Особенности определения формальной семантики императивного языка для верификации программ интеграции данных

С. А. Ступников

***Аннотация*—С увеличением числа неоднородных источников данных в науке и промышленности растет потребность в их интеграции. В различных предметных областях разрабатываются специализированные системы интеграции данных. Ввиду сложности программ интеграции данных, важным вопросом является формальная проверка их корректности – верификация. Корректность программы интеграции данных понимается как определяемое экспертом свойство, связывающее состояние совокупности источников данных и состояние целевой интегрирующей базы данных, выражаемое в виде логической формулы. Процесс формальной верификации сложен, однако оправдан, поскольку стоимость исправления ошибки после выпуска системы в производство превышает стоимость исправления ошибки на этапе разработки системы в десятки и сотни раз. На практике программы интеграции часто реализуются на императивных языках. В статье выделены особенности и суммирован опыт по определению формальной семантики императивных языков для верификации программ интеграции данных в областях астрономии, материаловедения, управления землепользованием. Рассмотрены особенности определения семантики библиотечных математических и строковых функций, функций преобразования данных, оператора присваивания и вызова вспомогательных функций, последовательной композиции операторов, условного оператора, операторов циклов. Приведены общие принципы применения формальной семантики императивного языка программирования для верификации программ интеграции данных с использованием автоматизированных средств доказательства.**

***Ключевые слова*—формальная семантика программ, верификация программ, интеграция данных.**

I. Введение

С увеличением числа неоднородных источников данных в науке и промышленности растет потребность в их интеграции [1]. Для этого в различных предметных областях разрабатываются системы интеграции данных. К примерам таких областей относятся астрономия [4], управление землепользованием [7], материаловедение [5], науки о Земле [8].

Ввиду сложности программ интеграции данных, важным вопросом является формальная проверка их корректности (верификация). *Корректность программы интеграции данных* – это определяемое экспертом свойство, связывающее состояние совокупности источников данных и состояние целевой интегрирующей базы данных, выражаемое в виде логической формулы. Например, корректность может состоять в том, что для всех объектов из источников в целевой базе данных в результате применения программы интеграции появляются их правильные образы (отсутствуют потерянные данные). Процесс формальной верификации сложен, однако оправдан, поскольку стоимость исправления ошибки после выпуска системы в производство превышает стоимость исправления ошибки на этапе разработки системы в десятки и сотни раз [12].

Основная идея подходов к верификации программ обычно состоит в том, чтобы определить их семантику в некотором формальном языке. Свойства программ, подлежащие проверке, представляются в виде выражений этого языка. Затем, с использованием формальных средств доказательства, спецификация, выражающая семантику конкретной программы интеграции данных, проверяется на соответствие необходимым свойствам.

Программы интеграции данных могут быть реализованы на различных языках: на логических правилах [15], декларативных (таких, как SQL [16] и HIL [14]), императивных (например, VBScript [17]). С концептуальной точки зрения для спецификации программ интеграции данных предпочтительны декларативные специализированные языки высокого уровня. Однако на практике программы интеграции нередко реализуются на императивных языках.

В данной статье выделены особенности и суммирован опыт по определению формальной семантики императивных языков для верификации программ интеграции данных в базе данных двойных звезд, разработанной в ИНАСАН [4][2][3]; в интегрированной системе баз данных по свойствам неорганических веществ и материалов Института металлургии и материаловедения им. А.А. Байкова РАН [5][6]; в системе «Умное землепользование», разрабатываемой в Почвенном институте имени В.В. Докучаева [7]. Для единообразия изложения примеры и образцы программного кода программ интеграции данных приведены на языке Python. Работа сосредоточена на том подмножестве императивного языка, которое

Статья получена 1 октября 2025.

С.А. Ступников – МГУ имени М.В. Ломоносова, Федеральный исследовательский центр «Информатика и управление» РАН (sstupnikov@ipiran.ru)

используется для реализации программ интеграции данных в конкретном наборе систем интеграции.

Существует достаточное количество работ, посвященных определению различных видов семантики (операционной, денотационной, аксиоматической) императивных языков программирования [18]. Существуют и работы, посвященные определению операционной семантики непосредственно языка Python [19][20]. Для определения семантики в известных работах используются различные формальные языки.

В качестве языка определения формальной семантики в данной работе используется Нотация абстрактных машин (AMN) [9] — язык, основанный на логике первого порядка и теории множеств. Язык AMN позволяет моделировать программные системы в виде абстрактных машин – спецификаций пространства состояний и поведения (определенного операциями на состояниях) систем. Спецификация состояния абстрактной машины задается переменными состояния вместе с инвариантами — ограничениями, которые должны всегда удовлетворяться. Операции определяются на основе расширения формализма охраняемых команд Дейкстры (*обобщенных подстановок*). AMN выбран как язык, поддержанный индустриальной технологией и средствами доказательства Atelier B [11]. Накоплен более чем двадцатилетний мировой опыт применения языка AMN и средств его инструментальной поддержки при разработке промышленных программных систем [13]. Данная работа демонстрирует, что язык AMN позволяет определить денотационную семантику императивных программ, которая позиционируется как подход, позволяющий наиболее полно и точно выразить значение конструкций языка программирования.

