Type Inference

This README file describe how to reproduce our tool to infer type information.

There are three steps in total:

step 1 is collecting the differential results;

step 2 is to parse the differential results to get the type indicator of basic types(bool, string and number) and array type;

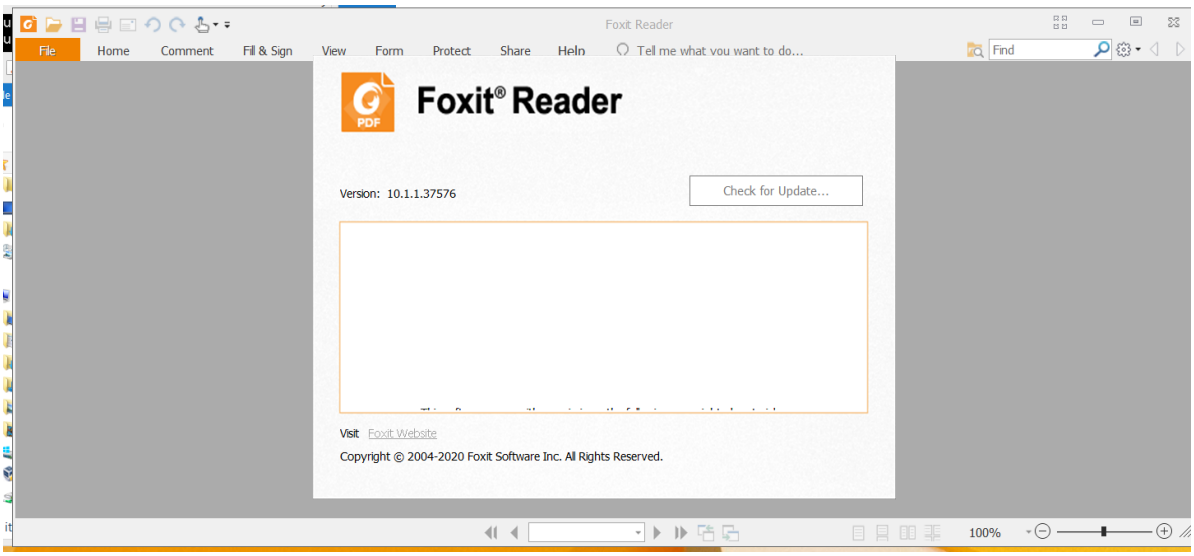
step 3 is to generate the type information of every binding calls.

The following will talk about reproduction in detail.

`Note:`

`1. for easier reproduction, the provided type inference's code is constructed based on Foxit Reader version 10.1.1.37576, please check the version before reproduction.`

`2. in our experience, the type information between different versions of Foxit Reader are usually the same, so we only need to infer type information through one version and use this infered type information to fuzz different versions of Foxit Reader(so does Adobe Reader)`



1. diff_collect

This step will collect the differential results. We use pin.exe to instrument the target PDF Reader to collect execution trace.

