CS421 Assignment 4 Tiger Abstract Syntax Generation Chen Gu CG736

Rule Modification

Handling grouping type declaration

In tiger, one can define mutually-recursive types using a consecutive sequence of type declarations. In order to support this feature, the rule of assignment3 parser should be modified. I rewrite the rule as follow (the action is omitted for simplicity)

```
tydecs : tydecs_aux ()

tydecs_aux: TYPE ID EQ ty.            ()
          | TYPE ID EQ ty tydecs_aux  ()
```

Handling grouping function declaration

Similarly, Function declarations may be mutually recursive. I rewrite the rule to support recursive declarations. There is simpler way to write to this rule, but that will result in shift/reduce conflict. In order to solve that conflict, I add redundancy in rules to achieve that.

```
fundecs: fundecs_aux ()

fundecs_aux :  FUNCTION ID LPAREN tyfields RPAREN EQ exp                  ()
        | FUNCTION ID LPAREN tyfields RPAREN COLON ID EQ exp              ()
        | FUNCTION ID LPAREN tyfields RPAREN EQ exp  fundecs_aux          ()
        | FUNCTION ID LPAREN tyfields RPAREN COLON ID EQ exp fundecs_aux ()
```

Difficulties Encounter during Implementation

Handling lvalue

Originally, I use a left-recursive rule to define lvalue, which will result in shift/reduce conflict regarding `LBRACK`. I rewrote the rule to solve this problem. By doing so, I make parsing lvalue more difficult. Because left recursive rule is the most natural way to parse lvalue.

```
lvalue_exp : lvalue      ()

lvalue : ID lvalue_aux   ()

lvalue_aux :                                        ()
           | DOT ID lvalue_aux                      ()
           | LBRACK exp RBRACK lvalue_aux           ()
```

I cannot call the constructor while reducing lvalue, because I need to know the preceding var in this case. To parse lvalue correctly, I define a new datatype called `lvalue_var`

```
datatype lvalue_var
    = Field of A.symbol * A.pos
    | Subscript of A.exp * A.pos
```

And I treat `lvalue_aux` as a list of `lvalue_var`. I keep concatenating the list while reducing. I call an auxiliary function `lvalue_gen` when reducing the whole expression into a lvalue.

```
fun lvalue_gen(v, []) = v
  | lvalue_gen(v, Field(s, p)::r) = lvalue_gen(A.FieldVar (v, s, p), r)
  | lvalue_gen(v, Subscript(e, p)::r) = lvalue_gen(A.SubscriptVar (v, e, p
), r)
```

Here is my choice of pos for different lvalue:
Single lvalue: IDleft
Field lvalue: DOTleft

Subscript lvalue: LBRACKleft

Transforming from tfield to formals.

According to appendix, both record creation and function declaration use `tyfields`. However, in `Absyn.sml` we can see that the field for record is called `tfield` and parameters for function is called `formals`. There is one slight difference between two types. So I need to write a recursive function to transforming `tfield` to `formals`.

```
fun formalize([]) = []
  | formalize(({name=n,typ=t,pos=p}:A.tfield)::r) = ({var={name=n, escape=
ref true}, typ=t, pos=p}:A.formals) :: formalize(r)
```

Test

I use the 6 provided test cases to test my program, I add a function in `parse.sml` to output the result to a file. Then I use `diff` command to compare my results with standard results.

```
fun writeFile (filename, content) =
    let val fd = TextIO.openOut filename
        val _ = TextIO.output (fd, content ^ "\n") handle e => (TextIO.clo
seOut fd; raise e)
        val _ = TextIO.closeOut fd
    in () end
```

The result of `diff` is nothing, which means all result are the same.

```
~/git/cs421/as/as4/tests $ diff test01.tig.std test01.tig.my
~/git/cs421/as/as4/tests $ diff test02.tig.std test02.tig.my
~/git/cs421/as/as4/tests $ diff test03.tig.std test03.tig.my
~/git/cs421/as/as4/tests $ diff test04.tig.std test04.tig.my
~/git/cs421/as/as4/tests $ diff test05.tig.std test05.tig.my
~/git/cs421/as/as4/tests $ diff test06.tig.std test06.tig.my
```