Construção de Compiladores

Bruna Felice da Silva
brunafelicesilva@gmail.com
Fernando Henrique de Oliveira
pets_frusciante@gmail.com
Guilherme Ferreira Ribeiro
guifrribeiro@hotmail.com

Faculdade de Computação
Universidade Federal de Uberlândia
Campus Santa Mônica

23 de fevereiro de 2015

Resumo

Este projeto é o resultado do estudo sobre "Construção de Compiladores". Aqui serão abordadas todos os processos envolvidos na construção e uso de um compilador MiniPython para Dalvik. Apresentaremos as diversas características de um compilador. Para a realização do projeto foram usados, o sistema operacional Ubuntu 14.04, as linguagens MiniPython, Dalvik e Ocaml.
A contribuição deste trabalho é compreender o funcionamento de um compilador e a respectiva documentação deste processo, visando o melhor entendimento e uso dos compiladores disponíveis no mercado, para criação de softwares mais eficientes.

Sumário

Lista de Figuras

Capítulo 1

Introdução

O presente relatório tem como objetivo principal efetuar a construção de um compilador MiniPython -> Dalvik.

O Dalvik Virtual Machine é uma máquina virtual baseada em registradores, projetada e escrita como parte da plataforma Android.

Ela é otimizada para requerer pouca memória, e é projetada para permitir que múltiplas instâncias da máquina virtual rodem ao mesmo tempo, deixando para o sistema operacional o isolamento de processos, o gerenciamento de memória e o suporte a threading.

A Google criou a máquina virtual Dalvik com o intuito de melhorar em certos pontos onde a JVM (Java Virtual Machine, que é bastante limitada quando se trata de mobilidade) falha quando o assunto é "Mobile"sendo assim, os pontos fortes que a Dalvik tem são, desempenho, segurança e tem a total liberdade para evoluir o Android sem a necessidade da especificação JSR (Especificação de requisitos Java) da Oracle, que é bastante burocrática.

Capítulo 2

Ferramentas

2.1 Sistema Operacional Ubuntu 14.04 LTS (Trusty Tahr

Para construir o compilador aqui proposto, faremos uso do sistema operacional Ubuntu 14.04 LTS, uma das várias distribuições Linux disponíveis.

2.1.1 Processo de instalação

Para instalar o Ubuntu 14.04 LTS, é necessário pelo menos 10 GB de espaço livre em HD, mas recomendamos 20 GB, além de 2 GB de memória RAM.

Passo 1 - Faça o download da imagem do CD de instalação do Ubuntu 14.04 no site: `http://www.ubuntu.com/download/desktop`. Basta escolher a versão do sistema, 32 ou 64 bits, recomendamos 64 bits.

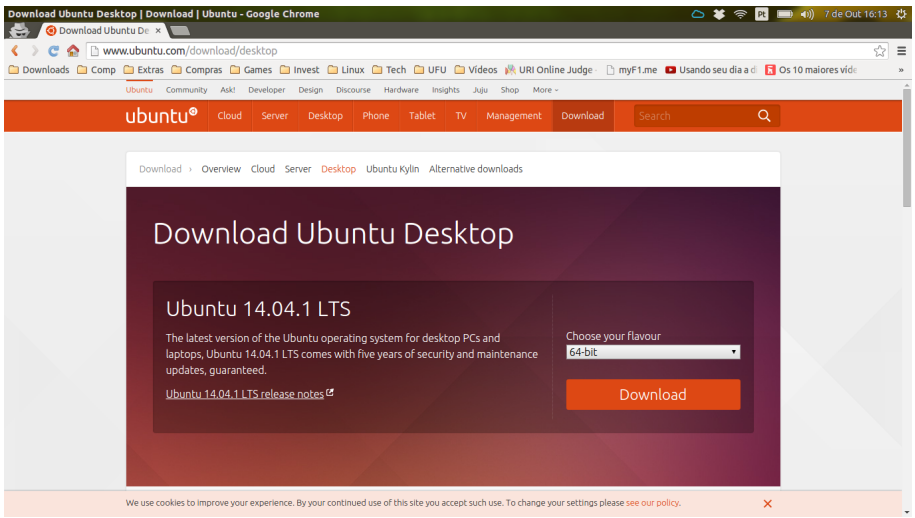

Figura 2.1: Download Ubuntu Desktop

Passo 2 - Grave a imagem em um disco, ou pen drive, então inicialize.

Figura 2.2: Tela inicial do instalador do Ubuntu 14.04

Passo 3 - Depois de inicializado, será apresentada a tela a seguir, onde você precisará escolher o seu idioma. Depois disso basta clicar em "Instalar o Ubuntu".

Figura 2.3: Bem-vindo a instalação do Ubuntu 14.04

Passo 4 - A tela seguinte é de preparação para instalação, onde é dada a possibilidade de indicar se pretende instalar algum software adicional para suporte a alguns formatos de arquivos, como .mp3, .mpeg. Essas opções não afetarão o nosso objetivo final com o sistema.

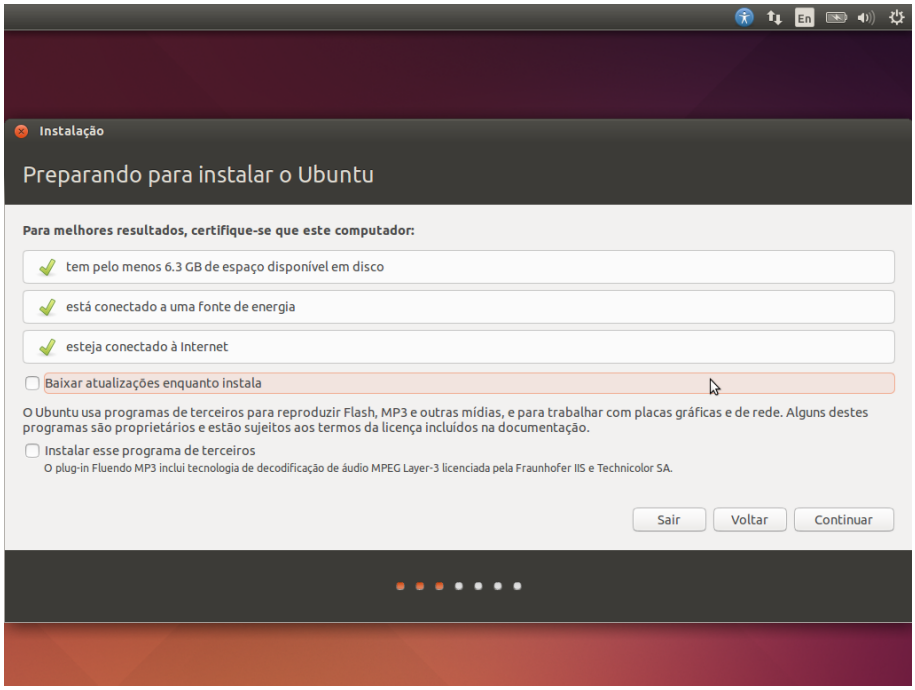

Figura 2.4: Preparando a instalação do Ubuntu 14.04

Passo 5 - Como estamos instalando o sistema em um HD sem outro sistema operacional, será apresentado apenas uma opção na tela Tipo de instalação, "Apagar disco e reinstalar o Ubuntu". Em diferentes circunstâncias teremos mais duas opções a disposição, "Atualizar o Ubuntu [versão] para Ubuntu 14.04"ou "Instalar o Ubuntu 14.04 ao lado do [Sistema operacional instalado na máquina]". Nesta tela, no nosso caso deixe a opção "Apagar disco e reinstalar o Ubuntu"marcada e clique em "Instalar agora".

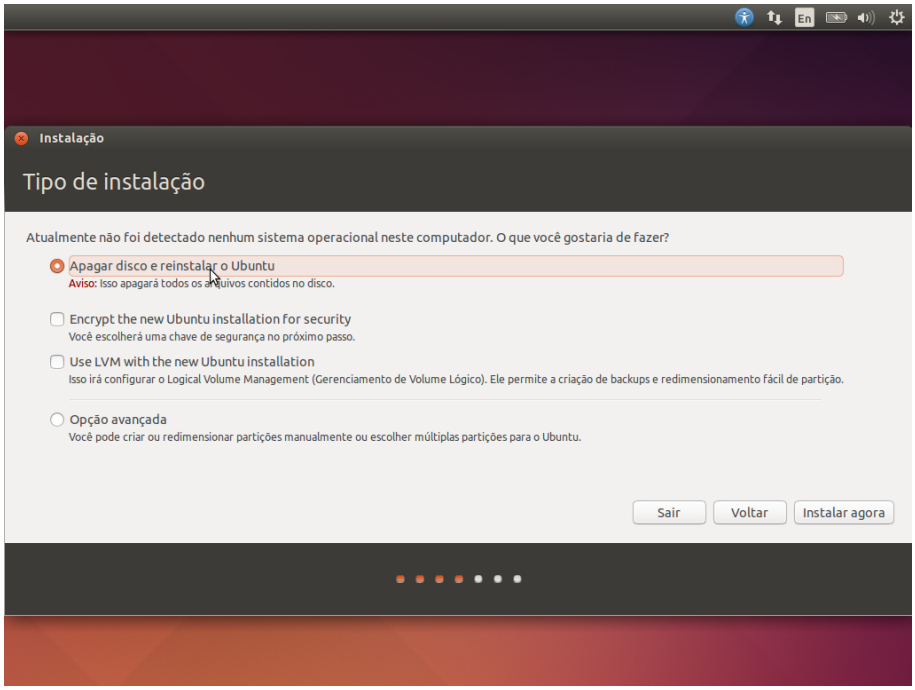

Figura 2.5: Tipo de instalação do Ubuntu 14.04

Passo 6 - Vamos agora definir nosso fuso horário.

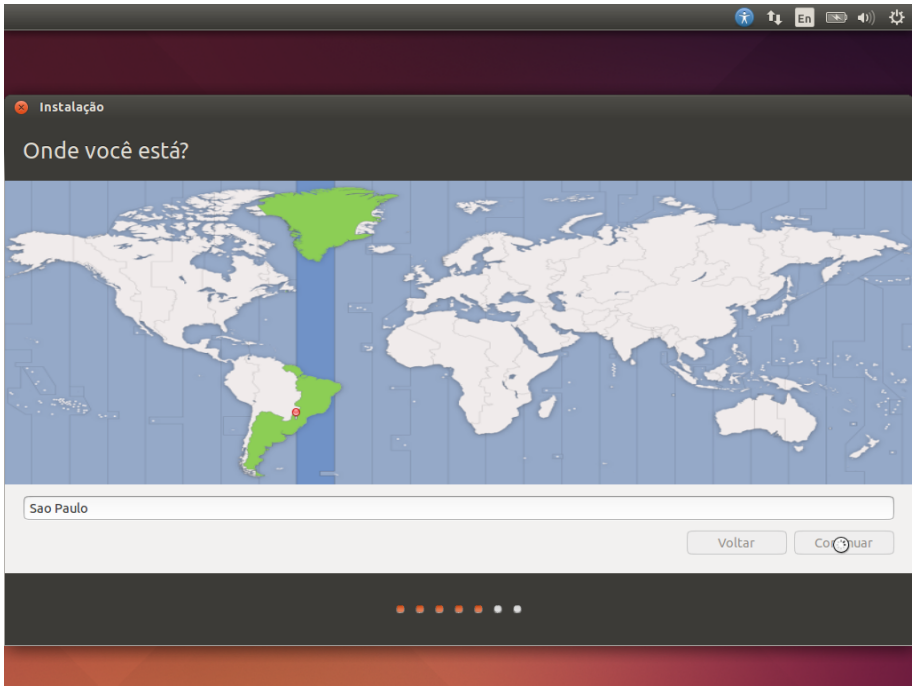

Figura 2.6: Janela de fuso horário do Ubuntu 14.04

Passo 7 - Agora indicamos o layout do teclado da máquina.

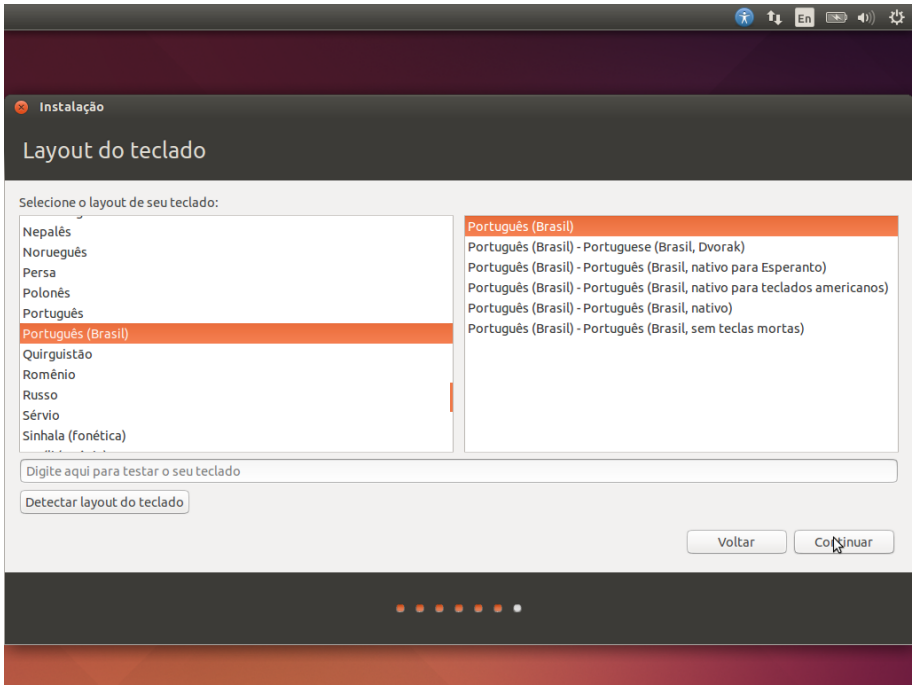

Figura 2.7: Layout do teclado no Ubuntu 14.04

Passo 8 - Nesse passo iremos ciar o usuário, assim como a senha. Será necessário também definir o nome da máquina.

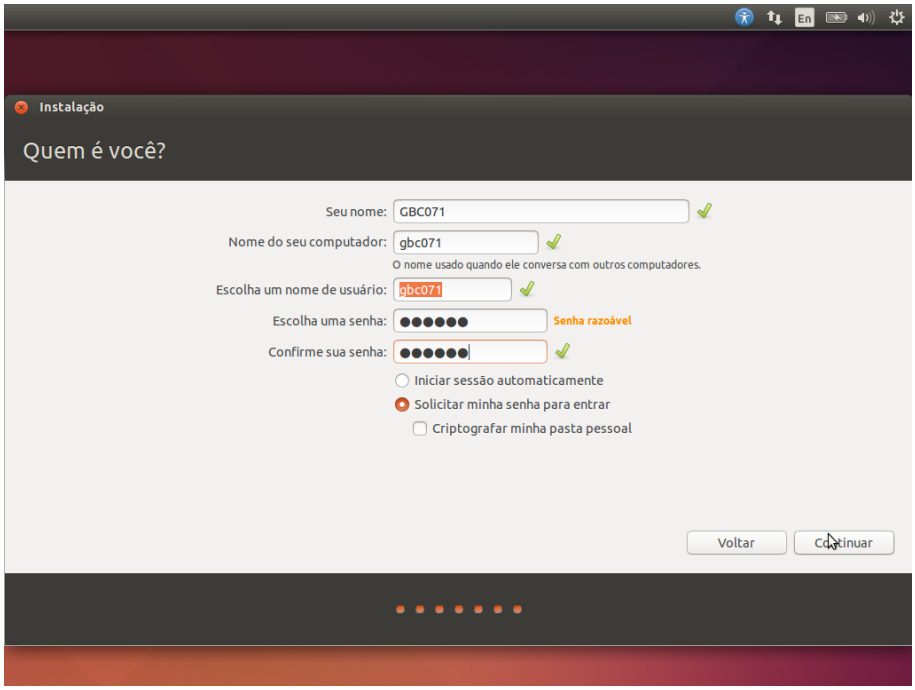

Figura 2.8: Identifique o usuário no Ubuntu 14.04

Passo 9 - Agora basta aguardar a instalação do Ubuntu 14.04 LTS na máquina. No final do processo será apresentado a seguinte tela, na qual será necessário "Reiniciar agora"a máquina.

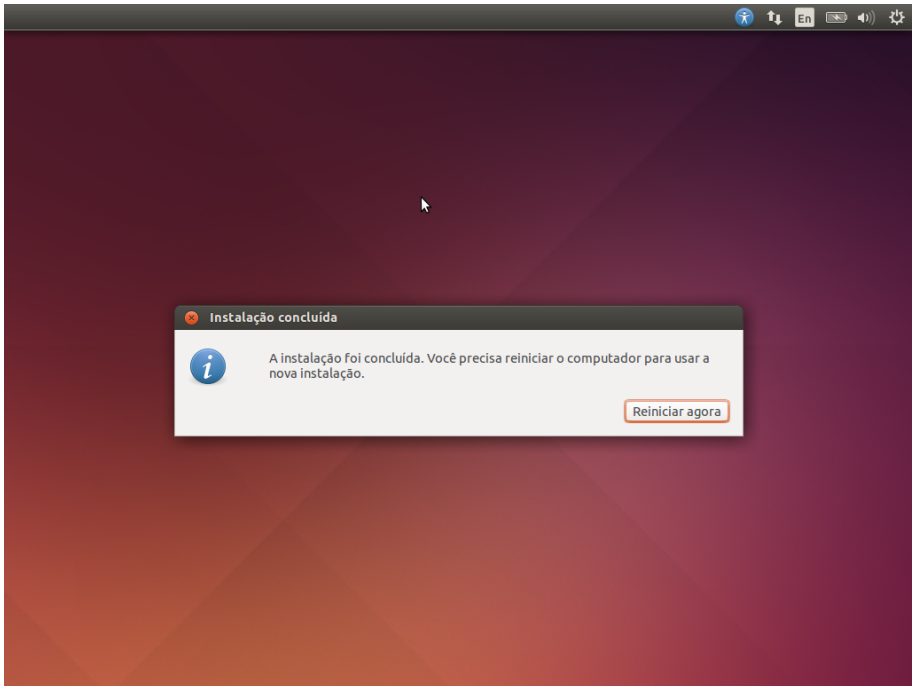

Figura 2.9: Final da instalação do Ubuntu 14.04

2.2 Dalvik

Apesar de as aplicações para Android serem escritas em Java, a Dalvik VM não pode ser considerada uma JVM, pois ela não interpreta Java bytecodes. Ao invés disso, a ferramenta *dx* transforma os arquivos .class compilados com um compilador Java normal em arquivos *.dex*, estes específicos para

execução na Dalvik. A Dalvik é otimizada para dispositivos com pouca memória, e foi desenhada de modo a permitir a execução de várias instâncias ao mesmo tempo de forma eficiente. Ela também deixa a cargo do kernel algumas tarefas, como o gerenciamento de memória e threads e para o isolamento entre processos.

Uma das medidas tomadas para salvar memória foi a de criar *constant pools* compartilhados para tipos específicos. Uma *constant pool* armazena valores literais constantes utilizados em uma classe, como strings e nomes de classes, interfaces, métodos e atributos. E ao utilizar *constant pools* compartilhados, ao invés de um para cada arquivo .class, como é feito com bytecode Java, evita-se a duplicação de constantes.

Outra estratégia utilizada para a otimização de aplicativos é o compartilhamento de código entre as instâncias da VM. Isso é feito através de um processo chamado Zygote, que é pai de todas as instâncias da Dalvik e é inicializado juntamente com o sistema, carregando todas as classes das bibliotecas que provavelmente serão utilizadas com frequência pelos aplicativos. Assim, quando uma aplicação for iniciada, um comando é enviado ao Zygote, que faz um *fork*, criando um processo que se torna uma nova Dalvik, minimizando o tempo de inicialização.

Em resumo, Dalvik VM é uma máquina virtual criada para o sistema Android e responsável por rodar os programas no mesmo.

Ela requer pouca memória e permite que várias instâncias (ou programas) sejam executados ao mesmo tempo. O sistema operacional Android cuida do isolamento dos programas, gerenciamento da memória e o controle de várias execuções ao mesmo tempo.

Alguns usuários acreditam que a máquina virtual Dalvik seja uma máquina virtual Java, mas isso não é verdade, pois o bytecode dela difere da JVM. Quando se cria um aplicativo Java em seu computador, a JVM executa tudo aquilo que foi compilado a partir do código-fonte. Este é geralmente o modo como o Java executa aplicativos nos populares sistemas operacionais e o Dalvik segue praticamente o mesmo roteiro no Android, mas difere no bytecode gerado.

2.2.1 Arquitetura Dalvik

Na realidade, o Android é mais do que um sistema operacional. Ele é na verdade um software stack composto por cinco camadas, como mostra a figura abaixo. A base do Android é uma versão modificado do kernel Linux 2.6, que provẽ vários serviços essenciais, como segurança, rede e gerenciamento de memória e processos, além de uma camada de abstração de hardware para as outras camadas de software.

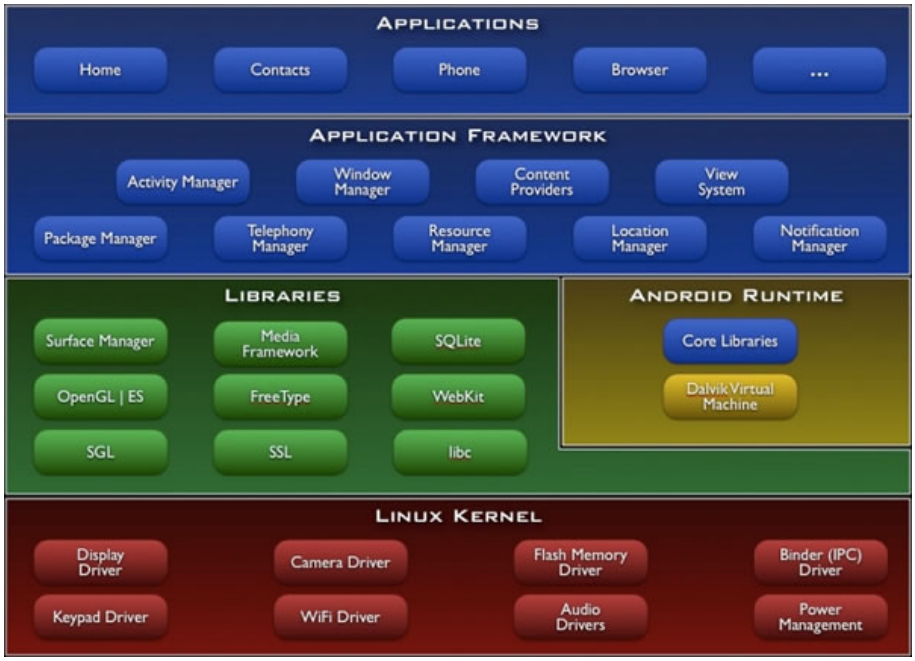

Figura 2.10: Arquitetura Dalvik

Acima do kernel ficam as bibliotecas C/C++ utilizadas por diversos dos componentes do sistema. No Android, aplicações escritas em Java são executadas em sua própria máquina virtual, que por sua vez é executada em seu próprio processo no Linux, isolando-a de outras aplicações e facilitando o controle de recursos. O Android Runtime é composto é composto pela máquina virtual Dalvik VM, onde as aplicações são executadas, e por um conjunto de bibliotecas que fornecem boa parte das funcionalidades encontradas nas bibliotecas padrão do Java. A Dalvik é baseada em registros.Ela é uma *process virtual machine*, o que quer dizer que a cada processo é gerada uma nova instância dela, diminuindo assim as chances de falhas se uma das instâncias dar erro.

Aplicações normalmente são compostas por atividades. No Android uma atividade é uma ação específica que um usuário pode realizar dentro de um aplicativo. Estas ações podem incluir a inicialização de outras atividades, tanto dentro como fora da aplicação a qual ela pertence, através de intenções, criando o que se chama de tarefa. Uma tarefa pode conter várias atividades, que são organizadas em uma pilha na ordem em que foram criadas, permitindo assim que o usuário volte para atividades em que estava anteriormente.

Quando uma nova tarefa é iniciada ou o usuário volta para a tela inicial, a tarefa anteriormente em primeiro plano passa para o segundo plano. Uma tarefa em segundo plano pode voltar ao primeiro plano e várias tarefas podem estar em segundo plano ao mesmo tempo. Entretanto, se muitas tarefas estiverem em segundo plano, o sistema pode começar a destruir atividades para recuperar memória, fazendo com que as atividades percam seus estados, o que não significa que o usuário não possa mais navegar de volta a esta atividade.

Todo o ciclo de vida das atividades, bem como a organização das mesmas em pilhas e tarefas é de responsabilidade do Gerenciador de Atividades (*Activity Manager*), visto na figura acima.

2.3 Python

Python é uma linguagem de programação dinâmica de altíssimo nível, orientada a objetos, de tipagem dinâmica e forte, interpretada e interativa. Uitlizada em larga escala por empresas como Google, Yahoo, Dreamworks e Industrial Light & Magic. No Brasil, é utilizada pela Locaweb, Globo.com, SERPRO, Interlegis (órgão vinculado ao Senado Federal), entre outros. Diversos softwares como GIMP, Inkscape e Blender3D utilizam a linguagem Python para extensões e criação de plugins.

Atualmente a linguagem Python possui um modelo de desenvolvimento comunitário, aberto e gerenciado pela organização Python Software Foundation. A terceira versão da linguagem foi lançada em dezembro de 2008, chamada Python 3.0.

