XBA: Uma linguagem orientada a objetos com tipagem estrutural

Henrique Stagni
Orientador: Prof. Francisco Reverbel
Instituto de Matemática e Estatistica - Universidade de São Paulo

IME-USP

Introdução

- Linguagens com **tipagem nominativa** (como *C* e *Java*) são capazes de detectar uma série de erros de programação, mas possuem um *sistema de tipos* muito *rígido* que pode limitar a arquitetura do sistema desenvolvido.
- Linguagens com "**tipagem dinâmica**" (como *Python* e *Ruby*) facilitam a produção de um código mais flexível, mas certos tipos de erros podem passar despercebidos até o momento da execução.
- Linguagens com **tipagem estrutural** oferecem uma espécie de meio termo entre a flexibilidade presente em linguagens dinâmicas e a detecção de certos tipos de erros. Nessas linguagens, os tipos descrevem a estrutura dos dados; relações de *subtipagem* são estabelecidas automaticamente.

Objetivo

O objetivo deste trabalho consistiu na especificação de uma linguagem de programação, denominada **XBA**, assim como no desenvolvimento de um compilador para ela. A linguagem **XBA** possui:

- Suporte a orientação a objetos.
- Um sistema de tipos estrutural.
- Um sistema de inferência de tipos que dispensa a inserção de *anotações de tipos* nos programas.
- Outras características como:
  - Funções como objetos de primeira classe.
  - Todos os métodos/funções suportam *currying*.

Exemplo de programa em XBA

O exemplo abaixo será usado para demonstrar algumas características da linguagem e o funcionamento do sistema de tipos:

```
class Circle with-radius: radius
  field radius = radius
  method radius! = radius #devolve o valor do campo radius

class ColorfulCircle with-radius: radius and-color:color
  field radius = radius
  field color = color
  method radius! = radius
  method color! = color

class Main
  method print-diameter: circle =
    print (2*circle radius!)

  method init! =  #Ponto de entrada do programa
    let circle = Circle with-radius:26,
        colorful = ColorfulCircle with-radius:4 and-color:0 in
      self print-diameter: circle    # OK
      self print-diameter: colorful  # OK
      self print-diameter: 88        # Erro em tempo de compilacao
```

Observações sobre o programa exemplo

- Algo importante a se notar neste exemplo é que o sistema de tipos é capaz de inferir tipos estruturais para os objetos criados, acusando uma incompatibilidade de tipos na última linha do código.
- Por meio da estrutura dos tipos inferidos, o sistema de tipos detecta uma relação de subtipagem entre as classes *Circle* e *ColorfulCircle*, não acusando erro na penúltima linha do código.
- Note, ainda, que todas essas informações foram inferidas, sem a necessidade de inserção de nenhuma *anotação de tipo*, por parte do programador.

O objetivo do sistema de tipos desenvolvido é impedir a compilação de programas que possam enviar mensagens a objetos que não possuam os métodos correspondentes.

Os tipos

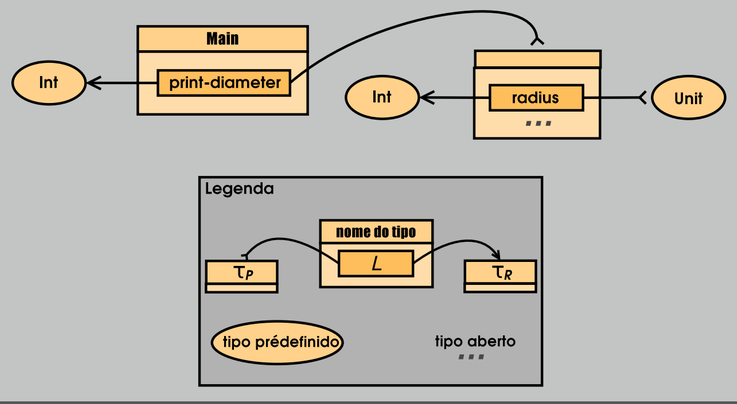
Na linguagem XBA, tipos descrevem a estrutura de um objeto. Um tipo é definido por um conjunto de triplas da forma (τ_R, L, τ_P), onde τ_R e τ_P são tipos e L é uma *label*
Cada tripla (τ_R, L, τ_P) está associada a um método do objeto que está sendo descrito pelo tipo:

- τ_R é o tipo de retorno do método em questão.
- τ_P é o tipo do parâmetro do método em questão.
- L é uma *label* associada ao seletor do método em questão.

Adicionalmente, tipos podem ser fechados ou abertos. Um tipo é fechado/aberto se representa um objeto contendo exatamente/pelo menos o conjunto de métodos descritos pelas suas triplas.

Exemplo

Expressões contendo instâncias de objetos da classe **Main** (ver programa de exemplo) possuem tipos que podem ser representados pela seguinte figura *(neste exemplo, vamos ignorar o método init)*:

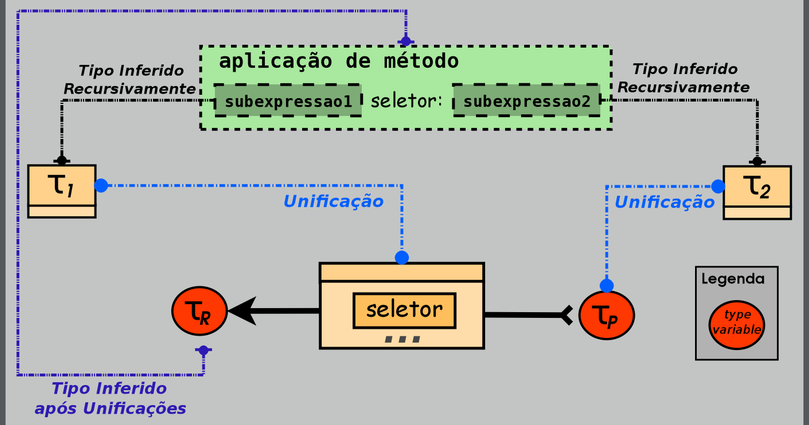


Inferência de tipos

É o processo que, em tempo de compilação, associa um tipo a cada *expressão* de um programa em XBA, acusando um erro caso isso não seja possível.

- É introduzido o conceito de *type variable*: uma variável que representa um tipo indeterminado.
- O processo todo se baseia no conhecido processo de **unificação** (lógica), aplicado ao contexto de tipos.
- A inferência é um processo recursivo que, para uma dada *expressão*:
  - infere o tipo de suas *subexpressões*.
  - constrói os tipos esperados das *subexpressões*.
  - **unifica** os tipos esperados com os tipos inferidos.

Segue, abaixo, um diagrama ilustrando como se dá a inferência de tipos de uma expressão que representa a *aplicação de um método*:



Implementação

- O compilador desenvolvido tem como alvo a **JVM** (*Java Virtual Machine*). A **JVM** foi escolhida por:
  - já possuir conceitos de orientação a objetos no nível dos bytecodes.
  - fazer otimizações agressivas durante a execução (*JIT*), não exigindo uma preocupação excessiva com a otimização do código gerado.
  - ter introduzido a instrução **invokedynamic**, que facilita o desenvolvimento e aumenta o desempenho de linguagens com características dinâmicas para a plataforma.
  - ser multiplataforma, possuir gigantesca coleção de bibliotecas, etc.
- Foi usada a ferramenta **JavaCC** para a geração do analisador sintático recursivo descendente da linguagem.
- Foi usada a biblioteca de manipulação de bytecodes **ASM**, facilitando a geração do código e a tranformação de classes feitas em Java (com anotações especiais) em classes **XBA** válidas.

stagni@gmail.com