Integrantes:

- Andres Fernando Roman Arevalo
- Alci René Ramírez Soto

ChocoPy

Nota: Para probar chocopy, utilizar el TryChocoPy que aparece aqui https://chocopy.org/

1. Introducción

- Es estáticamente tipado
- Diseñado para ser implementado como compilador
- Chocopy consiste en variables, funciones, clases y sentencias
- Chocopy no soporta diccionarios, funciones de primera clase(first-class function) y reflectividad introspectiva(reflective introspective)
- Solo un tipo durante toda la vida

2. Tour por ChocoPy

a. Notación:

i. {id} = Nombre de la variable
ii. {type} = Tipo de variable
iii. {literal}= Valor inicial
iv. {expr}= Cualquier valor del tipo especifico
v. {return type}= ->{type}

b. Variables

i. {id}:{type}={literal}
ii. Variables, funciones y clases deben tener distinto nombre

c. Funciones

i. Aparecen en la parte superior de un programa o dentro de funciones o métodos
ii. def{id} ({id}+{type},...,{id}+{type}){return type}
iii. {return type]= ->{type}
iv. Si el return type es vacío se debe retornar None
v. Una función crea un nuevo scope

d. Clases

i. class {id}({id}): {declarations}
ii. Existe una clase por default llamada object

e. Valores

i. Integers(int)=
    1. 32 bits= Hay que tener en cuenta el límite inferior y límite superior (-2147483648 a 2,147,483,647)
    2. Son inmutables
ii. Booleans(bool)= True or False (No admite 0 o 1)
iii. Strings(str)=

1. Se delimita por “ hola “. (No admite ‘hola’)
2. Soporta len, index s[i] y concatenación s1+s2
3. Solo admite entre 32 y 126 del rango ascii

iv. Listas=
1. Son mutables
2. [1,2,3]
3. Soporta las mismas operaciones que la string

v. Objetos= Se debe referencia por ejemplo x=cow()

vi. None= Solo se asigna a un objeto, clase o lista

vii. Lista vacía []= puede ser asignado a un objeto o lista

f. Expresiones

i. Literales e identificadores= Son de tipo str,bool e int

ii. Expresiones de listas=[]

iii. Expresiones aritméticas= int: {expr}+{expr}, {expr}-{expr},{expr}*{expr},{expr}//{expr} ,{expr}%{expr} y -{expr}. No soporta división porque da punto flotante y eso no es válido en Chocopy.

iv. Expresiones lógicas= Soporta not{expr},{expr} and {expr} y {expr} or { expr}

v. Expresiones de relación= int:{expr}<{expr}, {expr}<={expr}, {expr}>{expr}. También soporta {expr}=={expr} y {expr}!={expr} las cuales pueden ser de tipo int,bool y string

vi. Expresiones condicionales= Maneja la estructura {expr1} if {expr0} else { expr2}

vii. Concatenación de expresiones= {expr} + {expr} se puede usar para concatenar 2 string o listas

viii. Expresiones de acceso= {expr}.{id} ejemplo x.y

g. Oraciones

i. Condicionales y loops= Igual que en python.Soporta while y for

ii. Oraciones de asignación=
1. {id}={expr}
2. {expr}.{id}={expr}
3. {expr}[{expr}]={expr}
4. x=y.f=x[0]=1 Asigna el valor de 1 a toda la expresión

iii. Clases predefinidas y funciones=
1. print= Output str,int o bool el resto se aborda
2. input= Retorna una str

3. Estructura léxica

a. Estructura de línea

i. Líneas físicas=Son secuencias que terminan con fin de línea

ii. Líneas lógicas= ???

iii. Comentarios=Comienzan con # y terminan con una línea física

iv. Lineas vacias= Son ignoradas

v. Indentación=???

vi. Identificadores=Letras(Mayusculas o minusculas) de A a la Z, se puede barra al piso (_) y números excepto por el primer carácter
vii. Keywords= No deben ser utilizados como variables. Estas palabras son: False, None, True, and, as, assert, async, await, break, class, continue, def, del, elif, else, except, finally, for, from, global, if, import, in,is,lambda,nonlocal,not,or,or,pass,raise,return,try,while,with,yield,int, str.

