Haskython: Haskell and Python Conversion Library

David Tan Sang Tran
Univ. of Illinois at Urbana-Champaign
Urbana, Illinois, United States
davidtt2@illinois.edu

***Abstract*—Haskython [1] is a Python library that can convert Python code into Haskell and Haskell code into Python. This Python library will allow users to be freely translate code between Haskell and Python to utilize Haskell advantages, such as generating parsers and compilers. Having this repository as a Python library allows for ease of use since the majority of developers typically have more experience with procedural programming compared to functional programming.**

Keywords—Haskell, Python, Python library, conversion, programming languages, translation, Haskython, file parsing, compiler, parser, functional programming, programming

I. Introduction and Overview

A. Motivation

The motivation behind developing this Haskell-Python conversion library is to create an easy way for procedural developers learn functional programming. Haskell is a useful programming language with the ability to quickly generate parsers and compilers. With this being a Python library, more developers would be able to interact and utilize this library while also learning Haskell along the way.

B. Goals

The goal for this project is to create a fully functioning Python library that can convert code back and forth between Python and Haskell. First, the library should be developed to convert Haskell to Python with all of the basic functionalities built before moving on to the Python to Haskell library. After the basics have been developed, the more complex Haskell functions will be designed and implemented. Once a higher level of translation is established, the goal is to make this a public library with the ability to directly install the library using `pip install Haskython`. With Haskython being a public library, the next goal will be to then publish articles and documents to spread the message and increase its usage. As more users utilize Haskython, these users will find bugs and flaws within the code and then be able to contribute to this project. As such, the final goal would be to have a collaboration between the author and other open source developers to develop a full product that will be usable and maintainable for the long term.

C. Accomplishments

The code has the ability to parse a Haskell/Python file then recognize key features (functions, classes, parameters, return types, conditionals, lists, recursion) to generate a new Python/Haskell file. The main conversion files can take in a function name from command line call for an easier level of use. Afterwards, the Haskell/Python file is read in and then each line is properly translated into its Python/Haskell equivalence.

II. System Implementation

A. Tasks and Capabilities

The main capability for this Python library is the ability to read a Haskell/Python file and identify similar components in the other programming language to allow for automatic conversion. The first tasks of this project is to take in a file and parse it line-by-line to generate matching code in the other programming language (Haskell to Python, Python to Haskell). The tasks for the translator code include identifying function declaration, function parameters, return types, conditionals, lists, and recursion.

B. Components of the Code

There are many components of code that exist in the repository [1]. The two main files are the `haskell-to-python.py` and `python-to-haskell.py` files which are the main files to run to initiate conversion. For each file, the main function will take in a parameter from the command prompt call to choose which file to parse. Once the file name is taken, the file will be read line-by-line and each line will be parsed to determine its Python equivalence. For example, Haskython can determine function declarations, function parameters, return types, conditionals, lists, recursion, and more.

C. Project Status (Completed and Future Work)

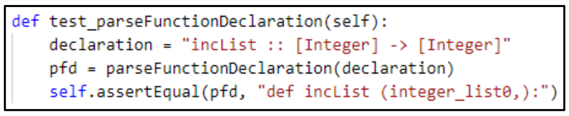
Currently, the basic functionality for converting Haskell code into Python code is working as expected. The `haskell_to_python.py` file can take in a file, parse through it, and convert simple Haskell into Python code (which would be a new file called `filename_haskython.py`). A fair amount of capability is not integrated yet, which will be work in progress and a stretch goal. The code is partially working and can parse basic functions, but a more complex function on multiple lines would be harder for the current code to determine. This is a

In terms of not implemented as planned, the code for converting Python into Haskell code is still in the very early stages and is not usable. The focus has been on converting Haskell into Python to familiarize with all of the features that Haskell has, which is not as heavily documented like Python. Compared to this project's proposal, the Haskell to Python converter is up and mostly working with matches the goal that was presented. As stated in the proposal, the Python to Haskell convert is a stretch goal and will be implemented in the future.

III. Tests

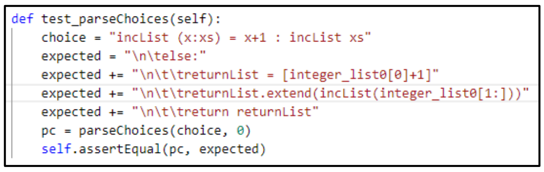
A. Unit Tests

Below are some of the unit tests designed for the Haskell to Python conversion file:



```python
def test_parseFunctionDeclaration(self):
    declaration = "incList :: [Integer] -> [Integer]"
    pfd = parseFunctionDeclaration(declaration)
    self.assertEqual(pfd, "def incList (integer_list0,):")
```

In the above code snippet, this test will retrieve the function declaration from the Haskell file and convert it into a Python function declaration using the `def` keyword and a counter to distinguish various parameters. The results can be seen in the function `assertEqual()`. With this test passing, it is proven that the library will be able to successfully convert a Haskell functional declaration into Python.