folder structure

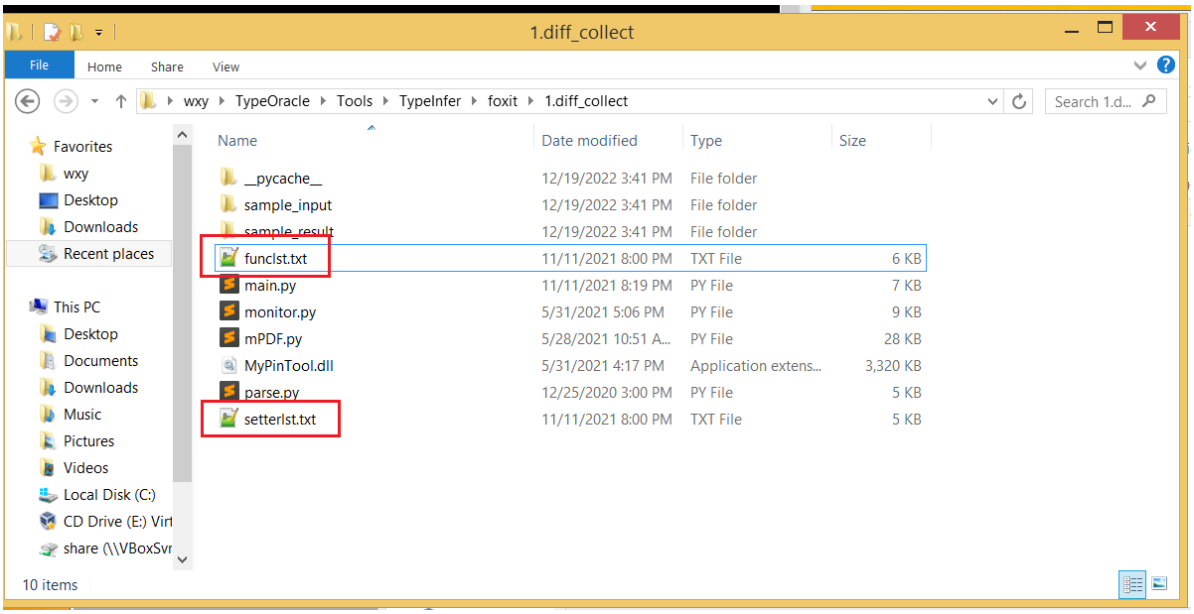
```
- sample_input
  - funclst.txt (list of all method names)
  - setterlst.txt (list of all accessors)

- sample_result (the folder store the sample result of this step)
  - save

- main.py

- monitor.py (to minitor the PDF Reader)

- mPDF.py (to embed javascript code to PDF)

- MyPinTool.dll (the instrumentation file)

- parse.py (parse the results recorded by pin.exe)
```

how to reproduce

1. make sure the Page Heap is turned off (execute following command and click yes, for more information about Page Heap, please refer to C:\Users\wxy\TypeOracle\Other\README.pdf)

```
"C:\Program Files (x86)\Windows Kits\8.1\Debuggers\x86\gflags.exe" /p /disable
"C:\Program Files (x86)\Adobe\Acrobat Reader DC\Reader\AcroRd32.exe"
```

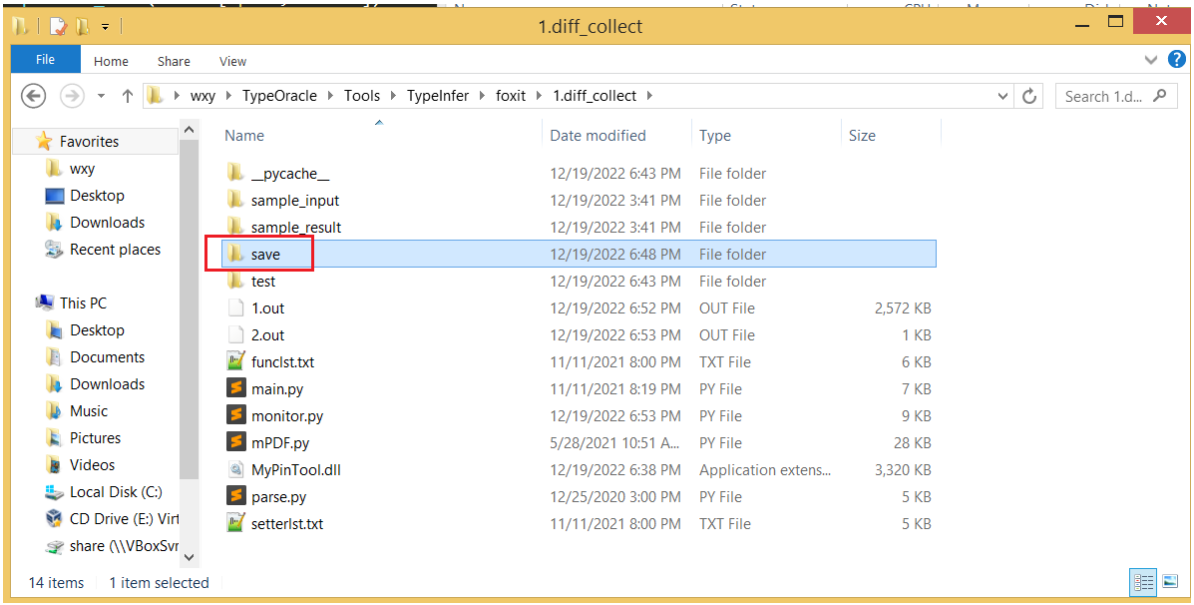
2. copy `funclst.txt` and `setterlst.txt` from `sample_input` folder to current folder



