty współprogramów mogą sobie przekazywać aktywność (tj. procesor) wykonując instrukcję $attach(x)$.
Niezmiennie $card(ActiveCoroutines) = 1$
W trakcie obliczeń programu loglanowskiego może powstać wiele kooperujących obiektów procesów. Obiekty te mogą być alokowane i wykonywane na jednym procesorze loglanowskim (daje to obliczenia współbieżne), bądź na wielu procesorach połączonych siecią (obliczenia rozproszone), bądź też inną obmyśloną miksturę tych dwu klas obliczeń.

Każdy obiekt procesu może rządzić wieloma obiektami współprogramów – a więc może zarządzać obliczeniami quasi-współbieżnymi.

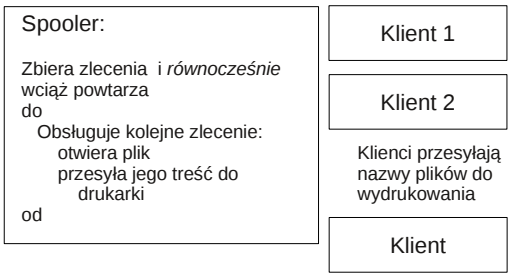
Każdy obiekt może tworzyć i zarządzać obiektami klas.

2.5.2 Wywołanie metody w obiekcie vs. obce wywolanie metody

Niech o będzie obiektem jakiejś klasy. Instrukcja
call $o.meth(params)$
powoduje, że program (proces) zaczyna wykonywać metodę $meth$ z obiektu o. To jest wywołanie metody $meth$ w obiekcie o.

Jeśli obiekt p został utworzony na podstawie pewnego procesu, tj. gdy p jest obiektem aktywnym, to tak samo wyglądająca instrukcja wykonywana w aktywnym obiekcie q
call $p.mtd(parametry)$
ma zupełnie inne znaczenie – nieformalnie możemy ją opisać tak: obiekt aktywny q zwraca się do obiektu aktywnego p z prośbą o wykonanie metody mtd z dostarczonymi przez obiekt q parametrami $parametry$.
To jest *obce wywołanie* metody mtd. Obiekt wzywany q może się zgodzić na przyjęcie takiej prośby wykonując instrukcję accept mtd bądź enable mtd lub ją zablokować wykonując instrukcję disable mtd.

Przykład 1. *(Spooler)*
Pewna liczba klientów wysyła od czasu do czasu zlecenia wydrukowania pliku do serwera.



Serwer o nazwie spooler ma dwa zadania:
a) przyjmować zlecenia,
b) obsługiwać je w kolejności zgłoszeń.

Spooler ma kolejkę Q i metodę *rejestruj*. Klienci rejestrują swoje zgłoszenia. Wątek spoolera (tj. program pomiędzy begin i end spooler) nie zajmuje się rejestrowaniem zadań. Robią to klienci prosząc spoolera o wykonanie procedury rejestruj.

```
unit spooler: process ...
  var Q: kolejka;
  rejestruj: procedure(f: string);
     Wstaw f  do kolejki Q
  end rejestruj
begin
    Weź pierwszego z kolejki Q;
    Obsłuż
    ...
    ...
    i na drukarkę
end Spooler;
```

```
Kolejka: class;
```

Na poniższym obrazku widać, że może dojść do konfliktu.

```
unit spooler: process ...
  var Q: kolejka;
   rejestruj: procedure(f: string);
      Wstaw f  do kolejki Q
   end rejestruj
begin
    Weź pierwszego z kolejki Q;
    Obsłuż
    ...
    ...
    i na drukarkę
end Spooler;
```

```
Kolejka: class;
```

Operacje *wstaw* i *weź* grożą konfliktem.

Zlecenia Q.*wstaw(...)* mogą nadchodzić (od klientów) przez cały czas.

Instrukcja disable blokuje wywołania procedury rejestruj na czas wyłaniania

pierwszego elementu z kolejki Q.

