Документация для симулятора μ-рекурсивных функций

1 Описание программы

Симулятор μ-рекурсивных функций представляет собой утилиту командной строки, позволяющую определять и симулировать вычисление μ-рекурсивных функции. Программа принимает на вход текстовый файл с определениями функций, формат которого описан в пункте 3 технического задания, а также стандартный поток ввода с запросами на вычисление функций, формат которых описан в пункте 4 технического задания.

Программа обладает пользовательским интерфейсом, позволяющим выполнять сборку в формате библиотеки. Пользовательский интерфейс представлен определениями структур для записи μ-рекурсивных функций, аргументов функций, а также функцией evaluate, симулирующей вычисление значения μ-рекурсивной функции на наборе операндов.

2 Пользовательский интерфейс

```
1 struct Function {...};
```

Мю-рекурсивная функция. Полиморфная структура для описания базисных и производных мю-рекурсивных функций. В случае производной функции определение содержит оператор и операнды, являющиеся ссылками на другие объекты Function.

```
1 struct Arguments{...};
```

Кортеж аргументов целого типа. Используется для вычисления мю-рекурсивных функций при заданном наборе аргументов.

```
1 int64_t evaluate(struct Function func, struct Argumetns arg);
```

Функция, симулирующая вычисление мю-рекурсивной функции.

```
1 void parse(FILE* stream, struct FuncList *registered\_functions);
```

Функция для парсинга определений мю-рекурсивных функций из потока чтения.

3 Внутренние структуры и методы

3.1 function.h

```
1 struct Function{...};
```

Элемент пользовательского интерфейса (см. раздел “пользовательский интерфейс”). Структура хранит метку примитивности описываемой функции, а также поля “оператор” и “операнды”. Для идентификации именованных функций используется поле “имя” фиксированного размера.

```
1 struct Operand{...};
```

Кортеж аргументов оператора, определяющего производную функцию. Хранит ссылки на объекты Function.

```
1 struct Arguments {...};
```

Элемент пользовательского интерфейса (см. Раздел “пользовательский интерфейс”). Структура хранит размер кортежа и массив целых чисел фиксированного размера.

3.2 basis.h

Базисные мю-рекурсивные функции.

```
1 int64_t Z(struct Arguments args);
```

Возвращает значение 0 на любом наборе аргументов.

```
1 int64_t S(int64_t x);
```

Возвращает следующее за x целое число (x + 1).

```
1 int64_t P(int64_t k, int64_t m, int64_t *args);
```

Возвращает из кортежа аргументов длины m k-ый элемент. При k > m выдаётся сообщение об ошибке и программа завершается.

3.3 evaluate.h

```
1 int64_t evaluate(struct Function func, struct Arguments arg);
```

Элемент пользовательского интерфейса (см. раздел “пользовательский интерфейс”). Симулирует вычисление значения func(arg).

```
1 int64_t O(struct Function func, struct Operand g, struct Arguments arg);
```

Вычисление значения мю-рекурсивной функции, определяемой оператором композиции: O(func, g)(arg).

```
1 int64_t R(struct Function h, struct Function g, struct Arguments arg);
```

Вычисление значения мю-рекурсивной функции, определяемой оператором примитивной рекурсии: R(h, g)(arg)

```
1 int64_t M(struct Function g, struct Arguments arg);
```

Вычисление значения мю-рекурсивной функции, определяемой оператором минимизации: M(g)(arg)

3.4 funclist.h

Содержит набор структур для хранения определённых в ходе работы программы мю-рекурсивных функций (объектов типа Function).

```
1 struct FuncList {...};
```

Односвязный список для хранения адресов определённых мю-рекурсивных функций.

```
1 struct FuncList *register_basis_functions();
```

Функция, регистрирующая базисные функции. Возвращает список с функциями S и Z.

```
1 void delete_funclist(struct FuncList* list);
```

Функция, удаляющая список функций и освобождающая выделенную динамически память из-под объектов Function.

3.5 parser.h

Содержит функции для парсинга потоков чтения с определениями мю-рекурсивных функций и запросов на вычисление функций.

```
enum PARSER_STATUS {...};
```

Описывает возможные состояния парсера.

```
void add_function(struct Function *func, struct FuncList *func_list);
```

Добавляет адрес функции func в список определённых функций.

```
struct Function *find_function(char *name, struct FuncList *func_list, struct FuncList **
    ↪ last_node);
```

Линейный поиск в списке по имени функции.

```
int getc_ns(FILE *stream);
```

Чтение потока до первого непробельного символа.

```
void parse(FILE *stream, struct FuncList *registered_functions);
```

Часть пользовательского интерфейса. Парсит поток чтения с определениями мю-рекурсивных функций и добавляет их в список определённых функций.

```
struct Function* parse_declaration(FILE *stream, struct FuncList *registered_functions);
```

Вызывается функцией parse для парсинга одного объявления именованной мю-рекурсивной функции.

```
struct Function* parse_definition(FILE *stream, struct FuncList *registered_functions);
```

Вызывается функцией parse_declaration для парсинга определений функций. Возможны рекурсивные вызовы при наличии операторов и вложенных определений. Добавляет новые определения функций к списку registered_functions.

```
struct Operand parse_operand(FILE *stream, struct FuncList *registered_functions);
```

Вызывается функцией parse_definition для парсинга операндов функциональных операторов. Внутри вызывает parse_definition для парсинга элементов кортежа, представляющего собой операнд, или единственной функции, представляющей операнд.

3.6 MRFS.c

Содержит функции, управляющие основным режимом работы программы: чтением определений мю-рекурсивных функций из файла и вычислением запросов из стандартного потока ввода.

3.7 tests.c

Тестирование вычисления базисных и производных функций, а также работы парсера.