3. execute `main.py` at the current folder to start collect the differential results of every binding call (this step will take more than ten hours)

```
python main.py
```

4. the result will be stored at `save` folder



2. parse_diffresult

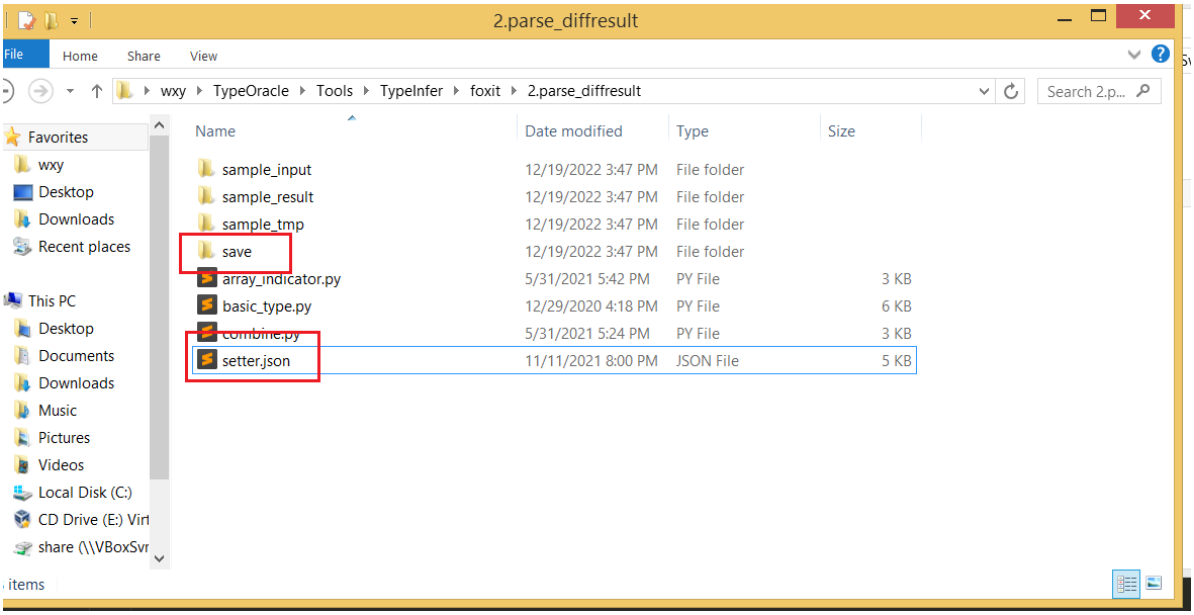
This step is to parse the differential results collected in step 1, and get the type indicator of bool, string , number and array type.