2.3.1 Construções

Construções de Python incluem estruturas de seleção como *if*, *else*; estruturas de repetição como *for* e *while*; construção de subrotinas (*def*), entre outros recursos.

2.3.2 Tipos

Como dito acima, a tipagem do Python é forte, pois os valores e objetos têm tipos bem definidos e não sofrem coerções como em C. Alguns dos tipos nativos são: *int*, *float*, *bool*.

2.3.3 Palavras Reservadas

O Python 2.5.2 define 31 palavras reservadas, entre elas; *while*, *if*, *else*, *print*, *def*, *return*,*for*.

2.3.4 Operadores

No Python temos os operadores básicos de comparação, como ==, <, >=. Estes operadores são usados em todos os tipos de dados, como inteiros e floats. Estes operadores serão os que usaremos para construir nosso compilador, além dos operadores binários de soma, subtração, multiplicação e divisão.

Para conhecer mais sobre o Python, acesso o link: `https://www.python.org/`

2.3.5 Instalando o Python no Ubuntu 14.04

Vamos agora aprender a instalar o Python no nosso Ubuntu 14.04. A instalação é feita de maneira simples, rápida e fácil, basta seguir os seguintes passos e não terá problema na instalação:

Passo 1 - Abra um terminal usando o Dash ou as teclas **Ctrl+Alt+T**;
Passo 2 - Se ainda não tiver, adicione os seguintes repositórios com o comando abaixo:

```
$ sudo add-apt-repository ppa: fkrull/deadsnakes
```

Passo 3 - Atualize o APT com o comando abaixo:

```
$ sudo apt-get update
```

Passo 4 - Vamos instalar o Python 3.4.0, para isso digite no terminal o seguinte comando:

```
$ sudo apt-get install python3.4
```

Passo 5 - Finalmente, para definir o python instalado como padrão, basta executar os comandos a seguir:

```
$ rm /usr/local/bin/python
$ ln -s /usr/local/bin/python3.4 /usr/local/bin/python
```

Depois de instalado, para executar seu fonte em Python basta usar a seguinte linha de comando:

```
$ python <arquivo.py>
```

2.4 Java JDK

Para desenvolver em Android, precisaremos utilizar o Java JDK da Oracle.
O Ubuntu já vem com um repositório Java completo, mas pode ser que não contenha o que realmente precisamos. Para resolver isso, existe um recurso no Ubuntu que vamos usar a nosso favor, é o PPA (Personal Package Archive). O PPA é uma plataforma usada para disponibilizar softwares empacotados por desenvolvedores para usuários de Linux.

Basta adicionar um PPA, e atualizar o repositório de softwares. Ao fazer isso o novo repositório estará ativo. Para fazer a instalação do Java adequado para nosso uso, faça uso dos seguintes comandos no terminal:

```
$ sudo add-apt-repository ppa:webupd8team/java
$ sudo apt-get update
$ sudo apt-get install oracle-java7-installer
```

Caso queira confirmar a instalação, basta usar o seguinte comando:

```
$ java-version
```

O comando deverá retornar algo parecido com o que está abaixo, caso contrário sua instalação não deu certo. Para corrigir, basta tentar a instalação novamente como explicado acima.

```
java version "1.7.0_72"
Java(TM) SE Runtime Environment (build 1.7.0_72-b14)
Java HotSpot(TM) 64-Bit Server VM (build 24.72-b04, mixed mode)
```

Caso esteja usando um Ubuntu 64 bits será necessáio instalar alguma bibliotecas de 32 bits. Para isso use o comando abaixo:

```
$ sudo apt-get install libc6-i386 lib32stdc++6 lib32gcc1 lib32ncurses5
   lib32z1 ia32-libs
```

2.5 Android SDK

O Android SDK é o kit de desenvolvimento criado pelo Google para desenvolvedores de aplicativos para o sistema Android. No nosso projeto aqui apresentado não vamos desenvolver para Android, então porque a necessidade do Android SDK?

Usaremos o Android SDK para executar códigos .smali. Como faremos isso veremos mais a frente, enquanto isso, vamos executar a instalação do Android SDK:

Para fazer o download, podemos acessar o site oficial:
`https://developer.android.com/sdk/index.html?hl=i`, ou podemos usar o terminal:

```
$ cd ~
$ wget http://dl.google.com/android/adt/22.6.2/adt-bundle-linux-x86_64
   -20140321.zip
```

Apenas fizemos o download do SDK, agora iremos executar o passo a passo de instalação para podermos utilizar o Android SDK:

Primeiro precisamos descompactar o arquivo baixado:

```
$ unzip adt-bundle-linux-x86_64-20140321.zip
```

Agora iremos fazer as configurações necessárias, para isso iremos acessar a pasta tools, e executar o configurador SDK:

```
$ cd adt-bundle-linux-x86_64-20140321/sdk/tools
$ ./android
```

A janela abaixo será exibida. Ela contém alguns pacotes de ferramentas para Android, sendo que os pacotes padrões já estarão marcados, portanto basta clicar no botão "Install package...".

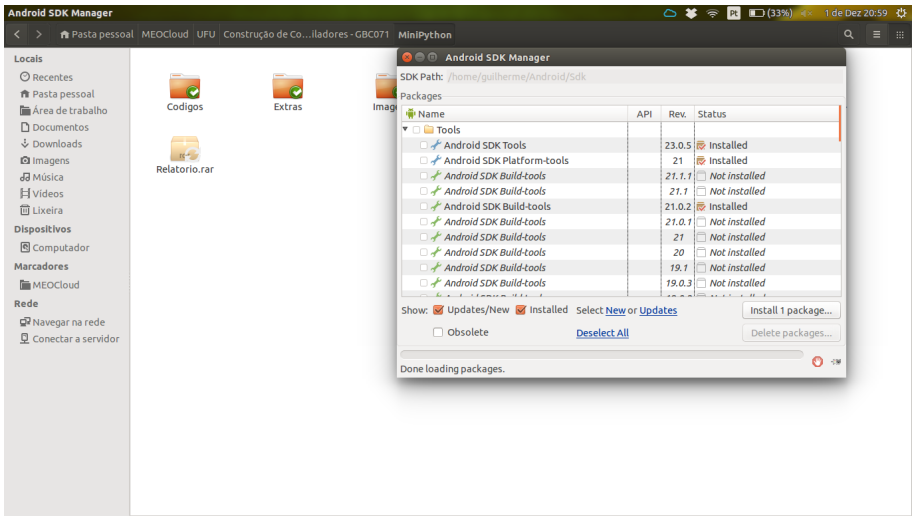

Figura 2.11: Android SDK Manager

2.5.1 Dispositivo Virtual Android - AVD

Não é necessário instalar uma aplicação em algum dispositivo real para testá-lo. Para facilitar os testes, existe o AVD (Android Virtual Device), que é um dispositivo virtual, afim de emular um aparelho real com o sistema Android, para que não seja necessário sempre instalar e reinstalar novas versões de uma aplicação que está em teste.

Para criar um novo dispositivo vitrual basta abrir o gerenciador de AVD, e clicar no botão "Create.."para criar um novo dispositivo virtual.

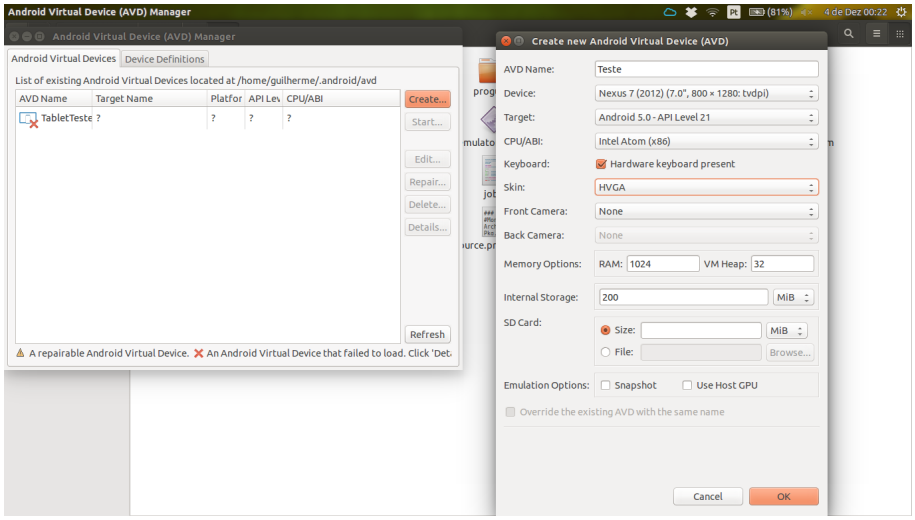

Figura 2.12: AVD Manager

2.6 Smali

O assembler para a máquina virtual Dalvik é o Smali. O Backsmali é usado para converter um arquivo .jar em um arquivo de código fonte .smali. A seguir temos um exemplo de um código fonte em python, e logo a seguir sua conversão para smali:

```
print(\"hello, world!\")
```

O programa equivalente em smali:

```
.class public LHelloWorld;

.super Ljava/lang/Object;

.method public static main([Ljava/lang/String;)V
        .registers 3

        sget-object v0, Ljava/lang/System;->out:Ljava/io/PrintStream;
        const-string v1, "hello, world!"
        invoke-virtual {v0, v1}, Ljava/io/PrintStream;->println(Ljava/lang/
            String;)V
        return-void
.end method
```

Vejamos mais alguns exemplos de fontes escritos em Smali, comentados para melhor entendimento da linguagem Smali:

```
# Nome da classe
.class public LExum;
# Nome da classe Pai
.super Ljava/lang/Object;

#Declaracao do metodo main
.method public static main([Ljava/lang/String;)V
        #Declara quantos registradores serao utilizados pelo metodo
        .registers 5

        #Armazena a referencia da classe em quesao em v0 de forma quesao
        #se possa usar metodos estaticos da classe atraves de v0
        sget-object v0, Ljava/lang/System;->out:Ljava/io/PrintStream;

        # Coloca o inteiro 1 registro v1
        const v1, 1

            # Coloca o inteiro 2 registro v2
            const v2, 2

            add-int/2addr v2, v1

            const v3, 3

            mul-int/2addr v3,v2

            add-int/2addr v3, v1

            const v4, 4

            sub-int/2addr v3, v4

        #Invoca um metodo com parametros v0 e v1 sendo o primeiro parametro
        #a instancia this, ou seja, o metodo chamado pertence a classe de v0
        #e v1 e o argumento do metodo chamado, no caso um inteiro
        invoke-virtual {v0, v3}, Ljava/io/PrintStream;->println(I)V

        #Retorno do metodo main
```

```
        return-void
#Fim do metodo
.end method
```

Listing 2.1: Smali 01

```
# Nome da classe
.class LExdois;

# Nome da classe Pai
.super Ljava/lang/Object;

# Esse e o metodo construtor da classe. Nesse exemplo ele apenas chama
# o construtor da classe Pai, que no caso e a classe Object
.method constructor <init>()V
        .registers 1

        invoke-direct {p0}, Ljava/lang/Object;-><init>()V

        return-void
.end method

#Comeco do metodo
.method public static main([Ljava/lang/String;)V
                #Declaracao de Registradres
                .registers 3

                #Cria uma instancia de Scanner em v0
                new-instance v0, Ljava/util/Scanner;

                #Cria uma referencia de InputStream em v1
                sget-object v1, Ljava/lang/System;->in:Ljava/io/InputStream;

                #Essa parte e equivalente ao new Scanner do java.]
                #0 <init> e a chamada para o construtor da classe.
                        invoke-direct {v0, v1}, Ljava/util/Scanner;-><init>(
                           Ljava/io/InputStream;)V

                #Chama a funcao nextInt() da classe Scanner instanciada em
                   v0
                invoke-virtual {v0}, Ljava/util/Scanner;->nextInt()I

                #v1 recebe o retorno da funcao anterior atraves do comando
                   move-result
                move-result v1

                #Cria a referencia de PrintStream em v2;
                sget-object v2, Ljava/lang/System;->out:Ljava/io/PrintStream
                   ;

                #Chamada do metodo println com parametro inteiro v1
                invoke-virtual {v2,v1}, Ljava/io/PrintStream;->println(I)V

                #retorno do metodo
                return-void
#Fim do metodo
.end method
```

Listing 2.2: Smali 02

```
# Nome da classe
.class LExtres;

# Nome da classe Pai
.super Ljava/lang/Object;

# Esse e o metodo construtor da classe. Nesse exemplo ele apenas chama
# o construtor da classe Pai, que no caso e a classe Object
.method constructor <init>()V
        .registers 1

        invoke-direct {p0}, Ljava/lang/Object;-><init>()V

        return-void
.end method

#Comeco do metodo
.method public static main([Ljava/lang/String;)V
                #Declaracao de Registradres
                .registers 6

                #Cria uma instancia de Scanner em v0
                new-instance v0, Ljava/util/Scanner;

                #Cria uma referencia de InputStream em v1
                sget-object v1, Ljava/lang/System;->in:Ljava/io/InputStream;

                #Essa parte e equivalente ao new Scanner do java.]
                #0 <init> e a chamada para o construtor da classe.
                        invoke-direct {v0, v1}, Ljava/util/Scanner;-><init>(
                           Ljava/io/InputStream;)V

                #Cria uma referencia de PrintStream em v5
                        sget-object v5, Ljava/lang/System;->out:Ljava/io/
                           PrintStream;

                #Armazena a String na variavel v2
                        const-string v2, "Digite um numero:"

                #Chamada do metodo println
                        invoke-virtual {v5, v2}, Ljava/io/PrintStream;->
                           println(Ljava/lang/String;)V

                #Chama a funcao nextInt() da classe Scanner instanciada em
                   v0
                invoke-virtual {v0}, Ljava/util/Scanner;->nextInt()I

                #v3 recebe o retorno da funcao anterior atraves do comando
                   move-result
                move-result v3

                #Atribuicao da String na variavel v2
                        const-string v2, "Digite outro numero:"

                #Chamada do metodo println
                invoke-virtual {v5,v2}, Ljava/io/PrintStream;->println(Ljava
                   /lang/String;)V
```

```
52
53                  #Chamada de metodo nextInt
54                          invoke-virtual {v0}, Ljava/util/Scanner;->nextInt()I
55
56                  #v4 recebe o retorno do ultimo metodo chamado
57                          move-result v4
58
59                  #Atribuicao da String na variavel v2
60                          const-string v2, "A soma dos numeros e: "
61
62                  #Chamada do metodo println
63                  invoke-virtual {v5,v2}, Ljava/io/PrintStream;->println(Ljava
                        /lang/String;)V
64
65                  #Armazena em v3 a soma de v3 e v4
66                          add-int/2addr v3, v4
67
68                  #Chamada do metodo println
69                  invoke-virtual {v5,v3}, Ljava/io/PrintStream;->println(I)V
70
71                  #retorno do metodo principal
72                  return-void
73
74  #Fim do metodo
75  .end method
```

Listing 2.3: Smali 03

```
1   # Nome da classe
2   .class LExquatro;
3   # Nome da classe Pai
4   .super Ljava/lang/Object;
5
6   #Construtor da classe
7   .method constructor <init>()V
8           .registers 1
9
10          invoke-direct {p0}, Ljava/lang/Object;-><init>()V
11
12          return-void
13  .end method
14
15  #Comeco do metodo
16  .method public static main([Ljava/lang/String;)V
17                  #Declaracao de Registradores
18                  .registers 6
19
20                  #Cria uma instancia de Scanner em v0
21                  new-instance v0, Ljava/util/Scanner;
22
23                  #Cria uma referencia de InputStream em v1
24                  sget-object v1, Ljava/lang/System;->in:Ljava/io/InputStream;
25
26                  #Essa parte e equivalente ao new Scanner do java.
27                          invoke-direct {v0, v1}, Ljava/util/Scanner;-><init>(
                                Ljava/io/InputStream;)V
28
29                  #Cria uma referencia de PrintStream em v5
```

```
30          sget-object v1, Ljava/lang/System;->out:Ljava/io/
                PrintStream;
31
32      #Armazena a String na variavel v2
33          const-string v2, "Digite um numero:"
34
35      #Chamada do metodo println
36          invoke-virtual {v1, v2}, Ljava/io/PrintStream;->
                println(Ljava/lang/String;)V
37
38      #Chama a funcao nextInt() da classe Scanner instanciada em
            v0
39      invoke-virtual {v0}, Ljava/util/Scanner;->nextInt()I
40
41      #v3 recebe o retorno da funcao anterior atraves do comando
            move-result
42      move-result v4
43
44      #Atribuicao da String na variavel v2
45          const-string v2, "Digite outro numero:"
46
47      #Chamada do metodo println
48      invoke-virtual {v1,v2}, Ljava/io/PrintStream;->println(Ljava
            /lang/String;)V
49
50      #Chamada de metodo nextInt
51          invoke-virtual {v0}, Ljava/util/Scanner;->nextInt()I
52
53      #v5 recebe o retorno do ultimo metodo chamado
54          move-result v5
55
56      #Atribuicao da String na variavel v2
57          const-string v2, "A subtracao dos numeros e: "
58
59      #Chamada do metodo println
60      invoke-virtual {v1,v2}, Ljava/io/PrintStream;->println(Ljava
            /lang/String;)V
61
62      #Armazena em v3 a subtracao de v4 e v5
63          sub-int v3, v4, v5
64
65      #Chamada do metodo println
66      invoke-virtual {v1,v3}, Ljava/io/PrintStream;->println(I)V
67
68      const-string v2, "A multiplicacao dos numeros e: "
69
70      invoke-virtual {v1, v2}, Ljava/io/PrintStream;->println(
            Ljava/lang/String;)V
71
72      #v3 recebe a multiplicacao de v4 e v5
73      mul-int v3, v4, v5
74
75      invoke-virtual {v1, v3}, Ljava/io/PrintStream;->println(I)V
76
77      const-string v2, "A divisao dos numeros e: "
78
79      invoke-virtual {v1, v2}, Ljava/io/PrintStream;->println(
            Ljava/lang/String;)V
```

```
80
81                  # v3 recebe a divisao de v4 e v5
82                  div-int v3, v4, v5
83
84                  invoke-virtual {v1, v3}, Ljava/io/PrintStream;->println(I)V
85
86                  #retorno do metodo principal
87                  return-void
88
89  #Fim do metodo
90  .end method
```

Listing 2.4: Smali 04

```
1   # Nome da classe
2   .class public LExcinco;
3   # Nome da classe Pai
4   .super Ljava/lang/Object;
5
6   #Declaracao do metodo main
7   .method public static main([Ljava/lang/String;)V
8       #Declara quantos registradores serao utilizados pelo metodo
9       .registers 5
10
11      #Armazena a referencia da classe em quesao em v0 de forma quesao
12      #se possa usar metodos estaticos da classe atraves de v0
13      sget-object v0, Ljava/lang/System;->out:Ljava/io/PrintStream;
14
15              # Coloca o inteiro 1 registro v1
16              const v1, 1
17
18              # Coloca o inteiro 2 registro v2
19              const v2, 2
20
21              add-int/2addr v2, v1
22
23              const v3, 32
24
25              const v4, 0
26
27              if-it v2, v3, target 39
28
29
30
31      #Invoca um metodo com parametros v0 e v1 sendo o primeiro parametro
32      #a instancia this, ou seja, o metodo chamado pertence a classe de v0
33      #e v1 e o argumento do metodo chamado, no caso um inteiro
34      invoke-virtual {v0, v4}, Ljava/io/PrintStream;->println(I)V
35
36      #Retorno do metodo main
37      return-void
38  #Fim do metodo
39  .end method
```

Listing 2.5: Smali 05

```
1   # Nome da classe
2   .class LExsete;
3   # Nome da classe Pai
```

```
4  .super Ljava/lang/Object;
5
6  # Esse e o metodo construtor da classe. Nesse exemplo ele apenas chama
7  # o construtor da classe Pai, que no caso e a classe Object
8  .method constructor <init>()V
9          .registers 1
10         invoke-direct {p0}, Ljava/lang/Object;-><init>()V
11         return-void
12 .end method
13
14 #Comeco do metodo
15 .method public static main([Ljava/lang/String;)V
16                 #Declaracao de Registradores
17                 .registers 8
18
19                 #Cria uma instancia de Scanner em v0
20                 new-instance v0, Ljava/util/Scanner;
21
22                 #Cria uma referencia de InputStream em v1
23                 sget-object v1, Ljava/lang/System;->in:Ljava/io/InputStream;
24
25                 #Essa parte e equivalente ao new Scanner do java.]
26                 #0 <init> e a chamada para o construtor da classe.
27                         invoke-direct {v0, v1}, Ljava/util/Scanner;-><init>(
                               Ljava/io/InputStream;)V
28
29                 sget-object v5, Ljava/lang/System;->out:Ljava/io/PrintStream
                       ;
30
31                 const-string v2, "a"
32
33                 invoke-virtual {v5, v2}, Ljava/io/PrintStream;->println(
                       Ljava/lang/String;)V
34
35
36                 const-string v4, "a"
37
38                 invoke-virtual{v4, v2}, Ljava/lang/String;->equals(Ljava/
                       lang/Object;)Z
39
40                 move-result v6
41
42                 if-eqz v6, :goto_fim
43                         sget-object v7, Ljava/lang/System;->out:Ljava/io/
                               PrintStream;
44                         const-string v2, "a"
45                         invoke-virtual {v7, v2}, Ljava/io/PrintStream;->
                               println(Ljava/lang/String;)V
46                 :goto_fim
47
48                 return-void
49 .end method
```

Listing 2.6: Smali 06

2.6.1 Como instalar o Smali e Backsmali

Para executar a instalação do Smali, você pode acessar o site: `https://code.google.com/p/smali/`, ou pode instalar diretamente pelo terminal, com a seguinte sequência de comandos:

```
$ cd ~
$ mkdir Dalvik
$ cd Dalvik
$ wget https://bitbucket.org/JesusFreke/smali/downloads/smali-2.0.3.jar
$ wget https://bitbucket.org/JesusFreke/smali/downloads/baksmali-2.0.3.jar
```

Pronto, agora basta usar o Smali para executar seus arquivos .smali.

Para utilizar o Smali, considerando termos o arquivo .smali salvo no mesmo diretório do smali-2.0.3.jar, execute os seguintes comandos:

```
$ cd ~/Dalvik
$ java -jar smali-2.0.3.jar -o classes.dex <arquivo>.smali
$ zip <arquivo>.zip classes.dex
```

Após estes primeiros comandos, necessitaremos de um dispositivo Android ligado ao computador através de conexão USB, e com o modo Depuração USB ativado (opção é ativada ou desativada na opção de configuração "Opções de desenvolvedor"), ou utilizando um emulador. Considerando um emulador com o nome *myEmulator* que deve ser criado como explicado anteriormente, execute o seguinte comando para abrir este emulador:

```
$ emulator -avd myEmulator &
```

Agora vamos utilizar o Android Debug Bridge (ADB) para copiar o arquivo zipado para dentro do sistema de arquivos da AVD que está sendo emulada:

```
$ adb push <arquivo>.zip /data/local
```

Agora basta executar o programa na AVD com os comandos abaixo:

```
$ adb shell
$ dalvikvm -cp /data/local/<arquivo>.zip <arquivo>
```

O backsmali.jar tem a função de desmontar (disassembles) o formato executável Dalvik (.dex) para uma versão de leitura, ou seja, para bytecode, um arquivo .smali. Para gerar o arquivo .smali a partir de um código java, com a ajuda do backsmali, utilizamos os comandos abaixo, considerando que o arquivo .java está pronto e compilado, no diretório do backsmali.

```
$ dx --dex --output=arquivo.dex <arquivo>.class
$ dx --dex --no-optimize --output=<arquivo>.dex <arquivo>.class
```

Como é bem visível nos dois comandos, a diferença entre eles é o termo *–no-optimize*, que faz como que o código não seja otimizado, ou seja, utilizando este comando conseguiremos visualizar como todos os registradores foram utilizados. Os comandos até o momento geraram um arquivo .class e um arquivo .dex. Agora, usamos o backsmali para converter o arquivo .dex em um arquivo .smali:

```
$ java -jar backsmali-2.0.3.jar -x <arquivo>.dex -o <arquivo>
```

Assim, poderemos verificar que foi criado no diretório corrente, um arquivo.smali correspondente ao arquivo.java criado anteriormente.

2.6.2 Dicionário Smali

A seguir apresentaremos alguns exemplos de sintaxe do Smali.

Opcodes Básicos

Nome	Explicação
.method	Inicia um método.
.endthod	Termina um método.
.register x	Declara x registros
new-instance vx, Ljava/ ...;	Instancia uma classe
invoke-virtual vx,.., vz Ljava/ ...	Executa um método
move-result vX	Armazena o retorno do último método executado em vX.
sget-object vX, Ljava/ ...;	Cria uma referência de uma classe em vX
move vX, vY	Move vY para vX. (vX = vY).
const-string vX, String	Armazena uma String em vX.
const vX, Valor	Armazena um valor inteiro em vX.
goto :area	Vai para uma área do programa pré-definida
return-void	Retorno para funções que retornam void.