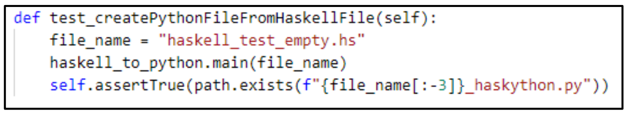


```python
def test_parseChoices(self):
    choice = "incList (x:xs) = x+1 : incList xs"
    expected = "\n\telse:"
    expected += "\n\t\treturnList = [integer_list0[0]+1]"
    expected += "\n\t\treturnList.extend(incList(integer_list0[1:]))"
    expected += "\n\t\treturn returnList"
    pc = parseChoices(choice, 0)
    self.assertEqual(pc, expected)
```

In the above image, this test will retrieve a function body from the Haskell file and convert it into Python. For this test, we will use the `incList (x:xs)` body to create the Python code that will create a list with the first element incremented with recursion to create the same output as the Haskell `incList` function. This test shows that after a function declaration, the Python script can convert a function body from Haskell to Python.

B. *Feature Tests*

One of the feature tests is to ensure that the file is being created as expected.

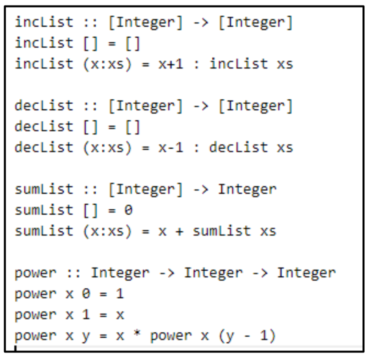


```python
def test_createPythonFileFromHaskellFile(self):
    file_name = "haskell_test_empty.hs"
    haskell_to_python.main(file_name)
    self.assertTrue(path.exists(f"{file_name[:-3]}_haskython.py"))
```

With the above test, it shows that the project is able to create new file from `haskell_test_empty.hs` in the same directory with the correct extension of `_haskython.py`. The output file would be called `haskell_test_empty_haskython.py`. With the file being correctly created, we can ensure that the Haskell file is converted into a Python file in the same directory where the file is added. The next subsection will test the functionality of the created Python file.

C. *Larger Tests*

The larger scale tests will validate the functionality after parsing a Haskell file to a Python file. For this, the program will utilize an existing sample Haskell file with basic functionality called `haskell_test.hs` and convert its functions into a Python file called `haskell_test_haskython.py`. Below is the original `haskell_test.hs` with some functions for incrementing a list, decrementing a list, getting the sum of a list, and getting the power of two integers.



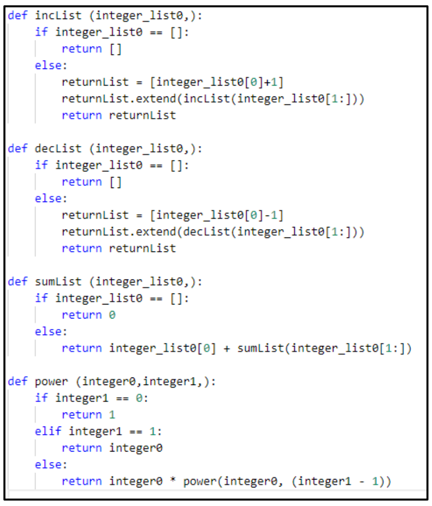
```haskell
incList :: [Integer] -> [Integer]
incList [] = []
incList (x:xs) = x+1 : incList xs

decList :: [Integer] -> [Integer]
decList [] = []
decList (x:xs) = x-1 : decList xs

sumList :: [Integer] -> Integer
sumList [] = 0
sumList (x:xs) = x + sumList xs

power :: Integer -> Integer -> Integer
power x 0 = 1
power x 1 = x
power x y = x * power x (y - 1)
```

After running the `haskell_to_python.py` file in the Haskython directory using `python haskell_to_python.py haskell_test.hs`, Haskython will create a new file called `haskell_test_haskython.py` with the contents shown below. All of the functions in the Haskell file has been properly converted into Python code.