folder structure

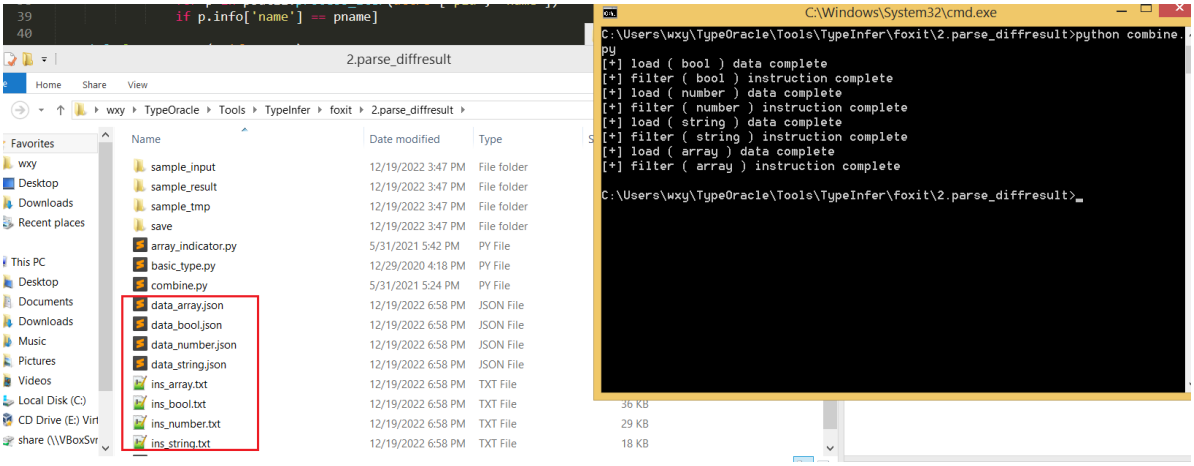
```
- sample_input
  - save (differential results collected in step 1)
  - setter.json (list of all accessors and its type)

- sample_result (the folder store the sample result of this step)
  - basic.txt
  - array.txt

- sample_tmp (store sample intermediate results)

- array_indicator.py (get the array indicator)

- basic_type.py (get basic types' indicator)

- combine.py (combine step 1's result)
```

how to reproduce

1. copy `save` folder in step 1 to this folder (you can also copy `save` from `sample_input` folder) , and copy setter.json in `sample_input` folder to this folder



2. execute combine.py, this will generate some intermediate result files(data_x.json ins_x.json)



3. execute basic.py to get basic types' type indicator, redirect the results to basic.txt

```
python basic_type.py > basic.txt
```

4. execute array_indicator.py，redirect the results to array.txt

```
python array_indicator.py > array.txt
```

3.arg_probe

This step uses the type indicator extracted previously to obtain type information.

folder structure

```
- sample_result  (the folder store the sample result of this step)

- sample_tmp (the folder store sample intermediate results)

- 1.json-9.json

- funclst.txt (list of all methods)

- init.pdf (PDF file to init Adobe Reader)

- jstemplate.js

- method.py

- monitor.py (monitor the execution of PDF Reader)

- mPDF.py (generate PDF file)

- parse.py

- pattern.py (generate javascript code for one binding call)

- record.py (script executed in windbg)

- reprobe.py

- runtime.py

- s1.py (combine the results)

- setterlst.txt (list of all accessors)
```

