Язык представления SCP-программ SimpleSCP

Введение

Язык SimpleSCP предназначен для короткого и удобочитаемого представления программ на языке SCP в варианте представления SCs.

Основы языка SimpleSCP

Программа на языке SimpleSCP представляет собой некоторую функцию, которая принимает на вход и возвращает некоторые параметры, и содержит некоторый набор операторов в теле. Тело SimpleSCP-программы состоит из линейных операторов и конструкций управления потоком.

Типичный линейный оператор языка выглядит следующим образом:

название(аргументы);

Где *название* – это ключевое слово языка SimpleSCP либо идентификатор пользовательской функции, а *аргументы* – конструкции вида:

[модификаторы, название]

название

[]

Тут *название* – идентификатор используемой константы или переменной, состоящий из нижних подчеркиваний, букв латинского алфавита и цифр, либо строковый литерал, взятый в кавычки. *Модификаторы* – список ключевых слов, обозначающих роль аргумента в операторе. Модификаторы могут быть назначены по умолчанию. Кроме того, аргумент может быть пустым – тогда при трансляции в SCP он не будет использован в операторе.

Примеры правильного вызова операторов SimpleSCP можно увидеть ниже:

Пример SimpleSCP	**Соответствующий код SCP**
search(_set, [assign, _arc],_element);	-> ..operator1 (* <- searchElStr3;; -> rrel_1: rrel_fixed: rrel_scp_var: _set;; -> rrel_2: rrel_scp_var: rrel_assign: _arc;; -> rrel_3: rrel_fixed: rrel_scp_var: _element;; *);;
sys_search(search_pattern, [assign, _result], parameters, [assign, _all_elements]);	-> ..operator1 (* <- sys_search;; -> rrel_1: rrel_fixed: rrel_scp_const: search_pattern;; -> rrel_2: rrel_scp_var: rrel_assign: _result;; -> rrel_3: rrel_fixed: rrel_scp_const: parameters;; -> rrel_4: rrel_scp_var: rrel_assign: _all_elements;; *);;
erase([erase, _element]);	-> ..operator1 (* <- eraseEl;; -> rrel_1: rrel_fixed: rrel_scp_var: rrel_erase: _element;; *);;
generate(_node1, [assign, _arc], [assign, _node2], [], [], [assign, _set]);	-> ..operator1 (* <- genSetStr3;; -> rrel_1: rrel_fixed: rrel_scp_var: _node1;; -> rrel_2: rrel_scp_var: rrel_assign: _arc;; -> rrel_3: rrel_scp_var: rrel_assign: _node2;; -> rrel_set_3: rrel_assign: rrel_scp_var: _set;; *);;
proc_of_user_function(_user_argument,	-> ..operator1 (* <- call;;

```
  nrel_some_relation
);
```

```
  -> rrel_1: rrel_fixed: rrel_scp_const:
proc_of_user_function;;
  -> rrel_2: ... (*
    -> rrel_1: rrel_fixed: rrel_scp_var: _user_argument;;
    -> rrel_2: rrel_fixed: rrel_scp_const:
nrel_some_relation;;
  *);;
  -> rrel_3: rrel_scp_var: rrel_assign: _process;;
  => nrel_goto: ..operator2;;
*);;
-> ..operator2 (*
  <- waitReturn;;
  -> rrel_1: rrel_fixed: rrel_scp_var: _process;;
*);;
```

Конструкции управления потоком устанавливают соответствующие связи nrel_goto, nrel_then и nrel_else между операторами программы. Конструкциями управления потоком в SimpleSCP являются условия и циклы.

Условные конструкции используются следующим образом:

if (оператор) {действия; если;}
else {действия; в; другом; случае;}

Здесь *оператор* – линейный оператор языка SimpleSCP, *действия если* – набор операторов, к которым будет осуществлен переход nrel_then от *оператора*, *действия в другом случае* - набор операторов, к которым будет осуществлен переход nrel_else от *оператора*.

Пример правильного использования условной конструкции

Пример SimpleSCP

```
if(search(some_class, [assign, _arc], _node)) {
  proc_do_some_actions(_node);
}
```

Соответствующий код SCP

```
-> ..operator1 (*
  <- searchElStr3;;
  -> rrel_1: rrel_fixed: rrel_scp_const: some_class;;
  -> rrel_2: rrel_scp_var: rrel_assign: _arc;;
  -> rrel_3: rrel_fixed: rrel_scp_var: _node;;
  => nrel_then: ..operator2;;
  => nrel_else: ..operator4;;
*);;
-> ..operator2 (*
  <- call;;
  -> rrel_1: rrel_fixed: rrel_scp_const: proc_do_some_actions;;
  -> rrel_2: ... (*
    -> rrel_1: rrel_fixed: rrel_scp_var: _node;;
  *);;
  -> rrel_3: rrel_scp_var: rrel_assign: _process;;
  => nrel_goto: ..operator3;;
*);;
-> ..operator3 (*
  <- waitReturn;;
  -> rrel_1: rrel_fixed: rrel_scp_var: _process;;
```

```
*);;
```

Циклы с предусловием в SimpleSCP описываются следующим образом:

while(оператор) {действия;}

Здесь *оператор* – линейный оператор языка SimpleSCP, *действия* – набор операторов, к которым будет осуществлен переход nrel_then от *оператора* и от которых будет осуществлен переход nrel_goto к *оператору*.