Operadores

Nome	Explicação
add-int /2addr vX, vY	vX = vX + vY
mul-int /2addr	vX = vX * vY
div-int /2addr x	vX = vX / vY
sub-int /2addr vX, vY, vZ	vX = vX - vY
rem-int /2addr vX, vY, vZ	vX = vX % vY
add-float /2addr vX, vY	vX = vX + vY (Float)
mul-float /2addr vX, vY	vX = vX * vY (Float)
div-float /2addr vX, vY	vX = vX / vY (Float)
sub-float /2addr vX, vY	vX = vX - vY (Float)

Operadores condicionais

Nome	Explicação
if-eq vX, vY, target	Se vX = vY, vai para target
if-ne vX, vY, target	Se vX != vY, vai para target
if-lt vX, vY, target x	Se vX < vY, vai para target
if-ge vX, vY, target	se vX >= vY, vai para target
if-gt vX, vY, target	Se vX > vY, vai para target
if-le vX, vY, target	Se vX <= vY, vai para target
if-eqz vX, target	Se vX = 0, vai para target
if-nez vX, target	Se vX != 0, vai para target
if-ltz vX, target	Se vX < 0, vai para target
if-gez vX, target	Se vX >= 0, vai para target
if-gtz vX, target	Se vX > 0, vai para target
if-lez vX, target	Se vX <= 0, vai para target

Todas as operações acima são realizadas considerando valores inteiros.

Capítulo 3

Python

Vamos agora descrever com mais detalhes a linguagem de programação Python. O MiniPython é um subconjunto da linguagem Python, o que quer dizer que todo programa MiniPython é um programa Python, mas com algumas características a menos. Para escrevermos um compilador para essa linguagem, precisaremos antes conhecer sua gramática, que será apresentado a seguir.

3.1 Gramática Completa

Logo abaixo encontra-se a gramática completa da linguagem Python, lembrando que não será usados toda a especificação, mas será a base para toda a implementação do nosso compilador.

```
# Grammar for Python

# Note:  Changing the grammar specified in this file will most likely
#        require corresponding changes in the parser module
#        (../Modules/parsermodule.c).  If you can't make the changes to
#        that module yourself, please co-ordinate the required changes
#        with someone who can; ask around on python-dev for help.  Fred
#        Drake <fdrake@acm.org> will probably be listening there.

# NOTE WELL: You should also follow all the steps listed in PEP 306,
# "How to Change Python's Grammar"

# Start symbols for the grammar:
#       single_input is a single interactive statement;
#       file_input is a module or sequence of commands read from an input
    file;
#       eval_input is the input for the eval() functions.
# NB: compound_stmt in single_input is followed by extra NEWLINE!
single_input: NEWLINE | simple_stmt | compound_stmt NEWLINE
file_input: (NEWLINE | stmt)* ENDMARKER
eval_input: testlist NEWLINE* ENDMARKER

decorator: '@' dotted_name [ '(' [arglist] ')' ] NEWLINE
decorators: decorator+
decorated: decorators (classdef | funcdef)
funcdef: 'def' NAME parameters ['->' test] ':' suite
parameters: '(' [typedargslist] ')'
typedargslist: (tfpdef ['=' test] (',' tfpdef ['=' test])* [','
       ['*' [tfpdef] (',' tfpdef ['=' test])* [',' '**' tfpdef] | '**'
          tfpdef]]
     |  '*' [tfpdef] (',' tfpdef ['=' test])* [',' '**' tfpdef] | '**'
        tfpdef)
```

```
tfpdef: NAME [':' test]
varargslist: (vfpdef ['=' test] (',' vfpdef ['=' test])* [','
       ['*' [vfpdef] (',' vfpdef ['=' test])* [',' '**' vfpdef] | '**'
          vfpdef]]
     |  '*' [vfpdef] (',' vfpdef ['=' test])* [',' '**' vfpdef] | '**'
         vfpdef)
vfpdef: NAME

stmt: simple_stmt | compound_stmt
simple_stmt: small_stmt (';' small_stmt)* [';'] NEWLINE
small_stmt: (expr_stmt | del_stmt | pass_stmt | flow_stmt |
             import_stmt | global_stmt | nonlocal_stmt | assert_stmt)
expr_stmt: testlist_star_expr (augassign (yield_expr|testlist) |
                    ('=' (yield_expr|testlist_star_expr))*)
testlist_star_expr: (test|star_expr) (',' (test|star_expr))* [',']
augassign: ('+=' | '-=' | '*=' | '/=' | '%=' | '&=' | '|=' | '^=' |
            '<<=' | '>>=' | '**=' | '//=')
# For normal assignments, additional restrictions enforced by the
   interpreter
del_stmt: 'del' exprlist
pass_stmt: 'pass'
flow_stmt: break_stmt | continue_stmt | return_stmt | raise_stmt |
   yield_stmt
break_stmt: 'break'
continue_stmt: 'continue'
return_stmt: 'return' [testlist]
yield_stmt: yield_expr
raise_stmt: 'raise' [test ['from' test]]
import_stmt: import_name | import_from
import_name: 'import' dotted_as_names
# note below: the ('.' | '...') is necessary because '...' is tokenized as
   ELLIPSIS
import_from: ('from' (('.' | '...')* dotted_name | ('.' | '...')+)
              'import' ('*' | '(' import_as_names ')' | import_as_names))
import_as_name: NAME ['as' NAME]
dotted_as_name: dotted_name ['as' NAME]
import_as_names: import_as_name (',' import_as_name)* [',']
dotted_as_names: dotted_as_name (',' dotted_as_name)*
dotted_name: NAME ('.' NAME)*
global_stmt: 'global' NAME (',' NAME)*
nonlocal_stmt: 'nonlocal' NAME (',' NAME)*
assert_stmt: 'assert' test [',' test]

compound_stmt: if_stmt | while_stmt | for_stmt | try_stmt | with_stmt |
   funcdef | classdef | decorated
if_stmt: 'if' test ':' suite ('elif' test ':' suite)* ['else' ':' suite]
while_stmt: 'while' test ':' suite ['else' ':' suite]
for_stmt: 'for' exprlist 'in' testlist ':' suite ['else' ':' suite]
try_stmt: ('try' ':' suite
           ((except_clause ':' suite)+
            ['else' ':' suite]
            ['finally' ':' suite] |
           'finally' ':' suite))
with_stmt: 'with' with_item (',' with_item)*  ':' suite
with_item: test ['as' expr]
# NB compile.c makes sure that the default except clause is last
except_clause: 'except' [test ['as' NAME]]
suite: simple_stmt | NEWLINE INDENT stmt+ DEDENT
```

```
82
83  test: or_test ['if' or_test 'else' test] | lambdef
84  test_nocond: or_test | lambdef_nocond
85  lambdef: 'lambda' [varargslist] ':' test
86  lambdef_nocond: 'lambda' [varargslist] ':' test_nocond
87  or_test: and_test ('or' and_test)*
88  and_test: not_test ('and' not_test)*
89  not_test: 'not' not_test | comparison
90  comparison: expr (comp_op expr)*
91  # <> isn't actually a valid comparison operator in Python. It's here for the
92  # sake of a __future__ import described in PEP 401
93  comp_op: '<'|'>'|'=='|'>='|'<='|'<>'|'!='|'in'|'not' 'in'|'is'|'is' 'not'
94  star_expr: '*' expr
95  expr: xor_expr ('|' xor_expr)*
96  xor_expr: and_expr ('^' and_expr)*
97  and_expr: shift_expr ('&' shift_expr)*
98  shift_expr: arith_expr (('<<'|'>>') arith_expr)*
99  arith_expr: term (('+'|'-') term)*
100 term: factor (('*'|'/'|'%'|'//') factor)*
101 factor: ('+'|'-'|'~') factor | power
102 power: atom trailer* ['**' factor]
103 atom: ('(' [yield_expr|testlist_comp] ')' |
104        '[' [testlist_comp] ']' |
105        '{' [dictorsetmaker] '}' |
106        NAME | NUMBER | STRING+ | '...' | 'None' | 'True' | 'False')
107 testlist_comp: (test|star_expr) ( comp_for | (',' (test|star_expr))* [','] )
108 trailer: '(' [arglist] ')' | '[' subscriptlist ']' | '.' NAME
109 subscriptlist: subscript (',' subscript)* [',']
110 subscript: test | [test] ':' [test] [sliceop]
111 sliceop: ':' [test]
112 exprlist: (expr|star_expr) (',' (expr|star_expr))* [',']
113 testlist: test (',' test)* [',']
114 dictorsetmaker: ( (test ':' test (comp_for | (',' test ':' test)* [','])) |
115                  (test (comp_for | (',' test)* [','])) )
116
117 classdef: 'class' NAME ['(' [arglist] ')'] ':' suite
118
119 arglist: (argument ',')* (argument [',']
120                          |'*' test (',' argument)* [',' '**' test]
121                          |'**' test)
122 # The reason that keywords are test nodes instead of NAME is that using NAME
123 # results in an ambiguity. ast.c makes sure it's a NAME.
124 argument: test [comp_for] | test '=' test  # Really [keyword '='] test
125 comp_iter: comp_for | comp_if
126 comp_for: 'for' exprlist 'in' or_test [comp_iter]
127 comp_if: 'if' test_nocond [comp_iter]
128
129 # not used in grammar, but may appear in "node" passed from Parser to
    Compiler
130 encoding_decl: NAME
131
132 yield_expr: 'yield' [yield_arg]
133 yield_arg: 'from' test | testlist
```

Listing 3.1: Especificação da Gramática Completa do Python

3.2 Exemplos Python

```
def contador(a, b):
    soma = 0
    for i in range(5):
        soma = soma + a + b
    return soma
def imprimeSoma():
    x = raw_input("Digite o primeiro numero: ")
    x = int(x)
    y = raw_input("Digite o segundo numero: ")
    y = int(y)
    soma = x + y
    print(soma)
count = contador(2, 3)
imprimeSoma()
```

Listing 3.2: Soma de dois inteiro em Python

```
numero = input("Digite um numero: ")
num = int(numero)
if 100 <= num <= 200:
    print("O numero esta no intervalo entre 100 e 200")
else:
    print("O numero nao esta no intervalo entre 100 e 200")
```

Listing 3.3: Verifica se valor está entre 100 e 200 em Python

```
num = 1
while num != 0:
    num = input("Digite um num:")
    num = int(num)
    if num > 10:
        print("O numero %d eh maior que 10" % num)
    else:
        print("O numero %d eh menor que 10" % num)
```

Listing 3.4: Verifica se valor é maior que 10

Capítulo 4

Construção do Compilador Python

4.1 Modificações na linguagem Python

Nossa linguagem pode ser dita baseada em Python, pois foram realizadas algumas modificações na mesma com o objetivo de facilitar a construção e entendimento do compilador, assim como a escrita e interpretação dos fontes dispostos em tal linguagem. Uma das modificações realizadas diz respeito a definição de funções, enquanto na linguagem original python a definição de função não necessita definir o tipo dos parâmetros, na nossa linguagem, decidimos acrescentar o tipo do parâmetro para que assim seja possível identificar este tipo com maior facilidade, além de mostrar qual o tipo do retorno da função. Em nível de código, a definição de uma função em Python deve ser da seguinte maneira:

```
def soma(a,b): # a e b sao os parametros da funcao
  soma = a + b
  return soma
```

Enquanto que na linguagem definida, teremos a mesma função definida da seguinte maneira:

```
def soma(a: int,b: int) -> int
  soma = a + b
  return soma
```

Veja abaixo mais alguns exemplos de fontes na nossa linguagem Python modificada:

```
num1 = 150
num2 = 100
if num1 == num2:
     print("Os numeros sao iguais")
else:
     print("Os numeros sao diferentes")
```

Listing 4.1: MiniPython modificado 1

```
num1 = 150
num2 = 100
if num1 > num2:
     print("O primeiro numero eh maior que o segundo")
else:
     print("O segundo numero eh maior que o primeiro")
```

Listing 4.2: MiniPython modificado 2

```
def contador(a: int) -> int:
    if a < 10:
       print("o numero eh menor do que 10")
    else:
```

```
5         print("o numero eh maior do que 10")
6     return a
```

Listing 4.3: MiniPython modificado 3

```
1 def retorna()->void:
2     print("teste")
3
4 retorna()
```

Listing 4.4: MiniPython modificado 4

4.2 OCaml

O OCaml (Objective Caml) é uma linguagem de programação funcional e imperativa da família ML, fortemente e estaticamente tipada. Trata-se da linguagem Caml com a adição de suporte de técnicas de orientação a objetos e algumas alterações e extensões de sintaxe.

Por ser uma linguagem herdeira de ML, possui várias características destas linguagens, entre elas:

- As funções são valores de "primeira classe": estas podem ser argumentos de outras funções ou mesmo o resultado de um cálculo;
- A linguagem é fortemente tipada: a qualquer valor está associado um tipo. A verificação sistemática e rigorosa da compatibilidade entre os tipos dos parâmetros formais e os tipos dos parâmetros efetivos é uma das características da linguagem;
- A tipagem é inferida automaticamente: o sistema de tipo do OCaml é particularmente expressivo e o problema de inferência de tipo continua no entanto decidível. Ou seja, não temos que declarar o tipo das expressões que definimos;
- A tipagem é estática: todos os tipos podem ser deduzidos na fase de compilação;
- A tipagem é polimórfica: o algoritmo de inferência de tipo do OCaml sabe reconhecer tipos polimórficos.

O OCaml permite dois tipos de compilação, para bytecode que corre numa máquina virtual ou para código de máquina nativo para um grande número de plataformas. Ela não é uma linguagem puramente funcional, permitindo a existência de valores mutáveis bem como de efeitos colaterais, tipicamente existentes apenas em linguagem imperativas. Esta característica distingue-a de outras linguagens puramente funcionais, e isto será ótimo para a construção do nosso compilador.

Essa será a linguagem que utilizaremos para construir o nosso compilador para a nossa linguagem MiniPython. Para instalar o OCaml no Ubuntu 14.04 basta usar o seguinte comando no Terminal:

```
1 $ sudo apt-get install ocaml
```

Depois de instalado, basta criar seu programa e salvá-lo com a extensão `.ml` e compilar. Para compilar um fonte.ml, iremos entrar no OCaml pelo terminal, e em seguida executar o comando `#use`. Siga o exemplo abaixo:

```
1 $ rlwrap ocaml
2                 OCaml version 4.01.0
3
4 # #use "teste.ml";;
```

Usamos o comando `rlwrap` antes do comando que abre o OCaml, esse comando faz com que o OCaml grave comandos digitados anteriormente, podendo estes serem acessados usando a seta para cima sem a necessidade de digitar o comando novamente.

4.3 Árvore Sintática Abstrata

O primeiro passo para a construção de um compilador é definir a Árvore Sintática Abstrata (ASA), que será utilizada pelo Analisador Sintático. Esta árvore é uma estrutura de dados em árvore que representam estruturas sintáticas de cadeias, de acordo com alguma gramático formal, porém os nós são diretamente valorados em seus símbolos terminais, não havendo uma representação de derivações por meio dos símbolos não terminais.

É uma representação simplificada da estrutura semântica de um fonte escrito em uma determinada linguagem, no nosso caso veremos uma árvore abstrata para a linguagem MiniPython, lembrando que esta linguagem foi levemente modificada. O código está abaixo:

```
1
2  (* Estrutura que define uma posicao*)
3  module Posicao =
4      struct
5          type t = { lin_inicial: int ;
6                     col_inicial: int ;
7                     lin_final: int ;
8                     col_final: int
9                                  }
10 let pos n =
11     let pos_inicial = Parsing.rhs_start_pos n in
12     let pos_final = Parsing.rhs_end_pos n in
13     let linha_inicial = pos_inicial.Lexing.pos_lnum
14
15 and coluna_inicial = pos_inicial.Lexing.pos_cnum - pos_inicial.Lexing.
       pos_bol + 1
16 and linha_final = pos_final.Lexing.pos_lnum
17 and coluna_final = pos_final.Lexing.pos_cnum - pos_final.Lexing.pos_bol in
18     {  lin_inicial = linha_inicial;
19        col_inicial = coluna_inicial;
20        lin_final = linha_final;
21        col_final = coluna_final
22     }
23 let npos n =
24     {  lin_inicial = Parsing.rhs_start(n);
25         col_inicial = Parsing.rhs_end(n);
26         lin_final = 0;
27         col_final = 0
28     }
29 end
30
31 (* Tipos base *)
32 type tipo_base =  TInt
33                           |TFloat
34                           |TString
35                           |TVoid(*
36                           |TGen *)
37                                    |TBool
38
39 (* Definicao de um comando e uma lista de comandos*)
40 and comandos = comando list
41 and comando = { vcmd: cmd;
42                         pcmd: Posicao.t
43                         }
44
45 (* Definicao de um parametro e uma lista de parametros*)
46 (* and parametro =  { entradaP: string * entradaVariavel} *)
```

```
47
48  and parametro = string * entradaVariavel
49
50  and parametros = parametro list
51
52  (* Definicao de um argumento e uma lista de argumentos*)
53  and argumento = expressao
54  and argumentos = argumento list
55
56  (* Definicao de uma funcao *)
57  and funcao = {    idF: string;
58                            paramsF: parametros;
59                            cmdsF: comandos;
60                            mutable returnF: tipo_base option;
61                            posF: Posicao.t;
62                            mutable varLocaisF:(string, entradaVariavel)
                                 Hashtbl.t;
63                        }
64
65  (* Definicao de um programa *)
66  and programa = {  funcsP: funcao list;
67                              cmdsP: comandos
68                 }
69
70  (* Tipos de comandos *)
71  and cmd = CmdAtrib of expressao * expressao
72          | CmdIf of expressao * comandos * comandos option
73          | CmdWhile of expressao * comandos
74          | CmdRange of int * int * int
75          | CmdFor of expressao * comando * comandos
76          | CmdFuncao of string * parametros * comandos
77          | ChamaFuncaoVoid of string * argumentos
78          | ChamaFuncaoAtrib of expressao * string * argumentos
79          | CmdReturn of expressao
80          | CmdPrint of expressao
81          | CmdInput of expressao * expressao
82          | CmdIntParse of expressao * expressao
83
84  (* Definicao de uma expressao *)
85  and expressao = {  mutable valor: expr option ;
86                                  mutable tipo: tipo_base option;
87                                  mutable pos: Posicao.t;
88                 }
89
90  (* Tipos de valor de uma expressao *)
91  and expr =   ExpInt of int
92                       | ExpFloat of float
93                       | ExpString of string
94                       | ExpVar of variavel
95                       | ExpBin of operadorBin * expressao * expressao(*
96                       | ExpGen *)
97                     | ExpBool of bool
98                     | ExpUn of operadorUn * expressao
99
100 (* Definicao de uma variavel *)
101 and variavel = VarSimples of string
102
103 and operadorUn = Not
```

```
104
105 (* Tipos de operadores binarios *)
106 and operadorBin = Mais | Menos | Mult | Div | Maior | Menor | Igual |
    Diferente | MaiorIgual | MenorIgual | Modulo | And | Or
107
108 (*TABELA DE SIMBOLOS*)
109 (* Definicao da tabela de simbolos geral *)
110 and ambiente = (string, entradaTabela) Hashtbl.t
111
112 (* Entrada de uma funcao na tabela de simbolos *)
113 and entradaFuncao = {  varLocais: (string, entradaVariavel) Hashtbl.t;
114                                   mutable tiporetorno: tipo_base option;
115                                   param: parametros
116                                       }
117
118 (* Entrada de uma variavel na tabela de simbolos *)
119 and entradaVariavel = {  mutable tipagem: tipo_base option;
120                                     v_inicial: expressao option;
121                                     mutable endereco: int option;
122                                     mutable valor_variavel: int
123                                         }
124
125 (* Tipos de entrada que a tabela de simbolos aceita *)
126 and entradaTabela = EntVar of entradaVariavel
127                               | EntFn of entradaFuncao
128
129
130
131
132
133 (* Cria uma entrada do tipo variavel para a tabela de simbolos de acordo com
    o tipo recebido *)
134 let cria_ent_var tipo = {  tipagem = tipo ;
135                                     v_inicial = None;
136                                     endereco = None;
137                                     valor_variavel = 0 }
138
139 (* Cria uma entrada do tipo funcao para a tabela de simbolos de acordo com o
    tipo e parametros recebidos *)
140 let cria_ent_func tipo par locais= { varLocais = locais;
141                                                  tiporetorno = Some tipo;
142                                                  param = par }
143
144 (* let cria_ent_param nome tipo = {  idP = nome;
145                                                 tipoP =  tipo;
146                                                 posP = {  lin_inicial =
                                                    0;
147                                                    col_inicial = 0;
148                                                    lin_final = 0;
149                                                    col_final = 0
150                                                 };
151                                                 valor_param = None
152                                             } *)
153
154 (* Procura um parametro numa lista de parametros *)
155 let rec procuraParam nome params =
156     match params with
157         [] -> None
```

```
158     | param :: params ->
159         if ((fst param) <> nome) then(
160            procuraParam nome params)
161         else
162            Some (snd param)
163
164
165 let busca_var_fun amb regfn nome =
166   match regfn with
167   EntFn regfn ->
168     (try (* tenta encontrar variavel local *)
169          Hashtbl.find regfn.varLocais nome
170      with Not_found -> (* tenta encontrar parametro *)
171            (match (procuraParam nome regfn.param) with
172                 Some v -> v
173               | None -> (* tenta encontrar variavel global *)
174                 (match (Hashtbl.find amb nome) with
175                   EntVar v -> v
176                   | _ -> failwith "busca_var_fun: erro"
177                 )
178             )
179     )
180     | _ -> failwith "busca_var_fun: erro"
181
182 (* let substitui amb reg nome=
183   let entrada = busca_var_fun amb reg nome in
184       Hashtbl.replace amb nome {entrada with valor_variavel = 2} *)
185
186
187
188 (* let busca_tipo_fun amb regfn nome =
189   match regfn with
190   EntFn regfn ->
191     (try (* tenta encontrar variavel local *)
192          Hashtbl.find regfn.varLocais nome
193             with Not_found -> (* tenta encontrar parametro *)
194                               (match (procuraParam nome regfn.param) with
195                                    Some v -> v;
196                                    v.tipagem
197                                  | None -> tenta encontrar variavel
                                              global
198                                             (match (Hashtbl.find amb
                                                 nome) with
199                                              EntVar v -> v;
200                                              v.tipagem
201                                              | _ -> failwith "
                                                 busca_var_fun: erro
                                                 "
202                                             )
203                                )
204     )
205     | _ -> failwith "busca_var_fun: erro" *)
206
207 (* Teste *)
208 let tabela_simb =
209     let entA = EntVar (cria_ent_var (Some TInt))
210     and params = [("x", cria_ent_var (Some TInt));
211                           ("y", cria_ent_var (Some TInt))]
```

```
212     and locais = (let tabl = Hashtbl.create 5 in
213                                 Hashtbl.add tabl "b" (cria_ent_var (Some TInt
                                        ));
214                                 tabl
215                           )
216     in
217     let entF = EntFn (cria_ent_func TInt params locais)
218     and tab = Hashtbl.create 5 in
219      ( Hashtbl.add tab "a" entA;
220       Hashtbl.add tab "f" entF;
221       tab )
222
223  let imprime_tipo_base t =
224      match t with
225      TInt -> print_endline("Int")
226     | TFloat -> print_endline("float")
227     | TString -> print_endline("string")
228     | TVoid -> print_endline("void")
229     | TBool -> print_endline("bool")
230
231 let imprime_tipagem t =
232   match t with
233       Some t -> imprime_tipo_base t
234     | None -> print_endline("None")
235
236  let imprime_operador op =
237      match op with
238      Mais -> print_endline("Mais")
239     | Menos -> print_endline("Menos")
240     | Mult -> print_endline("Mult")
241     | Div -> print_endline("Div")
242     | Maior -> print_endline("Maior")
243     | Menor -> print_endline("Menor")
244     | Igual -> print_endline("Igual")
245     | Diferente -> print_endline("Diferente")
246     | MaiorIgual -> print_endline("MaiorIgual")
247     | MenorIgual -> print_endline("MenorIgual")
248     | Modulo -> print_endline("Modulo")
249     | And -> print_endline("And")
250     | Or -> print_endline("Or")
251
252 (*
253 let imprime_entrada c v =
254   printf "%s: " c;
255   match v with
256     EntVar entVar -> (printf "var\n" ;
257                                 printf "tipagem: %s\n" (imprime_tipo_base
                                        entVar.tipagem)
258                               )
259
260   | EntFn entFn ->   ( printf "funcao\n"(* ;
261                                 printf "tipagem %s\n" (imprime_tipo_base_fun
                                        entFn.varLocais) *)
262                                )
263
264 let imprime_tbl amb =
265     Hashtbl.iter  imprime_entrada  amb
266
```

```
267
268 and entradaFuncao = {  varLocais: (string, entradaVariavel) Hashtbl.t;
269                                    mutable tiporetorno: tipo_base option;
270                                    param: parametros
271           }
272
273 (* Entrada de uma variavel na tabela de simbolos *)
274 and entradaVariavel = {  mutable tipagem: tipo_base option;
275                                      v_inicial: expressao option;
276                                      mutable endereco: int option;
277                                      mutable valor_variavel: expr option
278             }
279  *)
```

Listing 4.5: ASA para o MiniPython

4.4 Análise Léxica

A análise léxica é a primeira fase do compilador. A função do analisador léxico é ler o código fonte, buscando a separação e identificação dos componentes do programa fonte, denominados símbolos léxicos ou tokens. Essa fase também é responsável por eliminar elementos "decorativos", tais como espaços em branco e comentários.

Em resumo o analisador léxico varre o fonte, caracter a caracter e gera uma estrutura utilizando o que realmente importa no contexto do programa, como identificadores, palavras reservadas, constantes, operadores. Espaços em branco e comentários, são ignorados pelo analisador léxico. Nessa fase, erros de digitação de alguma palavra reservada, colocar vírgulas ou algum outro caracter no lugar errado são detectados e o mesmo não vai para as próximas fases enquanto o código não é corrigido.