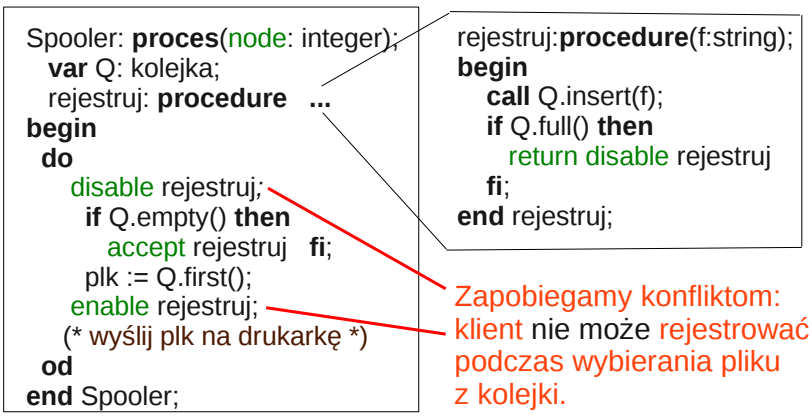
```
Spooler: proces(node: integer);
  var Q: kolejka;
  rejestruj: procedure(f: string);
begin
 do
    disable rejestruj;
      if Q.empty() then
        accept rejestruj   fi;
      plk := Q.first();
    enable rejestruj;
   (* wyślij plk na drukarkę *)
 od
end Spooler;
```

```
rejestruj:procedure(f:string);
begin
   call Q.insert(f);
   if Q.full() then
     return disable rejestruj
   fi;
end rejestruj;
```

Zwróćmy uwagę na instrukcje:

- `accept` rejestruj – gdy kolejka Q jest pusta, pozostaje nam tylko oczekiwanie na to by jakiś klient zarejestrował nowe zadanie,

- `return disable` rejestruj – w sytuacji gdy kolejka Q jest pełna, każdy klient pragnący zarejestrować plik będzie musiał zaczekać do momentu w którym z kolejki zostanie wyjety jakiś plik i wykonana zostanie instrukcja `enable` rejestruj.



Tworzymy system klientów ze spoolerem i puszczamy go w ruch.

```
program zlecenia_do_spoolera;
  spooler: process(node: integer); ... end spooler;
  klient: process(node: integer, s:spooler); ... end klient;
  var sp: spooler, k1, k2, k3, k4: klient
begin
   sp := new spooler(0);
   k1 := new klient(11, sp);
   k2 := new klient(25, sp);
   k3 := new klient(0, sp);
   k4 := new klient(11, sp);
   resume(sp); resume(k1);  ...  resume(k4);
end
```

Pięć obiektów aktywnych umieścimy na trzech wirtualnych procesorach loglanowskich i następnie puścimy je w ruch. Mamy tu i rozpraszanie obliczeń i obliczenia współbieżne.

Własności spoolera
Zakładamy, że instrukcje drukowania wybranego pliku kończą się (bez zawieszenia i bez zapętlenia).

Twierdzenie 2. *Opisany powyżej system ma dwie podstawowe własności:*

(i) Żadne zlecenie nie zostanie zgubione.

(ii) Obsługa zleceń nie dozna uszczerbku z powodu konfliktu zleceń bądź konfliktu zlecenie/obsługa zlecenia.

Nie może dojść do zapisania dwu plików w to samo miejsce w kolejce Q. Nie może dojść do zgubienia pliku z powodu przepełnienia kolejki.

Dowód. Przebiega w kilku punktach omawiających zdarzenia do jakich może dojść w trakcie pracy spoolera.

- Podczas wybierania pliku z kolejki nie dochodzi do nowej rejestracji.
  Ponieważ wykonano instrukcję ⌜`disable rejestruj`⌝.
- Podczas rejestracji wątek spoolera jest zawieszony.
  Ponieważ nastąpiło przerwanie.
- Podczas rejestracji inny klient musi oczekiwać na jej zakończenie.
  Zbiór ENABLED *jest pusty.*
- Po zapełnieniu kolejki kolejni klienci oczekują na miejsce w kolejce.
  Gdy po wpisaniu nazwy pliku do kolejki jest ona pełna, dokonuje się powrotu ⌜`return disable` rejestruj⌝, który *nie odtwarza zbioru* ENABLED.