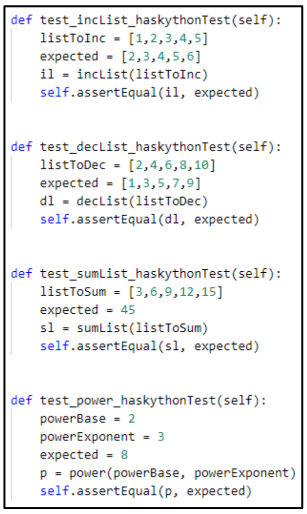
```python
def incList (integer_list0,):
    if integer_list0 == []:
        return []
    else:
        returnList = [integer_list0[0]+1]
        returnList.extend(incList(integer_list0[1:]))
        return returnList

def decList (integer_list0,):
    if integer_list0 == []:
        return []
    else:
        returnList = [integer_list0[0]-1]
        returnList.extend(decList(integer_list0[1:]))
        return returnList

def sumList (integer_list0,):
    if integer_list0 == []:
        return 0
    else:
        return integer_list0[0] + sumList(integer_list0[1:])

def power (integer0,integer1,):
    if integer1 == 0:
        return 1
    elif integer1 == 1:
        return integer0
    else:
        return integer0 * power(integer0, (integer1 - 1))
```

To test this newly created Python file, another test file is used to test all 4 of the functions within `haskell_test_haskython.py` which can be shown below.



```python
def test_incList_haskythonTest(self):
    listToInc = [1,2,3,4,5]
    expected = [2,3,4,5,6]
    il = incList(listToInc)
    self.assertEqual(il, expected)


def test_decList_haskythonTest(self):
    listToDec = [2,4,6,8,10]
    expected = [1,3,5,7,9]
    dl = decList(listToDec)
    self.assertEqual(dl, expected)


def test_sumList_haskythonTest(self):
    listToSum = [3,6,9,12,15]
    expected = 45
    sl = sumList(listToSum)
    self.assertEqual(sl, expected)


def test_power_haskythonTest(self):
    powerBase = 2
    powerExponent = 3
    expected = 8
    p = power(powerBase, powerExponent)
    self.assertEqual(p, expected)
```

With all of these tests passing as expected, it can be determined that the Haskell to Python is working as expected including reading a Haskell file, collecting the file information, generating the new Python file, and validating that the new functions perform successfully.

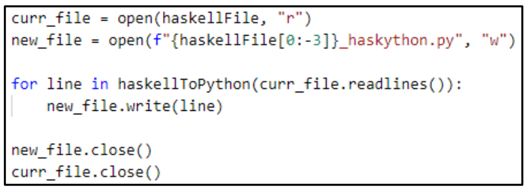
IV. Code Listing

Below are some of the core functions used for the Haskell to Python conversion file:

A. *Central Function 1: Read Haskell File, Write the Conversion to a New Python File*

The initial function for Haskython is within the `main()` of `haskell_to_python.py` which is designed to open a Haskell file which must be within the same directory as the converter and can be declared either by changing the `HASKELL_FILE` variable or by calling `python haskell_to_python.py file.hs`. The code for this function can be found below where `curr_file` represents the Haskell (`.hs`) file and `new_file` represents the

Python (`.py`) file that will contain the translated code. The Python file will also have an extension of `_haskython.py` to notify the user that this file was created by the Haskython library. After opening the Haskell file, the file is read line by line and each translated line is then returned and appended to the Python file.



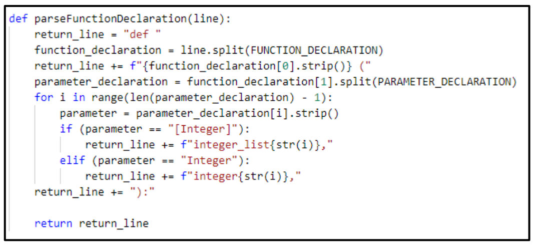
```python
curr_file = open(haskellFile, "r")
new_file = open(f"{haskellFile[0:-3]}_haskython.py", "w")

for line in haskellToPython(curr_file.readlines()):
    new_file.write(line)

new_file.close()
curr_file.close()
```

B. *Central Function 2: Recognize Function Declarations, Parameters, and Types*

After reading in the lines of the file, the next function in `haskell_to_python.py` is to notice function declarations, collect function parameters, and determine return type. Before the `parseFunctionDeclaration()` function, a function declaration is noticed in the Haskell line if the string "`::`" exists in that line. Once a line is determined as a function declaration in Haskell, the below code will run to convert that into a Python function declaration. The code below will append the keyword `def` to the line that will be returned along with the function name that can be found before the declaration string of "`::`". Following the declaration delimiter, the rest of the attributes will be the function parameters and the return type which are separated by the string "`->`". In this line, the final type after the "`->`" will be the return type with the rest being parameters. Each parameter is iterated through with counters to separate multiple same types (integer0, integer1, …).

