Compilers Project 3 'Type Checking' Report

2013-11431 Hyunjin Jeong

The third term project was to implement type checking base on the parser. In this report, I will write how I implemented type checking in parser.cpp, and how I implemented TypeCheck() and GetType() in ast.cpp, and what scheme I chose for integer constants range check.

0. Integer constant range check scheme

I chose 'SIMPLE' scheme. I checked a type of simpleexpr. If the first term of the simpleexpr is CASTConstant and 'integer', sign-folding is implemented. Otherwise, new node 'CAstUnaryOp' is made with sign.

1. In parser.cpp

1.1 module/subroutine identifier match (in CParser::module/subroutineDecl)

When the parser consumes first module/subroutine identifier, it saves in a token. When the parser consumes second module/subroutine identifier, the parser compares names between first identifier and second identifier. If they are not same, then the parser sets error.

1.2 duplicate subroutine/variable declaration (in CParser::subroutineDecl/varDeclParam/varDecl)

When new ident is consumed in variable declaration, the parser checks if symbol table has a same ident already. If current scope is module, the parser checks global symbol table. If current scope is procedure/module, the parser checks local symbol table. If symbol table has same ident with input, the parser sets error.

1.3 scalar return type check for functions (in CParser::type)

The type() function takes parameter 'bool isFunction'.. This Boolean value is true when the

function calls type(). If isFunction is true and type is not scalar, then the parser sets error.

1.4 use before declaration (in CParser::subroutineCall/assignment/factor)

When the parser wants to use symbols, the parser checks symbol tables. If both local symbol table and global symbol table don't have symbol, then the parser sets error.

1.5 subroutine calls require valid symbol (in CParser::subroutineCall/factor)

When subroutine calls require symbol, the parser checks symbol table. If symbol tables have symbol but the type is not 'CSymproc', then the parser sets error.

1.6 array dimensions provided for array declarations (in CParser::type)

When arrays are declared, the type() function consumes 'tNumber'. If consumed token is not 'tNumber', then the parser sets error.

However, arrays in parameter are allowed open arrays. The type() function takes parameter 'bool isParam'. So if 'isParam' is true, then open arrays can be handled.

1.7 array dimension constants > 0 (in CParser::type)

The type() function consumes 'tNumber', which means it only consumes positive constants. (negative sign is added in simpleexpr). Therefore, only positive array dimension constants are allowed.

1.8 integer range <= 2^31 (in CParser::factor)

In pasrer.cpp, Integer constants are checked range, 0 <= integer <= 2^31.

2. In ast.cpp

2.1 CAstScope

2.1.1 TypeCheck

I copied the reference implementation in pdf. First the parser checks its statements, and it checks children. If all type checking finishes well, then TypeCheck() returns true.

2.2 CAstStatAssign

2.2.1 TypeCheck

First the parser checks its left-hand side(LHS) and right-hand side(RHS). Then the parser checks LHS's GetType() is array. If then, the parser sets error because array assignments are not allowed. Then the parser checks LHS and RHS have same type using Match. If not, the parser sets error.

2.3 CAstStatReturn

2.3.1 TypeCheck

I copied the reference implementation in pdf. There are two cases in this node: function and procedure/module. If there exists an expression after return, then the parser sets error in procedure/module case.

In function, the parser sets error if expression doesn't exist. Then the parser checks expression. Then the parser checks return type is same with function type using match. If they are not same, the parse sets error.

2.4 CAstStatIf

2.4.1 TypeCheck

First the parser checks condition. Then the parser checks condition type is Boolean. If not, the parser sets error. Then the parser checks true body statements, and false body statements. All checks finished with no error, then the TypeCheck() returns true.

2.5 CAstStatWhile

2.5.1 TypeCheck

First the parser checks condition. Then the parser checks condition type is Boolean. If not, the parser sets error. Then the parser checks while body statements. All checks finished well, then the TypeCheck(0 returns true.

2.6 CAstBinaryOp

2.6.1 TypeCheck

First the parser checks LHS and RHS. Then if GetType() returns null, it means type check is false. Therefore, the parser sets error.

2.6.2 GetType

In '+', '-', '*', '/' cases, it returns integer when both LHS and RHS are integer.

In '&&', '||' cases, it returns Boolean when both LHS and RHS are Boolean.

In '=', '#' cases, it returns Boolean when both LHS and RHS have same type (integer, character, Boolean).

In '>', '>=', '<', '<=' cases, it returns Boolean when both LHS and RHS have same type (integer, character).

Otherwise, it returns null.

2.7 CAstUnaryOp

2.7.1 TypeCheck

First the parser checks an expression. Then if GetType() is null, then the parser sets error.

2.7.2 GetType

In '+' and '-' cases, it returns integer when expression type is integer.

In '!' cases, it returns Boolean when expression type is Boolean.

Otherwise, it returns null.

2.8 CAstSpecialOp

2.8.1 TypeCheck

First the parser checks an operand. Then if GetType() is null, the parser sets error.

2.8.2 GetType

In '&' cases, it returns pointer to type of operand. (GetPointer->(GetOperand()->GetType()).

Otherwise, it returns null because we don't support other operators in SnuPL/1.

2.9 CAstFunctionCall

2.9.1 TypeCheck

First the parser checks arguments number. If the number of arguments is more or less than parameters, then the parser sets error.

Then the parser checks each argument expression. Then the parser checks argument types are same with they are defined. If not, then the parser sets error.

If all checks finishes successfully, then TypeCheck() returns true.

2.10 CAstDesignator

2.10.1 TypeCheck

CAstDesignator::TypeCheck() always returns true because invalid variables are already checked in parser.cpp.

2.11 CAstArrayDesignator

2.11.1 TypeCheck

First the parser checks symbol type is pointer or array. If not, the parser sets error.

Then the parser checks dimension number. If the dimension number is less than index size, then the parser sets error. Opposite case is okay.

Then the parser checks each dimension expression and its type. The type of expression should be

integer, so if the type is not integer then the parser sets error.

Finally, if GetType() returns null, then the parser sets error. I think this condition will never return null if above checks finished well. It's just insurance.

2.11.2 GetType

First, the parser checks a type. If the type is pointer, then we need inside array. If the type is array, then it's good. Then the parser gets inner types of array according to index size. After the end of for loop, then the parser gets the type we wanted and returns it.

2.12 CAstConstant

2.12.1 TypeCheck

CAstConstant has three types: integer, Boolean, character.

In integer case, I checked the range is -2147483648 <= integer <= 2147483647. If the value is out of range, then the parser sets error.

In Boolean case, I checked the value is 0 or 1. Otherwise the parser sets error. I set the Boolean value 0 when false, 1 when true.

In character case, I don't do type check because invalid characters are checked in scanner, and they will be 'tUndefined' tokens. So 'tCharacter' tokens have always valid characters.

2.13 CAstStringConstant

2.13.1 TypeCheck

I don't do type check for string. The reason is same with character case in CAstConstant. Invalid string tokens are 'tUndefined'.

2.13.2 GetType

In SnuPL/1, strings are treated as arrays of characters. Therefore, GetType() returns array of characters.