CS305 – Programming Languages
Fall 2025-2026

HOMEWORK 4

Implementing a Semantic Analyzer for Calculator Scientifique (CS) using Scheme

Due date: December 17th, 2025 @ 23:55

NOTE

Only SUCourse submission is allowed. No submission by e-mail. Please see the note at the end of this document for late submission policy.

1 Introduction

In this homework, you will implement various semantic check functions for the **Calculator Scientifique (CS)** language using Scheme.

2 CS Language Format in Scheme

You are familiar with the Calculator Scientifique language from the previous homeworks. The Scheme functions that you will implement will take a syntactically correct CS program as a parameter and perform some semantic analysis.

Note that the syntax of CS is now modified to a Prefix Notation (e.g., `(+ x y)`), where the operator appears before the variables.

A CS program consists of two nested lists: the first contains the **Definitions**, and the second contains the **Calculations**.

Unlike previous versions of the language, there are no Derivation (D) or Integration (I) operations in this homework.

There is no implicit multiplication in this version of the language.

You can simply assume that the CS program will be grammatically correct. You do not need to check the grammar.

An example CS program with the new prefix syntax is given below.

```
(define example-program
  '(
    ;; Definitions
    (
      (f (x)   (= (+ (^ x 2) 5)) )
      (g (t u) (= (+ t u)) )
      (h (a)   (= (+ (* a 3) 2)) )
    )

    ;; Calculations
    (
      (calculate (+ (f x y) (g a)) )
      (calculate (+ (h b) (k x)) )
    )
  )
)
```

3 Semantic Rules

Your semantic analyzer should start by performing an analysis for the following semantic rules. For each violation of these rules, your semantic analyzer must print the identifiers or function names that violate the rules, in order that they appear in the program.

You will define a separate function for each semantic check explained below.

Rule 1 (Redefined Functions): *Every function name in the definition block must be unique.* If a function is defined more than once, it is a redefinition error.

Example:

```
(
 (f (x) (= (+ x 2)) )
 (g (x) (= (+ x 5)) )
 (f (y) (= (* y y)) )
 (f (x) (= (+ x 2)) )
 (g (x) (= (* x 3)) )
)
```

Output:

```
(f f g)
```

Rule 2 (Undefined Parameters): *All variables used on the right-hand side of a function definition must appear in the parameter list of that function.*

Example:

```
(
 (f (x) (= (+ x y z y)) )
 (g (t u) (= (+ t v)) )
)
```

Output:

```
(y z y v)
```

Rule 3 (Arity Contradiction): *In each calculation statement, the number of arguments passed to a function must match the number of parameters declared in its definition.*

Example:

```
;; Definitions
(
 (f (x) (= (* x 2)) )
 (h (z) (= (* z 2)) )
 (g (t u) (= (+ t u)) )
)

;; Calculations
(
  (calculate (+ (f x y) (h s) (g a)) )
)
```

Output:

```
(f g)
```

Rule 4 (Missing Function Name): *Each function call in a calculation must have a function name before its parentheses.*

Example:

```
(
  (calculate ((x y)) )
  (calculate ((a b c)) )
  (calculate (f (x y)) )
  (calculate (* ((a b (+ 1 2))) 3) )
  (calculate ((x y) (g 3)) )
  (calculate ((x y ((z)) )) )
  (calculate (((x y))) )
)
```

Output:

```
( ((x y)) ((a b c)) ((a b (+ 1 2))) ((x y) (g 3))
((x y ((z)) )) ((z)) (((x y))) ((x y)) )
```

Rule 5 (Undefined Functions): *Every function used in calculations must be defined in the definition block.*
Example:

```
(
    (f (x) (= (* x 2)) )
    (g (t u) (= (+ t u)) )
)

((calculate (+ (f x z) (h y))) )
```

Output:

```
(h)
```

Note that, even though the function `f` is defined in the definitions block to take only one parameter and the function `f` is given two parameters in the calculation statement, this is not a problem for the undefined function check. `f` is still defined. There is an arity error for the function `f`, but it will be detected by the arity–check procedure.

4 Functions

You need to define a separate function for each semantic check explained above. The functions that you will define must have the following names:

find-redefined-functions: *Implement Rule 1 and return redefined functions.*
find-undefined-parameters: *Implement Rule 2 and return undefined parameters.*
find-arity-contradictions: *Implement Rule 3 and return functions with arity mismatches.*
find-missing-function-names: *Implement Rule 4 and return missing function name expressions.*
find-undefined-functions: *Implement Rule 5 and return undefined functions used in calculations.*

Example:

```
(define example-program
  '(
    ;; Definitions
    (
        (f (x) (= (* x 3)) )
        (g (t u) (= (+ t v)) )
        (f (y) (= (* y y)) )
        (k (a b) (= (+ (* a b) c)) )
    )

    ;; Calculations
    (
        (calculate (+ (f x) (h y)) )
        (calculate (+ (f (o y)) (((x (b l)) y)) ((c b))  ) )
        (calculate (+ (g a)))
        (calculate (+ ((g a)) ) )
    )
  )
)
```

```
(find-redefined-functions example-program)
;; → (f)

(find-undefined-parameters example-program)
;; → (v c)

(find-arity-contradictions example-program)
;; → (g g)

(find-missing-function-names example-program)
;; → ( (((x (b l)) y)) ((x (b l)) y) ((c b)) ((g a)) )

(find-undefined-functions example-program)
;; → (h o x b c)
```

5 How to Use MIT-Scheme

To test your homework, you can use the MIT-Scheme interpreter installed on the server. Follow the steps below:

1. **SSH Connection:** Connect to the cs305.sabanciuniv.edu server.
2. **Start MIT-Scheme:** Once logged in, start the Scheme interpreter by typing:

   mit-scheme

   This will launch the interpreter, and you will see a prompt similar to `1 ]=>`.
3. **Load Files:** You can load your own Scheme file or the golden file into the interpreter using the `load` command.

   ```
   (load "username-hw4.scm")
       or
   (load "golden")
   ```

4. **Exit:** To exit the interpreter, type: (exit)

6 How to Submit

Submit your Scheme file named as `username-hw4.scm` where username is your SU-Course username. In this file you must have Scheme procedures defined with the names given in Section 4. In other words, you must have:

```
(define find-redefined-functions              (lambda (cs) ...))
(define find-undefined-parameters             (lambda (cs) ...))
(define find-arity-contradictions             (lambda (cs) ...))
(define find-missing-function-names           (lambda (cs) ...))
(define find-undefined-functions              (lambda (cs) ...))
```

You can have additional helper procedures, but we will only test and grade using the functions listed above.

7 Notes

- **Important**: Name your files as you are told and **do not zip them.** [-10 points otherwise]
- **Important: Since this homework is evaluated automatically make sure your output is exactly as it is supposed to be.**
  - Return empty lists `()` when no errors are found.
  - Maintain the order of identifiers as they appear in the input.
- If you are not sure about your outputs, compare them with the ones given by the golden.
- You may get help from friends or TAs, but **you must implement the homework yourself.**

LATE SUBMISSION POLICY

Late submission is allowed subject to the following conditions:

- Your homework grade will be decided by multiplying what you get from the test cases by a "submission time factor (STF)".
- If you submit on time (i.e. before the deadline), your STF is 1.
- If you submit late, you will lose 0.01 of your STF for every 5 mins of delay.
- We will not accept any homework later than 500 mins after the deadline.
- SUCourse's timestamp will be used for STF computation.
- If you submit multiple times, the last submission time will be used.