Interpretação e Compilação de Linguagens de Programação

Mestrado Integrado em Engenharia Informática
Departamento de Informática
Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa

2016-2017

João Costa Seco (joao.seco@di.fct.unl.pt)

Lecture 06
Compilação dirigida por informação de tipos

Unidade 7: Compilação tipificada

Os sistemas de tipos são ferramentas de análise estática que fornecem informação sobre programas concretos. O processamento é integrado na fase de compilação, termina sempre, e permite apurar a natureza concreta dos valores manipulados pelo programa.

Permite portanto a geração de código de forma especializada, mais eficiente. Ao preservar a informação de tipos permite-se uma verificação posterior (e.g. LLVM, JVM e CLR bytecode).

- Etiquetagem da AST
- Geração de código dedicado
- Verificação Low-Level

Compilador para uma linguagem (low-level) tipificada

- As linguagens intermédias e de baixo nível modernas (LLVM, CLR e JVM bytecode) são tipificadas.
- A Tipificação low-level permite:
  - Verificação em tempo de carregamento.
  - Validações de operações em runtime (a máquina não pára descontroladamente)
  - Optimização: geração de código específico

aload
iload
aaload
caload
dload

decl x = 1 **in**
decl y = 2 **in**
  x + y
end
end

```
new frame_1
dup
invokespecial frame_1/<init>()V
dup
sipush 1
putfield frame_1/loc_00 I
astore 0
new frame_2
dup
invokespecial frame_2/<init>()V
dup
aload 0
putfield frame_2/SL Lframe_1;
dup
sipush 2
putfield frame_2/loc_00 I
astore 0
aload 0
checkcast frame_2
getfield frame_2/SL Lframe_1;
getfield frame_1/loc_00 I
aload 0
checkcast frame_2
getfield frame_2/loc_00 I
iadd
aload 0
checkcast frame_2
getfield frame_2/SL Lframe_1;
astore 0
aconst_null
astore 0
```

Compilador de uma linguagem com variáveis na heap (arrays, objectos)

- Para implementar células de memória com o tempo de vida maior que o seu contexto de criação precisamos de alocação (no heap) de células de memória.

- Cada célula de memória é tipificada para conter um valor de um tipo único

Tipificação de microML

- Algoritmo **typecheck** para calcular o tipo de uma expressão qualquer da linguagem microML:

$$\text{typecheck} : \text{microML} \times \text{ENV} \rightarrow \text{TYPE}$$

typecheck(**var**(E) , *env*) ≜ [*t* = **typecheck**(E, *env*); ref{*t*}]

Tipificação de microML

- Algoritmo **typecheck** para calcular o tipo de uma expressão qualquer da linguagem microML:

$$typecheck : microML \times ENV \rightarrow TYPE$$

```
typecheck( assign(E1, E2) , env ) ≜
                                  [ t1 = typecheck( E1, env );
                                    t2 = typecheck( E2, env );
                                    if (t1 == ref{t2})
                                            then t2;
                                            else none ; ]
```

Compilador de uma linguagem com variáveis na heap (arrays, objectos)

- Para implementar células de memória com o tempo de vida maior que o seu contexto de criação precisamos de alocação (no heap) de células de memória.

- Cada célula de memória é tipificada para conter um valor de um tipo único.

- Logo, deve ser suportada por código tipificado com o tipo pré-determinado.