Пример правильного использования циклической конструкции

Пример SimpleSCP

```
while(search(_set, [assign, _arc], [assign, _node])){
  print_el(_node);
  proc_of_some_actions([_node]);
  erase(erase, _arc);
 }
```

Соответствующий код SCP

```
-> ..operator1 (*
  <- searchElStr3;;
  -> rrel_1: rrel_fixed: rrel_scp_var: _set;;
  -> rrel_2: rrel_scp_var: rrel_assign: _arc;;
  -> rrel_3: rrel_scp_var: rrel_assign: _node;;
  => nrel_then: ..operator2;;
  => nrel_else: ..operator6;;
*);;
-> ..operator2 (*
  <- printEl;;
  -> rrel_1: rrel_fixed: rrel_scp_var: _node;;
  => nrel_goto: ..operator3;;
*);;
-> ..operator3 (*
  <- call;;
  -> rrel_1: rrel_fixed: rrel_scp_const: proc_of_some_actions;;
  -> rrel_2: ... (*
    -> rrel_1: rrel_fixed: rrel_scp_var: _node;;
  *);;
  -> rrel_3: rrel_scp_var: rrel_assign: _process;;
  => nrel_goto: ..operator4;;
*);;
-> ..operator4 (*
  <- waitReturn;;
  -> rrel_1: rrel_fixed: rrel_scp_var: _process;;
  => nrel_goto: ..operator5;;
*);;
-> ..operator5 (*
  <- eraseEl;;
  -> rrel_1: rrel_fixed: rrel_scp_var: rrel_erase: _arc;;
  => nrel_goto: ..operator1;;
*);;
```

Для того, чтобы передать в SimpleSCP-функцию параметры, нужно в сигнатуре функции указать идентификаторы:

function example(имена, параметров)

Здесь *имена параметров* – перечисление идентификаторов параметров через запятую.
Для того, чтобы объявить в SimpleSCP-функции выходные параметры, нужно в начале тела функции указать:

return идентификатор;

Здесь *идентификатор* – название выходного параметра. Конструкций такого вида в программе может быть несколько.
Примеры правильного объявления входных и выходных параметров SimpleSCP-функции:

Пример SimpleSCP	**Соответствующий код SCP**

function example(_set) {
  return _set;
  return _answer;
}
	-> rrel_params: ... (*
	  -> rrel_1: rrel_in: _set;;
	  -> rrel_1: rrel_out: _set;;
	  -> rrel_2: rrel_out: _answer;;
	*);;

Приложение 1. Список операторов языка SimpleSCP
(Добавляются по мере необходимости)

Название	**Аргументы**	**Аналог в SCP**
generate	(#1)	->..operator (* <- genEl;; -> rrel_1: #1;; *);;

	(#1, #2, #3)	->..operator (*
		  <- genElStr3;;
		  -> rrel_1: #1;;
		  -> rrel_2: #2;;
		  -> rrel_3: #3;;
		*);;

	(#1, #2, #3, #4, #5)	->..operator (*
		  <- genElStr5;;
		  -> rrel_1: #1;;
		  -> rrel_2: #2;;
		  -> rrel_3: #3;;
		  -> rrel_4: #4;;
		  -> rrel_5: #5;;
		*);;

	(#1, #2, #3, #4, #5, #6)	->..operator (*
		  <- genSetStr3;;
		  -> rrel_1: #1;;
		  -> rrel_2: #2;;
		  -> rrel_3: #3;;
		  -> rrel_set_1: #4;;
		  -> rrel_set_2: #5;;
		  -> rrel_set_3: #6;;
		*);;