4.4.1 Indentação da Linguagem Python

Na linguagem Python, não utilizamos chaves ('{') para definir funções ou laços, é utilizado sua própria indentação para isso. Por isso espaçamentos e tabulações devem ser considerados. Todo programa é dividido em linhas lógicas que são separadas pelo token *NOVALINHA*, as linhas físicas são trechos de código divididos pelo caractere *enter*. Linhas lógicas não podem ultrapassar linhas físicas. Para a delimitação de blocos de códigos os delimitadores são colocados em uma pilha e diferenciados por sua indentação. Iniciando a pilha com valor 0 (zero) e colocando valores maiores que os anteriores na pilha. Para cada começo de linha, o nível de indentação é comparado com o valor do topo da pilha. Se o número da linha for igual ao topo da pilha, a pilha não é alterada. Se o valor for maior a pilha recebe o nível de indentação da linha e o nome *INDENTA* se o nível de indentação for menor, então é desempilhado até chegar a um nível de indentação recebendo o nome *DEDENTA* e se não encontrar nenhum valor é gerado um erro de indentação.

Python foi desenvolvido para ser uma linguagem de fácil leitura, com um visual agradável, frequentemente usando palavras e não pontuações como em outras linguagens. Para a separação de blocos de código, a linguagem Python usa espaços em branco e indentação ao invés de delimitadores visuais como chaves({) ou palavras. Logo, a indentação se torna obrigatória na linguagem Python. O aumento da indentação significa o início de um novo bloco, que termina na diminuição da indentação. Abaixo podemos ver um exemplo de permutação, para uma melhor compreensão de como funciona a indentação na linguagem Python:

```
1         def contador(a: int, b: int) -> int: NOVALINHA
2 INDENTA    soma = 0                          NOVALINHA
3            i = 0                               NOVALINHA
4            c = "2"                                   NOVALINHA
5            d = int(c)                                NOVALINHA
```

```
INDENTA    for i in range(5):                          NOVALINHA
             soma = 0 + 2                                    NOVALINHA
DEDENTA    return soma                                 NOVALINHA
```

Como podemos ver no código acima, os tokens que iremos utilizar serão o 'INDENTA' e o 'DEDENTA'. Também utilizaremos um token 'NOVALINHA' para indicar final de linha.

O código apresentado abaixo trata a indentação do código fonte Python para gerar os tokens de escopo, que se trata de um pré-processamento do fonte.

```
open Sintatico
open Printf

let preprocessa lexbuf =
        let pilha = Stack.create ()
        and npar = ref 0 in
                let _ = Stack.push 0 pilha  in
                let off_side toks nivel =
                if !npar != 0 (* nova linha entre parenteses *)
                then toks    (* nao faca nada *)
                else if nivel > Stack.top pilha
                         then begin
                                 Stack.push nivel pilha;
                                 INDENTA :: toks
                         end
                else if nivel = Stack.top pilha
                        then toks
                        else begin
                                let prefixo = ref toks  in
                                        while nivel < Stack.top pilha do
                                                ignore (Stack.pop pilha);
                                                if nivel > Stack.top pilha
                                                        then failwith "Erro
                                                           de indentacao"
                                                else prefixo := DEDENTA :: !
                                                   prefixo
                                        done;
                                        !prefixo
                                 end
  in
        let rec dedenta sufixo =
                if Stack.top pilha != 0
                        then let _ = Stack.pop pilha in
                        dedenta (DEDENTA :: sufixo)
                else sufixo
        in
        let rec get_tokens () =
                let tok = Lexico.preprocessador 0 lexbuf in
                match tok with
                LINHA(nivel,npars,toks) ->
                        let new_toks = off_side toks nivel in
                        npar := npars;
                        new_toks @ (if npars = 0 then NOVALINHA ::
                           get_tokens ()
                                        else get_tokens ())
                | _ -> dedenta []
                in get_tokens ()

(* Imprime na tela cada token gerado *)
let print_tok tok =
```

```
48          match tok with
49          | INT i    -> printf "INT %d\n" i
50          | FLOAT f -> printf "FLOAT %f\n" f
51          | ID s     -> printf "ID %s\n" s
52          | STRING s -> printf "STRING %s\n" s
53          | LINHA (i, p, toks) -> printf "LINHA (%d, %d, \n" i p
54          | MAIS -> print_endline("MAIS")
55          | MENOS -> print_endline("MENOS")
56          | VEZES -> print_endline("VEZES")
57          | DIVIDIDO -> print_endline("DIVIDIDO")
58          | TIPOINT ->print_endline("INT")
59          | TIPOFLOAT ->print_endline("FLOAT")
60          | TIPOBOOL ->print_endline("BOOL")
61          | TIPOVOID ->print_endline("VOID")
62          | TIPOSTRING ->print_endline("STRING")
63          | POT -> print_endline("POT")
64          | MAIOR -> print_endline("MAIOR")
65          | MENOR -> print_endline("MENOR")
66          | IGUAL -> print_endline("IGUAL")
67          | DIFERENTE -> print_endline("DIFERENTE")
68          | MAIORIGUAL -> print_endline("MAIORIGUAL")
69          | MENORIGUAL -> print_endline("MENORIGUAL")
70          | APAR -> print_endline("APAR")
71          | FPAR -> print_endline("FPAR")
72          | ACOL -> print_endline("ACOL")
73          | FCOL -> print_endline("FCOL")
74          | ACHA -> print_endline("ACHA")
75          | FCHA -> print_endline("FCHA")
76          | DPONTOS -> print_endline("DPTOS")
77          | PTO -> print_endline("PTO")
78          | PTVIRG -> print_endline("PTVIRG")
79          | IF -> print_endline("IF")
80          | ELSE -> print_endline("ELSE")
81          | WHILE -> print_endline("WHILE")
82          | FOR -> print_endline("FOR")
83          | IN -> print_endline("IN")
84          | RANGE -> print_endline("RANGE")
85          | VIRG -> print_endline("VIRG")
86          | DEF -> print_endline("DEF")
87          | RETURN -> print_endline("RETURN")
88          | INDENTA -> print_endline("INDENTA")
89          | DEDENTA -> print_endline("DEDENTA")
90          | NOT -> print_endline("NOT")
91          | TRUE -> print_endline("TRUE")
92          | FALSE -> print_endline("FALSE")
93          | ATRIB -> print_endline("ATRIB")
94          | ATRIBMAIS -> print_endline("ATRIBMAIS")
95          | ATRIBMENOS -> print_endline("ATRIBMENOS")
96          | ATRIBVEZES -> print_endline("ATRIBVEZES")
97          | ATRIBDIV -> print_endline("ATRIBDIV")
98          | ATRIBMOD -> print_endline("ATRIBMOD")
99          | EOF -> print_endline("EOF")
100         | AND -> print_endline("AND")
101         | OR -> print_endline("OR")
102         | IS -> print_endline("IS")
103         | FROM -> print_endline "FROM"
104         | YIELD -> print_endline "YIELD"
105         | NOVALINHA -> print_endline "NOVALINHA"
```

```
        | SETA -> print_endline "SETA"
        | MODULO -> print_endline "MODULO"
        | BOOL _ -> print_endline "BOOL"
        | PRINT -> print_endline "PRINT"
        | INPUT -> print_endline "INPUT"
        | INT_PARSE -> print_endline "INT_PARSE"

(* Chama o analisador lexico *)
let lexico =
        let tokbuf = ref None in
                let carrega lexbuf =
                        let toks = preprocessa lexbuf in
                        (match toks with
                        tok::toks ->
                                 tokbuf := Some toks;
                                 print_tok tok;
                                 tok
                        | [] -> print_endline "EOF";
                                EOF)
  in
  fun lexbuf ->
        match !tokbuf with
        Some tokens ->
                (match tokens with
                tok::toks ->
                        tokbuf := Some toks;
                        print_tok tok;
                        tok
                | [] -> carrega lexbuf)
        | None -> carrega lexbuf
```

Listing 4.6: Analisador léxico para o escopo do Python

4.4.2 Analisador Léxico

Depois de termos feito o pré-processamento do código utilizado anteriormente para identificar espaçamentos no arquivo fonte, precisamos terminar a análise léxica gerando os tokens restantes do fonte. Nossa intenção agora é definir as tipagens, por exemplo, quando um número é inteiro ou um ponto flutuante, quando uma string é ou não uma palavra reservada.

O código está abaixo:

```
{
    open Sintatico   (* o tipo token eh definido em sintatico.mli *)
    open Lexing
    open Printf


        (* contador de nivel de parentizacao - utilizado para identacao *)
    let nivel_par = ref 0

    (* incrementa a contagem de linhas *)
    let incr_nlinha lexbuf =
        let pos = lexbuf.lex_curr_p in
                lexbuf.lex_curr_p <- { pos with
                                       pos_lnum = pos.pos_lnum + 1;
                                       pos_bol = pos.pos_cnum;
                                     }
    (* imprime mensagem de erro *)
```

```
18      let msg_erro lexbuf c =
19          let pos = lexbuf.lex_curr_p in
20              let lin = pos.pos_lnum
21              and col = pos.pos_cnum - pos.pos_bol - 1 in
22              sprintf "%d-%d: Caracter Desconhecido %c" lin col c
23
24      (* cria tabela hasg  *)
25      let cria_tab_hash iniciais =
26          let tbl = Hashtbl.create (List.length iniciais) in
27              List.iter (fun (chave, valor) -> Hashtbl.add tbl chave valor)
                    iniciais;
28              tbl
29
30      (* palavras reservadas *)
31      let plv_res =
32          cria_tab_hash
33          [
34          ("def",    DEF);
35          ("else",   ELSE );
36          ("for",    FOR);
37          ("if",     IF);
38          ("in",     IN);
39          ("not",    NOT);
40          ("and",    AND);
41          ("or",     OR);
42          ("is",     IS);
43          ("yield",  YIELD);
44          ("from",   FROM);
45          ("return",RETURN);
46          ("while",  WHILE);
47          ("range",  RANGE);
48          ("print",  PRINT);
49          ("raw_input", INPUT);
50          ("int",    INT_PARSE);
51          ("void", TIPOVOID)
52          ]
53
54  (* Valores booleanos sao armazenados como 1 para true e 0 para false. *)
55  (* Operacoes com booleanos sao transformadas em operacoes com inteiros *)
56      let booleano nbool =
57      match nbool with
58      | "True" -> 1
59      | "False" -> 0
60      | _ -> failwith "Erro: nao eh valor booleano"
61  }
62
63  (* definicoes *)
64  let digito = ['0' - '9']
65  let letra = ['a'-'z' 'A'-'Z']
66  let id  = letra ( letra | digito | '_' )*
67  let comentario = '#' [^ '\n']*
68  let linha_em_branco = [' ' '\t' ]* comentario
69  let restante = [^ ' ' '\t' '\n' ] [^ '\n']+
70  let boolean = "True" | "False"
71  let strings = '"' id* digito* '"' | "'" id* digito* "'"
72  let floats = digito+ '.' digito+
73  let neg = '-' digito+
74
```

```
75 (* regras para identificar identacao para gerar tokens de abre e fecha
       escopo *)
76 rule preprocessador indentacao = parse
77   linha_em_branco      { preprocessador 0 lexbuf } (* ignora brancos *)
78 | [' ' '\t' ]+ '\n'    { incr_nlinha lexbuf;
79                          preprocessador 0 lexbuf } (* ignora brancos *)
80 | ' '                  { preprocessador (indentacao + 1) lexbuf }
81 | '\t'                 { let nova_ind = indentacao + 8 - (indentacao mod 8)
       in
82                          preprocessador nova_ind lexbuf }
83 | '\n'                 { incr_nlinha lexbuf;
84                          preprocessador 0 lexbuf }
85 | restante as linha    {
86                    let rec tokenize lexbuf =
87                      let tok = token lexbuf in
88                      match tok with
89                        EOF -> []
90                      | _ -> tok :: tokenize lexbuf in
91                    let toks = tokenize (Lexing.from_string linha) in
92                    LINHA(indentacao,!nivel_par, toks)
93   }
94 | eof                  { nivel_par := 0; EOF  }
95
96 (* identificacao dos tokens *)
97 and token  = parse
98 | ' '
99 | '\t'
100 | comentario          { token lexbuf }
101 | digito+ as numint    { let num = int_of_string numint in INT num }
102 | neg as numNeg        { let num = int_of_string numNeg in INT num }
103 | digito+ '.' digito+ as numfloat {let num = float_of_string numfloat in
       FLOAT num}
104 | boolean as nbool     { INT (booleano nbool)}
105 | floats as numfloat   { FLOAT (float_of_string numfloat) }
106 | id as palavra        { try Hashtbl.find plv_res palavra
107                          with Not_found -> ID (palavra)}
108 | '"'                  { let buffer = Buffer.create 1 in
109                          STRING (cadeia buffer lexbuf) }
110 | '''                  { let buffer = Buffer.create 1 in
111                          STRING (cadeia2 buffer lexbuf) }
112 | '='                      { ATRIB }
113 | '+'                  { MAIS }
114 | '-'                  { MENOS }
115 | '*'                  { VEZES }
116 | '/'                  { DIVIDIDO }
117 | ':'                      { DPONTOS }
118 | '>'                  { MAIOR }
119 | '<'                  { MENOR }
120 | ">="                 { MAIORIGUAL }
121 | "<="                 { MENORIGUAL }
122 | "=="                     { IGUAL }
123 | "!="                     { DIFERENTE }
124 | "+="                     { ATRIBMAIS}
125 | "-="                     { ATRIBMENOS}
126 | "*="                     { ATRIBVEZES}
127 | "/="                     { ATRIBDIV}
128 | "%="                     { ATRIBMOD}
129 | "%"                      { MODULO }
```

```
| '('                 { incr(nivel_par); APAR }
| '['                 { incr(nivel_par); ACOL }
| '{'                 { incr(nivel_par); ACHA }
| ')'                 { decr(nivel_par); FPAR }
| ']'                 { decr(nivel_par); FCOL }
| '}'                 { decr(nivel_par); FCHA }
| "->"                { SETA }
| '='                 { ATRIB }
| ':'                 { DPONTOS }
| ','                     { VIRG }
| ';'                     { PTVIRG }
| '.'                     { PTO }
| _ as c              { failwith (msg_erro lexbuf c); }
| eof                 { EOF }

(* para criar cadeias de strings *)
and cadeia buffer = parse
  | '"'         { Buffer.contents buffer }
  | "\\t"       { Buffer.add_char buffer '\t'; cadeia buffer lexbuf }
  | "\\n"       { Buffer.add_char buffer '\n'; cadeia buffer lexbuf }
  | '\\' '"'    { Buffer.add_char buffer '"';  cadeia buffer lexbuf }
  | '\\' '\\'   { Buffer.add_char buffer '\\'; cadeia buffer lexbuf }
  | _ as c      { Buffer.add_char buffer c;    cadeia buffer lexbuf }
  | eof         { failwith "string nao foi fechada" }

and cadeia2 buffer = parse
  | '''         { Buffer.contents buffer }
  | "\\t"       { Buffer.add_char buffer '\t'; cadeia2 buffer lexbuf }
  | "\\n"       { Buffer.add_char buffer '\n'; cadeia2 buffer lexbuf }
  | '\\' '''    { Buffer.add_char buffer '"';  cadeia2 buffer lexbuf }
  | '\\' '\\'   { Buffer.add_char buffer '\\'; cadeia2 buffer lexbuf }
  | _ as c      { Buffer.add_char buffer c;    cadeia2 buffer lexbuf }
  | eof         { failwith "string nao foi fechada" }
```

Listing 4.7: Analisador léxico do MiniPython

Usaremos o exemplo de fonte abaixo e verificaremos a saída do mesmo ao ser passado pelo analisador léxico criado.

```
def contador(a: int, b: int) -> int:
    soma = 0
    i = 0
    c = "2"
    d = int(c)
    for i in range(5):
        soma = 0 + 2
    return soma
```

Listing 4.8: Função Soma

Ao passar pela análise léxica, temos a seguinte saída para o fonte apresentado acima:

```
DEF
ID contador
APAR
ID a
DPTOS
INT_PARSE
VIRG
ID b
```

```
DPTOS
INT_PARSE
FPAR
SETA
INT_PARSE
DPTOS
NOVALINHA
INDENTA
ID soma
ATRIB
INT 0
NOVALINHA
ID i
ATRIB
INT 0
NOVALINHA
ID c
ATRIB
STRING 2
NOVALINHA
ID d
ATRIB
INT_PARSE
APAR
ID c
FPAR
NOVALINHA
FOR
ID i
IN
RANGE
APAR
INT 5
FPAR
DPTOS
NOVALINHA
INDENTA
ID soma
ATRIB
INT 0
MAIS
INT 2
NOVALINHA
DEDENTA
RETURN
ID soma
NOVALINHA
DEDENTA
EOF
```

Assim podemos verificar se todos os tokens realmente importantes foram pegos pela análise léxica, ou se algo ficou para trás, assim como também verificar se realmente foram ignorados os caracteres que devem ser ignorados, como os comentários. Podemos ver que todos os tokens relevantes para a nossa compilação foram realmente pegos pelo nosso analisador léxico.

4.5 Analisador Sintático

A análise sintática, ou análise gramatical é o processo de se determinar se uma cadeia de símbolos léxicos pode ser gerada por uma gramática, no nosso projeto faremos um analisador sintático para verificar se determinado fonte pode ser gerado para a gramática da linguagem MiniPython. O analisador sintático é responsável por verificar se os símbolos contidos no código fonte formam um programa válido. Abaixo é apresentado o fonte responsável pela análise sintática do nosso projeto:

```
%{
    open Asa;;
(* Cria uma expressao onde v eh o valor da expressao e o eh a ordem na
regra gramatical *)

let cria_exp o v =
    { valor = v;
       tipo = None;
       pos = Posicao.pos(o) }

(* Cria um comando onde c eh o comando e o eh a ordem na regra gramatical *)
let cria_cmd o c =
    { vcmd = c;
       pcmd = Posicao.pos(o) }
(* Cria um programa onde f sao as funcoes e c os comandos fora das funcoes
*)
let cria_programa f c =
    { funcsP = f;
       cmdsP = c }
(* Cria uma funcao onde o eh a ordem na regra gramatical, i eh o nome da
   funcao,
p sao os parametros e c sao os comandos dentro da funcao. O tipo de retorno
   inicia como None *)
let cria_funcao o i p t c =
    { idF = i;
       paramsF = p;
       cmdsF = c;
       returnF = t;
       posF = Posicao.pos(o);
       varLocaisF = Hashtbl.create 20 }
(* Cria um parametro onde o eh a ordem na regra gramatical e i o nome (id)
   do parametro.
O tipo do parametro inicia como TGen *)
let cria_parametro o i c = (i, cria_ent_var c)


%}


/* Definicao de tokens */

%token <int> INT
%token <float>  FLOAT
%token <bool> BOOL
%token <string> ID
%token <string> STRING
%token <int * int * token list> LINHA
%token INDENTA DEDENTA NOVALINHA
%token TIPOINT TIPOFLOAT TIPOSTRING TIPOVOID TIPOBOOL
%token PTO PTVIRG
```

```
48 %token DEF IS YIELD FROM RETURN
49 %token TRUE FALSE
50 %token APAR FPAR ACOL FCOL ACHA FCHA SETA
51 %token IF ELSE WHILE DPONTOS
52 %token FOR IN RANGE VIRG
53 %token NOT AND OR
54 %token ATRIB
55 %token MAIS MENOS VEZES DIVIDIDO MODULO POT
56 %token MAIOR MENOR IGUAL DIFERENTE MAIORIGUAL MENORIGUAL
57 %token ATRIBMAIS ATRIBMENOS ATRIBVEZES ATRIBDIV ATRIBMOD
58 %token EOF
59 %token PRINT INPUT INT_PARSE
60 /* o simbolo inicial da gramatica (ponto de entrada) */
61 %start programa
62 %type <Asa.programa> programa
63 %%
64
65 /* um programa eh definido por um conjunto de funcoes seguido por um
66 conjunto de comandos */
67 programa: funcoes comandos { cria_programa $1 $2 };
68
69 /* */
70 funcoes:  { [] }
71                 | funcoes funcao { $1 @ [ $2 ] }
72                 ;
73 /* define a estrutura de uma funcao e cria a funcao */
74 funcao: DEF ID APAR parametros FPAR SETA tipo DPONTOS
75 NOVALINHA INDENTA comandos DEDENTA
76 { cria_funcao 1 $2 $4 $7 $11 };
77 /* define a estrutura de uma lista de funcoes e um parametro e cria o
78 parametro */
79 tipo:   INT_PARSE {Some TInt}
80         | TIPOVOID {Some TVoid}
81
82         ;
83
84 parametros: { [] }
85                     | parametros parametro { $1 @ [ $2 ] }
86                     ;
87 /* um parametro pode estar seguido de virgula ou nao */
88 parametro:  ID DPONTOS tipo VIRG { cria_parametro 1 $1 $3 }
89                 |   ID DPONTOS tipo { cria_parametro 1 $1 $3 };
90
91 argumentos: { [] }
92                     | argumentos argumento { $1 @ [ $2 ] }
93                     ;
94 /* um argumento pode estar seguido de virgula ou nao */
95 argumento: expressao VIRG { cria_exp 2 $1.valor}
96                     | expressao { cria_exp 2 $1.valor}
97                     ;
98 /* a chamada de funcao pode ser ou nao uma atribuicao */
99 cmd_chamada_func: ID APAR argumentos FPAR NOVALINHA { cria_cmd 1 (
       ChamaFuncaoVoid ($1, $3))};
100                                 | expressao ATRIB ID APAR argumentos FPAR
                                       NOVALINHA {cria_cmd 1 (ChamaFuncaoAtrib (
                                       $1, $3, $5))}
101
102 /* tipos de comandos */
```

```
103 comando:  cmd_atrib { $1 }
104                 | cmd_if_else { $1 }
105                 | cmd_if { $1 }
106                 | cmd_while { $1 }
107                 | cmd_for { $1 }
108                 | cmd_range1 { $1 }
109                 | cmd_range2 { $1 }
110                 | cmd_range3 { $1 }
111                 | cmd_atribMAIS { $1 }
112                 | cmd_atribMENOS { $1 }
113                 | cmd_atribVEZES { $1 }
114                 | cmd_atribDIV { $1 }
115                 | cmd_atribMOD { $1 }
116                 | cmd_chamada_func { $1 }
117                 | cmd_retorno { $1 }
118                 | cmd_print { $1 }
119                 | cmd_input { $1 }
120                 | cmd_int_parse { $1 }
121             ;
122 /* definicao da estrutura dos comandos */
123 comandos: { [] }
124                 | comandos comando { $1 @ [ $2 ] }
125                 ;
126 cmd_int_parse: expressao ATRIB INT_PARSE APAR expressao FPAR NOVALINHA
127     { cria_cmd 1 ( CmdIntParse ( $1, $5 ))}
128 cmd_print: PRINT APAR expressao FPAR NOVALINHA
129     { cria_cmd 1 ( CmdPrint( $3 ) )}
130 cmd_input: expressao ATRIB INPUT APAR expressao FPAR NOVALINHA
131     { cria_cmd 1 ( CmdInput ( $1, $5 ) )}
132 cmd_retorno: RETURN expressao NOVALINHA
133     { cria_cmd 1 ( CmdReturn ( $2 ) ) }
134 cmd_atrib: expressao ATRIB expressao NOVALINHA
135     { cria_cmd 2 ( CmdAtrib ( $1, $3 ) ) }
136 cmd_if: IF expressao DPONTOS NOVALINHA INDENTA comandos DEDENTA
137     { cria_cmd 1 ( CmdIf ( $2, $6, None ) ) }
138 cmd_if_else: IF expressao DPONTOS NOVALINHA INDENTA comandos DEDENTA
139                     ELSE DPONTOS NOVALINHA INDENTA comandos DEDENTA
140     { cria_cmd 1 ( CmdIf ( $2, $6, Some( $12 ) ) ) }
141 cmd_while: WHILE expressao DPONTOS NOVALINHA INDENTA comandos DEDENTA
142     { cria_cmd 1 ( CmdWhile ( $2, $6 ) ) }
143 cmd_for: FOR expressao IN comando DPONTOS NOVALINHA INDENTA comandos DEDENTA
144     { cria_cmd 1 ( CmdFor ( $2, $4, $8 ) ) }
145 /* se o comando range tiver um parametro a range varia de 0 ao parametro
146 com incremento 1*/
147 cmd_range1: RANGE APAR INT FPAR
148     { cria_cmd 1 ( CmdRange ( 0, $3, 1 ) ) }
149 /* se o comando range tiver dois parametros a range varia entre os dois
150 parametros com incremento 1*/
151 cmd_range2: RANGE APAR INT VIRG INT FPAR
152     { cria_cmd 1 ( CmdRange ( $3, $5, 1) ) }
153 /* se o comando range tiver tres parametros os tres sao passados para o
    comando */
154 cmd_range3: RANGE APAR INT VIRG INT VIRG INT FPAR
155     { cria_cmd 1 ( CmdRange ( $3, $5, $7 ) ) }
156 /* para comandos +=, -= cria uma expressao e coloca na atribuicao */
157 cmd_atribMAIS: expressao ATRIBMAIS expressao
158     { let exp = cria_exp 2 (Some( ExpBin(Mais, $1, $3 ) )) in
159         cria_cmd 2 ( CmdAtrib( $1, exp ) ) }
```

```
160 cmd_atribMENOS: expressao ATRIBMENOS expressao
161     { let exp = cria_exp 2 (Some (ExpBin ( Menos, $1, $3 ) ) )in
162     cria_cmd 2 ( CmdAtrib ( $1, exp ) ) }
163 cmd_atribVEZES: expressao ATRIBVEZES expressao
164     { let exp = cria_exp 2 (Some( ExpBin ( Mult, $1, $3 ) )) in
165     cria_cmd 2 ( CmdAtrib ( $1, exp ) ) }
166 cmd_atribDIV: expressao ATRIBDIV expressao
167     { let exp = cria_exp 2 (Some( ExpBin ( Div, $1, $3 ) )) in
168     cria_cmd 2 ( CmdAtrib( $1, exp ) ) }
169 cmd_atribMOD: expressao ATRIBMOD expressao
170     { let exp = cria_exp 2 (Some( ExpBin ( Modulo, $1, $3 ) )) in
171     cria_cmd 2 ( CmdAtrib( $1, exp ) ) }
172
173 /* criando expressoes */
174 expressao : expressao AND expr1 {cria_exp 5 (Some(ExpBin (And, $1, $3)))}
175                         |expressao OR expr1 {cria_exp 5 (Some(ExpBin (Or, $1
                               , $3)))}
176                         | expr1 {$1}
177                         ;
178
179 expr1 : expr1 MAIOR expr2 { cria_exp 4 (Some( ExpBin ( Maior, $1, $3 ) )) }
180           | expr1 MENOR expr2 { cria_exp 4 (Some( ExpBin ( Menor, $1, $3 ) ))
                 }
181           | expr1 IGUAL expr2 { cria_exp 4 (Some( ExpBin ( Igual, $1, $3 ) ))
                 }
182           | expr1 DIFERENTE expr2 { cria_exp 4 (Some( ExpBin ( Diferente, $1,
                 $3 ) )) }
183           | expr1 MAIORIGUAL expr2 { cria_exp 4 (Some( ExpBin ( MaiorIgual,
                 $1, $3 ) )) }
184           | expr1 MENORIGUAL expr2 { cria_exp 4 (Some( ExpBin ( MenorIgual,
                 $1, $3 ) )) }
185           | expr1 MODULO expr2 { cria_exp 4 (Some( ExpBin ( Modulo, $1, $3 )
                 )) }
186           | expr2 { $1 }
187
188 expr2 : expr2 MAIS expr3 { cria_exp 3 (Some( ExpBin ( Mais, $1, $3 ) )) }
189                   | expr2 MENOS expr3 { cria_exp 3 (Some( ExpBin ( Menos, $1
                        , $3 ) )) }
190                   | expr3 { $1 }
191            ;
192
193 expr3: expr3 VEZES expr4 { cria_exp 2 (Some( ExpBin ( Mult, $1, $3 ) )) }
194           | expr3 DIVIDIDO expr4 { cria_exp 2 (Some( ExpBin ( Div, $1, $3 ) )
                 ) }
195           | expr4 { $1 }
196           ;
197
198 expr4: NOT expr4 {cria_exp 1 (Some(ExpUn(Not, $2)))}
199           | expr5 {$1}
200           ;
201
202 expr5: operando  { cria_exp 0 $1 }
203      | variavel  { cria_exp 0 (Some( ExpVar $1 ) )}
204      | APAR expressao FPAR { $2 }
205      ;
206 /* criando operando e variavel */
207 operando: INT {Some (ExpInt $1) }
208                   | FLOAT { Some (ExpFloat $1) }
```

```
                    | STRING { Some (ExpString $1) }
                    |  BOOL { Some (ExpBool $1) }
;
variavel: ID { VarSimples $1 }
;
```