Compilador de uma linguagem com variáveis na heap (arrays, objectos)

decl x = **var**(1) in **!**x

```
new frame_1
dup
invokespecial frame_1/<init>()V
dup
new ref_int
dup
invokespecial ref_int/<init>()V
sipush 1
putfield ref_int/value I
putfield frame_1/loc_00 Lref_int;
astore 0
aload 0
checkcast frame_1
getfield frame_1/loc_00 Lref_int;
getfield ref_int/value I
aconst_null
astore 0
```

Compilador de uma linguagem com variáveis na heap (arrays, objectos)

decl x = **var**(1) in **!**x

```
.source frame_1.j
.class frame_1
.super java/lang/Object
.implements frame

.field public loc_00 Lref_int; ; x

.method public <init>()V
aload_0
invokespecial java/lang/Object/<init>()V
return
.end method
```

Compilador de uma linguagem com variáveis na heap (arrays, objectos)

decl x = **var**(1) in **!**x

```
.source ref_int.j
.class ref_int
.super java/lang/Object

.field public value I

.method public <init>()V
aload_0
invokespecial java/lang/Object/<init>()V
return
.end method
```

Compilador de uma linguagem com variáveis na heap (arrays, objectos)

decl x = **var**(**var**(1)) in **!!**x

```
.source ref_ref_int.j
.class ref_ref_int
.super java/lang/Object

.field public value Lref_int;

.method public <init>()V
aload_0
invokespecial java/lang/Object/<init>()V
return
.end method
```

Compilador de uma linguagem com variáveis na heap (arrays, objectos)

decl x = **var**(**var**(1)) in **!!**x

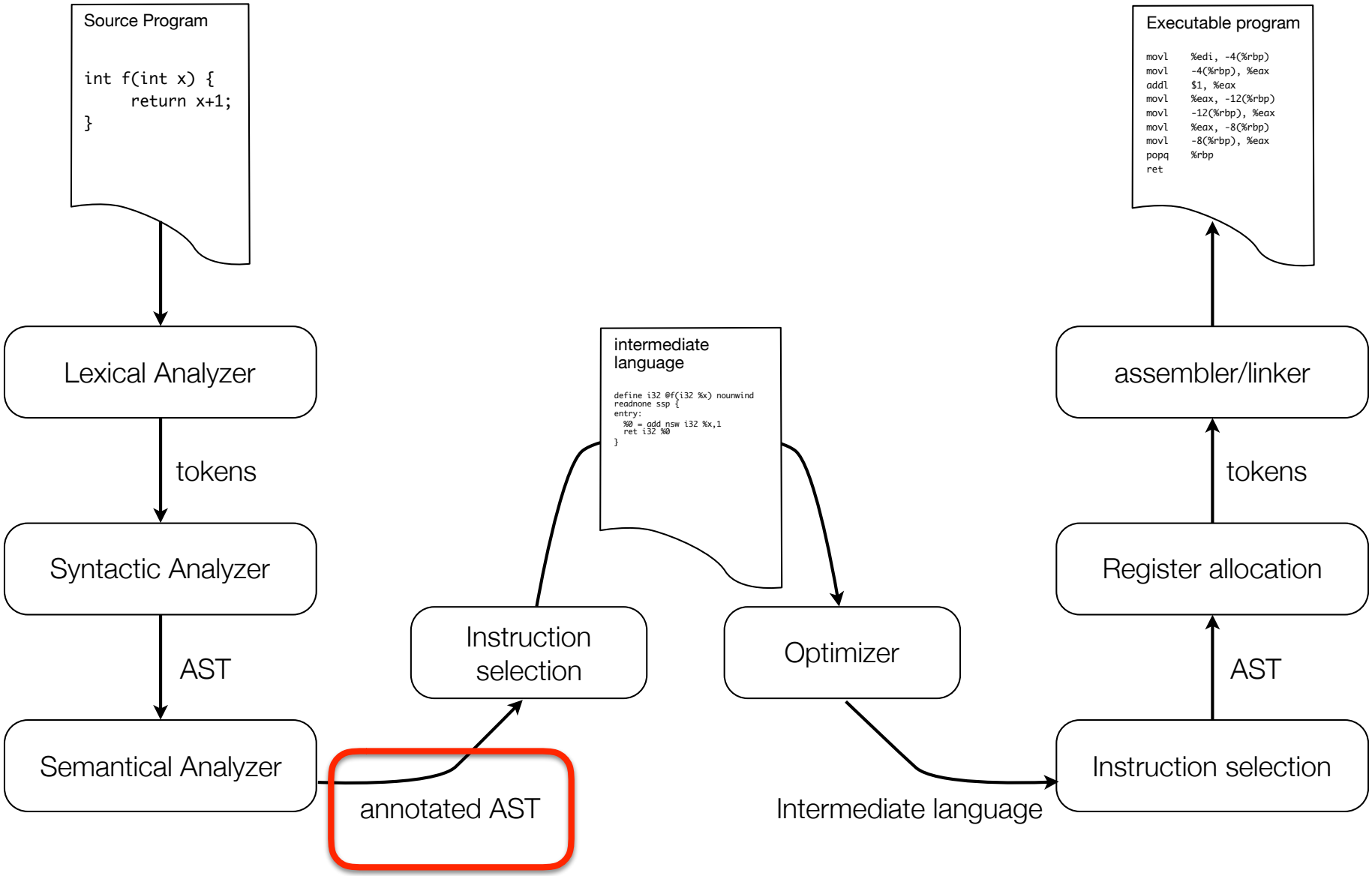
```
new frame_1
dup
invokespecial frame_1/<init>()V
dup
new ref_ref_int
dup
invokespecial ref_ref_int/<init>()V
new ref_int
dup
invokespecial ref_int/<init>()V
sipush 1
putfield ref_int/value I
putfield ref_ref_int/value Lref_int;
putfield frame_1/loc_00 Lref_ref_int;
astore 0
aload 0
checkcast frame_1
getfield frame_1/loc_00 Lref_ref_int;
getfield ref_ref_int/value Lref_int;
getfield ref_int/value I
aconst_null
astore 0
```