- Gdy kolejka jest pusta wątek oczekuje na zarejestrowanie jakiegoś pliku. Wykonanie instrukcji ⌜`accept` rejestruj⌝ powoduje oczekiwanie na rejestrację.
- Podczas opracowywania pliku i przesyłania go do drukarki może dokonywać się bezkonfliktowa rejestracja zgłoszeń. Operacja ⌜rejestruj⌝ *przerywa na chwilę* wątek spoolera.

□

2.6 Jednolity mechanizm programowania obliczeń rozproszonych i współbieżnych

W Loglanie'82 programowanie współbieżne i rozproszone odbywa się przy użyciu *wbudowanych w język* klasy `process` oraz instrukcji: obcego wołania procedur, spotkania – accept, oraz instrukcji enable, disable zmieniających stan metod procesu:

$$prywatna \rightarrow publiczna \rightarrow prywatna \rightarrow publiczna \rightarrow \ldots$$

Dostarczamy prosty i intuicyjny sposób łączenia wirtualnych maszyn loglanowskich w rozproszony wieloprocesorowy wirtualny komputer loglanowski. Na takim komputerze obliczenia mogą być rozpraszane, można też obliczenia wykonywać współbieżnie lub w kombinacji wykonującej część obliczeń w rozproszeniu a część współbieżnie. Mozna w nim uruchamiać na raz wiele programów.

Zauważmy, że w znanych nam systemach obliczenia współbieżne programuje się inaczej, a obliczenia rozproszone inaczej. Np. w Javie obliczenia współbieżne programujemy tworząc wątki – threads, a obliczenia rozproszone wymagają zastosowania mechanizmu Java RMI.

2.7 Każdy moduł może dziedziczyć klasę

W Loglanie'82 każdy rodzaj modułu może dziedziczyć (tj. rozszerzać) klasę. Mozemy więc "wyciągnąć przed nawias" wspólną część kilku algorytmów, zawrzeć ja w klasie C, a później deklarując funkcję f lub procedurę p zapowiedzieć, że rozszerza ona klasę C. Oprócz Loglanu'82 tylko język BETA ma taką możliwość.
przykład
W strukturze danych drzewa binarnych poszukiwań BST mamy trzy operacje:

- funkcję boolowska szukaj, sprawdzającą czy dany element e jest w dzrewie,

- operację wstaw, wstawiajacą dany element do drzewa,

- operacje usuń, usuwajacą element z dzrewa

rozwiń ten przykład

2.8 Krótkie porównanie języków programowania obiektowego

Poniższa tablica jest argumentem na rzecz następującej tezy;

Twierdzenie 3. *Żaden z istniejących języków programowania nie posiada cech występujących w Loglanie'82.*

Features	Simula 67	obj. Pascal	C++	Modula 3	Smalltalk	Eiffel	Ada	Beta	Java	**Loglan 82**
Modularisation										
nesting of modules	+	+	–	–	–	–	+	+	+	+
inheritance	+	+	+	+	+	+	–	+	+	+
– multilevel	–	–	–	–	–	–	–	+	+	+
– multiple	–	–	+	–	+	+	–	–	+	–
– in functions	–	–	–	–	–	–	–	+	–	+
static binding	+	+	–	+	–	–	+	+	+	+
Classes & Objects	+	+	+	+	+	+	+	+	+	+
Coroutines	+	–	–	+	–	–	–	+	–	+
Processes	–	–	–	+	–	–	+	+	+	+
– alien calls	–	–	–	–	–	–	–	–	–	+
Signals & Exceptions	–	–	+	–	–	–	+	+	+	+
Safety										
safe dealocation	–	–	–	–	–	–	–	–	–	+
type checking	+	+	–	–	–	–	+	+	+	+
protection of private	+	–	–	+	–	–	+	+	+	+
Genericity										
types as formal parameters	–	–	–	–	–	–	–	–	–	+
virtual methods	+		+		+	+	–	+	+	+
overloading	–		+	+		+	+	+	+	–