how to reproduce

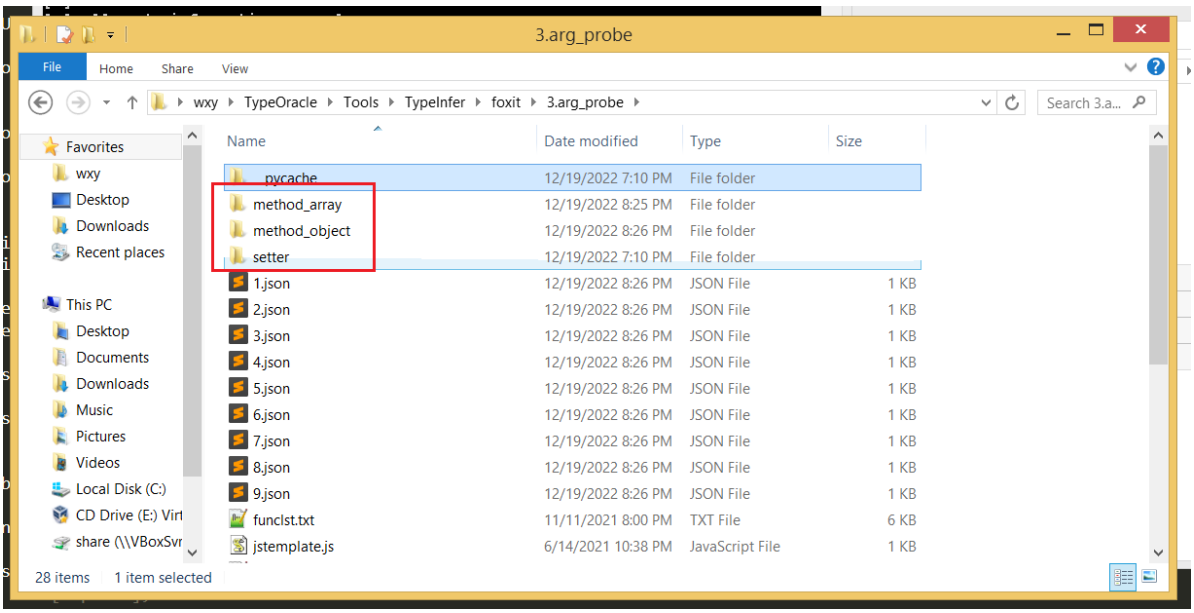
1. fill the type indicator infered before in `record.py`

record.py

```python
import os
import json

import pykd

CUR_PATH = r'C:\Users\wxy\TypeOracle\Tools\TypeInfer\foxit\3.arg_probe'

nothing_start = 0x9d6580  # + 0x400000
# start of app.platform (wrapper)

toolbar_start = 0xa0ab70
# start of app.platform (real)
toolbar_ret = 0x9d682c
# end of app.platform (wrapper)

dispatcher_start = 0x2410169
dispatcher_end = 0x241016b

setter_start = 0x24103d0
setter_end = 0x24103d2

jsonkey1 = 0x240a57b
# push dword ptr [eax]
jsonkey2 = 0x240c8fc
# push dword ptr [eax]

rbool = 0x246d657
# test al, FEh
rnum = 0x2430f29
# mov ecx, [esi]
rstr = 0x2412b2a
# mov [ebp-14h], ecx
rarr = 0x2635aea
# mov ecx, [ecx+bh]
```

2. execute `method.py` (it will take about five hours), this will generate three folders: `method_array`, `method_object`, `setter`

```
python method.py
```



3. execute s1.py to combine all results, the type information is store in `data` folder

```
python s1.py
```

4. some exceptions

```
Some APIs, such as app.log will not produce any results, because there are bugs
in the implementation of these apis of Foxit Reader. If you use windbg to attach
to the program and run it, you will see null pointer dereference (if you do not
use windbg, it will be handled by the built-in exception handling mechanism of
Foxit Reader, so that the program will not crash). In this way, these apis will
interrupt the running of windbg, causing the tool to fail

There are 8 apis that have bugs in the implementation:
*methods:
this.addPageOpenJSMessage
this.app.addFocusedDoc
this.app.log
this.app.removeEncryption
this.convertTocPDF
this.saveAsNewcPDF
*accessors:
this.app.fs.isFullScreen
this.app.fullscreen

We also list some other exceptions:
1. this.print
When infering this api's type information, the constructed test case will
generate a pop-up window that can not be closed by alt+f4, which is how the tool
closes pop-up windows. Therefore, you need to manually close the pop-up window
when the program is running to get the results

2. this.addAnnot
Some keys in this.addAnnot will cause the Foxit Reader to stuck. After the Foxit
Reader is forced to exit, the tmpthis_addAnnot.json will be obtained, which is
assumed as the intermediate result
At this time, you need to run reprobe.py, start infering from the intermediate
results, and finally get this_addAnnot.json as the result
```

illustration of type information

api： api name

apitype: 0-method 1-accessor

root： root object's index

info:

23x generic object

22x array

req_type: required argument

opt_type: optional argument

5x value: multiple types of parameters are allowed

0 bool

1 number

3 string

2 built-in object

5 any type