Рассмотрим на примере модельного императивного языка программирования IPL с одним типом int, константами типа int, операциями доступа к значению переменной, присваивания значения переменной и сложения целых чисел, условным оператором и оператором-блоком.

1. Документ со спецификацией языка IPL на естественном языке.

* множество простых типов данных языка IPL:
  + тип int, представляет множество целых чисел;
* множество составных типов данных языка IPL:
  + тип array<T>[n] – массив константной длины n, элементы которого имеют одинаковый тип T;
* множество констант типа int включает целочисленные константы в стандартной нотации, например -2, 6, 0 и т. п.;
* множество констант типа array<T>[n] включает массивы длины n, элементы которого являются константами типа T;
* выражения языка IPL имеют тип и определяются следующим образом:
  + если c – константа типа T, то c – выражение типа T;
  + если x – переменная типа T, то x – выражение типа T;
  + если e1 и e2 – выражения типа int, то e1+e2 – выражение типа int;
  + если x – переменная типа T, и e – выражение типа T, то x = e – выражение присваивания типа T;
  + если e – выражение типа T, то (e) – выражение типа T;
  + если e1, …, en – выражения типа T, то e1, …, en – выражение типа T;
* семантика выражений языка IPL определяются следующим образом:
  + выражение (для константы) c возвращает c;
  + выражение x возвращает значение переменной x. Если такая переменная не определена, то ошибка;
  + выражение e1+e2 возвращает сумму значений выражений e1 и e2 типа int;
  + выражение x = e; вычисляет выражение e, присваивает переменной x значение этого выражения и возвращает вычисленное значение выражения e. Если такой переменной нет, то заводится новая переменная типа как у выражения e;
  + выражение (e) возвращает значение выражения e;
  + выражение e1, …, en вычисляет выражения e1, …, en и возвращает значение выражения en;
* операторы языка IPL определяются следующим образом:
  + последовательность операторов определяется следующим образом: если s1, …, sn – операторы, то s1, …, sn  – последовательность операторов;
  + break - оператор передачи управления break (*далее для краткости, вместо оператор передачи управления будем писать ПУ-оператор*);
  + continue – ПУ-оператор continue;
  + return – ПУ-оператор return;
  + если e – выражение, то e – оператор;
  + если e – выражение типа int, s – оператор, то if(e) s – условный оператор;
  + если e – выражение типа int, s – ПУ-оператор, то if(e) s – ПУ-оператор;
  + если e – выражение типа int, s, s′ – операторы, то if(e) s else s′ – условный оператор;
  + если e – выражение типа int, s, s′ – операторы или ПУ-операторы, из которых по крайней мере один является ПУ-оператором, то if(e) s else s′ – условный ПУ-оператор;
  + если s – последовательность операторов, то {s} – оператор блока;
  + если s1, …, sn – операторы или ПУ-операторы, из которых по крайней мере один ПУ-оператор, то {s1,…, sn} – ПУ-оператор блока;
  + если e – выражение, s – оператор или ПУ-оператор, то while(e) s – оператор (цикла while);
  + если e – выражение, s – оператор или ПУ-оператор, то do-while s (e) – оператор (цикла do-while);
  + если e, e′, e′′ – выражения, s – оператор или ПУ-оператор, то for e′ e e′′ s – оператор (цикла for);
  + если e – выражение, с1, …, сn – константы типа int, s1, …, sn –последовательности операторов, то switch(e) case с1 s1  … case сn sn – оператор switch;
  + если e – выражение, с1, …, сn – константы типа int, s1, …, sn+1 –последовательности операторов, то switch(e) case с1 s1  … case сn sn default sn+1 - оператор switch;
* семантика операторов языка IPL определяется следующим образом:
  + оператор if(e) s else s′ вычисляет выражение e. Если e возвращает не 0, то выполняем оператор s. Если e возвращает 0, то выполняем оператор s′;
  + оператор if(e) s вычисляет выражение e. Если e возвращает не 0, то выполняем оператор s. Если e возвращает 0, то ничего не делаем;
  + оператор блока {s1, …, sn} последовательно выполняет операторы s1, …, sn;
  + оператор switch(e) case с1 s1  … case сn sn  вычисляет выражение e и выполняет последовательно все последовательности операторов sk , для которых значение выражения e совпадает с константой сk , пока не встретит оператора передачи управления (вне вложенного цикла). Если встретит оператор break, то заканчивает выполнение;
  + оператор switch(e) case с1 s1  … case сn sn default sn+1 вычисляет выражение e и выполняет последовательно все последовательности операторов sk , для которых значение выражения e совпадает с константой сk, а также последовательность операторов sn+1. Выполнение происходит, пока не встретится оператор передачи управления;
  + оператор while(e) s вычисляет выражение e. Если e возвращает не 0, то выполняем оператор s. Повторяет данные действия, пока e возвращает не 0. Если e возвращает 0, то заканчивает работу. Если при выполнении оператора s встречается оператор break (вне вложенных операторов цикла или switch), то выполнение оператора while заканчивается. Если при выполнении оператора s встречается оператор continue (вне вложенных операторов цикла или switch), то выполнение оператора s заканчивается;
  + оператор do-while s (e) выполняет оператор s, вычисляет выражение e. Если e возвращает не 0, то повторяет данные действия, пока e возвращает не 0. Если e возвращает 0, то заканчивает работу. Если при выполнении оператора s встречается оператор break (вне вложенных операторов цикла или switch), то выполнение данного оператора цикла заканчивается. Если при выполнении оператора s встречается оператор continue (вне вложенных операторов цикла или switch), то выполнение оператора s заканчивается;
  + оператор for e′ e e′′ s вычисляет выражение e′, вычисляет выражение e. Пока оно отлично от нуля выполняет оператор s и вычисляет выражение e′′. Если при выполнении оператора s встречается оператор break (вне вложенных операторов цикла или switch), то выполнение данного оператора цикла заканчивается. Если при выполнении оператора s встречается оператор continue (вне вложенных операторов цикла или switch), то выполнение оператора s заканчивается;
  + оператор break может встречаться только внутри операторов, входящих в состав switch, while, for;
  + оператор continue может встречаться только внутри операторов, входящих в состав while, for;