Rozdział 3

Jutro: problemy i zadania

Skromnymi środkami zrobiono bardzo wiele. Uzyskano eksperymentalne potwierdzenie przydatności opracowanych narzędzi w procesie tworzenia oprogramowania. Zbudowane kompilatory i środowiska nie mają jednak charakteru komercyjnego. Nie musimy się tym zrażać. Doświadczenia z projektami GNU i Linux pokazują, że istnieje szansa wytworzenia czegoś wartościowego i akceptowanego przez myślących programistów i manadżerów. Poniżej przytaczam zadania do zrealizowania i problemy do rozwiazania. Wspominam też o nowym, dużym, trudnym i ambitnym projekcie badawczym.

3.1 Problemy

W tej części wyliczamy kilka problemów otwartych. Dziś nie znamy odpowiedzi na te pytania.

- Czy protokół alien call jest implementowalny w Javie? Czy protokół ten jest do zaimplementowania w C++?

- Czy można zaimplementować w Javie system współprogramów o takich własnościach jak współprogramy w Loglanie'82?

- udowodnić, że instrukcja współprogramów attach(x) może być postrzegana jako cykliczny obrót stosu rekordów aktywacji. W wyniku tego obrotu na wierzchołku stosu znajduje się najśieższy rekord aktywacji znajdujący się w łańcuchu dynamicznym espółprogramu x.

- Czy można przenieść do Javy (odpowiednio do C++) Kreczmara system zarzadzania pamięcia obiektów?

- Czy protokół alien call da się rozszerzyć na maszyny równoległe, wieloprocesorowe? Jeśli by to się udało, to mielibyśmy jednolity mechanizm programowania współbieżnego, rozproszonego i równoległego.

- Czy potrafimy zbudować wzorcowa aksjomatyczną semantykę Loglanu?

3.1.1 Alien call

- Czy protokół alien call jest implementowalny w Javie?

  - Spróbujmy. Część synchroniczną protokołu potrafię zaprogramować. Ale asynchroniczne obce wołania wymagają, albo ingerencji w byte code, albo modyfikacji maszyny JVM, lub użycia preprocessora, który po kazdej instrukcji(tj. przy średniku) wstawi instrukcję wywołujacą procedurę sprawdzania czy należy przerwać obliczenia wątku. Te trzy sposoby nie moga byc zaakceptowane. Pozostaje pewna szansa polegająca na wykorzystaniu metody interrupt(). Nie wiem jednak czy pójscie tą drogą powiedzie się. Jeśli się uda, to wierzę, że protokół obcego wołania metody w obiekcie klasy thread pozyska wielu zwolenników wśród użytkowników Javy.
  - Jeśli nie można zaimplementować protokołu alien call w Javie, to fakt taki trzeba udowodnić i opublikować. Stanowić to będzie argument na rzecz modyfikacji Javy (gruntownej) i/lub na rzecz stworzenia nowego języka.

- Czy można rozszerzyć protokół alien call na programowanie równoległe? Sądzę, ze tak. Dziś komputery z wieloma rdzeniami, z wieloma procesorami nie są drogie i są blisko nas. Zaoferowanie narzędzia, dzięki któremu można będzie planować pracę procesorów powinno spotkać się z zainteresowaniem użytkowników. W tej chwili planowaniem zajmuje się system operacyjny, użytkownik nie ma wpływu na obciążanie rdzeni bądź procesorów.

- Porównać alien call z pojęciem akordu (*ang.* 'chord') w Polyphony - proponowanym rozszerzeniu języka C#.

3.1.2 Współprogramy

Zacznijmy od hasła *coroutine* w wikipedii. Łatwo zauważymy, że definicja tego pojęcia odnosi się do wczesnych lat 60 XX wieku, i nie ma związku z pojęciem współprogramu wprowadzonym w języku Simula67. Nie będziemy

szerzej rozwijać tego tematu. W tym samym haśle wikipedii znajdujemy omówienie czterech sposobów implementacji współprogramóww Javie.

c_1) zmodyfikować JVM wirtualną maszynę Javy,

c_2) modyfikować bytecode uzyskiwany w procesie kompilacji,

c_3) wykorzystać odpowiednie dla konkretnej platformy(Windows, Linux, XOS, ...) mechanizmy JNI zaimplementowane w języku C lub systemie operacyjnym,

c_4) oprzeć implementację współprogramów na pojęciu watku (klasa thread).