	(#1, #2, #3, #4, #5, #6, #7, #8, #9, #10)	-> ..operator(*

		<- genSetStr5;; -> rrel_1: #1;; -> rrel_2: #2;; -> rrel_3: #3;; -> rrel_4: #4;; -> rrel_5: #5;; -> rrel_set_1: #6;; -> rrel_set_2: #7;; -> rrel_set_3: #8;; -> rrel_set_4: #9;; -> rrel_set_5: #10;; *);;
search	(#1, #2)	->..operator (* <- searchSet;; -> rrel_1: #1;; -> rrel_set_1: #2;; *);;
	(#1, #2, #3)	->..operator (* <- searchElStr3;; -> rrel_1: #1;; -> rrel_2: #2;; -> rrel_3: #3;; *);;
	(#1, #2, #3, #4, #5)	->..operator (* <- searchElStr5;; -> rrel_1: #1;; -> rrel_2: #2;; -> rrel_3: #3;; -> rrel_4: #4;; -> rrel_5: #5;; *);;
	(#1, #2, #3, #4, #5, #6)	->..operator (* <- searchSetStr3;; -> rrel_1: #1;; -> rrel_2: #2;; -> rrel_3: #3;; -> rrel_set_1: #4;; -> rrel_set_2: #5;; -> rrel_set_3: #6;; *);;
	(#1, #2, #3, #4, #5, #6, #7, #8, #9, #10)	-> ..operator(* <- searchSetStr5;; -> rrel_1: #1;; -> rrel_2: #2;; -> rrel_3: #3;; -> rrel_4: #4;; -> rrel_5: #5;; -> rrel_set_1: #6;; -> rrel_set_2: #7;; -> rrel_set_3: #8;;

		-> rrel_set_4: #9;; -> rrel_set_5: #10;; *);;
erase	(#1)	->..operator (* <- eraseEl;; -> rrel_1: #1;; *);;
	(#1, #2, #3)	->..operator (* <- eraseElStr3;; -> rrel_1: #1;; -> rrel_2: #2;; -> rrel_3: #3;; *);;
	(#1, #2, #3, #4, #5)	->..operator (* <- eraseElStr5;; -> rrel_1: #1;; -> rrel_2: #2;; -> rrel_3: #3;; -> rrel_4: #4;; -> rrel_5: #5;; *);;
	(#1, #2, #3, #4, #5, #6)	->..operator (* <- eraseSetStr3;; -> rrel_1: #1;; -> rrel_2: #2;; -> rrel_3: #3;; -> rrel_set_1: #4;; -> rrel_set_2: #5;; -> rrel_set_3: #6;; *);;
	(#1, #2, #3, #4, #5, #6, #7, #8, #9, #10)	-> ..operator(* <- eraseSetStr5;; -> rrel_1: #1;; -> rrel_2: #2;; -> rrel_3: #3;; -> rrel_4: #4;; -> rrel_5: #5;; -> rrel_set_1: #6;; -> rrel_set_2: #7;; -> rrel_set_3: #8;; -> rrel_set_4: #9;; -> rrel_set_5: #10;; *);;
sys_generate	(#1, #2, #3, #4)	->..operator (* <- sys_gen;; -> rrel_1: #1;; -> rrel_2: #2;; -> rrel_3: #3;; -> rrel_4: #4;;

		*);;
sys_search	(#1, #2, #3, #4)	->..operator (* <- sys_search;; -> rrel_1: #1;; -> rrel_2: #2;; -> rrel_3: #3;; -> rrel_4: #4;; *);;
var_assign	(#1, #2)	->..operator (* <- varAssign;; -> rrel_1: #1;; -> rrel_2: #2;; *);;
cont_assign	(#1, #2)	->..operator (* <- contAssign;; -> rrel_1: #1;; -> rrel_2: #2;; *);;
has_value	(#1)	->..operator (* <- ifVarAssign;; -> rrel_1: #1;; *);;
add	(#1, #2, #3)	->..operator (* <- contAdd;; -> rrel_1: #1;; -> rrel_2: #2;; -> rrel_3: #3;; *);;
subtract	(#1, #2, #3)	->..operator (* <- contSub;; -> rrel_1: #1;; -> rrel_2: #2;; -> rrel_3: #3;; *);;
multiple	(#1, #2, #3)	->..operator (* <- contMult;; -> rrel_1: #1;; -> rrel_2: #2;; -> rrel_3: #3;; *);;
divide	(#1, #2, #3)	->..operator (* <- contDiv;; -> rrel_1: #1;; -> rrel_2: #2;; -> rrel_3: #3;; *);;
cos	(#1, #2)	->..operator (* <- contCos;;

		-> rrel_1: #1;; -> rrel_2: #2;; *);;
sin	(#1, #2)	->..operator (* <- contSin;; -> rrel_1: #1;; -> rrel_2: #2;; *);;
acos	(#1, #2)	->..operator (* <- contACos;; -> rrel_1: #1;; -> rrel_2: #2;; *);;
asin	(#1, #2)	->..operator (* <- contASin;; -> rrel_1: #1;; -> rrel_2: #2;; *);;
greater	(#1, #2)	->..operator (* <- ifGr;; -> rrel_1: #1;; -> rrel_2: #2;; *);;
print	(#1)	->..operator (* <- print;; -> rrel_1: #1;; *);;
show	(#1)	->..operator (* <- printEl;; -> rrel_1: #1;; *);;
is	(#1, #2)	->..operator (* <- ifCoin;; -> rrel_1: #1;; -> rrel_2: #2;; *);;
equals	(#1, #2)	->..operator (* <- ifEq;; -> rrel_1: #1;; -> rrel_2: #2;; *);;
greater	(#1, #2)	->..operator (* <- ifGr;; -> rrel_1: #1;; -> rrel_2: #2;; *);;