Convenciones para el AFD
No admite caracteres fuera del rango solo ASCII del 32 a 126

Tipos de Datos		
A	Letras	A…Z,a…z
B	Números	0…9
C	Símbolo ASCII(32 al 126)	&,%,etc..
F	Espacio	

Tipos de token

- <tipo_de_token,lexema,fila,columna>
- <tipo_de_token,fila,columna>

Tokens	
Convención de nombres	**Token**
Keywords	Se autoidentifica
tk_par_izq	(
tk_dos_puntos	:
tk_par_der	)
tk_asig	=
tk_cadena	“Animal”

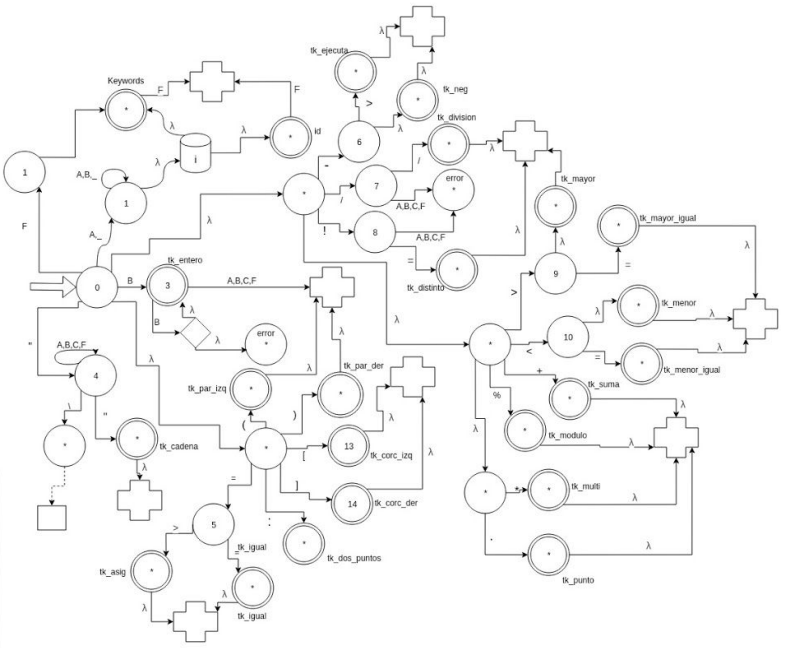
tk_ejecuta	->
tk_escape	\n,\\,\",\t
tk_punto	.
tk_distinto	!=
tk_igual	==
tk_menor	<
tk_mayor	>
tk_mayor_igual	>=
tk_menor_igual	<=
tk_suma	+
tk_neg	-
tk_multi	*
tk_division	//
tk_modulo	%
tk_corch_der	]
tk_corch_izq	[
tk_entero	123 o cualquier número
tk_coma	,
id	Identificador
tk_igual_igual	==

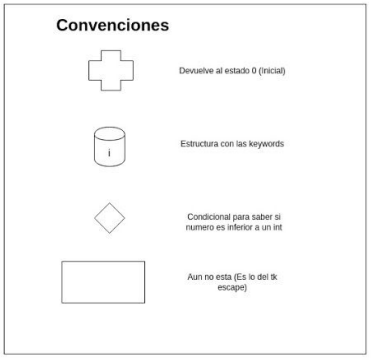
Fake AFD

Se construyó un AFD que no cumple con todas las reglas para serlo (Fake AFD) pero permite visualizar a gran escala la idea que se utilizó para construir el código respectivo y también para facilitar el desarrollo dentro de los integrantes del equipo. Esta un poco desactualizado respecto al codigo porque cuando estuvimos programando nos dimos cuenta de cosas que faltaban pero se nos olvido actualizarlo

Tipos de Datos		
A	Letras	A…Z,a…z
B	Números	0…9
C	Símbolo ASCII(32 al 126)	&,%,etc..
F	Espacio	

Tokens	
Convención de nombres	**Token**
Keywords	Se autoidentifica
tk_par_izq	(
tk_dos_puntos	:
tk_par_der	)
tk_asig	=
tk_cadena	"Animal"
tk_ejecuta	->
tk_escape	\n,\\,\",\t
__init__	__init__
tk_punto	.
tk_distinto	!=
tk_igual	==
tk_menor	<
tk_mayor	>
tk_mayor_igual	>=
tk_menor_igual	<=
tk_suma	+
tk_neg	-
tk_multi	*
tk_division	//
tk_modular	%
tk_cor_der	]
tk_cor_izq	[
tk_entero	123 o cualquier número





https://drive.google.com/file/d/1AmtCJVbl4ghdgw_O7iof6li4slfEy1Id/view?usp=sharing

Nota:

- No se considera la palabra __init__ como reservada debido a que está a comparación las palabras reservadas se puede utilizar como nombre de variable. Es decir el siguiente código donde __init__ es nombre de variable es perfectamente válido en chocopy

  __init__:int=1

print(__init__)

- Falta en el analizador léxico por implementar el apartado de los comentarios multilineas y las listas.