Pierwsze dwie propozycje wyprowadzają poza język Java. Trzecia narusza przenaszalność bytecodu pomiędzy platformami. Czwarta propozycja jest obarczona sporym narzuconym kosztem i ma inne mankamenty. Realizowanie współprogramów jako wątków wydaje się niedorzeczne, z tego powodu, że współprogramy to narzędzia programowania quasi-współbieżnego a wątki realizują obliczenia współbieżne.

W naszym przekonaniu problem realizacji współprogramów w Javie pozostaje wciąż otwarty. Warto zbadać czy można zdefiniować i zrealizować w Javie system współprogramów o cechach zrealizowanych w Loglanie'82? W szczególności, czy można i czy warto by każdy wątek mógł posiadać własny system współprogramów?

3.1.3 System zarządzania obiektami

Warto zbadać na ile wyposażenie Javy, C++, C# w system Kreczmara jest wykonalne i czy przynosi realne korzyści twórcom oprogramowania. Spodziewamy się, że wprowadzenie bezpiecznej dealokacji pozwoli na rzadsze wykonywanie operacji odśmiecania *gc()*. Koszt tej operacji rośnie wraz z rosnącym rozmiarem pamięci. Programista zostanie zwolniony z obowiązku pamiętania o wszystkich referencjach do obiektu *o* jaki uznał za niepotrzebny.

3.1.4 Programowanie równoległe

Pytania:

- Czy rozwiązania przyjete w Loglan VLP nadaja się do programowania rozproszonego w dużej skali?
- Czy alien call i łączenie maszyn wirtualnych w wirtualny rozproszony komputer loglanowski wystarczą by zrealizować programowanie równoległe na maszynach wieloprocesorowych?

3.1.5 Aksjomatyczny opis semantyki

Wydaje się nam, że doświadczenia z logiką algorytmiczną i z Loglanem'82 pozwalają z optymizmem patrzeć na możliwość zbudowania zestawu aksjomatów i reguł opisujacych semantykę Loglanu'82 bądź nowego języka.

3.2 Zadania

Lista zadań jakie dzisiaj widzę:

1. Ponowna kompilacja kompilatora loglanu. Dwa programy: loglan i gen zostały skompilowane lata temu. Ponowna ich kompilacja wymaga włożenia pewnej pracy. Oceniam ją na ok. 40 godzin. W efekcie uzyskamy bardziej przenośne oprogramowanie całości.

2. Przepisanie środowiska VLP na nowo tym razem w qt4. Obecna wersja środowiska VLP została zaprojektowana dla qt1 - dziś praktycznie nieistniejącą wersję biblioteki qt. W 2010 O. Świda, autor VLP, przystosował VLP do współpracy z qt3. Ale byłoby lepiej przepisać VLP od nowa i dostosować do współpracy z biblioteką qt4. Jeśli podejmiesz się tego zadania: nie tylko nam pomożesz ale i zyskasz wiedzę o qt4. Będziesz mógł sie tym pochwalić Twemu pracodawcy.
   A może wykorzystać środowisko wxwidgets?

3. Przeniesienie środowiska VLP na platformę Windows. Zrobiliśmy krok w tym kierunku. VLP działa (niestety bez kompilatora loglanu - patrz punkt 1) w terminalu linuksowym *cygwin* na Windowsie. A więc jest to możliwe. Następny krok to wykorzystanie kompilatora *mingw*, który pozwala kompilować na platformie Windows programy napisane dla platformy Linuks. Trzeba też zbudować bibliotekę qt3 dla Windows. To jest do zrobienia - przepis znalazłem w sieci. Taki produkt byłby bardzo ważny dla propagowania Loglanu i dla sprawdzenie jak działa rozpraszanie programu loglanowskiego na hybrydowej sieci komputerów Windowsowych i Linuksowych.