Listing 4.9: Analisador Sintático do MiniPython

Ao executar o analisador sintático no mesmo fonte que passamos pelo analisador léxico, teremos a seguinte saída:

```
- : Asa.programa =
{funcsP =
  [{idF = "contador";
    paramsF =
     [("a",
       {tipagem = Some TInt; v_inicial = None; endereco = None;
        valor_variavel = 0});
      ("b",
       {tipagem = Some TInt; v_inicial = None; endereco = None;
        valor_variavel = 0})];
    cmdsF =
     [{vcmd =
        CmdAtrib
         ({valor = Some (ExpVar (VarSimples "soma")); tipo = None;
           pos =
            {Asa.Posicao.lin_inicial = 8; col_inicial = 17; lin_final = 8;
             col_final = 16}},
         {valor = Some (ExpInt 0); tipo = None;
          pos =
           {Asa.Posicao.lin_inicial = 8; col_inicial = 17; lin_final = 8;
            col_final = 16}});
       pcmd =
        {Asa.Posicao.lin_inicial = 8; col_inicial = 17; lin_final = 8;
         col_final = 16}};
      {vcmd =
        CmdAtrib
         ({valor = Some (ExpVar (VarSimples "i")); tipo = None;
           pos =
            {Asa.Posicao.lin_inicial = 8; col_inicial = 17; lin_final = 8;
             col_final = 16}},
         {valor = Some (ExpInt 0); tipo = None;
          pos =
           {Asa.Posicao.lin_inicial = 8; col_inicial = 17; lin_final = 8;
            col_final = 16}});
       pcmd =
        {Asa.Posicao.lin_inicial = 8; col_inicial = 17; lin_final = 8;
         col_final = 16}};
      {vcmd =
        CmdAtrib
         ({valor = Some (ExpVar (VarSimples "c")); tipo = None;
           pos =
            {Asa.Posicao.lin_inicial = 8; col_inicial = 17; lin_final = 8;
             col_final = 16}},
         {valor = Some (ExpString "2"); tipo = None;
          pos =
           {Asa.Posicao.lin_inicial = 8; col_inicial = 17; lin_final = 8;
            col_final = 16}});
       pcmd =
```

```
49          {Asa.Posicao.lin_inicial = 8; col_inicial = 17; lin_final = 8;
50           col_final = 16}};
51        {vcmd =
52          CmdIntParse
53           ({valor = Some (ExpVar (VarSimples "d")); tipo = None;
54             pos =
55              {Asa.Posicao.lin_inicial = 8; col_inicial = 17; lin_final = 8;
56               col_final = 16}},
57           {valor = Some (ExpVar (VarSimples "c")); tipo = None;
58            pos =
59             {Asa.Posicao.lin_inicial = 8; col_inicial = 17; lin_final = 8;
60              col_final = 16}});
61         pcmd =
62          {Asa.Posicao.lin_inicial = 8; col_inicial = 17; lin_final = 8;
63           col_final = 16}};
64        {vcmd =
65          CmdFor
66           ({valor = Some (ExpVar (VarSimples "i")); tipo = None;
67             pos =
68              {Asa.Posicao.lin_inicial = 8; col_inicial = 17; lin_final = 8;
69               col_final = 16}},
70           {vcmd = CmdRange (0, 5, 1);
71            pcmd =
72             {Asa.Posicao.lin_inicial = 8; col_inicial = 17; lin_final = 8;
73              col_final = 16}},
74           [{vcmd =
75              CmdAtrib
76               ({valor = Some (ExpVar (VarSimples "soma")); tipo = None;
77                 pos =
78                  {Asa.Posicao.lin_inicial = 8; col_inicial = 17;
79                   lin_final = 8; col_final = 16}},
80                {valor =
81                  Some
82                   (ExpBin (Mais,
83                     {valor = Some (ExpInt 0); tipo = None;
84                      pos =
85                       {Asa.Posicao.lin_inicial = 8; col_inicial = 17;
86                        lin_final = 8; col_final = 16}},
87                     {valor = Some (ExpInt 2); tipo = None;
88                      pos =
89                       {Asa.Posicao.lin_inicial = 8; col_inicial = 17;
90                        lin_final = 8; col_final = 16}}));
91                tipo = None;
92                pos =
93                 {Asa.Posicao.lin_inicial = 8; col_inicial = 17; lin_final =
                     8;
94                  col_final = 16}});
95              pcmd =
96               {Asa.Posicao.lin_inicial = 8; col_inicial = 17; lin_final = 8;
97                col_final = 16}}]);
98         pcmd =
99          {Asa.Posicao.lin_inicial = 8; col_inicial = 17; lin_final = 8;
100          col_final = 16}};
101       {vcmd =
102         CmdReturn
103          {valor = Some (ExpVar (VarSimples "soma")); tipo = None;
104           pos =
105            {Asa.Posicao.lin_inicial = 8; col_inicial = 17; lin_final = 8;
```

```
106             col_final = 16}};
107       pcmd =
108        {Asa.Posicao.lin_inicial = 8; col_inicial = 17; lin_final = 8;
109         col_final = 16}}];
110    returnF = Some TInt;
111    posF =
112     {Asa.Posicao.lin_inicial = 8; col_inicial = 17; lin_final = 8;
113      col_final = 16};
114    varLocaisF = <abstr>}];
115 cmdsP = []}
```

Perceba que nossa tabela foi preenchida corretamente com as variáveis, funções, e tudo mais, exceto seus tipos. Os tipos serão inferidos ao passarmos pelo analisador semântico, que será a próxima etapa do nosso compilador.

4.6 Analisador Semântico

O analisador semântico é o terceiro módulo de um compilador, o qual recebe a árvore abstrata passada pelo analisador sintático e verifica se há algum erro semântico. As tarefas básicas desempenhadas pelo analisador semântico incluem a verificação de tipos, a verificação do fluxo de controle e a verificação de unicidade da declaração de variáveis. O fonte do analisador semântico está abaixo:

```
1  open Asa;;
2  open Printf;;
3
4  let current_func = ref ""
5  (* Funcoes para mostrar os erros na tela *)
6  let erro nome pos msg =
7      let nlin = pos.Posicao.lin_inicial
8      and ncol = pos.Posicao.col_inicial in
9      let msglc = sprintf "Erro na linha %d, coluna %d" nlin ncol in
10     print_endline msglc;
11     print_endline (msg ^ nome);
12     failwith "Erro semantico"
13 (* Tabela de simbolos para as operacoes *)
14 let tab2list tab = Hashtbl.fold (fun c v ls -> (c,v) :: ls) tab []
15 let ambfun =
16     let amb = Hashtbl.create 23 in
17         Hashtbl.add amb Mais [ (TInt, TInt, TInt) ; (TFloat, TFloat, TFloat)
               ; (TInt,TFloat,TFloat); (TFloat,TInt,TFloat)] ;
18         Hashtbl.add amb Menos [ (TInt, TInt, TInt); (TFloat, TFloat, TFloat)
               ; (TInt,TFloat,TFloat); (TFloat,TInt,TFloat)] ;
19         Hashtbl.add amb Mult [(TInt, TInt, TInt); (TFloat, TFloat, TFloat);
               (TInt,TFloat,TFloat); (TFloat,TInt,TFloat)] ;
20         Hashtbl.add amb Div [(TInt, TInt, TInt); (TFloat, TFloat, TFloat); (
               TInt,TFloat,TFloat); (TFloat,TInt,TFloat)] ;
21         Hashtbl.add amb Menor [ (TInt, TInt, TBool); (TFloat, TFloat, TBool)
               ; (TInt,TFloat,TBool);(TFloat,TInt,TBool)] ;
22         Hashtbl.add amb Maior [ (TInt, TInt, TBool); (TFloat, TFloat, TBool)
               ; (TInt,TFloat,TBool);(TFloat,TInt,TBool)] ;
23         Hashtbl.add amb Igual [(TInt, TInt, TBool); (TFloat, TFloat, TBool);
                (TInt,TFloat,TBool);(TFloat,TInt,TBool)] ;
24         Hashtbl.add amb Diferente [(TInt, TInt, TBool); (TFloat, TFloat,
               TBool); (TInt,TFloat,TBool);(TFloat,TInt,TBool)] ;
25         Hashtbl.add amb MaiorIgual [(TInt, TInt, TBool); (TFloat, TFloat,
               TBool); (TInt,TFloat,TBool);(TFloat,TInt,TBool)] ;
```

```
26          Hashtbl.add amb MenorIgual [(TInt, TInt, TBool); (TFloat, TFloat,
               TBool); (TInt,TFloat,TBool);(TFloat,TInt,TBool)] ;
27          Hashtbl.add amb Modulo [ (TInt, TInt, TInt); (TFloat,TFloat, TFloat)
               ] ;
28          (*nao admite outros tipos de and e or que nao seja com bool*)
29          Hashtbl.add amb And [ (TBool, TBool, TBool)];
30          Hashtbl.add amb Or [ (TBool, TBool, TBool)];
31      amb
32  let tipo e = e.tipo
33  (* Verifica se os tipos dos termos sao compativeis com o tipo do operador*)
34  let rec verifica_op t1 t2 ls =
35      match ls with
36          (a1,a2,r)::ls -> begin
37                  if ((t1 == a1) && (t2 == a2)) then r
38                  else verifica_op t1 t2 ls
39          end
40          | [] -> print_endline("O tipo dos operandos deveria ser o mesmo");
41                    failwith "Erro semantico: verifica_op"
42
43  (* Insere variavel em uma tabela e retorna o tipo, caso ja exista apenas
       retorna o tipo *)
44  let insere_var amb nome tipo current =
45      if(current<>"") then
46      (
47          let tabVar = Hashtbl.find amb current in
48          match tabVar with
49                  EntFn tabFn ->
50              (try (* tenta encontrar variavel local *)
51                 let reg = Hashtbl.find tabFn.varLocais nome in
52                  reg.tipagem
53                              with Not_found -> (* tenta encontrar parametro
                                    *)
54                        (match (procuraParam nome tabFn.param) with
55                          Some v ->  v.tipagem;
56                            | None ->(* tenta encontrar variavel global dentro
                                  da funcao*)
57                          try (match (Hashtbl.find amb nome) with
58                                    (EntVar v) ->   (* imprime_tbl; *)v.tipagem
59                                         | _ -> Hashtbl.add (tabFn.varLocais)
                                               nome  (cria_ent_var (Some tipo));
60                                      (Some TInt)
61                                    )
62                               with Not_found -> Hashtbl.add (tabFn.varLocais)
                                    nome  (cria_ent_var (Some tipo));
63                                              (Some TInt)
64                          )
65                )
66                | EntVar _ -> print_endline("O nome '" ^ current ^ "' nao
                     esta associado a uma funcao.");
67                      failwith("Erro semantico: insere_var")
68      ) else
69      ( (*Tenta encontrar variavel global fora da funcao*)
70          try
71           let ent = Hashtbl.find amb nome in
72          ( match ent with
73             (EntVar t) -> t.tipagem
74             | EntFn _ ->print_endline("A variavel '" ^ nome ^ "' esta
                  associado a uma funcao.");
```

```
                        failwith "Erro semantico: insere_var"
        )
        with
        Not_found ->  Hashtbl.add amb nome (EntVar (cria_ent_var (Some tipo)
           ));
                (Some TInt)
    )

(*Verifica se a variavel existe*)
let verifica_var amb nome current =
    if(current<>"") then
        let tabVar = Hashtbl.find amb current in
        match tabVar with
          EntFn tabFn ->
            (try (* tenta encontrar variavel local *)
                 let reg = Hashtbl.find tabFn.varLocais nome in
                 reg.tipagem
                with Not_found -> (* tenta encontrar parametro *)
                  (match (procuraParam nome tabFn.param) with
                             Some v ->   v.tipagem
                           | None ->  (* tenta encontrar variavel global *)
                          let entrada = Hashtbl.find amb nome in
                             (match entrada with
                           (EntVar v) ->   v.tipagem
                             | EntFn _ -> print_endline("O nome '" ^ nome ^
                                "' esta associado a uma funcao");
                                   failwith "Erro Semantico: verifica_var"
                        )
                  )
              )
        | _ -> print_endline("O nome '" ^ current ^ "' esta associado a uma
           variavel");
                    failwith "Erro Semantico: verifica_var"
    else(
        try
                 let entradaGlobal = Hashtbl.find amb nome in
                (match entradaGlobal with
                  (EntVar v) ->   v.tipagem
                  | EntFn _ -> print_endline("O nome '" ^ nome ^ "' esta
                     associado a uma funcao");
                 failwith "Erro Semantico: verifica_var"
                )
         with e-> raise e
        )

(*Verifica variavel a direita*)
let rec verifica_var_dir amb pos var current =
    match var with
        | VarSimples nome ->  verifica_var amb nome current

(*Verifica variavel a esquerda*)
let rec verifica_var_esq amb pos var tipo current =
    match var with
        | VarSimples nome ->  insere_var amb nome tipo current

(*converte tipo_base para tipo_base option*)
and converte_tipo t1 =
    match t1 with
```

```
129         TInt -> (Some TInt)
130         | TFloat -> (Some TFloat)
131         | TString -> (Some TString)
132         | TBool -> (Some TBool)
133         | TVoid -> (Some TVoid)
134
135 (*converte tipo_base option para tipo_base*)
136 and converte_tipo_option t1 =
137     match t1 with
138         (Some TInt) -> TInt
139         | (Some TFloat) -> TFloat
140         | (Some TBool) -> TBool
141         | (Some TVoid) -> TVoid
142         | (Some TString) -> TString
143         | _ -> print_endline("O tipo da parcela nao existe");
144             failwith("Erro Semantico: converte_tipo_option")
145
146 (* Verifica se as parcelas correspondem aos tipos dos operadores *)
147 and verifica_primitiva op t1 t2 =
148     try
149         let tipos_op = Hashtbl.find ambfun op in
150          match t1 with
151             (Some TInt) -> verifica_op TInt (converte_tipo_option t2)
                tipos_op
152               | (Some TFloat) -> verifica_op TFloat (converte_tipo_option
                    t2) tipos_op
153               | (Some TBool) -> verifica_op TBool (converte_tipo_option
                    t2) tipos_op
154               | (Some TVoid) -> verifica_op TVoid (converte_tipo_option
                    t2) tipos_op
155               | (Some TString) -> verifica_op TString (
                    converte_tipo_option t2) tipos_op
156               | _ -> print_endline("O tipo das parcelas nao foi
                    encontrado");
157                             failwith("Erro Semantico: verifica_primitiva")
158      with e-> print_endline "Erro: verifica_primitiva";
159          raise e
160
161 (* Verifica expressao a direita*)
162 and verifica_exp_dir amb expr current =
163     match expr.valor with
164         Some (ExpInt _ )-> expr.tipo <- (Some TInt)
165         | Some(ExpFloat _ )-> expr.tipo <- (Some TFloat)
166         | Some (ExpString _) -> expr.tipo <- (Some TString)
167         | Some(ExpBool _)    -> expr.tipo <- (Some TBool)
168         | Some(ExpVar v) -> expr.tipo <-  (verifica_var_dir amb expr.pos v
              current)
169         | Some(ExpUn (not,expressao)) -> verifica_exp_dir amb expressao
              current;
170                                                 expr.tipo <-(Some TBool)
171         | Some(ExpBin (op,e1,e2)) ->( verifica_exp_dir amb e1 current;
172                                 verifica_exp_dir amb e2 current;
173                                 expr.tipo <- (converte_tipo (
                                    verifica_primitiva op  (tipo e1)  (tipo
                                    e2))))
174         | _ -> print_endline ("A expressao a direita contem erros: ");
175                 failwith "Erro semantico: verifica_exp_dir"
176
```

```
177 (*Verifica expressao a esquerda*)
178 and verifica_exp_esq amb expr tipo current =
179     match expr.valor with
180         Some (ExpInt _ )-> expr.tipo <- (Some TInt)
181         | Some(ExpFloat _ )-> expr.tipo <- (Some TFloat)
182         | Some (ExpString _) -> expr.tipo <- (Some TString)
183         | Some(ExpBool _)    -> expr.tipo <- (Some TBool)
184         | Some(ExpVar v) -> expr.tipo <-  (verifica_var_esq amb expr.pos v
              tipo current)
185         | Some(ExpUn (not,expressao)) -> verifica_exp_dir amb expressao
              current;
186                                          expr.tipo <-(Some TBool)
187         | Some(ExpBin (op,e1,e2)) -> verifica_exp_dir amb e1 current;
188                                              verifica_exp_dir amb e2 current;
189                                           expr.tipo <- (converte_tipo (
                                             verifica_primitiva op  e1.tipo
                                             e2.tipo))
190         | _ -> print_endline ("A expressao a esquerda contem erros: ");
191                failwith "Erro semantico: verifica_exp_esq"
192
193 (* Verifica o retorno de uma funcao *)
194 and verifica_retorno_func amb nomeFunc =
195     try
196         let reg = Hashtbl.find amb nomeFunc in
197         match reg with
198         | EntFn tab -> tab.tiporetorno
199         | _ -> print_endline("O nome '" ^ nomeFunc ^ "' nao esta associado a
              uma funcao.");
200                failwith "Erro Semantico: verifica_retorno_func"
201  with e-> print_endline "Erro Semantico: verifica_retorno_func";
202               raise e
203
204
205 (* Verifica o retorno de uma funcao void *)
206 let retorna_tipo_funcao amb nomeFuncao =
207     try
208         let tab = Hashtbl.find amb nomeFuncao in
209         (match tab with
210            | EntFn entFun -> (match entFun.tiporetorno with
211                 Some tipo -> entFun.tiporetorno
212            | None -> print_endline ("A funcao '" ^ nomeFuncao ^ "' nao tem
                 um tipo definido.");
213                  failwith "Erro semantico: retorna_tipo_funcao")
214              | _ -> print_endline ("O nome '" ^ nomeFuncao ^ "' esta
                   associado a uma variavel.");
215               failwith "Erro semantico: retorna_tipo_funcao")
216         with
217            Not_found -> print_endline ("A funcao '" ^ nomeFuncao ^ "' nao
                 foi definida.");
218            failwith "Erro semantico: retorna_tipo_funcao"
219
220 (* Verifica os argumentos que sao variaveis *)
221 let verifica_var_arg amb var =
222     try
223         let entrada = Hashtbl.find amb var in
224         (match entrada with
225            | EntVar entVar -> entVar.tipagem
226            | _ -> print_endline ("O nome '" ^ var ^ " 'esta associado a uma
```

```
                funcao.");
227              failwith "Erro semantico: verifica_var_arg")
228         with
229         Not_found -> print_endline ("Variavel '" ^ var ^ "' nao definida.");
230         failwith "Erro semantico: verifica_var_arg"
231
232 (* Verifica argumentos *)
233 let rec verifica_args args amb=
234         (match args with [] -> ignore()
235         | arg :: args -> verifica_arg arg amb;
236                  verifica_args args amb)
237
238 (* Verifica argumento *)
239 and verifica_arg arg amb=
240     (match arg.valor with
241         (Some ExpInt _ )-> arg.tipo <- (Some TInt)
242         | (Some ExpFloat _ ) -> arg.tipo <- (Some TFloat)
243         | (Some ExpString _ )-> arg.tipo <- (Some TString)
244         | (Some ExpVar (VarSimples v)) -> arg.tipo <- (verifica_var_arg amb
              v)
245         | _ -> print_endline ("Nao e permitido usar operacao como argumento
              de chamada de funcao");
246              failwith "Erro semantico: verifica_arg")
247
248 (* Verifica comandos *)
249 let rec verifica_cmds amb cmds current param =
250     match cmds with [] -> ignore()
251         | cmd :: cmds -> verifica_cmd amb cmd current param;
252                          verifica_cmds amb cmds current param
253
254
255 (* Verifica comando *)
256 and verifica_cmd amb cmd current param =
257     match cmd.vcmd with
258         | ChamaFuncaoAtrib (e1, nomeFunc, arg) -> let tiporetorno =
259             verifica_retorno_func amb nomeFunc in
260         (match tiporetorno with
261            | (Some TInt) -> verifica_exp_esq amb e1 TInt current
262            | (Some TFloat) -> verifica_exp_esq amb e1 TFloat current
263            | (Some TString) -> verifica_exp_esq amb e1 TString current
264            | (Some TBool) -> verifica_exp_esq amb e1 TBool current
265            | _ -> print_endline("Tipo de retorno nao implementado");
266                 failwith("Erro Semantico: ChamaFuncaoAtrib")
267         );
268         verifica_args arg amb
269         | ChamaFuncaoVoid (nomeFunc, arg) -> ignore()(* retorna_tipo_funcao
              amb nomeFunc *)
270         | CmdPrint (e) -> verifica_exp_dir amb e current
271         | CmdInput (e1, e2) -> e1.tipo <- (Some TString);
272                                verifica_exp_esq amb e1 TString current;
273                                verifica_exp_dir amb e2 current
274         | CmdIntParse (e1, e2) ->  e1.tipo <- (Some TInt);
275                                    verifica_exp_esq amb e1 TInt current;
276                                    verifica_exp_dir amb e2 current
277         | CmdAtrib (e1,e2) ->  verifica_exp_dir amb e2 current;
278         let t1=tipo e2 in
279         (match t1 with
280            | (Some TInt) -> verifica_exp_esq amb e1 TInt current
```

```
281             | (Some TFloat) -> verifica_exp_esq amb e1 TFloat current
282             | (Some TString) -> verifica_exp_esq amb e1 TString current
283             | (Some TBool) -> verifica_exp_esq amb e1 TBool current
284                 | _ -> print_endline("O tipo da expressao a direita nao foi
                        encontrado");
285                  failwith("Erro Semantico: CmdAtrib")
286         )
287         | CmdIf (e, ce, cs) -> verifica_exp_dir amb e current;
288         begin verifica_cmds amb ce current param;
289         (match cs with
290                 | None -> ignore()
291                 | Some cmds -> verifica_cmds amb cmds current param)
292         end
293         | CmdWhile (e, cs) -> verifica_exp_dir amb e current;
294                 verifica_cmds amb cs current param
295         | CmdFor (v, range, cmds) -> verifica_exp_dir amb v current;
296           v.tipo <- (Some TInt);
297              (match range.vcmd with
298                 | CmdRange (ini, fim, inc) -> ignore ()
299                 | _ -> erro "range" cmd.pcmd "Range invalida");
300                   verifica_cmds amb cmds current param
301         | CmdReturn (e) -> verifica_exp_dir amb e current
302         | _ -> erro "verifica_cmd" cmd.pcmd "Comando nao definido. Erro
              Semantico."
303
304 (*Insere uma nova funcao na tabela*)
305 let insere_nova_funcao amb func =
306     try
307         let entrada = Hashtbl.find amb func.idF in
308         (match entrada with
309             | EntVar _ -> print_endline ("O nome '" ^ func.idF ^ "' esta
                  associado a uma variavel.");
310                       failwith "Erro semantico: insere_nova_funcao "
311             | _ -> print_endline ("A funcao '" ^ func.idF ^ "' ja foi
                  definida."
312         );
313         failwith "Erro semantico: insere_nova_funcao")
314         with
315         Not_found -> let t1= func.returnF in
316                  (match t1 with
317                   | (Some TInt) -> Hashtbl.add amb func.idF (EntFn (
                        cria_ent_func TInt func.paramsF func.varLocaisF))
318                   | (Some TFloat) -> Hashtbl.add amb func.idF (EntFn (
                        cria_ent_func TFloat func.paramsF func.varLocaisF))
319                   | (Some TString) -> Hashtbl.add amb func.idF (EntFn (
                        cria_ent_func TString func.paramsF func.varLocaisF))
320                   | (Some TBool) -> Hashtbl.add amb func.idF (EntFn (
                        cria_ent_func TBool func.paramsF func.varLocaisF))
321                   | (Some TVoid) -> Hashtbl.add amb func.idF (EntFn (
                        cria_ent_func TVoid func.paramsF func.varLocaisF))
322                   | _ -> print_endline("O tipo dessa funcao nao foi
                        implementado");
323                       failwith("Erro Semantico: insere_nova_funcao")
324                  )
325
326 (* Verificacao das funcoes *)
327 let rec verifica_funcs amb funcs =
328     match funcs with [] -> ignore()
```

```
329         | func :: funcs -> verifica_func amb func; verifica_funcs amb funcs
330
331 (* Verificacao de funcao *)
332 and verifica_func amb func =
333     insere_nova_funcao amb func;
334     current_func := func.idF;
335     verifica_cmds amb func.cmdsF !current_func func.paramsF;
336     let entFun = Hashtbl.find amb !current_func in
337      (match entFun with
338         | EntFn funcao -> let novo_reg = { varLocais = funcao.varLocais;
339                                            tiporetorno = funcao.
                                               tiporetorno;
340                                            param = funcao.param } in
341                 Hashtbl.replace amb !current_func (EntFn novo_reg);
342                                 func.varLocaisF <- funcao.varLocais
343         | _ -> print_endline ("O nome '" ^ !current_func ^ "' esta associado
                a uma variavel.");
344                failwith "Erro semantico: verifica_func");
345                try
346                     let entTab = Hashtbl.find amb !current_func in
347                        (match entTab with
348                                 | EntFn entFunc -> (if (entFunc.tiporetorno
                                      == None) then
349                                    entFunc.tiporetorno <- Some TVoid);
350                                          func.returnF <- entFunc.
                                             tiporetorno
351                            | _ -> print_endline ("O nome '" ^ !current_func
                                ^ "' esta associado a uma varivel.");
352                                failwith "Erro semantico: verifica_func")
353                with e-> print_endline("Erro: A funcao '" ^ !current_func ^
                    "' nao foi encontrado.");
354                  raise e
355
356 (* Verifica o programa *)
357 let verifica_prog amb arv =
358     verifica_funcs amb arv.funcsP;
359     current_func := "";
360     verifica_cmds amb arv.cmdsP !current_func []
361
362 let semantico arv =
363     let ambiente = Hashtbl.create 23 in
364         verifica_prog ambiente arv;
365         ambiente
```

