Implementačná dokumentácia k 1. úlohe do IPP 2024/2025
Meno a priezvisko: Juraj Mesík
Login: xmesikj00

V mojom programe som pre dôkladnú analýzu vstupného kódu použil knižnicu 'lark', pretože mi umožňuje jednoducho definovať LL gramatiku uvedenú v zadaní a taktiež bez väčších problémov odchytiť syntaktické chyby vo vstupnom súbore. Nad rámec zadania som v gramatike definoval okrem základného identifikátoru *<id>* aj identifikátory, ktoré musia výhradne začínať alebo končiť znakom ':', hlavne kvôli odlíšeniu situácii, kedy sa jedná o parameter bloku, meno premennej alebo selektor. Pre toto rozlíšenie som použil regulárne výrazy. Taktiež som v gramatike špecifikoval ignorovanie bielych znakov alebo komentárov, ktoré začínajú a sú korektné ukončené úvodzovkami. Pre konštrukciu abstraktného stromu podľa definovanej gramatiky používam funkciu `Lark()` a neskôr `parse()`. Na základe tohto stromu viem efektívne vytvárať prvky a podprvky XML stromu, ktorý bude výstupom.
Dôležité funkcie:

1. `parseClasses(tree: Tree)`

   Táto funkcia je volaná hneď na začiatku programu a stará sa o to, aby boli správne vytvorené všetky triedy pod koreňovým stromom `program` (**1**). Ako argument je funkcii predaný abstraktný strom typu `Tree` knižnice lark, cez ktorého potomkov funkcia iteruje. Ak narazí na potomka, ktorého typ je tiež strom a jeho identifikátor `.data` je "`class`"(**2**) , znamená to, že môžeme vytvoriť XML podstrom - do premenných `name` a `parent`(**3**) vložíme meno triedy a meno rodiča (čím je koreň `program`) a cez zoznam `attrs` predáme tieto atribúty funkcii `constructXml` (**4**). Ak funkcia narazí na potomka, ktorého identifikátor `.data` je `program`, funkcia je rekurzívne zavolaná, ale teraz nad daným potomkom (**5**) nakoľko ďalšie triedy sú v lark strome zahniezdene (viz. ukážku lark AST).

Jednoduchý príklad vstupu SOL25

```
class MyInt : Integer { }
class Main : Object {
    run [|
        x := Integer from: 1.
    ]
}
```

lark AST

```
program    1
class      2
  MyInt    3
  Integer  3
program    5
  class    2
    Main   3
    Object 3
    method
      selector        run
      block
        blockstat
          x
          expr
            exprbase  Integer
            exprtail
              exprsel
                from:
                exprbase
```

funkcia

```
def parseClasses(tree: Tree):
    for child in tree.children:

        if type(child) == Tree and child.data == "class":  2

            name = str(child.children[0])
            parent = str(child.children[1])  3

            attrs = []
            attrs.append(name)
            attrs.append(parent)

            if name not in defined_classes:
                defined_classes.append(name)
            else:
                sys.stderr.write(f"error 35: Class \'{name}\' was already defined.
                sys.exit(35)

            constructXml(t_root, "class", attrs , tree.children[0]  4

        elif type(child) == Tree and child.data == "program":
            parseClasses(child)  5
```

2. `constructXml(parent: ET, roottype: str, rootattr: list, tree: Tree)`

   Funkcia je prvý krát volaná z `parseClasses()` a funguje na podobnom rekurzívnom princípe. Podľa atribútu `roottype` rozlišuje, aký XML podstrom sa vytvorí, podľa atribútu `parent` je určený rodičovský koreň typu `ElementTree`, zoznam `rootattr` obsahuje všetky potrebné dáta pre vytvorenie daného prvku (meno premennej, hodnota arity, meno selektorov,...) a `tree` je príslušný `Lark` podstrom, cez ktorý funkcia prípadne iteruje. V ukážke je vystrihnutá časť funkcie, ktorá konkrétne vytvára `var` XML podstrom. Premenná `t_var` ukladá podstrom typu ET, ktorého rodič je `parent` určený parametrom funkcie (**1**), prvok `var` potrebuje iba meno premennej, ktoré sa nachádza v zozname atribútov `rootattr` na prvom mieste (**2**). Podľa rodičovského stromu som bol schopný previesť jednoduchú sémantickú kontrolu(**3**), ktorá kontroluje či použitá premenná už bola definovaná, resp. pridáva sa do zoznamu definovaných.V ukážke nie je použitá rekurzia, pretože podstrom typu `var` by nemal mať potomkov vo výsledku, ale v závislosti od `roottype` sú v programe implementované rôzne sémantické kontroly a generované relevantné XML prvky.

```
# -------------VAR-------------

if roottype == "var":

    t_var = ET.SubElement(parent, "var")  1
    t_var.set("name", rootattr[0])  2

    if parent.tag == "assign":
3       defined_variables.append(rootattr[0])

    elif rootattr[0] not in defined_variables:
        sys.stderr.write(f"error 32: Use of undefined variable \'{rootattr[0]}\'. \n")
        sys.exit(32)
```