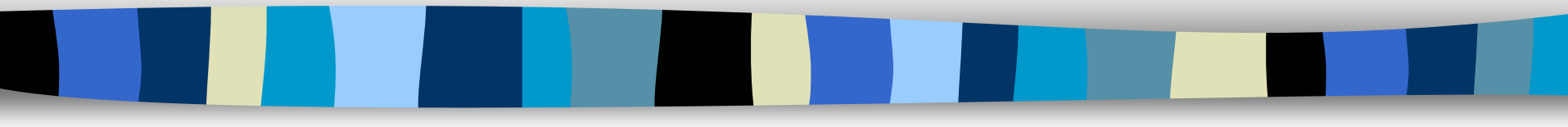
Elżbieta Bajkowska



Strongtalk

Plan prezentacji

- Czym jest Strongtalk
- Historia Strongtalk
- System typów w Strongtalk
- Podsumowanie
- Źródła

Czym jest Strongtalk

Składnia i semantyka Smalltalk'a-80 plus:

- Większa wydajność – najszybsza implementacja Smalltalk
- System typów – opcjonalny i przyrostowy; silny, statyczny system typów działający niezależnie od metody kompilacji (nie typowany kod Smalltalka wykonuje się równie szybko)
- Typowana biblioteka klas według Smalltalk Blue Book

Historia Strongtalk

Dwa równoległe i niezależne wątki rozwoju przyszłego systemu Strongtalk

- Zachodnie wybrzeże – prace grupy związanej z językiem Self nad innowacyjną maszyną wirtualną
- Wschodnie wybrzeże – prace nad systemem typów dla języka Smalltalk

Historia Strongtalk
Zachód – maszyna wirtualna

- Cel – poprawienie wydajności czysto obiektowego języka Self
- Satysfakcjonujące odśmiecanie
- Problem efektywnej kompilacji dla dynamicznie typowanego języka, używającego konstrukcji bloku gdzie wszystkie dane są obiektami (analogia do Smalltalk’a) - Urs Hoelzle

Historia Strongtalk Wschód – system typów

- Dave Griswold i Gilad Bracha opracowują Strongtalk - (nazwa systemu wywodzi się od systemu typów) – konferencja OOPSLA ’93
- Baza – biblioteki ParcPlace Smalltalk, nie odzwierciedlające w pełni hierarchii typów Strongtalka
- Zainteresowanie technologią grupy pracującej nad kompilatorem Self dla poprawienia wydajności

Historia Strongtalk Połączenie technologii

- Animorphic Systems - Dave Griswold zaczyna współpracę z Urs'em Hoelzle (reszta grupy: Lars Bak, Gilad Bracha, Steffen Grarup, Robert Griesemer Srdjan Mitrovic)
- 1994 – 96 wstępna wersja systemu Strongtalk, prezentacja na OOPSLA'96

Historia Strongtalk Zmierzch

- 1997 – Animorphic przejęty przez Sun Microsystems (elementy Animorphic VM w Javie).
- Zawieszenie prac
- 4.12.2001 – wersja 1.1. udostępniona za zgodą Sun, dla celów niekomercyjnych

System typów w Strongtalk

- Czemu statyczne typowanie
- Elementy systemu typów Strongtalk

System typów w Strongtalk Czemu statyczne typowanie

- Możliwość wykrycia niektórych błędów podczas kompilacji a nie wykonania – Smalltalk do dużych projektów
- ‘Programs are more often read then written’ – G.Bracha & D.Griswold paper
- Większa czytelność kodu z anotacjami
- Większa wydajność (ale nie w Strongtalk – niezależność od metody kompilacji)

System typów w Strongtalk Czemu statyczne typowanie – potencjalne problemy

- Prawdopodobieństwo ograniczenia elastyczności języka
- Niechęć programistów do porzucenia swobody dynamicznie typowanego języka
- Zwiększenie nakładu pracy programisty (aczkolwiek tej związanej z ‘pisaniem’ programu)

System typów w Strongtalk Czemu statyczne typowanie – opcjonalność

- System nie wymaga dostępu do wnętrza klasy lub metody, by typować jej użycie – typowanie interfejsów, enkapsulacja
- Kod w Strongtalk'u trywialnie konwertowany do czystego Smalltalk'a przez usunięcie anotacji
- Poprawny kod w Smalltalku jest też poprawny w Strongtalku

System typów w Strongtalk
Elementy systemu typów

- Podstawowa semantyka anotacji
- Protokoły – oddzielenie typów od klas
- Typowane bloki
- Generyczne typy
- Parametryzowany polimorfizm
- Typy – unie
- Fundamentalne zasady dla typowania

System typów w Strongtalk Podstawowa semantyka anotacji

- Po typowanym wyrażeniu wpisujemy anotację:
- *<nazwa typu>*, ^*<nazwa zwracanego typu>*

dodaj: liczba <Int> ^<Int>

Sygnatura metody przyjmującej parametr typu Int i zwracającej parametr tego typu:

