Final Assignment CS 3520

Matthew Duffin

Additions

- Covariance and contravariance:
    - Feature with 2 stars
- Private declaration
    - Feature 4 with 2 stars
- Java Style Null
    - Feature 8 with 2 stars
- If0I
    - Feature 3 with 2 stars
- **STAR TOTAL:** 8 stars

Testing

In each section I will specify test locations where I implemented tests for each feature. These are primarily in typed_parse and inherit_parse. However, I do have tests in all the files that test more specific methods for each feature that I have not specifically pointed out.

Covariance and Contravariance

My general approach for this was to use the type checker to climb up the tree to determine if result types were covariant, or if argument types were contravariant. To do this, I needed to create a helper method that utilized the is_subtype method but would call it on an entire tree of inheritance. The method is called is_any_subtype and can be found on line 388 of typed_class. This method is then called in check_override method on line 368 of the same file. Basically it will call is_any_subtype on both the arg_type and result_type, but with there parameters flipped in the two calls. This flipping gives us contravariance on the arg_type and covariance on the result_type. Important to note that in my interpretation of this problem, that anything that inherits from object is valid for both covariance and contravariance. I felt this was appropriate because it didn't make sense to me to only allow some inheritance but not inheritance from objects. To see tests that are rejected by the type checker in this case, see lines 117-135 for contravariance, and lines 243-282 in typed_parse for covariance. To see how my type checker now accepts contravariance and covariance, see lines 136-241 for contravariance and lines 283-385 for covariance in the same file.

Private Declaration

To implement private I felt it important to also implement public. The reason I felt the need to also do public, is because I thought this would make it very clear which context we are in when type checking and interpreting expressions. I felt that the easiest way to do this is to implement similar syntax to how we did types, which resulted in something like this:

Method get_dog :: public (arg :: Int) :: Int :

Method get_dog :: private (arg :: Int) :: Int :

This then allowed me to parse out the type in typed parse into a variable called method_ty (lines 32-37). The next step was to implement the various types needed in both values, expI, types

and exp. A private and public was needed for each of these various types because this allowed us to traverse the entire code process from typechecker to interp. In interp however, not much is done, a simple publicV or privateV is returned indicating the type. The real interesting part is in type checker. There it checks the type for overrides in typed_class line 368-386. Basically it just determines that a public cant override a private and vice versa but that something with the same method type can override. Some of this type checking is also done in type_check_send on lines 340-354. However, this determines if a .this or .super keyword are calling the private method. By keeping track of the context, we can confirm if the method is being called in the proper places. My private tests can be found on line 659-843 of typed_parse and showcase a variety of test cases.

Java Style Null

To implement null I need to add the appropriate null types to value, exp, expI, and type. I then in all of the parsers needed to add the ability to parse null out as an argument, argument type and return type. This can be seen in the parser files. The main thing with null was just returning the proper null type when doing interping, but with type checking, it was important to make sure it could be passed as an argument, a field, method result and that it cannot be used as a number. This was primarily done by changing is_subtype to always treat null as a subtype of object in typed_class line 70-83. However, if null is used as an object to access a method or field, this will pass the type checker, but fail in interp. I felt this was the best place to throw an error because I think that since null is treated like an object, it should pass any type checking where an object is needed, but fail when trying to access methods and fields from it for an expression when interpreting. I have null tests that showcase all the behavior listed out in the requirements that can be found in inherit_parse lines 242-338 and in typed_parse lines 845-1067.

If0I

To implement if0I, it was required that I implement if0E. Then in interp I handled the logic of an if statement the same we have in all the previous other interpretations of if0 that we have done in this class. This can be found in class.rhm lines 107-114. Again the most interesting parts of the implementation are found in the typechecker. In typed_class lines 177-202, I handle the various cases of if0I. It first checks that the condition is only an integer. It then determines if the return types of the true and false cases are also the same types. For int and null, it determines that they are both ints or nulls. If not it throws a type error. In objT, things get a bit more complicated. It first determines if both the true and false conditions return an objT, if not, it throws an error. If they are, it then calls a helper method I created called is_any_subclass. This method can be found on lines 254-266. Basically, it will call is_subclass all the way up the tree until they are in fact subclasses, or we get to object. When we get to object, I elected to return #true still, because everything inherits from object. What you might notice then is that in this case, false can never be returned. This is because everything will eventually inherit from object and I felt that you should be able to return things that are both objects rather than not permitting them when you reach that level of inheritance. However, this could easily be changed to then return #false when we get to object should the need arise. My tests for if0I are very

comprehensive and go into as many test cases as I could think of. They can be found in typed_parse lines 386-843. I also have tests in inherit_parse from lines 157-229.