4. Nowy kompilator starego Loglanu. Programy loglan i gen mają ponad 30 lat. Działają, ale nie możemy ich modyfikować, dokonywać eksperymentów. Dlatego zaczynamy pracę nad nowym kompilatorem Loglanu82. Pierwsza część parser będzie niedługo gotowa. Następne: statyczna analiza semantyczna i generowanie kodu bajtowego oczekują na swych twórców.

5. **Nowa wersja języka Loglan**.
   Loglan zaprojektowano ponad 30 lat temu. Mógłbym powiedzieć, że język ten trzyma się całkiem nieźle chociaż jest staruszkiem: Odróżnia się pozytywnie od wielu nowszych języków np. oryginalnym protokołem współpracy procesów tzw alien call, koncepcją łączenia maszyn wirtualnych, ofertą współprogramów tzn. coroutin, bezpieczną dealokacją niepotrzebnych obiektów, przekazywaniem typów jako parametrów. Ale warto by zmienić ortografię, zmodernizować system sygnałów i wyjątków, ... Jednym słowem zapraszam do pracy nad nowym językiem programowania. Łącznie z wymyśleniem nazwy dla niego.

6. Rozszerzenie oferty obliczeń na obliczenia równoległe. Jako jedyny, Loglan oferuje jednolity mechanizm dla programowania obliczeń współbieżnych i rozproszonych. Ale nadszedł czas by ofertę tę poszerzyć na obliczenia równoległe. Nie jest dziś problemem zakup komputera z kilkoma rdzeniami lub kilkoma procesorami. Trzeba opracować koncepcję jak to ma wyglądać od strony użytkownika - programisty i jak to zrealizować. To chyba najbardziej ambitna część naszego projektu. Na pewno zaowocuje publikacjami, może nawet doprowadzić do doktoratu i dalej.

7. Rozwijanie zastosowań Loglanu Żaden język nie może się rozwijać jeśli nie będą powstawać aplikacje w nim tworzone. Witamy wszelkie próby tworzenia większych aplikacji.

8. Porządkowanie dokumentacji, tworzenie materiałów dydaktycznych, tutoriali etc. Mamy sporo dokumentacji. Chętnie przyjmiemy pomoc w porządkowaniu dokumentacji, tworzeniu tutoriali etc.

9. Poszukiwanie odpowiedzi na pytanie: VLP czy Eclipse? Posiadamy wtyczkę loglanowską do Eclipse - daje ona współpracę z prawdziwym edytorem, uporządkowane tworzenie projektów loglanowskich, kolorowanie składni itp. Niestety nie obejmuje to VLP: łączenia maszyn wirtualnych, alokacji procesów na zdalnych komputerach, etc. Nie wiemy czy da się pogodzić te dwie koncepcje?

10. Jak tworzyć biblioteki klas?
    Skłaniam się ku przyjęciu koncepcji jaką przyjęto w Javie, Adzie i in. Wprowadźmy pojęcie pakietu i compilation unit.

Ta lista może się zmieniać, poszerzać.

3.3 Nowy projekt: SpecVer

Od pewnego czasu przymierzamy się do nowego projektu – nazwaliśmy go SpecVer. Celem tego projektu jest stworzenie środowiska w którym zespoły będa mogły tworzyć oprogramowanie od poczatku – specyfikacja, do końca – weryfikacja. Oczywiście projekt SpecVer ma też wspomagać tworzenie oprogramowania.
Projekt SpecVer ma być emanacją dwu wcześniejszych projektów, w których uczestniczyliśmy i je inspirowaliśmy:

- AL - logika algorytmiczna
  - pokusimy się o pełny opis semantyki języka programowania LEM'12 w terminach aksjomatów i reguł wnioskowania opisujących konstrukcje programistyczne,
  - zapewni narzędzia do specyfikacji algorytmów tj. procedur, funkcji, konstruktorów, wątków oraz do specyfikacji klas, współprogramów i procesów,
  - AL dostarczy też narzędzi do weryfikowania oprogramowania względem specyfikacji,
- Loglan - (a właściwie nowy język programowania LEM'12) dostarczy narzędzi do tworzenia oprogramowania.