Listing 4.10: Analisador Semântico do MiniPython

Ao executarmos o analisador semântico no nosso código fonte usado como exemplo, devemos ter a seguinte saída:

```
1  Asa.programa * (string, Asa.entradaTabela) Hashtbl.t =
2  ({funcsP =
3     [{idF = "contador";
4       paramsF =
5        [("a",
6          {tipagem = Some TInt; v_inicial = None; endereco = None;
7           valor_variavel = ExpInt 0});
8         ("b",
9          {tipagem = Some TInt; v_inicial = None; endereco = None;
10          valor_variavel = ExpInt 0})];
```

```
    cmdsF =
     [{vcmd =
        CmdAtrib
         ({valor = Some (ExpVar (VarSimples "soma")); tipo = Some TInt;
           pos =
            {Asa.Posicao.lin_inicial = 8; col_inicial = 17; lin_final = 8;
             col_final = 16}},
         {valor = Some (ExpInt 0); tipo = Some TInt;
          pos =
           {Asa.Posicao.lin_inicial = 8; col_inicial = 17; lin_final = 8;
            col_final = 16}});
       pcmd =
        {Asa.Posicao.lin_inicial = 8; col_inicial = 17; lin_final = 8;
         col_final = 16}};
      {vcmd =
        CmdAtrib
         ({valor = Some (ExpVar (VarSimples "i")); tipo = Some TInt;
           pos =
            {Asa.Posicao.lin_inicial = 8; col_inicial = 17; lin_final = 8;
             col_final = 16}},
         {valor = Some (ExpInt 0); tipo = Some TInt;
          pos =
           {Asa.Posicao.lin_inicial = 8; col_inicial = 17; lin_final = 8;
            col_final = 16}});
       pcmd =
        {Asa.Posicao.lin_inicial = 8; col_inicial = 17; lin_final = 8;
         col_final = 16}};
      {vcmd =
        CmdAtrib
         ({valor = Some (ExpVar (VarSimples "c")); tipo = Some TInt;
           pos =
            {Asa.Posicao.lin_inicial = 8; col_inicial = 17; lin_final = 8;
             col_final = 16}},
         {valor = Some (ExpString "2"); tipo = Some TString;
          pos =
           {Asa.Posicao.lin_inicial = 8; col_inicial = 17; lin_final = 8;
            col_final = 16}});
       pcmd =
        {Asa.Posicao.lin_inicial = 8; col_inicial = 17; lin_final = 8;
         col_final = 16}};
      {vcmd =
        CmdIntParse
         ({valor = Some (ExpVar (VarSimples "d")); tipo = Some TInt;
           pos =
            {Asa.Posicao.lin_inicial = 8; col_inicial = 17; lin_final = 8;
             col_final = 16}},
         {valor = Some (ExpVar (VarSimples "c")); tipo = Some TString;
          pos =
           {Asa.Posicao.lin_inicial = 8; col_inicial = 17; lin_final = 8;
            col_final = 16}});
       pcmd =
        {Asa.Posicao.lin_inicial = 8; col_inicial = 17; lin_final = 8;
         col_final = 16}};
      {vcmd =
        CmdFor
         ({valor = Some (ExpVar (VarSimples "i")); tipo = Some TInt;
           pos =
            {Asa.Posicao.lin_inicial = 8; col_inicial = 17; lin_final = 8;
```

```
69                col_final = 16}},
70            {vcmd = CmdRange (0, 5, 1);
71             pcmd =
72              {Asa.Posicao.lin_inicial = 8; col_inicial = 17; lin_final = 8;
73               col_final = 16}},
74            [{vcmd =
75               CmdAtrib
76                ({valor = Some (ExpVar (VarSimples "soma")); tipo = Some TInt;
77                  pos =
78                   {Asa.Posicao.lin_inicial = 8; col_inicial = 17;
79                    lin_final = 8; col_final = 16}},
80                {valor =
81                  Some
82                   (ExpBin (Mais,
83                     {valor = Some (ExpInt 0); tipo = Some TInt;
84                      pos =
85                       {Asa.Posicao.lin_inicial = 8; col_inicial = 17;
86                        lin_final = 8; col_final = 16}},
87                     {valor = Some (ExpInt 2); tipo = Some TInt;
88                      pos =
89                       {Asa.Posicao.lin_inicial = 8; col_inicial = 17;
90                        lin_final = 8; col_final = 16}}));
91                 tipo = Some TInt;
92                 pos =
93                  {Asa.Posicao.lin_inicial = 8; col_inicial = 17;
94                   lin_final = 8; col_final = 16}});
95              pcmd =
96               {Asa.Posicao.lin_inicial = 8; col_inicial = 17; lin_final = 8;
97                col_final = 16}}]);
98          pcmd =
99           {Asa.Posicao.lin_inicial = 8; col_inicial = 17; lin_final = 8;
100           col_final = 16}};
101        {vcmd =
102          CmdReturn
103           {valor = Some (ExpVar (VarSimples "soma")); tipo = Some TInt;
104            pos =
105             {Asa.Posicao.lin_inicial = 8; col_inicial = 17; lin_final = 8;
106              col_final = 16}};
107         pcmd =
108          {Asa.Posicao.lin_inicial = 8; col_inicial = 17; lin_final = 8;
109           col_final = 16}}];
110      returnF = Some TInt;
111      posF =
112       {Asa.Posicao.lin_inicial = 8; col_inicial = 17; lin_final = 8;
113        col_final = 16};
114      varLocaisF = <abstr>}];
115   cmdsP = []},
116  <abstr>)
```

Podemos perceber que a tabela devolvida pelo analisador semântico é a mesma devolvida pelo analisador sintático, sendo que agora temos todos os tipos inferidos, como o tipo de retorno da função, os tipos das expressões contidas no fonte Python, etc.

4.7 Interpretador

Interpretadores são "programas de computador"que leem um código fonte de uma linguagem interpretada e o converte em código executável. Seu funcionamento pode variar de acordo com a imple-

mentação. Nosso objetivo com o interpretador é, logo após a análise semântica, o código fonte estando limpo de erros, fazer a interpretação linha por linha, do fonte, verificando o que o fonte deseja realizar, operações matemáticas, etc. O fonte do interpretador é apresentado a seguir:

```
open Asa;;
open Printf;;

let current_func = ref ""
(* Funcoes para mostrar os erros na tela *)
let erro nome pos msg =
    let nlin = pos.Posicao.lin_inicial
    and ncol = pos.Posicao.col_inicial in
    let msglc = sprintf "Erro na linha %d, coluna %d" nlin ncol in
    print_endline msglc;
    print_endline (msg ^ nome);
    failwith "Erro semantico"

let tipo e = e.tipo

let analisa_var amb var current =
  match var with
    | VarSimples nome -> (
      if(current<>"") then
        let tabVar = Hashtbl.find amb current in
           match tabVar with
             EntFn tabFn ->
               (try (* tenta encontrar variavel local *)
                 let reg = Hashtbl.find tabFn.varLocais nome in
                       reg.valor_variavel
                       with Not_found -> (* tenta encontrar parametro *)
                         (match (procuraParam nome tabFn.param) with
                            Some v ->   v.valor_variavel
                            | None ->  (* tenta encontrar variavel global *)
                               try
                                 let entrada = Hashtbl.find amb nome in
                                   (match entrada with
                                      (EntVar v) ->   v.valor_variavel
                                      | _ -> failwith "busca_var_fun: erro"
                                    )
                             with e-> print_endline "Erro: verifica_var";
                                 raise e
                           )
                )
        | _ -> failwith("Erro ao analisar variavel")
      else (
        try
           let entradaGlobal = Hashtbl.find amb nome in
             (match entradaGlobal with
               (EntVar v) ->   v.valor_variavel
               | _ -> failwith "busca_var_fun: erro"
             )
         with e-> print_endline "Erro: verifica_var global";
            raise e
       )
  )
```

```
55 let converte_tipo_expr t1 =
56        match t1 with
57     ExpInt v -> Some(ExpInt v)
58     | ExpFloat v -> Some (ExpFloat v)
59     | ExpString v -> Some (ExpString v)
60     | ExpBool v -> Some (ExpBool v)
61     | ExpVar v -> Some (ExpVar v)
62     | _ -> failwith("converte_tipo_expr: tipo expressao nao encontrado")
63
64 let converte_tipo_expr_option t1 =
65        match t1 with
66     Some(ExpInt v) -> ExpInt v
67     | Some (ExpFloat v) -> ExpFloat v
68     | Some (ExpString v) -> ExpString v
69     | Some (ExpBool v) -> ExpBool v
70     | _ -> failwith("converte_tipo_expr_option: tipo nao encontrado")
71
72
73 let rec avalia_exp  expr amb current =
74     match expr.valor with
75         Some (ExpInt v )-> Some (ExpInt v )
76         | Some(ExpFloat v ) -> Some(ExpFloat v )
77         | Some (ExpString v) -> Some (ExpString v)
78         | Some(ExpBool v)    -> Some(ExpBool v)
79         | Some(ExpUn (not,expressao)) -> avalia_exp expressao amb current
80         | Some(ExpVar v) -> converte_tipo_expr (analisa_var amb v current)
81         | Some(ExpBin (op,e1,e2)) -> avalia_bin (op, e1, e2) amb current
82         | _ -> print_endline ("avalia_exp: A expressao contem erros: ");
83         failwith "Erro interpretador: avalia_exp"
84
85
86 and avalia_op_int_int op v1 v2 =
87   match op with
88       Igual -> (Some (ExpBool  (v1 == v2)))
89     | Diferente -> (Some (ExpBool (v1 != v2)))
90     | Maior -> (Some (ExpBool (v1 > v2)))
91     | MaiorIgual -> (Some (ExpBool  (v1 >= v2)))
92     | Menor -> (Some (ExpBool  (v1 < v2)))
93     | MenorIgual -> (Some (ExpBool  (v1 <= v2)))
94     | Div -> (Some (ExpInt  (v1 / v2)))
95     | Mais -> (Some (ExpInt  (v1 + v2)))
96     | Menos -> (Some (ExpInt  (v1 - v2)))
97     | Mult -> (Some (ExpInt  (v1 * v2)))
98     | _ -> failwith "Operacao invalida"
99
100 and avalia_op_int_float op v1 v2 =
101   match op with
102       Igual -> (Some (ExpBool  (float(v1) == v2)))
103     | Diferente -> (Some (ExpBool  (float(v1) != v2)))
104     | Maior -> (Some (ExpBool  (float(v1) > v2)))
105     | MaiorIgual -> (Some (ExpBool  (float(v1) >= v2)))
106     | Menor -> (Some (ExpBool  (float(v1) < v2)))
107     | MenorIgual -> (Some (ExpBool  (float(v1) <= v2)))
108     | Div -> (Some (ExpFloat  (float(v1) /. v2)))
109     | Mais -> (Some (ExpFloat  (float(v1) +. v2)))
110     | Menos -> (Some (ExpFloat  (float(v1) -. v2)))
111     | Mult -> (Some (ExpFloat  (float(v1) *. v2)))
112     | _ -> failwith "Operacao invalida"
```

```
113
114 and avalia_op_float_int op v1 v2 =
115   match op with
116       Igual -> (Some (ExpBool  (v1 == float(v2))))
117     | Diferente -> (Some (ExpBool  (v1 != float(v2))))
118     | Maior -> (Some (ExpBool  (v1 > float(v2))))
119     | MaiorIgual -> (Some (ExpBool  (v1 >= float(v2))))
120     | Menor -> (Some (ExpBool  (v1 < float(v2))))
121     | MenorIgual -> (Some (ExpBool  (v1 <= float(v2))))
122     | Div -> (Some (ExpFloat  (v1 /. float(v2))))
123     | Mais -> (Some (ExpFloat  (v1 +. float(v2))))
124     | Menos -> (Some (ExpFloat  (v1 -. float(v2))))
125     | Mult -> (Some (ExpFloat  (v1 *. float(v2))))
126     | _ -> failwith "Operacao invalida"
127
128 and avalia_op_float_float op v1 v2 =
129   match op with
130       Igual -> (Some (ExpBool  (v1 == v2)))
131     | Diferente -> (Some (ExpBool  (v1 != v2)))
132     | Maior -> (Some (ExpBool  (v1 > v2)))
133     | MaiorIgual -> (Some (ExpBool  (v1 >= v2)))
134     | Menor -> (Some (ExpBool  (v1 < v2)))
135     | MenorIgual -> (Some (ExpBool  (v1 <= v2)))
136     | Div -> (Some (ExpFloat (v1 /. v2)))
137     | Mais -> (Some (ExpFloat (v1 +. v2)))
138     | Menos -> (Some (ExpFloat (v1 -. v2)))
139     | Mult -> (Some (ExpFloat (v1 *. v2)))
140     | _ -> failwith "Operacao invalida"
141
142 and avalia_op_string_string op v1 v2 =
143   match op with
144       Igual -> (Some (ExpBool   (v1 == v2)))
145     | Diferente -> (Some (ExpBool   (v1 != v2)))
146     | Maior -> (Some (ExpBool   (v1 > v2)))
147     | MaiorIgual -> (Some (ExpBool   (v1 >= v2)))
148     | Menor -> (Some (ExpBool   (v1 < v2)))
149     | MenorIgual -> (Some (ExpBool   (v1 <= v2)))
150     | _ -> failwith "Operacao invalida"
151
152 and avalia_op_bool_bool op v1 v2 =
153   match op with
154       Igual -> (Some (ExpBool   (v1 == v2)))
155     | Diferente -> (Some (ExpBool   (v1 != v2)))
156     | And -> (Some (ExpBool   (v1 && v2)))
157     | Or -> (Some (ExpBool   (v1 || v2)))
158     | _ -> failwith "Operacao invalida"
159
160
161
162 and avalia_bin (op, exp1, exp2) amb current =
163   match (avalia_exp exp1 amb current) with
164     (Some ExpInt v1)  ->
165       (match (avalia_exp exp2 amb current) with
166           (Some ExpInt v2) -> avalia_op_int_int op v1 v2
167         | (Some ExpFloat v2) -> avalia_op_int_float op v1 v2
168         | _ -> failwith "Operador invalido")
169     | (Some ExpFloat v1) ->
170       (match (avalia_exp exp2 amb current) with
```

```
171             (Some ExpInt v2) -> avalia_op_float_int op v1 v2
172           | (Some ExpFloat v2) -> avalia_op_float_float op v1 v2
173           | _ -> failwith "Operador invalido")
174       | (Some ExpString v1) ->
175         (match (avalia_exp exp2 amb current) with
176             (Some ExpString v2) -> avalia_op_string_string op v1 v2
177           | _ -> failwith "Operador invalido")
178        | (Some ExpBool v1) ->
179         (match (avalia_exp exp2 amb current) with
180           | (Some ExpBool v2)  -> avalia_op_bool_bool op v1 v2
181           | _ -> failwith "Operador invalido")
182     | _ -> failwith "Operador invalido"
183
184
185
186   let avalia_print e =
187   match e with
188         (Some (ExpInt v)) -> print_int(v); print_char('\n')
189       | (Some (ExpFloat v)) -> print_float(v); print_char('\n')
190       | (Some (ExpString v)) -> print_string(v); print_char('\n')
191       | (Some (ExpBool v)) -> if (v == true) then
192                                             print_string ("true")
193                                         else print_string("false")
194       | _ -> ignore()
195
196 (* let avalia_atrib v expr amb current=
197   let valor = avalia_exp expr amb current  in
198        match v with
199         | VarSimples nome ->
200             let entrada = analisa_var amb v current  in
201                 entrada.valor_variavel <- valor *)
202
203 let avalia_atrib v expr amb current=
204   let valor = avalia_exp expr amb current  in
205        match v.valor with
206        | (Some ExpVar var) -> (match var with
207             | VarSimples nome ->
208                   if(current <> "") then
209                       let reg = Hashtbl.find amb current in
210                           let entrada = busca_var_fun amb reg nome  in
211                               entrada.valor_variavel <-
                                       converte_tipo_expr_option (valor)
212
213                   else( let entrada2 = busca_var amb nome in
214                                   entrada2.valor_variavel <-
                                           converte_tipo_expr_option (valor)
215                         )
216                 )
217        | _ -> failwith("Erro ao avaliar atribuicao")
218
219
220 let rec avalia_cmds amb cmds current =
221     match cmds with [] -> ignore()
222         | cmd :: cmds -> avalia_cmd amb cmd current;
223                                   avalia_cmds amb cmds current
224
225 and avalia_cmd amb cmd current =
226     match cmd.vcmd with
```

```
227               CmdPrint (e) -> avalia_print (e.valor)
228             | CmdInput  (e,amb) -> ignore()(* avalia_input (e.valor) amb *)
229             | CmdIntParse _ (*e1, e2*) -> ignore() (*avalia_intparse e1 e2
                   amb*)
230             | CmdAtrib (v,e) ->  avalia_atrib v e amb current
231             | CmdIf (e,cmd1,cmd2)-> avalia_if e cmd1 cmd2 amb current
232             | CmdWhile (e, cs) ->  avalia_while e cs amb current
233             | CmdFor (v, range, cmds)-> avalia_for v cmds range amb current
234             | CmdReturn(e) -> avalia_return e amb current
235             | _ -> failwith("avalia_cmd: erro")
236
237 and avalia_if exp cmd1 cmd2 ambiente current=
238       match avalia_exp exp ambiente current with
239            (Some ExpBool v) ->
240             if (v) then
241               avalia_cmds ambiente cmd1 current
242             else
243               (match cmd2 with
244                   Some cmd ->  avalia_cmds ambiente cmd current
245                 | _ -> ignore())
246         | _ -> failwith "Condicao invalida"
247
248 and avalia_for exp cmds range ambiente current =
249   (match range.vcmd with
250         | CmdRange (ini, fim, inc) -> ignore ()
251 (*          | _ -> erro "range" cmds.pcmd "Range invalida"); *)
252         | _ -> failwith "Range invalida");
253         avalia_cmds ambiente cmds current
254
255 and avalia_return exp ambiente current =
256   match avalia_exp exp ambiente current with
257        (Some (ExpInt v)) -> print_int(v); print_char('\n')
258       | (Some (ExpFloat v)) -> print_float(v); print_char('\n')
259       | (Some (ExpString v)) -> print_string(v); print_char('\n')
260       | (Some (ExpBool v)) -> if (v == true) then
261                                              print_string ("true")
262                                          else print_string("false")
263       | _ -> failwith("Erro ao avaliar retorno")
264
265   (* avalia_cmd ambiente cmd1 current;
266   (match cmd1 with
267       CmdAtrib (v, exp1) ->
268         (match (avalia_exp v ambiente current) with
269           ExpInt (Some val1) ->
270             (match (avalia_exp exp ambiente current) with
271                 ExpInt (Some val2) ->
272                   for var = val1 to val2 do
273                     avalia_cmd ambiente cmd2 current
274                   done
275               | _ -> failwith "Invalido")
276         |_ -> failwith "Valor invalido")
277     | _ -> failwith "Nao eh atribuicao") *)
278
279
280
281
282 and avalia_while exp cmd ambiente current=
283   match (avalia_exp exp ambiente current) with
```

```
      (Some (ExpBool v)) ->
        (let value = ref v in
            while !value do
              avalia_cmds ambiente cmd current;
              match (avalia_exp exp ambiente current) with
                   (Some (ExpBool v)) -> value := v
                 | _ -> failwith "Condicao invalida"
            done)
    | _ -> failwith "Condicao invalida"


let rec avalia_funcs amb funcs =
    match funcs with [] -> ignore()
        | func :: funcs -> avalia_func amb func;
                           avalia_funcs amb funcs

and avalia_func amb func =
                current_func := func.idF;
                avalia_cmds amb func.cmdsF !current_func

let avalia_prog amb arv = avalia_funcs amb arv.funcsP;
                                       current_func := "";
                                       avalia_cmds amb arv.cmdsP !
                                           current_func


let interpretador amb arv  =
        avalia_prog amb arv;
        amb
```