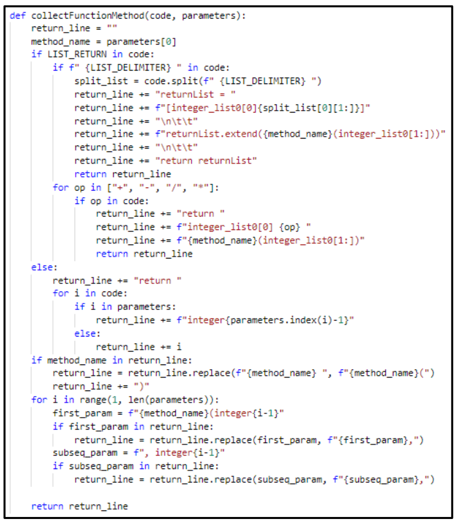


```python
def parseFunctionDeclaration(line):
    return_line = "def "
    function_declaration = line.split(FUNCTION_DECLARATION)
    return_line += f"{function_declaration[0].strip()} ("
    parameter_declaration = function_declaration[1].split(PARAMETER_DECLARATION)
    for i in range(len(parameter_declaration) - 1):
        parameter = parameter_declaration[i].strip()
        if (parameter == "[Integer]"):
            return_line += f"integer_list{str(i)},"
        elif (parameter == "Integer"):
            return_line += f"integer{str(i)},"
    return_line += "):"

    return return_line
```

C. *Central Function 3: Recognize Conditionals, Loops, Lists, and Recursion*

Once the function declaration is parsed, the program will know that the next lines will be related to the declared function until a new function declaration is read into the input stream from the Haskell file. From here, the majority of work is done by recognizing conditionals, loops, lists, and recursion within the Haskell file. Below is the `collectFunctionMethod()` functions used for this which can iterate through the function body to determine what should be returned to the Python file. With how Python is configured, this program will need to have the correct amount of indenting to allow a functioning program. This function uses constants such as a `LIST_RETURN` and `LIST_DELIMITER` variables which are commonly "`xs`" and "`:`". After recognizing these list determinators as well as the return type from the function, the program will create a new list called `returnList` to match the return type of the Haskell function. This function will also recognize recursion if the method name is the next parsed value. If recursion is present, the system will call the function again with the `method_name` parameter along with the needed parameters.



```python
def collectFunctionMethod(code, parameters):
    return_line = ""
    method_name = parameters[0]
    if LIST_RETURN in code:
        if f" {LIST_DELIMITER} " in code:
            split_list = code.split(f" {LIST_DELIMITER} ")
            return_line += "returnList = "
            return_line += f"[integer_list0[0]{split_list[0][1:]}]"
            return_line += "\n\t\t"
            return_line += f"returnList.extend({method_name}(integer_list0[1:]))"
            return_line += "\n\t\t"
            return_line += "return returnList"
            return return_line
        for op in ["+", "-", "/", "*"]:
            if op in code:
                return_line += "return "
                return_line += f"integer_list0[0] {op} "
                return_line += f"{method_name}(integer_list0[1:])"
                return return_line
    else:
        return_line += "return "
        for i in code:
            if i in parameters:
                return_line += f"integer{parameters.index(i)-1}"
            else:
                return_line += i
    if method_name in return_line:
        return_line = return_line.replace(f"{method_name} ", f"{method_name}(")
        return_line += ")"
    for i in range(1, len(parameters)):
        first_param = f"{method_name}(integer{i-1}"
        if first_param in return_line:
            return_line = return_line.replace(first_param, f"{first_param},")
        subseq_param = f", integer{i-1}"
        if subseq_param in return_line:
            return_line = return_line.replace(subseq_param, f"{subseq_param},")

    return return_line
```

V. Conclusion

The Haskython project provides a easy and quick way to convert back and forth between Haskell and Python. The functions cover the majority of basic Haskell functionalities and will be continuously added upon to provide a deeper level of usage. Haskython is created with the thought that procedural programmers can have an opportunity to slowly learn functional programming through the translation process. The future goal for Haskython is to make it publicly available as a pip package and to develop an open-source community between the author and other developers. With that, Haskython can gain experienced knowledge from other engineers to provide a refined product.

References

[1] Tran, David T.S., "Haskython: Haskell and Python Conversion (Python Library)". GitHub, Inc. *CS421: Programming Languages Course*. Summer 2021. https://github.com/davidtstran/haskython

[2] Beckman, Mattox, "CS 421: Programming Languages" Coursera. University of Illinois at Urbana-Champaign. Summer 2021. https://www.coursera.org/learn/cs421/home/welcome

[3] Python Wiki, "Python vs Haskell". MoinMoin Wiki Engine. Python. https://wiki.python.org/moin/PythonVsHaskell

[4] D. Can, V. R. Martinez, P. Papadopoulos and S. S. Narayanan, "Pykaldi: A Python Wrapper for Kaldi," *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018, pp. 5889-5893, doi: 10.1109/ICASSP.2018.8462463. https://ieeexplore.ieee.org/document/8462463

[5] S. Behnel, R. Bradshaw, D. S. Seljebotn, G. Ewing, W. Stein, G. Gellner, et al. "Welcome to Cython's Documentation". Cython 3.0.0a9 Documentation. https://cython.readthedocs.io/en/latest/