Nowy Loglan stanie się częścią ogromnego projektu SpecVer - środowiska dla pracy nad:

- specyfikacją oprogramowania,
- konstrukcją programów i modułów takich jak klasy,
- weryfikacją oprogramowania względem specyfikacji.

W środowisku SpecVer potrzebne będa narzędzia wspomagajace pracę twórców specyfikacji.

Potrzebne będa narzędzia do pracy nad oprogramowaniem,a więc kompilatory, ale także narzędzia do eksperymentowania z programami i graficznej wizualizacji, np graficzny debugger

Narzędzia dla audytorów, czyli specjalistów badających czy oprogramowanie P jest zgodne ze specyfikacją S będą dopiero powstawać. Ta praca nie da się zautomatyzować – i bardzo dobrze! Ale pomocą mogą być proof checkery, provery i specjalistyczne edytory.. W związku z tym pojawia się **zasadnicze pytanie**: Czy można zdefiniować nowy język programowania o następujących cechach:

- nowy język zachowa wszystkie osiągnięcia Loglanu'82,
- wykorzysta co tylko się da z dorobku Javy, C#, i innych nowszych języków programowania,
- w miejsce modułów `interface` wprowadzi moduły `specification`, w celu stworzenia środowiska SpecVer. (To środowisko ma stworzyć wspólne ramy i wspomagać prace nad specyfikacją, implementacją i weryfikacją oprogramowania.)

3.4 "razem młodzi przyjaciele"

Zapraszam wszystkich chętnych do współpracy.
Zapewne padną pytania o finansowanie tych badań. Moje doświadcznia pokazują, że środowisko akademickie będzie blokować te projekty. Trzykrotnie w latach 90 ubiegłego wieku składałem wniosek o grant do KBN (Komitetu Badań Naukowych) i trzy razy otrzymałem decyzję odmowną. Dwa razy składałem wniosek o skromny grant dla doktoranta i te wnioski też zostały odrzucone.
A jednak obaj doktoranci znakomicie wywiązali się ze swych zadań: dzięki jednemu z nich mamy wyniki dotyczące problemu znajdowania bezpośrednich superklas w Javie, drugi opracował i zrealizował system VLP pozwalający tworzyć rozproszoną wieloprocesorową maszynę wirtualną Loglanu i wykonywać w niej obliczenia współbieżne, rozproszone i mieszane, z wykorzystaniem protokołu alien call.
Wysnuwam stąd nadzieję, że młodzi koledzy potrafią wiele zrobić, nawet jeśli nie zapewnimy im specjalnie atrakcyjnych warunków finansowych.
Mam nadzieję, że tak jak to już było wcześniej studenci i doktoranci różnych krajów skrzykną się w internecie by wspólnie stworzyć nową jakość i cieszyć się nią.

Dołącz do nas, lub przynajmniej zaproponuj nazwę dla nowego języka.

Rozdział 4

Podziękowania i wyjaśnienia

Pragnę podziękować wszystkim, którzy pomogli zrealizować projekt badawczy Loglan'82. W kolejności chronologicznej: prof. Andrzejowi Janickiemu, który wierzył, ze potrafimy to zrobić i zaangażował się w doprowadzenie do grantu dzięki, któremu zyskaliśmy, w roku 1978, dwa minikomputery MERA 400 i środki finansowe na wspomożenie pracowników, prof. Tomaszowi Müldnerowi, profesor Hannie Oktabie, dr Wiesławie Bartol z którymi opracowaliśmy wstępny projekt języka Loglan77.

Jestem i będę zawsze wdzięczny Antkowi Kreczmarowi(1945 - 1996), kiedyś studentowi, później przyjacielowi i koledze. Bez niego nic by się nie udało. Jego dorobek pozostaje dziś zapoznany, mam jednak nadzieję, że to się zmieni i że jego wyniki będą szerzej wykorzystywane. To profesor Antoni Kreczmar opracował cały running system Loglanu'82, w którego skład weszły m.in. całkowicie oryginalne podsystemy: zarządzania pamięcią obiektów, zarządzania współprogramami, i in.