AST Etiquetada

- A análise semântica do sistema de tipos é independente da geração de código, mas pode influenciá-la.

- Para isso é possível colorir (etiquetar) a árvore com todos os resultados intermédios da verificação de tipos.

Arquitectura de um compilador (Lecture 03)



AST Etiquetada

- A análise semântica do sistema de tipos é independente da geração de código, mas pode influenciá-la.

- Para isso é possível colorir (etiquetar) a árvore com todos os resultados intermédios da verificação de tipos.

```
public class ASTNum implements ASTNode {

      private int value ;

      Num(int v) { value = v; }

      IValue eval(Env<Value> env) { return value; }

      IType typecheck(Env<IType> env) { return IntType.singleton; }

      IType getType() { return IntType.singleton; }

}
```

Types in Java

- Os tipos podem ser representados por um tipo indutivo IType, definido por um interface e um conjunto de construtores (classes).
- O uso de alguns padrões torna as implementações mais convenientes (singleton pattern - permite o uso de ==).

```
public interface IType {}

public class IntType implements IType {

   private IntType();

   public singleton = new IntType();

}

public class RefType implements IType {

   private IType type;

   public RefType(IType type) { this.type = type; }

   public Type getType() { return type; }

}
```

AST Etiquetada

- A análise semântica do sistema de tipos é independente da geração de código, mas pode influenciá-la.

- Para isso é possível colorir (etiquetar) a árvore com todos os resultados intermédios da verificação de tipos.

```
public class ASTAdd implements ASTNode {

    private int value ;

    private IType type;

    Num(int v) { value = v; }

    IValue eval(Env<Value> env) { return value; }

    IType typecheck(Env<IType> env) { … type = IntType.singleton }

    IType getType() { return type; }

}
```

Desafio de concepção

- Como compilar uma linguagem imperativa com a seguinte variante para as expressões de manipulação de memória:

```
var a = 1 in E end        a := E                    a
```

- O tempo de vida da variável `a` é a expressão `E` e os seus usos são sempre implicitamente desreferenciados.

- (JVM bytecode) Usar variáveis locais dos métodos para alojar as variáveis locais.