Análisis Sintáctico

1. **Precedencia:** De menor a mayor

   - · if · else ·
   - or
   - and
   - not
   - ==, !=, <, >, <=, >= , is
   - +, − (binary)
   - *, // , %
   - − (unary)
   - ., []

2. **Gramática:**

```
program ::= [var_def | func_def | class_def]* stmt*
class_def ::= class ID ( ID ) : NEWLINE INDENT class_body DEDENT
class_body ::= pass NEWLINE
             | [var_def | func_def]+
func_def ::= def ID ( [typed_var [, typed_var]*]* ) [-> type]* : NEWLINE INDENT func_body DEDENT
func_body ::= [global_decl | nonlocal_decl | var_def | func_def]* stmt+
typed_var ::= ID : type
type ::= ID | IDSTRING | [ type ]
global_decl ::= global ID NEWLINE
nonlocal_decl ::= nonlocal ID NEWLINE
var_def ::= typed_var = literal NEWLINE
stmt ::= simple_stmt NEWLINE
       | if expr : block [elif expr : block ]* [else : block]*
       | while expr : block
       | for ID in expr : block
simple_stmt ::= pass
              | expr
              | return [expr]*
              | [ target = ]+ expr
block ::= NEWLINE INDENT stmt+ DEDENT
literal ::= None
          | True
          | False
          | INTEGER
          | IDSTRING | STRING
expr ::= cexpr
       | not expr
       | expr [and | or] expr
       | expr if expr else expr
cexpr ::= ID
        | literal
        | [ [expr [, expr]*]* ]
        | ( expr )
        | member_expr
        | index_expr
        | member_expr ( [expr [, expr]*]* )
        | ID ( [expr [, expr]*]* )
        | cexpr bin_op cexpr
        | - cexpr
bin_op ::= + | - | * | // | % | == | != | <= | >= | < | > | is
member_expr ::= cexpr . ID
index_expr ::= cexpr [ expr ]
target ::= ID
         | member_expr
         | index_expr
```

Figure 3: Grammar describing the syntax of the ChocoPy language.

3. **Gramática adaptada(LL1)**

Estrategia a usar:

- Utilizamos un multimap para almacenar la gramática donde el lado izquierdo de la gramática es el la key y el lado derecho de la gramática es el valor que será representado por un pair. Donde el primer valor del pair es la posición actual de del procesamiento de la lista o vector y el segundo es la lista o vector en si
- Utilizaremos también un memo que será una stack que se guardan los estados anteriores de la gramática

Negrilla son tokens terminales, los otros son no terminales.Identado puede ser cualquier cosa desde un espacio hasta el infinito :v

No terminales en minúscula y los terminales en mayuscula

inicial -> var_def
inicial -> func_def
inicial -> class_def
inicial -> stmt
class_def -> **class** id_n **(** id_n | id_o **) : newline indentado** class_bod **dedent**
id_o -> **object**
id_n -> **ID**
tipo_dato -> **int**
tipo_dato -> **str**
tipo_dato -> **bool**
tipo_dato -> **list**
class_bod -> **pass newline**
class_bod -> var_def
class_bod -> func_def
func_def -> **def** id_n (typed_var) -> type: **newline indentado** func_bod **dedent**
func_bod -> global_var **newline**
func_bod -> nonlocal_var **newline**
func_bod -> var_def **newline**
func_bod -> func_def **newline**
func_bod -> stmt_simple
func_bod -> multi_bod **newline**
multi_bod -> func_bod **newline** func_bod
typed_var -> id_n: type
typed_var -> multi_var
typed_v -> id_n: type
multi_var -> typed_var **,** multi_var
type -> id_n
type -> [type]
type -> tipo_dato
type -> IDSTRING
global_var -> **global** id_n | tipo_dato
nonlocal_var -> **nonlocal** id_n | tipo_dato
var_def -> typed_v **=** literal **newline**
literal -> None | True | False | integer | IDSTRING | string
stmt -> stmt_simple **newline** | stmt_elif | stmt_while | stmt_for | multi_stmt
multi_stmt -> expr **newline** stmt
stmt_if -> expr **if** expr **else** expr
stmt_simple -> **pass |** expr **| return** expr | target **=** expr
stmt_for -> **for** id_n **in** expr: block
stmt_while -> **while** expr : block
stmt_elif -> **if** expr **:** block elif_mas **else** expr **:** block
elif_mas -> **elif** expr**:**block | **elif** expr**:**block elif_mas | :v
condiciones -> expr_arit | expr_lo