1. Документ на естественном языке с описанием рекурсивной функции rep, сопоставляющей инструкциям языка IPL выражения на языке ASL.
   * rep(c) = c;
   * rep(x) = variableAccess(x);
   * rep(e1+e2) = plus(rep(e1), rep(e2));
   * rep((e)) = rep(e);
   * rep(x = e;) = variableAssignment(x, rep(e));
   * rep(e) = expressionStatement(rep(e));
   * rep(if(e) s else s′) = ifStatement(rep(e), rep(s), rep(s′));
   * rep(if(e) s) = ifStatement(rep(e), rep(s));
   * rep({s1, …, sn}) = blockStatement(rep(e), rep(s1), …, rep(sn));
   * rep(while(e) s) = whileStatement(rep(e), rep(s));
   * rep(do-while s (e)) = rep({s, while(e) s};
   * rep(for e′ e e′′ s) = rep({e′, if(e) do-while s (e′′, e)});

//для последовательностей операторов S1, S2, …, Sn, S (здесь с большой буквы, чтобы отличать последовательности операторов от операторов, т.е. Si – это последовательность операторов si1, si2, ..., sik)

* + rep(switch(e) case с1 S1) = rep(if(e == c1) {S1});
  + rep(switch(e) case с1 S1 … case сn Sn) = rep(if(e == c1) {S1, switch(e) case с2 S2 … case сn Sn} else switch(e) case с2 S2 … case сn Sn);
  + rep(switch(e) case с1 S1 default S) = rep(if(e == c1) {S1, S} else {S});
  + rep(switch(e) case с1 S1 … case сn Sn default S) = rep(if(e == c1) {S1, switch(e) case с2 S2 … case сn Sn} else switch(e) case с2 S2 … case сn Sn default S);

1. Программа на языке ASL с определениями конструкторов для инструкций.

defun variableAccess($variable) {

return(conz(

variableAccess, concept, variable, $variable

));

}

defun plus($expression1, $expression2) {

return(conz(

qname("+"), concept,

arguments, seqOne($expression1, $$expression2)

));

}

defun variableAssignment($variable, $expression) {

return(

conz(

variableAssignment, concept,

variable, $variable,

expression, $expression

)

);

}

defun ifStatement($condition, $thenBranch,

$elseBranch...