Listing 4.11: Interpretador do MiniPython

4.8 Gerador

O gerador é a última etapa do compilador. Pegando a árvore devolvida pelo analisador semântico que contém toda a estrutura do código, incluindo os tipos, o gerador será o responsável por fazer a análise desta árvore e gerar o código assembly do fonte correspondente. O gerador usará o dicionário Smali apresentado na seção 2.6.2 para traduzir nossos arquivos .py para .smali. O fonte do gerador segue abaixo:

```
open Asa;;
open Printf;;

(* Variaveis globais *)
let contador_registrador = ref 0
let first_reg = ref 0
let contador_rotulo = ref 0

let current_func = ref ""
let nomePrograma = ref ""

(* Retorna um novo numero de registrador. *)
let prox_registrador() =
    incr(contador_registrador);
    !contador_registrador

(* Retorna um novo label *)
```

```
18 let prox_rotulo() =
19     incr(contador_rotulo);
20     sprintf ":label_%d" !contador_rotulo
21
22 let tipo e = e.tipo
23
24 (*Procura os parametros da funcao *)
25 let procura_param ts id =
26     let reg = Hashtbl.find ts id in
27         match(reg) with
28                 EntFn ent -> ent.param
29                 | _ -> failwith("Erro gerador: procura_param")
30
31 (* Converte expressao para tipo option *)
32 let converte_tipo_expr t1 =
33        match t1 with
34        ExpInt v -> Some(ExpInt v)
35     | ExpFloat v -> Some (ExpFloat v)
36     | ExpString v -> Some (ExpString v)
37     | ExpBool v -> Some (ExpBool v)
38     | _ -> failwith("converte_tipo_expr: tipo nao encontrado")
39
40 (* Converte expressao option para expressao nao option *)
41 let converte_tipo_expr_option t1 =
42        match t1 with
43     Some(ExpInt v) -> ExpInt v
44     | Some (ExpFloat v) -> ExpFloat v
45     | Some (ExpString v) ->ExpString v
46     | Some (ExpBool v) -> ExpBool v
47     | Some (ExpVar v ) -> ExpVar v
48     | _ -> failwith("converte_tipo_expr_option: tipo nao encontrado")
49
50 (* Converte tipo_base para tipo_base option*)
51 let converte_tipo_base t1 =
52        match t1 with
53     TInt -> (Some TInt)
54     | TFloat -> (Some TFloat)
55     | TString -> (Some TString)
56     | TVoid -> (Some TVoid)
57     | TBool -> (Some TBool)
58
59 (*Converte tipo_base option para tipo_base*)
60 let converte_tipo_base_option t1 =
61        match t1 with
62      (Some TInt) ->TInt
63     | (Some TFloat) -> TFloat
64     | (Some TString) -> TString
65     | (Some TVoid) -> TVoid
66     | (Some TBool) -> TBool
67     | _ -> failwith("converte_tipo_base_option: tipo nao encontrado")
68
69 (*Converte int option para int*)
70 let converte_int t1 =
71        match t1 with
72      Some int -> int
73     | _ -> failwith("tipo invalido")
74
75 (* Recebe uma variavel ou parametro e retorna em qual registrador ela esta
```

```
ou adiciona um registrador pra ela *)
let endereco amb e current =
 match e with
  | ExpVar v ->(
    match v with
  | VarSimples nome -> (
    if(current<>"") then
     let tabVar = Hashtbl.find amb current in
        match tabVar with
          EntFn tabFn ->
            (try (* tenta encontrar variavel local *)
              let entrada = Hashtbl.find tabFn.varLocais nome in
                 (match (entrada.endereco) with
                    Some endr -> (Some endr)
                    | None -> let endr = prox_registrador() in
                       Hashtbl.add amb nome (EntVar { entrada with endereco
                          = Some endr});
                          (Some endr))
             with Not_found -> (* tenta encontrar parametro *)
               (match (procuraParam nome tabFn.param) with
                  Some v ->  (match (v.endereco) with
                      Some endr -> (Some endr)
                      | None -> let endr = prox_registrador() in
                          Hashtbl.add amb nome (EntVar { v with endereco =
                             Some endr});
                           (Some endr))
                  | None -> (* tenta encontrar variavel global *)
                      try
                       let entradaV = Hashtbl.find amb nome in
                         (match entradaV with
                           | EntVar entVar ->
                             (match (entVar.endereco) with
                                 Some endr -> (Some endr)
                                 | None -> let endr = prox_registrador()
                                    in
                                   Hashtbl.add amb nome (EntVar { entVar
                                      with endereco = Some endr});
                                   (Some endr))
                                 | _ -> failwith("erro gerador - endereco:
                                    variavel da funcao nao eh global")
                         )
                         with e-> print_endline "Erro gerador: endereco";
                             raise e
                 )
            )
    | _ -> failwith("endereco: funcao nao contem essa variavel")
     else (
      try
        let entradaMain = Hashtbl.find amb nome in
          (match entradaMain with
             | EntVar entVar ->
               (match (entVar.endereco) with
                  Some endr -> (Some endr)
                  | None -> let endr = prox_registrador() in
                     Hashtbl.add amb nome (EntVar { entVar with endereco =
                        Some endr});
                      (Some endr))
                  | _ -> failwith "erro gerador - endereco: variavel
```

```
                        invalida"
128           )
129           with e-> print_endline "Erro gerador: endereco";
130             raise e
131       )
132       ) )
133     |_ -> failwith "endereco: funcao esta sendo usada como variavel"
134
135
136 (* Formatacao das operacoes com 1 parametro*)
137 let op inst a = sprintf "%s %d" inst a
138
139 (* Formatacao das operacoes com 2 parametros*)
140 let opBinRR inst a1 a2 = sprintf "%s v%d, v%d" inst a1 a2
141 let opBinRN inst a1 a2 = sprintf "%s v%d, %d" inst a1 a2
142 let opBinRString inst a1 a2 = sprintf "%s v%d, \"%s\"" inst a1 a2
143
144 (* Formatacao das operacoes com 3 parametros*)
145 let opBinRNFloat inst a1 a2 = sprintf "%s v%d, %f" inst a1 a2
146 let opTriRRR inst a1 a2 a3 = sprintf "%s v%d v%d, v%d" inst a1 a2 a3
147
148 (* Formatacao das operacoes IF com 2 e 3 parametros*)
149 let opBinIF inst a target = sprintf "%s v%d, %s" inst a target
150 let opTriIF inst a1 a2 target = sprintf "%s v%d, v%d, %s" inst a1 a2 target
151
152 (* Tipos de opcodes *)
153 type dalvik_op =
154         Move of tipo_base * expr
155     | Const16 of tipo_base * expr
156     | OpBin of tipo_base * operadorBin * expr * expr
157 (* Operacoes binarias inteiro smali que utilizam dois registradores*)
158 let opr_int op =
159     match op with
160         Mais -> "add-int/2addr"
161     | Menos -> "sub-int/2addr"
162     | Mult -> "mul-int/2addr"
163     | Div -> "div-int/2addr"
164     | Modulo -> "rem-int/2addr"
165     | Igual -> "if-eq"
166     | Diferente -> "if-ne"
167     | Maior -> "if-gt"
168     | Menor -> "if-lt"
169     | MaiorIgual -> "if-ge"
170     | MenorIgual -> "if-le"
171     | _ -> failwith("opr_int: operador nao implementado")
172
173 (* Operacoes binarias float smali que utilizam dois registradores*)
174 let opr_float op =
175     match op with
176         Mais -> "add-float"
177     | Menos -> "sub-float"
178     | Mult -> "mul-float"
179     | Div -> "div-float"
180     | Modulo -> "rem-float"
181     | _ -> failwith "Operacao nao definida para float"
182
183 (* Operacoes binarias bool smali que utilizam dois registradores*)
184   let opr_bool op =
```

```
185   match op with
186       Igual -> "if-eq"
187     | Diferente -> "if-ne"
188     | Maior -> "if-gt"
189     | Menor -> "if-lt"
190     | MaiorIgual -> "if-ge"
191     | MenorIgual -> "if-le"
192     | _ -> failwith "Operacao nao definida para bool"
193
194 (* Tipos de Retorno *)
195 let tipoRetorno op =
196     match op with
197         TInt -> "I"
198     | TFloat -> "F"
199     | TVoid -> "V"
200     | TString -> "Ljava/lang/String;"
201     | _ -> failwith "Retorno nao definido para esse tipo"
202
203 (* Escolhe o operador de acordo com o tipo dos parametros *)
204 let operador t op =
205     match t with
206         TInt -> opr_int op
207     | TFloat -> opr_float op
208     | TBool -> opr_bool op
209     | _ -> failwith "erro tipo de operador nao encontrado"
210
211
212
213 (* Gera o codigo da expressao ou operacao *)
214 let rec emite ts instr current=
215     match instr with
216     | Move (TInt, ExpInt i) ->
217         let reg = prox_registrador() in
218         ([ opBinRN "move" reg i ], reg)
219     | Const16 (TInt, ExpInt i) ->
220         let reg = prox_registrador() in
221         ([ opBinRN "const/16" reg i ], reg)
222     | Const16 (TFloat, ExpFloat i) ->
223         let reg = prox_registrador() in
224         ([ opBinRNFloat "const/16" reg i ], reg)
225     | Const16 (TString, ExpString i) ->
226         let reg = prox_registrador() in
227         ([ opBinRString "const-string" reg i ], reg)
228     | OpBin (tipo, op, ExpBin (op2, a1, a2), ExpBin (op3, a3, a4)) ->
229         let (lista1, reg1) = emite ts (OpBin (tipo, op2,
              converte_tipo_expr_option a1.valor, converte_tipo_expr_option a2.
              valor)) current
230         and (lista2, reg2) = emite ts (OpBin (tipo, op3,
              converte_tipo_expr_option a3.valor, converte_tipo_expr_option a4.
              valor)) current
231              in
232         let (lista3, reg3) = criaCodigoOperador tipo op reg1 reg2
233         in (lista1 @ lista2 @ lista3, reg3)
234     | OpBin (tipo, op, ExpBin (op2, a1, a2), i) ->
235         let (lista1, reg1) = emite ts (OpBin (tipo, op2,
              converte_tipo_expr_option a1.valor, converte_tipo_expr_option a2.
              valor)) current
236         and (lista2, reg2) = criaCodigo tipo i ts current in
```

```
237         let (lista3, reg3) = criaCodigoOperador tipo op reg1 reg2
238         in (lista1 @ lista2 @ lista3, reg3)
239     | OpBin (tipo, op, i, ExpBin (op2, a1, a2)) ->
240         let (lista1, reg1) = emite ts (OpBin (tipo, op2,
                converte_tipo_expr_option a1.valor, converte_tipo_expr_option a2.
                valor)) current
241         and (lista2, reg2) = criaCodigo tipo i ts current in
242         let (lista3, reg3) = criaCodigoOperador tipo op reg1 reg2
243         in (lista2 @ lista1 @ lista3, reg3)
244     | OpBin (tipo, op, i, j) ->
245         let (lista1, reg1) = criaCodigo tipo i ts current
246         and (lista2, reg2) = criaCodigo tipo j ts current in
247         let (lista3, reg3) = criaCodigoOperador tipo op reg1 reg2
248         in(lista1 @ lista2 @ lista3, reg3)
249     | _ -> failwith "Erro emite: opcao nao implementada"
250
251
252
253
254
255 (* Cria codigo para o emite de acordo com o tipo do termo*)
256 and criaCodigo tipo termo ts current=
257     match termo with
258     | ExpInt t -> let reg = prox_registrador() in
259                         ([ opBinRN "const/16" reg t ], reg)
260
261     | ExpVar t -> let endr = endereco ts (ExpVar t) current in ([],
            converte_int endr)
262
263     | ExpFloat t -> let reg = prox_registrador() in
264                             ([ opBinRNFloat "const/high16" reg t ], reg)
265
266     | ExpString t -> let reg = prox_registrador() in
267                             ([ opBinRString "const-string" reg t ], reg)
268
269     | _ -> failwith "CriaCodigo: termo nao implementado"
270
271 (* Cria o codigo do operador de acordo com o numero de parametros*)
272     and criaCodigoOperador tipo op reg1 reg2 =
273     match tipo with
274         | TInt -> ([opBinRR (operador tipo op) reg1 reg2], reg1)
275         | TFloat -> let reg = prox_registrador() in
276             let (lista, regist) = ([opTriRRR (operador tipo op) reg reg1
                    reg2], reg)
277                 in (lista, regist)
278         |_ -> failwith "CriaCodigoOperador: tipo de operador nao
                implementado"
279
280 (* Gera a expressao chamando a funcao emite *)
281 let rec gen_expressao ts exp current=
282     match exp.valor with
283     | (Some ExpBin (op, a1, a2)) ->
284         emite ts (OpBin (converte_tipo_base_option exp.tipo, op,
                converte_tipo_expr_option a1.valor, converte_tipo_expr_option a2.
                valor)) current
285     | _ -> failwith "gen_expressao: expressao nao implementada"
286
287 (* Gera uma condicao *)
```

```
let rec gen_condicao ts exp rotulo current =
    match exp.valor with
    | (Some ExpBin(op, a1, a2)) ->
        let (lista1, reg1) = criaCodigo (converte_tipo_base_option exp.tipo)
            (converte_tipo_expr_option a1.valor) ts current
        and (lista2, reg2) = criaCodigo (converte_tipo_base_option exp.tipo)
            (converte_tipo_expr_option a2.valor)  ts current in
        (lista1 @ lista2 @ [opTriIF (operador (converte_tipo_base_option exp
           .tipo) op) reg1 reg2
        rotulo] , reg1, reg2)
| _ -> failwith "gen_condicao: condicao nao implementada"

(* a.valor -> recebe um numero e retorna um numero de registrador. Se
necessario, ele gera codigo para mover o numero para o registrador *)

(* Gera a atribuicao *)
let gen_atrib amb esq dir current=
    let reg1 = endereco amb (converte_tipo_expr_option esq.valor) current in
    (match dir.valor with
      | (Some ExpInt i) -> let (lista, reg2) = emite amb (Const16 (
        converte_tipo_base_option dir.tipo, ExpInt i)) current in
                                      lista @ [opBinRR "move"  (
                                         converte_int reg1) reg2]
      | (Some ExpFloat i) -> let (lista, reg2) = emite amb (Const16 (
        converte_tipo_base_option dir.tipo, ExpFloat i)) current in
                                       lista @ [opBinRR "move" (
                                          converte_int reg1) reg2]
      | (Some ExpString i) -> let (lista, reg2) = emite amb (Const16 (
        converte_tipo_base_option dir.tipo, ExpString i)) current in
                                        lista @ [opBinRR "move-object"
                                           (converte_int reg1) reg2]
      | (Some ExpVar (VarSimples _)) -> let reg2 = endereco amb (
        converte_tipo_expr_option dir.valor ) current in
            (match converte_tipo_base_option dir.tipo with
                 | TInt -> [opBinRR "move" (converte_int reg1) (
                    converte_int reg2)]
                 | _ -> [opBinRR "move-object" (converte_int reg1) (
                    converte_int reg2)])
                 | _ -> let (lista, reg2) = gen_expressao amb dir current
                    in lista @ [opBinRR "move" (converte_int reg1) reg2])

(* Gera o comando if/else *)
let rec gen_if amb teste cmdsEntao cmdsSenao current =
    let lE = prox_rotulo()
        and l1 = prox_rotulo() in
    let lS = match cmdsSenao with
        | None -> l1
        | Some _ -> prox_rotulo() in
    let (cod_teste, reg_teste1, reg_teste2) = gen_condicao amb teste lE
       current in
    let cod_entao = gen_cmds amb cmdsEntao current in
    let cod_senao =
            match cmdsSenao with
        | None -> []
        | Some cmdsS -> gen_cmds amb cmdsS current in
              let codigo = cod_teste @ cod_senao @ [sprintf "goto %s" lS]
                 @ [sprintf "%s" lE] @ cod_entao @ [sprintf "%s" lS] in
                 codigo
```

```
330
331 (* Gera o comando while *)
332 and gen_while amb teste cmds current =
333     let labelE = prox_rotulo()
334     and labelE1 = prox_rotulo ()
335     and labelS = prox_rotulo() in
336     let (cod_teste, _, _) = gen_condicao amb teste labelE1 current in
337     let cod_cmd = gen_cmds amb cmds current in
338     let codigo = [sprintf "%s" labelE] @ cod_teste @ [sprintf "goto %s"
        labelS] @ [sprintf "%s" labelE1] @ cod_cmd @ [sprintf "goto %s"
        labelE] @ [sprintf "%s" labelS]
339
340     in codigo
341
342 (* Gera o comando for *)
343 and gen_for amb var cmd cmds current=
344     let lfor = prox_rotulo()
345         and lout = prox_rotulo()
346         and laux = prox_rotulo() in
347     match cmd with
348     | CmdRange (fim, ini, inc) ->
349         let (cod_range, reg_fim, reg_ini, reg_inc) = gen_range fim ini inc
            in
350             let reg_var = endereco amb var current in
351                 let cod_inicio = [ sprintf "const/16 v%d, %d" (converte_int
                    reg_var) fim ] in
352             let cod_teste = [opTriIF "if-lt" (converte_int reg_var) reg_fim
                laux] in
353             let cod_cmds = gen_cmds amb cmds current in
354             let cod_incremento = [opBinRR (opr_int Mais) (converte_int
                reg_var) reg_inc] in
355             let codigo = cod_range @ cod_inicio @ [sprintf "%s" lfor] @
                cod_teste @ [sprintf "goto %s" lout] @ [sprintf "%s" laux] @
                cod_cmds @ cod_incremento @ [sprintf "goto %s" lfor] @ [
                sprintf "%s" lout]
356             in codigo
357     | _ -> []
358
359 (* Gera o comando range *)
360 and gen_range ini fim inc =
361     let reg_fim = prox_registrador()
362     and reg_ini = prox_registrador()
363     and reg_inc = prox_registrador() in
364     ([opBinRN "const/16" reg_ini ini] @ [opBinRN "const/16" reg_fim fim] @ [
        opBinRN "const/16" reg_inc inc], reg_fim, reg_ini, reg_inc)
365 (* Gera o comando print *)
366 and gen_print amb exp current=
367     match exp.valor with
368     | (Some ExpString e) -> let reg1 = prox_registrador() in
369         let cod1 = [sprintf "sget-object v%d, Ljava/lang/System;->out: Ljava
            /io/PrintStream;" reg1] in
370         let (cod2, reg2) = emite amb (Const16 (TString, ExpString e))
            current in
371         let cod3 = [sprintf "invoke-virtual {v%d, v%d}, Ljava/io/PrintStream
            ;->println(Ljava/lang/String;)V" reg1 reg2] in
372         let codigo = cod1 @ cod2 @ cod3 in codigo
373     | (Some ExpVar _ ) -> let reg1 = prox_registrador() in
374                           let cod1 = [sprintf "sget-object v%d, Ljava/lang/
```

```
                                System;->out:Ljava/io/PrintStream;" reg1] in
375                           let reg2 = endereco amb (
                                converte_tipo_expr_option exp.valor ) current
                                in
376         (match exp.tipo with
377         | (Some TInt) -> let cod3 = [sprintf "invoke-virtual {v%d, v%d},
               Ljava/io/PrintStream;->println(I)V" reg1 (converte_int reg2)] in
378                          let codigo = cod1 @ cod3 in codigo
379         | (Some TString) -> let cod3 = [sprintf "invoke-virtual {v%d, v%d},
               Ljava/io/PrintStream;->println(Ljava/lang/String;)V" reg1 (
               converte_int reg2)] in
380                             let codigo = cod1 @ cod3 in codigo
381         | _ -> failwith "Print so aceita variaveis int e string")
382
383 | _ -> failwith "Print soh esta aceita strings e variaveis"
384 (* Gera o comando raw_input *)
385 and gen_input amb exp1 exp2 current=
386   (match exp2.tipo with
387    | (Some TString) ->
388      (let reg1 = prox_registrador() in
389       let cod1 = [sprintf "new-instance v%d, Ljava/util/Scanner;" reg1] in
390       let reg2 = prox_registrador() in
391       let cod2 = [sprintf "sget-object v%d, Ljava/lang/System;->in:Ljava/io
             /InputStream;" reg2] in
392       let cod3 = [sprintf "invoke-direct {v%d, v%d}, Ljava/util/Scanner;-><
             init>(Ljava/io/InputStream;)V" reg1 reg2] in
393       let codigo = cod1 @ cod2 @ cod3 in
394        (match exp1.tipo with
395           | (Some TInt) -> let codInt = codigo @ [sprintf "invoke-virtual {
                 v%d}, Ljava/util/Scanner;->nextInt()I" reg1] @
396                                          [sprintf "move-result v%d" reg2] in
                                             gen_print amb exp2 current @
                                             codInt
397           | (Some TString) -> let moveReg = endereco amb (
                 converte_tipo_expr_option exp1.valor) current in
398                                            let codStr = codigo @
399                                              [sprintf "invoke-virtual {v%d
                                                   }, Ljava/util/Scanner;->
                                                   nextLine()Ljava/lang/String
                                                   ;" reg1] @
400                                              [sprintf "move-result-object v
                                                   %d" reg2] @
401                                              [sprintf "move-object v%d, v%d
                                                   " (converte_int moveReg)
                                                   reg2] in
402                                              gen_print amb exp2 current @
                                                   codStr
403           | _ -> [sprintf "input soh esta aceitando strings e inteiros"]))
404     | _ -> [sprintf "input soh esta aceitando strings dentro nos parametros"
          ])
405 (* Gera o comando x = int(str) *)
406 and gen_int_parse amb exp1 exp2 current=
407  let regF = endereco amb ( converte_tipo_expr_option exp1.valor) current in
408   (match exp2.valor with
409     | (Some ExpString e) -> let (cod1, reg1) = emite amb (Const16 (TString,
          ExpString e)) current in
410        let cod2 = [sprintf "invoke-static {v%d}, Ljava/lang/Integer;->
             parseInt(Ljava/lang/String;)I" reg1] in
```

```
411         let reg2 = prox_registrador() in
412         let cod3 = [sprintf "move-result v%d" reg2] in
413         let codigo = cod1 @ cod2 @ cod3 @ [sprintf "move v%d v%d" (
                converte_int regF) reg2] in codigo
414     | (Some ExpVar _ )-> let endr = endereco amb ( converte_tipo_expr_option
            exp2.valor ) current in
415         (match exp2.tipo with
416           | (Some TString) -> let cod2 =
417               [sprintf "invoke-static {v%d},Ljava/lang/Integer;->parseInt(
                    Ljava/lang/String;)I" (converte_int endr)] in
418               let reg2 = prox_registrador() in
419               let cod3 = [sprintf "move-result v%d" reg2] in
420               let codigo = cod2 @ cod3 @ [sprintf "move v%d, v%d" (
                    converte_int regF) reg2] in codigo
421           | _ -> failwith "apenas string para o parse")
422     | _ -> failwith "parse soh aceita strings e variaveis")
423 (* Gera o comando return *)
424 and gen_return ts exp current =
425     let entrada = Hashtbl.find ts !current_func in
426     (match entrada with
427     | EntFn entFun ->
428         let retorno = entFun.tiporetorno in
429         (match retorno with
430             | Some _ -> (match exp.valor with
431             | (* (Some ExpVar _) -> let reg = endereco entFun.varLocais (
                    converte_tipo_expr_option exp.valor) current in
432                                                                 [sprintf "return
                                                                      v%d" (
                                                                     converte_int
                                                                     reg)] *)
433               (Some ExpVar _) -> let reg = endereco ts (
                    converte_tipo_expr_option exp.valor) current in
434                                                                 [sprintf "return
                                                                      v%d" (
                                                                     converte_int
                                                                     reg)]
435             | (Some ExpInt i) -> let reg = prox_registrador() in [opBinRN "
                    const/16" reg i] @ [sprintf "return v%d" reg]
436             | (Some ExpString s) -> let reg = prox_registrador() in [
                    opBinRString "const-string" reg s] @ [sprintf "return v%d"
                    reg]
437             | _ -> [sprintf "return aceitando apenas variaveis, int e string
                    "] )
438             | _ -> failwith "gen_return"
439         )
440     | _ -> failwith "gen_return - variavel encontrada")
441 (* Gera o comando chamada de funcao *)
442 and gen_chamadaFuncao ts nomeFun arg current =
443     let entrada = Hashtbl.find ts nomeFun in
444         (match entrada with
445             | EntFn entFun ->
446                 let params = entFun.param in
447                 let retorno = entFun.tiporetorno in
448                 let cod1 = "invoke-static {" in
449                 let (cod2, cod0) = regist_arg arg ts params current in
450                 let cod3 = "}, L" ^ !nomePrograma ^ ";->" ^ nomeFun ^ " ("
                        in
451                 let cod4 = (tipos_params params) ^ ")" in
```

```
452                 (match retorno with
453                     | Some a -> let cod5 = tipoRetorno a in
454                       let codFinal = cod0 ^ cod1 ^ cod2 ^ cod3 ^ cod4 ^
                              cod5 in [codFinal]
455                     | _ -> failwith "funcao com retorno none:
                          gen_chamadaFuncao")
456                     | _ -> failwith "gen_chamadaFuncao - variavel encontrada
                          ")
457
458 (* Gera o comando chamada de funcao com atribuicao *)
459 and gen_chamadaFuncaoAtrib ts exp nomeFun arg current =
460     let lista = gen_chamadaFuncao ts nomeFun arg current in
461     let entrada = Hashtbl.find ts nomeFun in
462     (match entrada with
463         | EntFn entFun ->
464             let regExp = endereco ts (converte_tipo_expr_option exp.valor)
                    current in
465             let reg = prox_registrador() in
466             lista @ [sprintf "move-result v%d" reg] @ [sprintf "move v%d, v%
                    d" (converte_int regExp) reg]
467         | _ -> failwith "gen_chamadaFuncaoAtrib - variavel encontrada"
468     )
469
470 (* Gera comando *)
471 and gen_cmd ts c current=
472     match c with
473     | CmdIf (teste, cmdsE, cmdsS) -> gen_if ts teste cmdsE cmdsS current
474     | CmdAtrib (esq,dir) -> gen_atrib ts esq dir current
475     | CmdWhile (teste,cmds) -> gen_while ts teste cmds current
476     | CmdFor (var, cmd, cmds) -> gen_for ts (converte_tipo_expr_option var.
          valor) cmd.vcmd cmds current
477     | CmdFuncao (id, param, cmds) -> gen_func ts id param cmds
478     | CmdPrint (exp) -> gen_print ts exp current
479     | CmdInput (exp1, exp2) -> gen_input ts exp1 exp2 current
480     | CmdReturn (exp) -> gen_return ts exp current
481     | ChamaFuncaoAtrib (exp, nomeFun, argum) -> gen_chamadaFuncaoAtrib ts
          exp nomeFun argum current
482   (*    | ChamaFuncaoVoid (nomeFun, argum) -> gen_chamadaFuncaoVoid ts
        nomeFun argum  *)
483     | CmdIntParse (exp1, exp2) -> gen_int_parse ts exp1 exp2 current
484     | _ -> failwith "gen_cmd: Comando nao implementado"
485
486 (* Gera comandos *)
487 and gen_cmds ts cs current=
488     match cs with
489         | c :: cs -> let lista = gen_cmd ts c.vcmd current in lista @
              gen_cmds ts cs current
490         | [] -> []
491
492 (* Gera string com os tipos de parametros (IIF) *)
493 and tipos_params ps =
494     match ps with
495         | (p1,p2) :: ps -> let str = tipoRetorno(converte_tipo_base_option
              p2.tipagem)
496                 in str ^ tipos_params ps
497         | [] -> ""
498
499 (* Retorna os tipos dos argumentos e codigo necessario caso argumento seja
```

```
   variavel *)
(* Retorna tupla (tipo de retornos, codigo de move's) *)
and regist_arg ars ts parametros current =
    match ars with
        | a :: [] ->
            (match a.valor with
                | (Some ExpInt i) ->
                    let reg1 = prox_registrador() in
                    let cod1 = sprintf "v%d" reg1 in
                    let cod3 = opBinRN "const/16" reg1 i in (cod1, cod3 ^ "\
                       n")
                | (Some ExpString s) ->
                    let reg1 = prox_registrador() in
                    let cod1 = sprintf "v%d" reg1 in
                    let cod3 = opBinRString "const-string" reg1 s in (cod1,
                       cod3 ^ "\n")
                | (Some ExpVar (VarSimples v)) ->
                    let entrada = Hashtbl.find ts v in
                        (match entrada with
                            | (EntVar entVar) -> let reg1 = endereco ts  (
                               converte_tipo_expr_option a.valor) current in
                                let cod1 = sprintf "v%d" (converte_int reg1)
                                    in (cod1, "")
                            | _ -> failwith "regist_arg"
                        )
                | _ -> failwith "Argumentos desse tipo nao estao sendo
                   tratados: regist_arg ts parametros")
                | a :: ars ->
                    (match a.valor with
                        | (Some ExpInt i) ->
                            let (cod2, cod4) = regist_arg ars ts parametros
                               current in
                            let reg1 = prox_registrador() in
                            let cod1 = sprintf "v%d, " reg1 in
                            let cod3 = opBinRN "const/16" reg1 i in (cod1 ^
                               cod2, cod3 ^ "\n" ^ cod4)
                        | (Some ExpString s) ->
                            let (cod2, cod4) = regist_arg ars ts parametros
                               current in
                            let reg1 = prox_registrador() in
                            let cod1 = sprintf "v%d, " reg1 in
                            let cod3 = opBinRString "const-string" reg1 s in
                                (cod1 ^ cod2, cod3 ^ "\n" ^ cod4)
                        | (Some ExpVar (VarSimples v)) ->
                            let entrada = Hashtbl.find ts v in
                            (match entrada with
                                | EntVar entVar -> let reg1 = endereco ts (
                                   converte_tipo_expr_option a.valor)
                                   current in
                                    let cod1 = sprintf "v%d, " (converte_int
                                        reg1) in
                                    let (cod2, cod4) = regist_arg ars ts
                                       parametros current in (cod1 ^ cod2,
                                       cod4)
                                | _ -> failwith "regist_arg"
                            )
                        | _ -> failwith "Argumentos desse tipo nao estao
                           sendo tratados: regist_arg ts parametros")
```

```
542                         | [] -> ("", "")
543
544 (* Gera String dos parametros *)
545 and coloca_param_tabela ps =
546     match ps with
547     | (p1,p2) :: ps ->
548         let reg = prox_registrador() in
549                 (match p2.tipagem with
550                 | (Some TInt) -> [sprintf "move v%d, p%d " reg reg] @
                        coloca_param_tabela ps
551                 | _ -> [sprintf "move-object v%d, p%d " reg reg] @
                        coloca_param_tabela ps)
552                 | [] -> []
553
554
555
556 (* Gera funcao*)
557 and gen_func ts id param cmds=
558  let paramToReg = coloca_param_tabela (procura_param ts id) in
559     let entrada = Hashtbl.find ts id in
560         (match entrada with
561             |EntFn entFun ->
562                 (match entFun.tiporetorno with
563                     | Some t ->
564                         (    let comandos = gen_cmds ts cmds id in
565                             let inicio = ".method public static " ^ id ^ "("
                                    ^ (tipos_params param) ^ ")" ^ (tipoRetorno
                                t) in
566                             let numR = prox_registrador() in
567                             let regs = [sprintf ".registers %d \n" numR] in
568                             let pfim = [sprintf "%s" inicio] @ regs @
                                paramToReg @ comandos in
569                             let fim = [sprintf "\n.end method"] in
570                                 (match t with
571                                     | TVoid -> pfim @ [sprintf "return-void"
                                        ] @ fim
572                                     | _ -> pfim @ fim
573                                 )
574                         )
575                     | None -> failwith "Erro gen_func tipo retorno not found
                        ")
576                     | _ -> failwith "Variavel encontrada com o nome da
                        funcao: gen_func")
577
578
579 (* Gera funcoes *)
580 and gen_funcs ts fs=
581     contador_registrador := -1;
582     contador_rotulo := 0;
583     match fs with
584         | f :: fs -> current_func := f.idF;
585         let lista = gen_func ts f.idF f.paramsF f.cmdsF in
586                         lista @ gen_funcs ts fs
587         | [] -> []
588
589
590 (* Gera programa *)
591 and gen_prog ts prog nomeProg=
```

```
592     nomePrograma := nomeProg;
593     let funcs = prog.funcsP in
594     let cmds = prog.cmdsP in
595     let funcoes = gen_funcs ts funcs in contador_registrador := -1;
596                           contador_rotulo := 0;
597                           current_func := "";
598     let comandosMain = gen_cmds ts cmds !current_func in
599     let prologoMain = [sprintf "\n.method public static main([Ljava/lang/
        String;)V "] in
600     let numReg = prox_registrador() in
601     let registMain = [sprintf ".registers %d\n" numReg] in
602     let numReg2 = prox_registrador() in
603     let registMain2 = [sprintf ".registers %d\n" numReg2] in
604     let epilogoMain = [sprintf "\nreturn-void\n.end method"] in
605         if (!contador_registrador - 1 <> 0) then
606         funcoes @ prologoMain @ registMain @ comandosMain @ epilogoMain
607         else
608         funcoes @ prologoMain @ registMain2 @ comandosMain @ epilogoMain
609
610 let rec gerador ts prog nomeProg =
611         gen_prog ts prog nomeProg
```