Приложение 2. Упрощенные операторы языка SimpleSCP

Оператор	**Упрощение**
has_value(_a)	_a

Приложение 3. Модификаторы аргумента SimpleSCP
(Добавляются по мере необходимости)

Модификатор	**Аналог в SCP**
fixed	rrel_fixed: arg
assign	rrel_assign: arg
pos_const_perm	rrel_pos_const_perm: arg
constant	rrel_const: arg
node	rrel_node: arg
arc	rrel_arc: arg
link	rrel_link: arg
erase	rrel_erase: arg
scp_variable	rrel_scp_var: arg
scp_constant	rrel_scp_const: arg
common	rrel_common: arg

Приложение 4. Модификаторы аргумента по умолчанию.

Роль	До обработки	После обработки
Константа	argument	[fixed, scp_constant, argument]
Переменная	_argument	[fixed, scp_variable, _argument]

Приложение 5. Примеры программ SimpleSCP.

Hello World
SimpleSCP

function example1() {
  print("Hello world!");
}

SCP

scp_program -> example1 (*
  -> rrel_params: ... (*
  *);;
  -> rrel_operators: ... (*
  ->rrel_init: ..operator50693 (*
  <- print;;
  -> rrel_1: rrel_fixed: rrel_scp_const: [Hello world!];;
  => nrel_goto: ..operator26856;;
  *);;
  ->..operator26856 (*
  <- return;;

```
      *);;
   *);;
*);;
```

Программа нахождения квадрата числа

SimpleSCP

```
function example3(_x) {
  return _y;
  multiple([assign, _y], _x, _x);
}
```

SCP

```
scp_program -> example3 (*
    -> rrel_params: ... (*
       -> rrel_1: rrel_in: _x;;
       -> rrel_2: rrel_out: _y;;
    *);;
    -> rrel_operators: ... (*
       ->rrel_init: ..operator53806 (*
          <- contMult;;
          -> rrel_1: rrel_scp_var: rrel_assign: _y;;
          -> rrel_2: rrel_fixed: rrel_scp_var: _x;;
          -> rrel_3: rrel_fixed: rrel_scp_var: _x;;
          => nrel_goto: ..operator16416;;
       *);;
       ->..operator16416 (*
          <- return;;
       *);;
   *);;
*);;
```

Программа нахождения конструкций по шаблону

SimpleSCP

```
function proc_of_demo(_pattern, _results) {
  sys_search(_pattern, results, parameters, _results)
  if(search(_results, [assign, _arc], [assign, _result]))
    search(
      _pattern,
      [assign, _arc],
      [assign, constant, _element],
      [], [], _results
    );
  else
    print("Nothing!");
}
```

SCP

```
scp_program -> example (*
  -> rrel_params: ... (*
    -> rrel_1: rrel_in: _pattern;;
    -> rrel_2: rrel_out: _results;;
  *);;
```

```
  -> rrel_operators: ... (*
    ->rrel_init: ..operator46740 (*
      <- sys_search;;
      -> rrel_1: rrel_fixed: rrel_scp_var: _pattern;;
      -> rrel_2: rrel_fixed: rrel_scp_const: results;;
      -> rrel_3: rrel_fixed: rrel_scp_const: parameters;;
      -> rrel_4: rrel_scp_var: rrel_assign: _results;;
      => nrel_goto: ..operator2311;;
    *);;
    ->..operator2311 (*
      <- ifVarAssign;;
      -> rrel_1: rrel_fixed: rrel_scp_var: _results;;
      => nrel_then: ..operator64829;;
      => nrel_else: ..operator32945;;
    *);;
    ->..operator64829 (*
      <- searchSetStr3;;
      -> rrel_1: rrel_fixed: rrel_scp_var: _pattern;;
      -> rrel_2: rrel_scp_var: rrel_assign: _arc;;
      -> rrel_3: rrel_scp_var: rrel_assign: _element;;
      -> rrel_set_3: rrel_fixed: rrel_scp_var: _results;;
      => nrel_goto: ..operator20621;;
    *);;
    ->..operator32945 (*
      <- print;;
      -> rrel_1: rrel_fixed: rrel_scp_const: [Nothing!];;
      => nrel_goto: ..operator20621;;
    *);;
    ->..operator20621 (*
      <- return;;
    *);;
  *);;
*);;
```