Jestem wdzięczny twórcom kompilatora na Merę 400: Andrzejowi I. Litwiniukowi, Danucie Szczepańskiej, Wojtkowi Nykowskiemu, Markowi Lao.

Podziękowania należą się pracownikom Zakładu Teorii Obliczeń w Instytucie Informatyki UW (1977 - 1992) i wielu studentom, którzy wykazali się wielkim zaangażowaniem w prace nad Loglanem i jego zastowaniami.

Bibliografia

[B+84] W.M. Bartol et al. *Report on the Loglan'82 programming language*. PWN, Warszawa, 1984.

[BKLO83] W.M. Bartol, A. Kreczmar, A. Litwiniuk, and H. Oktaba. Semantics and implementation of prefixing at many levels. In A. Salwicki, editor, *Proc. Logic of Programs and Their Applications*, volume 148 of *LNCS*, pages 45–80. Springer Verlag, 1983.

[CDG+89] Luca Cardelli, James Donahue, Lucille Glassman, Mick Jordan, Bill Kalsow, and Greg Nelson. Modula-3 report (revised). Technical Report 52, DEC, November 1989.

[Cie88] B. Ciesielski. Realizacja procesów rozproszonych w loglanie'82. Master's thesis, Instytut Informatyki UW, Warszawa, 1988.

[CK84] G. Cioni and A. Kreczmar. Programmed deallocation without dangling references. *Information Processing Letters*, 18:179–187, 1984.

[DMN70] O. J. Dahl, B. Myhrhaug, and K. Nygaard. Common base language (simula67). Technical Report S-22, NCC, Oslo, 1970.

[GM95] James Gosling and Henry McGuilton. The java language environment, a white paper. Technical report, Sun Microsystems, Mountain View, October 1995.

[K+83] A. Kreczmar et al. *Loglan-82*. Uniwersytet Warszawski, Zaborów, 1983.

[KKLS84] M. Krause, A. Kreczmar, H. Langmaack, and A. Salwicki. Specification and implementation problems of programming languages proper for hierarchical data types. Technical Report 8410, Institut fuer Informatik Kiel University, Kiel, 1984.

[KLKW86] M. Krause, H. Langmaack, A. Kreczmar, and M. Warpechowski. Concatenation of program modules, an algebraic approach to the semantic and implementation problems. In *Proc. Computation Theory*, volume 208 of *LNCS*, pages 134–156. Springer Verlag, 1986.

[KR03] D. Knuth and F. Ruskey. Efficient coroutine generation of constrained gray sequences (aka deconstructing coroutines),. In *From Object-Orientation to Formal Methods: Dedicated to The Memory of Ole-Johan Dahl*. Springer Verlag, 2003.

[LSW08] H. Langmaack, A. Salwicki, and M. Warpechowski. On the correctness and completeness of a deterministic algorithm elaborating direct super classes in java-like languages. *Fundamenta Informaticae*, 85:343–357, 2008.

[LSW09] H. Langmaack, A. Salwicki, and M. Warpechowski. On an algorithm determining direct superclasses in java and similar languages with inner classes - its correctness, completeness and uniqueness of solutions. *Information and Computation*, 207:389–410, 2009.

[MS87] G. Mirkowska and A. Salwicki. Problems and theories inspired by the loglan project. In *Algorithmic Logic*, pages 298–347. PWN & D.Reidel, 1987.

[Sal85] A. Salwicki, editor. *Język programowania Loglan'82, jesienna szkoła PTI*. PTI, Instytut Informatyki UW, 1985.

[SBOM77] A. Salwicki, W.M. Bartol, H. Oktaba, and T. Müldner. Loglan 77 - definicja języka programownia. Technical Report 20, Instytut Maszyn Matematycznych MERA, Warszawa, 1977.

[Wan82] A. Wang. Coroutine sequencing in simula, i - iii. Technical report, Norvegian Computing Center, Oslo, 1982.