Listing 4.12: Gerador (Tradutor) do MiniPython

4.9 Makefile

Com todos os passos do nosso compilador construídos, faremos uso de um artifício para facilitar a compilação de todos os arquivos criados. Esse artifício será o *Makefile*. Com uso deste arquivo, basta acessarmos o terminal, na pasta corrente do arquivo *Makefile*, e executarmos o comando "`make`"para compilarmos todas as etapas do nosso compilador. E caso queiramos voltar ao passo inicial, sem os arquivos gerados pela compilação, que serão arquivos com as extensões *.cmi*, *.cmo* e *.output*, basta usarmos o comando "`make clean`".

Abaixo é apresentado o nosso Makefile criado:

```
1  CAMLC=ocamlc
2  CAMLLEX=ocamllex
3  CAMLYACC=ocamlyacc
4
5  # compInter: compSem interpretador.cmo
6  compGer: compSem gerador.cmo
7
8  compSem: compSint semantico.cmo
9
10 compSint: lexIndenta.cmo sintatico.cmo
11
12 gerador.cmo: asa.cmi gerador.ml
13         $(CAMLC) -c gerador.ml
14 # interpretador.cmo: asa.cmi interpretador.ml
15 #       $(CAMLC) -c interpretador.ml
16
17
18 asa.cmi: asa.ml
19         $(CAMLC) -c asa.ml
20
21 sintatico.cmo: sintatico.cmi sintatico.ml
22         $(CAMLC) -c sintatico.ml
23
```

```
sintatico.cmi: sintatico.mli
	$(CAMLC) -c sintatico.mli

sintatico.ml: asa.cmi sintatico.mly
	$(CAMLYACC) -v sintatico.mly

sintatico.mli: asa.cmi sintatico.mly
	$(CAMLYACC) -v sintatico.mly

semantico.cmo: asa.cmi semantico.ml
	$(CAMLC) -c semantico.ml

lexico.cmo: sintatico.cmi lexico.ml
	$(CAMLC) -c lexico.ml

lexico.cmi: sintatico.cmi lexico.ml
	$(CAMLC) -c lexico.ml

lexico.ml: lexico.mll
	$(CAMLLEX) lexico.mll

lexIndenta.cmo: sintatico.cmi lexico.cmi lexIndenta.ml
	$(CAMLC) -c lexIndenta.ml

clean:
	rm *.cmo *.cmi lexico.ml sintatico.ml sintatico.mli *.output
```

Listing 4.13: Arquivo Makefile

Abaixo teremos como deve ser a saída depois de executarmos os comandos apresentados anteriormente:

```
 $ make clean
rm *.cmo *.cmi lexico.ml sintatico.ml sintatico.mli *.output
 $ make
ocamlc -c asa.ml
ocamlyacc -v sintatico.mly
ocamlc -c sintatico.mli
ocamllex lexico.mll
78 states, 1083 transitions, table size 4800 bytes
ocamlc -c lexico.ml
ocamlc -c lexIndenta.ml
ocamlc -c sintatico.ml
ocamlc -c semantico.ml
ocamlc -c gerador.ml
```

4.10 Arquivo carregador.ml

A partir dos analisadores já criados, será necessário criartambém faremos uso de um arquivo fonte com nome "**carregador.ml**", para facilitar o uso do compilador criado em arquivos fontes da linguagem MiniPython. Como o próprio nome do arquivo diz, o carregador será responsável por carregar os módulos responsáveis por cada etapa do nosso compilador. O código fonte do carregador segue abaixo:

```
#load "asa.cmo";;
#load "sintatico.cmo";;
#load "lexico.cmo";;
#load "lexIndenta.cmo";;
#load "semantico.cmo";;
```

```
#load "gerador.cmo"
(* #load "interpretador.cmo";; *)

open Asa;;
open Sintatico;;
open Semantico;;
(* open Interpretador;; *)
open Gerador;;
open Printf;;
open Filename;;

let sintatico lexbuf =
    try
        Sintatico.programa LexIndenta.lexico lexbuf
    with exn ->
    begin
        let tok = Lexing.lexeme lexbuf in
        let pos = lexbuf.Lexing.lex_curr_p in
        let nlin = pos.Lexing.pos_lnum in
        let ncol = pos.Lexing.pos_cnum - pos.Lexing.pos_bol - String.length
            tok in
        let msg1 = sprintf "Erro na linha %d, coluna %d" nlin ncol in
        let msg2 = sprintf "\tA palavra \"%s\" nao era esperada aqui." tok
           in
            print_endline msg1;
            print_endline msg2;
            flush stdout;
            raise exn
   end

let sint_str str =
  let lexbuf = Lexing.from_string str in
  sintatico lexbuf

let sint_arq arq =
   let ic = open_in arq in
   let lexbuf = Lexing.from_channel ic in
   let arv = sintatico lexbuf in
   close_in ic;
   arv

let sem_arq arq =
  let arv = sint_arq arq in
  semantico arv

let emite_prologo oc arq = fprintf oc ".class public L%s; \n.super Ljava/
    lang/Object;
\n.method public constructor <init>()V\n\t.registers 3\n\n\tmove-object v0,
    p0\n\tmove-object v1, v0\n\n\t
invoke-direct {v1}, Ljava/lang/Object;-><init>()V\n\n\treturn-void\n.end
    method\n\n" (chop_extension arq)
let emite_epilogo oc = fprintf oc ""
let emite oc str = fprintf oc "%s\n" str
let rec emite_cod oc cod =
match cod with
c::cs -> emite oc c; emite_cod oc cs
| [] -> emite oc ""

```

```
(* Compila um arquivo .py e gera seu codigo .smali*)
let compila arq =
  let ic = open_in arq in
  let lexbuf = Lexing.from_channel ic in
  let asa = sintatico lexbuf in
  let _ = close_in ic in
  let tabSimb = semantico asa in
  let codigo = gerador tabSimb asa (chop_extension arq) in
  let saida = (chop_extension arq) ^ ".smali" in
  let oc = open_out saida in
  emite_prologo oc arq;
  emite_cod oc codigo;
  emite_epilogo oc;
  close_out oc;
  close_in ic

(* Compila uma string em python e gera seu codigo .smali*)
let compila_str str =
  let lexbuf = Lexing.from_string str in
  let asa = sintatico lexbuf in
  let tabSimb = semantico asa in
  gerador tabSimb asa
```

Listing 4.14: Carregador

Para usá-lo, basta abrir o OCaml como ensinado anteriormente, e usar o seguinte comando:

```
# #use "carregador.ml";;
```

Depois deste comando executado ele deve devolver a seguinte mensagem:

```
# #use "carregador.ml";;
val sintatico : Lexing.lexbuf -> Asa.programa = <fun>
val sint_str : string -> Asa.programa = <fun>
val sint_arq : string -> Asa.programa = <fun>
val sem_arq : string -> (string, Asa.entradaTabela) Hashtbl.t = <fun>
val arvtab_str :
  string -> Asa.programa * (string, Asa.entradaTabela) Hashtbl.t = <fun>
val emite_prologo : out_channel -> string -> unit = <fun>
val emite_epilogo : out_channel -> unit = <fun>
val emite : out_channel -> string -> unit = <fun>
val emite_cod : out_channel -> string list -> unit = <fun>
val compila : string -> unit = <fun>
val compila_str : string -> string -> string list = <fun>
```

Os comandos a frente de "val"são as funções que foram implementadas no arquivo carregador, e poderão ser utilizadas no nosso fonte MiniPython, passando este fonte por cada uma das etapas. Por exemplo, o *compila* será a função responsável por gerar o assembly do nosso fonte MiniPython; enquanto o *sem_ arq*, executará do analisador léxico até o analisador semântico no nosso fonte.

Capítulo 5

Testes

Neste capítulo executaremos nosso compilador em vários fontes na linguagem que foi definida neste projeto. Executaremos o nosso compilador afim de gerar o código assembly (.smali), e executaremos este assembly na Dalvik, assim verificando suas saídas.

5.0.1 Teste 1:

```
num1 = 150
num2 = 100
if num1 == num2:
     print("Os numeros sao iguais")
else:
     print("Os numeros sao diferentes")
```

Listing 5.1: Teste 1

```
.class public Lex1;
.super Ljava/lang/Object;

.method public constructor <init>()V
        .registers 3

        move-object v0, p0
        move-object v1, v0


invoke-direct {v1}, Ljava/lang/Object;-><init>()V

        return-void
.end method


.method public static main([Ljava/lang/String;)V
.registers 8

const/16 v1, 150
move v0, v1
const/16 v3, 100
move v2, v3
if-eq v0, v2, :label_1
sget-object v6, Ljava/lang/System;->out: Ljava/io/PrintStream;
const-string v7, "Os numeros sao diferentes"
invoke-virtual {v6, v7}, Ljava/io/PrintStream;->println(Ljava/lang/String;)V
goto :label_3
```

```
:label_1
sget-object v4, Ljava/lang/System;->out: Ljava/io/PrintStream;
const-string v5, "Os numeros sao iguais"
invoke-virtual {v4, v5}, Ljava/io/PrintStream;->println(Ljava/lang/String;)V
:label_3

return-void
.end method
```

Listing 5.2: Teste 1 - Smali gerado

5.0.2 Teste 2:

```
num1 = 150
num2 = 100
if num1 > num2:
     print("O primeiro numero eh maior que o segundo")
else:
     print("O segundo numero eh maior que o primeiro")
```

Listing 5.3: Teste 2

```
.class public Lex2;
.super Ljava/lang/Object;

.method public constructor <init>()V
        .registers 3

        move-object v0, p0
        move-object v1, v0


invoke-direct {v1}, Ljava/lang/Object;-><init>()V

        return-void
.end method


.method public static main([Ljava/lang/String;)V
.registers 8

const/16 v1, 150
move v0, v1
const/16 v3, 100
move v2, v3
if-gt v0, v2, :label_1
sget-object v6, Ljava/lang/System;->out: Ljava/io/PrintStream;
const-string v7, "O segundo numero eh maior que o primeiro"
invoke-virtual {v6, v7}, Ljava/io/PrintStream;->println(Ljava/lang/String;)V
goto :label_3
:label_1
sget-object v4, Ljava/lang/System;->out: Ljava/io/PrintStream;
const-string v5, "O primeiro numero eh maior que o segundo"
invoke-virtual {v4, v5}, Ljava/io/PrintStream;->println(Ljava/lang/String;)V
:label_3

return-void
.end method
```

Listing 5.4: Teste 2 - Smali gerado

5.0.3 Teste 3:

```
i=1
soma=0
for i in range(5):
    soma = soma + 2
```

Listing 5.5: Teste 3

```
.class public Lex3;
.super Ljava/lang/Object;

.method public constructor <init>()V
        .registers 3

        move-object v0, p0
        move-object v1, v0


invoke-direct {v1}, Ljava/lang/Object;-><init>()V

        return-void
.end method


.method public static main([Ljava/lang/String;)V
.registers 8

const/16 v1, 1
move v0, v1
const/16 v3, 0
move v2, v3
const/16 v5, 0
const/16 v4, 5
const/16 v6, 1
const/16 v0, 0
:label_1
if-lt v0, v4, :label_3
goto :label_2
:label_3
const/16 v7, 2
add-int/2addr v2, v7
move v2, v2
add-int/2addr v0, v6
goto :label_1
:label_2

return-void
.end method
```

Listing 5.6: Teste 3 - Smali gerado

5.0.4 Teste 4:

```
num = 15
while num != 0:
    if num > 10:
        print("O numero eh maior que 10")
    else:
        print("O numero eh menor que 10")
    num = num - (1)
```

Listing 5.7: Teste 4

```
.class public Lex4;
.super Ljava/lang/Object;

.method public constructor <init>()V
        .registers 3

        move-object v0, p0
        move-object v1, v0


invoke-direct {v1}, Ljava/lang/Object;-><init>()V

        return-void
.end method


.method public static main([Ljava/lang/String;)V
.registers 9

const/16 v1, 15
move v0, v1
:label_1
const/16 v2, 0
if-ne v0, v2, :label_2
goto :label_3
:label_2
const/16 v3, 10
if-gt v0, v3, :label_4
sget-object v6, Ljava/lang/System;->out: Ljava/io/PrintStream;
const-string v7, "O numero eh menor que 10"
invoke-virtual {v6, v7}, Ljava/io/PrintStream;->println(Ljava/lang/String;)V
goto :label_6
:label_4
sget-object v4, Ljava/lang/System;->out: Ljava/io/PrintStream;
const-string v5, "O numero eh maior que 10"
invoke-virtual {v4, v5}, Ljava/io/PrintStream;->println(Ljava/lang/String;)V
:label_6
const/16 v8, 1
sub-int/2addr v0, v8
move v0, v0
goto :label_1
:label_3

return-void
.end method
```

Listing 5.8: Teste 4 - Smali gerado

5.0.5 Teste 5:

```
con=0
x=1
for x in range(0, 80):
    num = 30
    if num >= 10:
       if num <= 150:
          con = con + 1
```

Listing 5.9: Teste 5

```
.class public Lex5;
.super Ljava/lang/Object;

.method public constructor <init>()V
        .registers 3

        move-object v0, p0
        move-object v1, v0


invoke-direct {v1}, Ljava/lang/Object;-><init>()V

        return-void
.end method


.method public static main([Ljava/lang/String;)V
.registers 12

const/16 v1, 0
move v0, v1
const/16 v3, 1
move v2, v3
const/16 v5, 0
const/16 v4, 80
const/16 v6, 1
const/16 v2, 0
:label_1
if-lt v2, v4, :label_3
goto :label_2
:label_3
const/16 v8, 30
move v7, v8
const/16 v9, 10
if-ge v7, v9, :label_4
goto :label_5
:label_4
const/16 v10, 150
if-le v7, v10, :label_6
goto :label_7
:label_6
const/16 v11, 1
add-int/2addr v0, v11
move v0, v0
:label_7
:label_5
```

```
add-int/2addr v2, v6
goto :label_1
:label_2

return-void
.end method
```

Listing 5.10: Teste 5 - Smali gerado

5.0.6 Teste 6:

```
def contador(a: int) -> int:
    if a < 10:
        print("o numero eh menor do que 10")
    else:
        print("o numero eh maior do que 10")
    return a
```

Listing 5.11: Teste 6

```
.class public Lex6;
.super Ljava/lang/Object;

.method public constructor <init>()V
        .registers 3

        move-object v0, p0
        move-object v1, v0


invoke-direct {v1}, Ljava/lang/Object;-><init>()V

        return-void
.end method

.method public static contador(I)I
.registers 10

move v0, p0
const/16 v2, 20
move v1, v2
const/16 v4, 10
if-lt v3, v4, :label_1
sget-object v7, Ljava/lang/System;->out: Ljava/io/PrintStream;
const-string v8, "o numero eh maior do que 10"
invoke-virtual {v7, v8}, Ljava/io/PrintStream;->println(Ljava/lang/String;)V
goto :label_3
:label_1
sget-object v5, Ljava/lang/System;->out: Ljava/io/PrintStream;
const-string v6, "o numero eh menor do que 10"
invoke-virtual {v5, v6}, Ljava/io/PrintStream;->println(Ljava/lang/String;)V
:label_3
return v9

.end method

.method public static main([Ljava/lang/String;)V
.registers 1
```

```
39
40
41 return-void
42 .end method
```

Listing 5.12: Teste 6 - Smali gerado

5.0.7 Teste 7:

```
1 def contador(a: int) -> int:
2     while a < 10:
3           print("o numero eh menor do que 10")
4           a=a-(1)
5     return a
6
7 x = contador(5)
```

Listing 5.13: Teste 7

```
1  .class public Lex7;
2  .super Ljava/lang/Object;
3
4  .method public constructor <init>()V
5          .registers 3
6
7          move-object v0, p0
8          move-object v1, v0
9
10
11 invoke-direct {v1}, Ljava/lang/Object;-><init>()V
12
13         return-void
14 .end method
15
16 .method public static contador(I)I
17 .registers 8
18
19 :label_1
20 const/16 v1, 10
21 if-lt v0, v1, :label_2
22 goto :label_3
23 :label_2
24 sget-object v2, Ljava/lang/System;->out: Ljava/io/PrintStream;
25 const-string v3, "o numero eh menor do que 10"
26 invoke-virtual {v2, v3}, Ljava/io/PrintStream;->println(Ljava/lang/String;)V
27 const/16 v6, 1
28 sub-int/2addr v5, v6
29 move v4, v5
30 goto :label_1
31 :label_3
32 return v7
33
34 .end method
35
36 .method public static main([Ljava/lang/String;)V
37 .registers 3
38
39 const/16 v0, 5
```

```
invoke-static {v0}, Lex7;->contador (I)I
move-result v2
move v1, v2

return-void
.end method
```

Listing 5.14: Teste 7 - Smali gerado

5.0.8 Teste 8:

```
def imprime(a:int)->void:
    if a < 10:
        print("o numero eh menor do que 10")
    else:
        print("o numero eh maior do que 10")
```

Listing 5.15: Teste 8

```
.class public Lex8;
.super Ljava/lang/Object;

.method public constructor <init>()V
        .registers 3

        move-object v0, p0
        move-object v1, v0


invoke-direct {v1}, Ljava/lang/Object;-><init>()V

        return-void
.end method

.method public static imprime(I)V
.registers 6

const/16 v1, 10
if-lt v0, v1, :label_1
sget-object v4, Ljava/lang/System;->out: Ljava/io/PrintStream;
const-string v5, "o numero eh maior do que 10"
invoke-virtual {v4, v5}, Ljava/io/PrintStream;->println(Ljava/lang/String;)V
goto :label_3
:label_1
sget-object v2, Ljava/lang/System;->out: Ljava/io/PrintStream;
const-string v3, "o numero eh menor do que 10"
invoke-virtual {v2, v3}, Ljava/io/PrintStream;->println(Ljava/lang/String;)V
:label_3
return-void

.end method

.method public static main([Ljava/lang/String;)V
.registers 1


return-void
.end method
```

Listing 5.16: Teste 8 - Smali gerado

5.0.9 Teste 9:

```
def contador(a: int, b: int)->int:
    soma=0
    i=0
    for i in range(5):
        soma = soma + 2
    return soma
```

Listing 5.17: Teste 9

```
.class public Lex9;
.super Ljava/lang/Object;

.method public constructor <init>()V
        .registers 3

        move-object v0, p0
        move-object v1, v0


invoke-direct {v1}, Ljava/lang/Object;-><init>()V

        return-void
.end method

.method public static contador(II)I
.registers 14

move v0, p0
move v1, p1
const/16 v3, 0
move v2, v3
const/16 v5, 0
move v4, v5
const/16 v7, 0
const/16 v6, 5
const/16 v8, 1
const/16 v9, 0
:label_1
if-lt v9, v6, :label_3
goto :label_2
:label_3
const/16 v12, 2
add-int/2addr v11, v12
move v10, v11
add-int/2addr v9, v8
goto :label_1
:label_2
return v13

.end method

.method public static main([Ljava/lang/String;)V
```

```
44 .registers 1
45
46
47 return-void
48 .end method
```

Listing 5.18: Teste 9 - Smali gerado

Referências Bibliográficas

[1] **LLVM**. Disponível em: <http://pt.wikipedia.org/wiki/Dalvik_virtual_machine>. Acesso em: 07 out. 2014.

[2] **Dalvik virtual machine**.<http://www.dalvikvm.com/>. Acesso em: 07 nov. 2014.

[3] **MiniC**. Disponível em: <https://docs.python.org/3/reference/grammar.html>. Acesso em: 18 out. 2014.

[4] **OCaml**. Disponível em: <http://pt.wikipedia.org/wiki/OCaml>. Acesso em: 18 out. 2014.

[5] **Compilador**. Disponível em: <http://pt.wikipedia.org/wiki/Compilador>. Acesso em: 19 out. 2014.