cexpr -> id_n | literal | **[**multi_expr**]** | (expr) | member_expr | index_expr |

member_expr(expr) | id_n(expr) | bin_op | **-** cexpr

multi_expr -> expr,expr

expr -> cexpr | **not** expr | condiciones | stmt_if
expr_arit -> expr bin_op expr
expr_lo -> expr bin_lo expr
bin_lo -> **and** | **or**
bin_op -> + | - | * | // | % | == | != | <= | >= | < | > | **is**
block -> **newline indentado** stmt **dedent**
target -> tipo_dato | id_o| id_n | member_expr | index_expr

member_expr -> cexpr **.** id_n

index_expr -> cexpr[expr]

member_expr -> cexpr member
member -> tk_punto id_n
member -> tk_corch_izq expr tk_corch_der

Primeros:

#	First sets
Inicial	def class id pass return target if while for none true false integer iDSTRING string [(- + * // % == != <= >= < > is
Class_def	class
Class_bod	pass def id
Func_def	def
Func_bod	def global nonlocal pass return target int str bool list id none true false integer iDSTRING string [(- + * // % == != <= >= < > is
Multi_bod	def global nonlocal pass return target int str bool list id none true false integer iDSTRING string [(- + * // % == != <= >= < > is
Typed_var	id
Typed_v	id
Multi_var	id
Type	[id int str bool list
Global_var	global int str bool list
Nonlocal_var	nonlocal int str bool list
Var_def	id
Stmt	pass return target if while for none true false integer iDSTRING string [(- id + * // % == != <= >= < > is
Multi_stmt	none true false integer iDSTRING string [(- id + * // % == != <= >= < > is
Stmt_if	none true false integer iDSTRING string [(- id + * // % == != <= >= < > is
Stmt_simple	pass return target none true false integer iDSTRING string [(- id + * // % == != <= >= < > is
Stmt_for	for
Stmt_while	while
Stmt_elif	if
Elif_mas	elif

Block	newline
Literal	none true false integer iDSTRING string
Expr	none true false integer iDSTRING string [(- id + * // % == != <= >= < > is
Expr_arit	none true false integer iDSTRING string [(- id + * // % == != <= >= < > is
Expr_lo	none true false integer iDSTRING string [(- id + * // % == != <= >= < > is
Multi_expr	none true false integer iDSTRING string [(- id + * // % == != <= >= < > is
Cexpr	none true false integer iDSTRING string [(- id + * // % == != <= >= < > is
Bin_lo	and or
Bin_op	+ - * // % == != <= >= < > is
Member_expr	none true false integer iDSTRING string [(- id + * // % == != <= >= < > is
Index_expr	none true false integer iDSTRING string [(- id + * // % == != <= >= < > is
Target	none true false integer iDSTRING string [(- id int str bool list + * // % == != <= >= < > is

Condiciones	none true false integer iDSTRING string [(- id + * // % == != <= >= < > is
Id_o	object
Id_n	id
Tipo_dato	int str bool list

siguientes:

#	Predict sets
1	id
2	def
3	class
4	pass return target if while for none true false integer iDSTRING string [(- id + * // % == != <= >= < > is
5	class
6	pass
7	id
8	def
9	def
10	global int str bool list
11	nonlocal int str bool list

#	Follow sets
Inicial	┤
Class_def	┤
Class_bod	dedent
Func_def	┤ dedent newline
Func_bod	dedent newline
Multi_bod	newline
Typed_var	) ,
Typed_v	=
Multi_var	) ,
Type	:)] , =
Global_var	newline
Nonlocal_var	newline

