Compiladores
Roteiro de Laboratório 04 – Verificação de Tipos

1 Introdução

A tarefa deste laboratório é implementar um verificador de tipos (*type checker*) para a linguagem EZLang. Juntamente com a tarefa do Laboratório 03, a tarefa de hoje conclui o desenvolvimento do analisador semântico para a linguagem.

Conforme visto no conteúdo teórico do Módulo 04, a verificação de tipos é elemento fundamental do processo de análise semântica. Como praticamente todas as linguagens de alto nível são tipadas, sempre é necessário verificar se o uso dos tipos está de acordo com a semântica da linguagem. Essa verificação (*type checking*) ocorre tanto em linguagens estáticas quanto dinâmicas, diferindo apenas no momento em que ela ocorre: em tempo de compilação nas linguagens estáticas, e em tempo de execução nas linguagens dinâmicas.

A linguagem EZLang segue o modelo clássico de linguagens imperativas derivadas de Algol, com uma verificação de tipos totalmente estática. A semântica do sistema de tipos da linguagem é descrita a seguir.

2 O sistema de tipos de EZLang

O sistema de tipos de EZLang é muito simples, possuindo somente quatro tipos primitivos, sem tipos compostos ou estruturados (*structs*). Os tipos básicos da linguagem são **inteiro**, **real**, **Booleano** e ***string***. Só é possível declarar uma variável por linha e não é permitido a inicialização de variáveis junto com a declaração.

As operações aritméticas são as quatro fundamentais, entretanto o operador + é sobrecarregado (mais detalhes abaixo). A linguagem também possui dois operadores de comparação, um comando de atribuição e um comando para impressão de valores na tela.

2.1 Verificação e conversão de tipos

Valem as seguintes regras para o sistema de tipos de EZLang:

- Tipo inteiro sofre *widening* implícito para real. Não é permitido *narrowing*. (Veja *slides* da Aula 04 para as explicações desses termos.)
- O operador + é sobrecarregado. Os tipos resultantes são dados pela tabela abaixo (i – inteiro, r – real, b – Booleano, s – *string*):

+	i	r	b	s
i	i	r	i	s
r	r	r	r	s
b	i	r	b	s
s	s	s	s	s

  Assim, vemos pela tabela acima que em uma expressão como `2 + 4.2`, o operando da esquerda (`2`) sofre *widening* para real (`2.0`) e a expressão como um todo tem real como tipo resultante. É fácil perceber que o operador + é sobrecarregado: se quaisquer dos operandos for uma *string*, a expressão resultante também é uma *string*, e o operador

representa uma concatenação de *strings*. Além disso, também é possível "somar" valores Booleanos (de forma similar ao C), aonde `false` vale 0 e `true` vale 1. A soma de dois Booleanos é equivalente à operação lógica de OU (*or*).

- Os demais operadores aritméticos manipulam os tipos como abaixo (e – erro):

$-*/$	i	r	b	s
i	i	r	e	e
r	r	r	e	e
b	e	e	e	e
s	e	e	e	e

- Operadores de comparação **sempre geram um Booleano como resultado**. Entretanto, tais operadores só admitem tipos compatíveis, conforme a tabela abaixo (✓– OK, e – erro):

$<, =$	i	r	b	s
i	✓	✓	e	e
r	✓	✓	e	e
b	e	e	e	e
s	e	e	e	✓

- O comando de atribuição só admite tipos idênticos, exceto no caso de inteiro sofrendo *widening* para real, conforme a tabela abaixo:

:=	i	r	b	s
i	✓	e	e	e
r	✓	✓	e	e
b	e	e	✓	e
s	e	e	e	✓

  Como um exemplo de uma verificação que o seu compilador deve fazer, suponha um comando de atribuição como

  ```
  x := "Valor: " + 2;
  ```

  aonde a variável `x` foi previamente declarada com o tipo real. Esse exemplo possui um erro de tipos, pois o resultado da expressão da direita da atribuição (RHS – *right-hand side*) é do tipo *string*, enquanto que a variável do lado esquerdo (LHS – *left-hand side*) é do tipo real. Segundo a tabela acima, essa atribuição de tipos é inconsistente.
- Uma última verificação de tipos da linguagem requer que todas as expressões de teste dos comandos de `if` e `repeat` sejam do tipo Booleano.

2.2 Mensagens do verificador de tipos

O seu verificador de tipos deve exibir as seguintes mensagens.

Operadores binários com tipos incompatíveis. Se os operadores aritméticos, de comparação, ou de atribuição possuírem operandos incompatíveis, exiba a seguinte mensagem de erro no terminal:

```
SEMANTIC ERROR (XX): incompatible types for operator 'OP',
                     LHS is 'LT' and RHS is 'RT'.
```

Aonde `XX` é o número da linha do programa onde o erro foi detectado, `OP` é o operador binário da expressão com erro, e `LT` e `RT` são os tipos das sub-expressões da esquerda e direita, respectivamente.

Expressões de teste. Se as expressões de teste dos comandos de `if` e `repeat` não forem do tipo Booleano, exiba a seguinte mensagem de erro no terminal:

```
SEMANTIC ERROR (XX): conditional expression in 'OP'
                     is 'ET' instead of 'bool'.
```

Aonde `XX` é o número da linha do programa onde o erro foi detectado, `OP` é o comando que contém a expressão, e `ET` é o tipo inferido da expressão de teste.

Demais mensagens. As demais mensagens de erros léxicos, sintáticos e semânticos são as mesmas do Laboratório 03 e devem continuar sendo exibidas como antes.

3 Implementado o verificador de tipos

Para poder implementar o verificador de tipos de EZLang, você deve ser capaz de atribuir tipos adequados para todas as expressões da gramática, seguindo as regras de tipagem estabelecidas na Seção 2. A computação (inferência) dos tipos das expressões deve ser feita de forma recursiva, em um processo de visitação *top-down* da *parse tree* igual ao que foi feito no exemplo da calculadora do laboratório passado. A diferença é que em vez de calcular os valores numéricos das expressões, você vai inferindo e propagando os tipos adequadamente, até o momento de fazer as verificações de compatibilidade.

Assim, o *visitor* desenvolvido anteriormente para a tarefa passada precisa ser estendido para também computar os tipos das expressões e verificar o seu uso apropriado. Só lembrando mais uma vez que no *visitor* você é responsável por chamar recursivamente o caminhamento nos filhos de um nó. Isso permite então que a gente determine os tipos dos filhos antes, através da recursão, e no final, quando ela retorna, a gente decide o tipo do nó pai. Para isso funcionar, você precisa que os métodos de visitação da *parse tree* agora retornem um valor da enumeração `Type`. Isto é definido com a declaração do analisador semântico segundo o cabeçalho abaixo:

```
1 public class SemanticChecker extends EZParserBaseVisitor<Type>
```

ATIVIDADE: Modifique a sua solução do laboratório anterior para realizar a verificação de tipos conforme descrita nesse documento.

Algumas observações importantes:

- O seu compilador pode terminar a execução ao encontrar o primeiro erro no programa de entrada.
- Os programas de entrada para teste são os mesmos de sempre (`in.zip`). As saídas esperadas desta tarefa estão no arquivo `out04_java.zip`.
- Uma implementação de referência para este laboratório será disponibilizada pelo professor em um futuro próximo. No entanto, você é *fortemente* encorajado a realizar a sua implementação completa antes de ver uma solução em outro lugar.