) {

return(

conz(

ifStatement, concept,

condition, $condition,

thenBranch, $thenBranch,

elseBranch, elseBranch[1]

)

);

}

defun blockStatement($statementSequence...) {

return(

conz(

blockStatement, concept,

statementSequence, $statementSequence

)

);

}

defun whileStatement($condition, $whileBody) {

return(

conz(

whileStatement, concept,

condition, $condition,

whileBody, $whileBody

)

);

}

1. Документ на естественном языке с описанием модели состояния императивной программы на языке PL. Состояние ψ характеризуется следующими глобальными атрибутами:

* @variableState хранит значения переменных. Если @variableState.x = v, то переменная x имеет значение v в состоянии ψ. Если @variableState.x = undef, то переменная x не определена в состоянии ψ.

1. Программа на языке ASL с реализацией операционной семантики онтологических моделей для онтологических моделей инструкций языка IPL.

// общее правило для всех инструкций

// (операторов и выражений)

defun execute:instruction($instruction) {

if ($instruction.variableAccess == concept)

then return(execute:variableAccess($instruction));

else if($instruction.variableAssignment == concept)

then execute:variableAssignment($instruction);

else if($instruction.qname("+") == concept)

then return(execute:plus($instruction));

else if($instruction.ifStatement == concept)

then return(execute:ifStatement($instruction));

else if($instruction.blockStatement == concept)

then return(execute:blockStatement($instruction));

else if($instruction.whileStatement == concept)

then return(execute:whileStatement($instruction));

else return($instruction) // случай константы

}

// правило для доступа к значению переменной

defun execute:variableAccess($variableAccess) {

$value = @variableState.(variableAccess.variable);

if(isDef($value))

then return($value);

else jump(

execute:variableAccess:Error, $variableAccess

);

}

// правило для операции сложения

defun execute:plus($plus) {

$arguments = $plus.arguments;

return(

execute:instruction($arguments[1]) +

execute:instruction($arguments[2])

);

}

// правило для присваивания

defun execute:variableAssignment($variableAssignment) {

@variableState.($variableAssignment.variable) =

execute:instruction($variableAssignment.expression);

}

// правило для условного оператора

defun execute:ifStatement($ifStatement) {

if(execute:instruction($ifStatement.condition) == 0)

then execute:instruction($ifStatement.thenBranch);

else {

$elseBranch = $ifStatement.elseBranch;

if(isDef($elseBranch))

then execute:instruction($elseBranch);

}

}

// правило для блока

defun execute:blockStatement($blockStatement) {

foreach $i $statement

SeqAsc($blockStatement.statementSequence) {

execute:instruction($statement);

}

}

// правило для оператора while

defun execute:whileStatement($whileStatement) {

$condition = execute:instruction($whileStatement.condition);

if($condition != 0) then {

execute:instruction($whileStatement.body);

execute:whileStatement($whileStatement);

}

catch(break);

}

1. Набор юнит-тестов для онтологических моделей инструкций языка IPL и результаты их выполнения.

> variableAccess(x);

Done in 0 msec, returned value is not set due to jump-in-progress:

Jump type @res[1] is x

Jump value @res[2] is y

> variableAssignment(x, 1);

Done in 0 msec, returned value @res[3] is undef

> variableAccess(x);

Done in 0 msec, returned value @res[4] is 1

> plus(x, 1)

Done in 0 msec, returned value @res[5] is 2

> ... // остальные тесты