CS3840: Software Language Engineering

Part 2 Write-Up

100927938

Name: PLIP - A Polite Language for 2D Image Processing

Domain: 2D Image Processing

Introduction

Between the three choices, the domain of 2D image processing was chosen. The reason for this is as follows; firstly, regarding the domain of music, this was a topic that I am not particularly passionate for, which is why it was a not a strong option. The domain of 3D modelling was quite interesting and 3D modelling struck me as more open-ended which was intriguing. However, 2D image processing seemed as though it could have more impactful commands with obvious effects on the image, which is why it ended up as my final choice – the usefulness of my language feels stronger in this domain.

1. Informal Language Specification Description

The informal language specification provides a description of key signatures that have been implemented in the language. There are three main categories of signatures present in this document. Firstly, arithmetic specifications are described. This is followed by comparison specifications. This feeds into the ‘if’ ‘else’ and ‘while’ specifications (as these specifications rely on comparative ability). Finally, numerous image manipulation specifications are described.

2. Informal Syntax Signatures Description

The informal syntax signatures are provided in a simple text file. They contain every (main) signature present in the PLIP language; however, they are described informally meaning that:

1. Repeated signatures are excluded from this file.
2. Each signature has its functional purpose explicitly stated

Additionally, each signature has its parameter types explicitly stated.

Certain repeated signatures may have a functional difference, in which case these special cases are included in the informal syntax (for example: an alternate sequence, singular 'if', etc.)

3. eSOS Rules Description

This is a major component of PLIP. The eSOS rules determine the low-level functionality of PLIP. Each previously described document extensively goes over functionality, but this file is a non-abridged version as it highlights support signatured to be referenced by main signatures. The rules present in this file reference the value user plugin, where image manipulation takes place using JavaFX.

This file also includes !try statements which have been commented out. This was submitted during Part 1 and can still be uncommented and ran.

A UNIX script "runner.sh" is include in the folder "/plip/support_scripts_unix" and this allows the eSOS rules to be run with the value user plugin.

4. External Syntax Parser (Translator) Description

The purpose of the external parser is to translate PLIP programs written by the user into abstract terms that can then be internally interpreted using the eSOS rules. There are numerous different signatures defined using promotion operators to create the external syntax of PLIP.

Some interesting examples of external syntax defined by plip_exToInt include the following:

X 'shallHenceforthBe' __int32	Assigns a variable X to an integer.
pleaseShow(__string)	Loads an image when given a path
kindlyInvert(pleaseShow)	Inverts a loaded image.
pritheeTilt (pleaseShow, __int32)	Rotates a loaded image by a specified degree.

Note: Attribute Interpreter and ExToInt share the same external syntax – thus these examples apply to both files.

'plip_exToInt' can be run by using its runner "exToESOS_runner.sh". This runner does the following:

- It calls the support script 'parse.sh' with specified PLIP program and '+showAll'. This will compile ART and use 'plip_exToInt.art' to create a file 'terms.txt' which contains generated eSOS statement.
- This file will then be concatenated as such '!try' + 'terms.txt' + '__map'. The concatenation will be added to a copy of 'plip_eSOS.art' called 'temp.art'.
- 'temp.art' is then ran using the support script 'runner.sh'. As it contains the generated '!try' statement, the eSOS rules of PLIP will be utilised to fulfil the externally written request.
- Finally, upon termination of the program all excess files are removed using the support script 'clean.sh'.

5. Attribute Interpreter Description

The purpose of the Attribute Interpreter is to directly interpret PLIP programs and use the value user plugin backend. There are numerous different grammar rules written to define the external syntax of PLIP.

Some interesting examples of external syntax defined by plip_attribute include the following:

prayFlip(pleaseShow, __string)	Flips a loaded image as specified (diagonally, horizontally, vertically)
graciouslyEnhance(pleaseShow)	Increases the saturation and contrast of a loaded image
benevolentlyResize(__string, __int32, __int32)	Loads an image in a resized, specified width and height.

Note: Attribute Interpreter and ExToInt share the same external syntax – thus these examples apply to both files.

'plip_attribute' can be run using its runner 'attribute_runner.sh'. This runner does the following:

- Compiles ValueUserPlugin with ART.
- Calls support script 'parseFX.sh' with specified PLIP program and '+showAll'.
- This will compile 'plip_attribute.art' with ART and will call JavaFX. Thus, running the PLIP program.

6. Example Domain Specific Programs

There are four example PLIP programs included (in plip/plip_scripts), each testing different components of the language.

(Note: there is a guide.txt provided to help run these programs on a Unix system.)

1. ps1.str

This is a basic test showing off arithmetic in PLIP. Numbers are assigned to values using multiplication, division, subtraction, or simply no arithmetic. This is followed by showing that these values can be dereferenced and changed (as b is modified to be 5. Finally, BODMAS is demonstrated as two very similar arithmetic expressions are assigned to a new variable, however the placement of brackets mean that their final value differs.

2. ps2.str

This test is to demo the use of blocks and comparison expressions. The while loops shows that it can run multiple statements. The if loop shows that an else block is optional, and in both cases, numbers are manipulated in the loop such that all comparators (greater than, equal, not equal, less than) are run at some stage.

3. ps3.str

This test simply highlights and runs all image manipulation commands. Something of note is that they can be ran one after the other – thus the order they are called can be changed.

4. ps4.str

This puts everything from the previous tests together – numbers are manipulated in a while loop and comparators are used in 'if' statements to manipulate images in various ways. Outside (after) the loop, commands continue to work.

Implementation Aspects: Achievements and Reflection

To sum up, these are some of the main differentiating aspects of my current implementation of PLIP:

Full arithmetic has been implemented. The language features addition, subtraction, multiplication, and division. It also accounts for brackets. All of this allows for full use of BODMAS, meaning that complex equations can be calculated using the language.

The specifications for my language use "polite" keywords and each function/script works as expected across both the translator and interpreter, featuring numerous different kinds of image manipulations. This "polite" theme makes the language more engaging to use and the functions are designed such that they allow for numerous different manipulations of an image.

The 'if' and 'while' statements of PLIP work such that they both allow for multiple expressions to run in order within each block. Additionally, for the 'if' statements, an 'else' block is entirely optional and does not have to be included.

There are also multiple comparison expressions that have been added for use in statements, such as comparing equality, non-equality, greater-than and less-than, etc.

To reflect on some of the things I'm particularly proud of; I'm very happy with the functions I've implemented as I feel they are useful in this domain. For example, my 'prayFlip' command allows words in images to be read in a mirror or for directions of an arrow to be changed. – it is a feature that is not a native image function in most basic/built-in image manipulation programs. Or another example is my 'graciouslyEnhance' function which automatically adds more flair and pop to an image without a user having to understand how saturation or contrast works. And of course, I'm very entertained by my polite theme – I wanted the language to have an absurdist air of dignity to it, which is why the keywords chosen are deliberately over the top. I purposefully only did this in the front-end and used normal expressions in the backend, as I also wanted to make the language features to be simple to understand for those improving PLIP in the future.