Var_def	┤ dedent newline
Stmt	┤ dedent
Multi_stmt	┤ dedent
Stmt_if	newline if else dedent : + - * // % == != <= >= < > is and or ,)]
Stmt_simple	dedent newline
Stmt_for	┤ dedent
Stmt_while	┤ dedent
Stmt_elif	┤ dedent
Elif_mas	else
Block	┤ elif else elif_mas dedent
Literal	newline if else dedent : . [+ - * // % == != <= >= < > is and or ,)]

Expr	newline if else dedent : + - * // % == != <= >= < > is and or ,)]
Expr_arit	newline if else dedent : + - * // % == != <= >= < > is and or ,)]
Expr_lo	newline if else dedent : + - * // % == != <= >= < > is and or ,)]
Multi_expr	]
Cexpr	newline if else dedent : . [+ - * // % == != <= >= < > is and or ,)]
Bin_lo	none true false integer iDSTRING string [(- id + * // % == != <= >= < > is
Bin_op	none true false integer iDSTRING string [(- id + * // % == != <= >= < > is newline if else dedent : . and or ,)]
Member_expr	newline if else dedent : (. [+ - * // % == != <= >= < > is and or ,)]
Index_expr	newline if else dedent : . [+ - * // % == != <= >= < > is and or ,)]

Target	
Condiciones	newline if else dedent : + - * // % == != <= >= < > is and or ,)]
Id_o	
Id_n	( ) : newline in if else dedent] , = . [+ - * // % == != <= >= < > is and or
Tipo_dato	:)] newline , =

Predicciones:

12	id
13	def
14	pass return target none true false integer iDSTRING string [(- id + * // % == != <= >= < > is
15	def global nonlocal pass return target int str bool list id none true false integer iDSTRING string [(- + * // % == != <= >= < > is
16	def global nonlocal pass return target int str bool list id none true false integer iDSTRING string [(- + * // % == != <= >= < > is
17	id
18	id
19	id
20	id
21	id
22	[
23	int str bool list
24	global
25	int str bool list
26	nonlocal
27	int str bool list
28	id
29	pass return target none true false integer iDSTRING string [(- id + * // % == != <= >= < > is
30	if
31	while
32	for
33	none true false integer iDSTRING string [(- id + * // % == != <= >= < > is
34	none true false integer iDSTRING string [(- id + * // % == != <= >= < > is
35	none true false integer iDSTRING string [(- id + * // % == != <= >= < > is
36	pass
37	none true false integer iDSTRING string [(- id + * // % == != <= >= < > is

38	return
39	target
40	for
41	while
42	if
43	elif
44	elif
45	newline
46	none
47	true
48	false
49	integer
50	iDSTRING

51	string
52	none true false integer iDSTRING string [(- id + * // % == != <= >= < > is
53	none true false integer iDSTRING string [(- id + * // % == != <= >= < > is
54	none true false integer iDSTRING string [(- id + * // % == != <= >= < > is
55	none true false integer iDSTRING string [(- id + * // % == != <= >= < > is
56	none true false integer iDSTRING string [(- id + * // % == != <= >= < > is
57	none true false integer iDSTRING string [(- id + * // % == != <= >= < > is
58	none true false integer iDSTRING string [(- id + * // % == != <= >= < > is
59	id
60	none true false integer iDSTRING string
61	[
62	(
63	none true false integer iDSTRING string [(- id + * // % == != <= >= < > is
64	none true false integer iDSTRING string [(- id + * // % == != <= >= < > is

65	none true false integer iDSTRING string [(- id + * // % == != <= >= < > is
66	id
67	+ - * // % == != <= >= < > is
68	-
69	and
70	or
71	+
72	-
73	*
74	//
75	%
76	==
77	!=

78	<=
79	>=
80	<
81	>
82	is
83	none true false integer iDSTRING string [(- id + * // % == != <= >= < > is
84	none true false integer iDSTRING string [(- id + * // % == != <= >= < > is
85	id
86	int str bool list
87	none true false integer iDSTRING string [(- id + * // % == != <= >= < > is
88	none true false integer iDSTRING string [(- id + * // % == != <= >= < > is
89	none true false integer iDSTRING string [(- id + * // % == != <= >= < > is
90	none true false integer iDSTRING string [(- id + * // % == != <= >= < > is

91	object
92	id
93	int
94	str
95	bool
96	list