System typów w Strongtalk
Protokoły

- Definiują hierarchię typów, oddzieloną od hierarchii klas
- Protokół – kolekcja metod i ich sygnatur
- Każda klasa indukuje protokół, można również definiować niezależne protokoły
- Dziedziczą z Object class protocol, tak jak klasy z Object class
- Self – zmienna typowa reprezentująca interfejs odbiorcy komunikatu, Instance – reprezentująca interfejs instancji odbiorcy komunikatu

System typów w Strongtalk
Protokoły

```
Protocol PlanarPoint
    x^<Int>.
    x:<Int> ^<Int>.
    y^<Int>.
    y:<Int> ^<Int>.
    + <Self> ^<Self>.
```

Protokół PlanarPoint

Self oznacza typ odbiorcy komunikatu

Instance jego instancję

```
class SpacialPoint
subclassOf:BasicPlanarPoint
instance var z <Int>.
instance methods
    z ^<Int>
        ^z.
    z: zval <Int>
        z:= zval.
    + p <Self> ^<Self>
        ^super + z: self z + p z.
```

SpacialPoint nie jest podtypem BasicPlanarPoint

Self – SpacialPoint

```
class BasicPlanarPoint
instance var x <Int>
instance var y <Int>
class methods
    new ^<Instance>
    ^super new init.
instance methods
    init
        x:= y:= 0.
    x ^<Int>
        ^x.
    x: xval <Int>
        x:= xval.
    y ^<Int>
        ^y.
    y: yval <Int>
        y:= yval.
    + p <Self> ^<Self>
        ^(self class new x: self x + p x)
            y: self y + p y.
```

BasicPlanarPoint

implementuje protokół PlanarPoint

Self oznacza typ BasicPlanarPoint

Instance instancję BasicPlanarPoint

System typów w Strongtalk
Typowane bloki - składnia

- [*typParametru1, typParametru2,…, typParametruN,^zwracanyTyp*]
- [*typParametru1, typParametru2,…, typParametruN*] – jeśli zwracany typ to Object
- Sygnatura: [Int,^Int] blok: [:n <Int>|^n+1]

```
aMessage: aStringBlock <[^Str]> ^<Int>
| result <Int> |
result := aStringBlock value size.
^result
```

System typów w Strongtalk
Typy generyczne

- <TypGen[X,Y,Z…]>, np. <Collection[E]>
- Za zmienną zostanie podstawiony typ

```
oneElement: anElement <^Str> ^<Collection[Str]>
| result <Collection[Str]> |
result := Collection[Str] new.
^result.
```

System typów w Strongtalk
Parametryzowany polimorfizm

- Możliwość wywnioskowania typu
- Może wydawać się nieporęczne, ale używane jest tylko przy definicji metody, a nie przy każdym użyciu
- Mechanizm wymuszony przy typowaniu biblioteki

Sygnatura metody z Collecton[T]
collect: blk <[T, ^X def]> ^ <Collection[X]>
Typ X będzie wywnioskowany z pierwszego argumentu metody – typ obiektu zwracanego przez ‘blk’
Użycie – wnioskujemy Integer:

```
| nums <Collection[Integer]> |
nums := 'Fee Fi Fo Fum' collect:[:c <Character> | c asciiValue].
```

System typów w Strongtalk
Typy - unie

- Gdy możemy mieć do czynienia z więcej niż jednym typem
- <*wariant1*|*wariant2*|…>

at: key <KEY>
ifAbsent: blk <[^X def]> ^<VALUE|X>

Sygnatura metody Dictionary[KEY,VALUE]
at: ifAbsent:

System typów w Strongtalk Fundamentalne zasady typowania

Zgodność sygnatur z

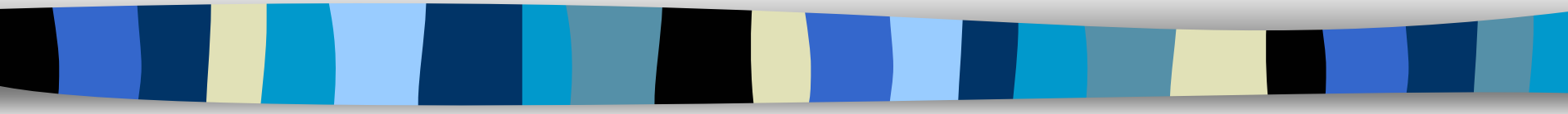
- Wołanymi metodami (argumenty, odbiorca, zwracany typ)
- Przypisaniami
- Sygnaturami metod w podklasach

Podsumowanie

- Strongtalk - silnie, statycznie typowany Smalltalk plus zwykły Smalltalk
- Typowanie opcjonalne i pozostawiające wiele walorów elastycznego Smalltalk’a
- System typów niezależny od maszyny wirtualnej, można zintegrować z dowolną wersją Smalltalk’a
- Małe szanse na komercyjne powodzenie – wbrew intencjom twórców

Źródła

- Strongtalk - http://www.cs.ucsb.edu/projects/strongtalk/
- Gilad Bracha - http://www.bracha.org
- G.Bracha D.Griswold – „Strongtalk: Typechecking Smalltalk in a Production Environment”
- Self - http://research.sun.com/self/



Koniec