В разделе II настоящей статьи суммированы особенности определения семантики подмножества конструкций императивного языка, используемого для определения программ интеграции данных: библиотечных математических и строковых функций, функций преобразования данных, оператора присваивания и вызова вспомогательных функций, последовательной композиции операторов, условного оператора, циклов. Основная часть иллюстративных примеров использует фрагменты программ интеграции данных в базу данных двойных звезд. Отдельные детали программ интеграции опущены для упрощения изложения. В разделе III изложены общие принципы применения формальной семантики императивного языка программирования для верификации программ интеграции данных.

II. Особенности определения формальной семантики конструкций императивного языка

A. Особенности определения семантики библиотечных математических функций

При определении семантики библиотечных математических функций можно абстрагироваться от конкретных реализаций этих функций и задать их свойства при помощи явных выражений или формул логики предикатов первого порядка. Ниже этот принцип продемонстрирован на примере функций *trunc* и *sqrt* модуля *Math* языка Python, используемых при определении программ интеграции данных в базе данных двойных звезд (Листинг 1).

```
MACHINE
  Math
ABSTRACT_CONSTANTS
  trunc,
  sqrt,
PROPERTIES
  trunc = %xx.(xx: REAL | floor(xx)) &
  sqrt: REAL +-> REAL &
  !xx.(xx: REAL & xx >= 0 =>
    sqrt(xx)*sqrt(xx) = xx)
END
```

Листинг 1. Пример определения семантики библиотечных математических функций

Для модуля определена одноименная спецификация AMN вида MACHINE. Функции определены как абстрактные константы спецификации.

Если значение функции может быть задано арифметическим выражением, параметром которого является аргумент функции (например, функция *trunc*), то функция задается в AMN соответствующим лямбда-выражением %. В случае функции *trunc* это выражение крайне простое (*floor(xx)*), поскольку в AMN существует встроенная функция округления вещественного числа *floor* (“floor represents the floor function, defined for ℝ in Z. floor(x) represents the floor of x, i.e the only integer n such that n <= x < n+1” [10]).

Если встроенных арифметических функций AMN недостаточно, чтобы выразить семантику функции в виде выражения (например, *sqrt*), то соответствующая абстрактная константа типизируется как частичная функция (+->) из типа аргумента в тип возвращаемого значения. Дополнительно определяется формула – логическая импликация (=>) под квантором всеобщности (!) по аргументу функции, в которой для любого значения аргумента функции задается связь между значением аргумента и возвращаемым значением функции. В случае функции *sqrt* эта связь задается предикатом равенства (Листинг 1):

```
sqrt(xx)*sqrt(xx) = xx
```

B. Особенности определения семантики функций преобразования данных

В случаях, когда при определении программ интеграции данных выделяются модули повторно используемых функций относительно простых преобразований данных, семантика этих модулей определяется в AMN с использованием отдельных спецификаций вида MACHINE. Функции при этом представляются операциями спецификации. Пример функции преобразования астрономических данных (преобразующей относительные координаты объекта на небесной сфере в абсолютные), используемой в программе интеграции данных в базе данных двойных звезд выглядит следующим образом:

```
def rel2abs(bRA,bDE,theta,rho):
  theta=theta-360.0*math.trunc(theta/360.0)
  if theta<0:
```

```
    theta=360.0+theta
  th=theta*math.pi/180.0
  dd=rho/math.sqrt(1.0+(math.tan(th)**2))
  da=abs(dd)*abs(math.tan(th))
  if theta<90.0:
    if bDE<0.0:
      dd=-dd
  elif theta<180.0:
    if bDE>0.0:
      dd=-dd
  elif theta<270.0:
    da=-da
    if bDE>0.0:
      dd=-dd
  else:
    da=-da
    if bDE<0.0:
      dd=-dd
  return (bRA+da,bDE+dd)
```

Листинг 2. Пример определения функции преобразования астрономических данных

Семантика функции *rel2abs* выражается в виде одноименной операции спецификации *Conversions*:

```
MACHINE
  Conversions
SEES Math
OPERATIONS
ra, de <-- rel2abs(bRA, bDE, theta, rho) =
PRE
  bRA: REAL & bDE: REAL & theta: REAL & rho: REAL
THEN
  ANY theta, theta1, th, dd, dd1, da, da1 WHERE
   theta: REAL &
   theta = theta -
     360.0*real(trunc(theta/360.0)) &
   theta1: REAL &
   (theta < 0.0 => theta1 = 360.0 + theta) &
   (theta >= 0.0 => theta1 = theta) &
   th: REAL & th = theta0 * pi/180.0 &
   dd: REAL & dd = rho/sqrt(1.0 + (tan(th)**2)) &
   da: REAL & da = abs(dd)*abs(tan(th)) &
   dd1: REAL & da1: REAL &
   (theta1 < 90.0 & bDE < 0.0 =>
      dd1 = 0.0 - dd & da1 = da) &
   (theta1 >= 90.0 & theta1 < 180.0 & bDE > 0.0 =>
      dd1 = 0.0 - dd & da1 = da) &
   (theta1 >= 180.0 & theta1 < 270.0 & bDE> 0.0 =>
     da1 = 0.0 - da & dd1 = 0.0 - dd) &
   (theta1 >= 180.0 & theta1< 270.0 & bDE<= 0.0 =>
     da1 = 0.0 - da & dd1 = dd) &
   (theta1 >= 270.0 & bDE < 0.0 =>
     da1 = 0.0 - da & dd1 = 0.0 - dd) &
   (theta1 >= 270.0 & bDE >= 0.0 =>
     da1 = 0.0 - da & dd1 = dd)
  THEN
   ra, de := bRA + da1, bDE + dd1
  END
END
```

Листинг 3. Пример определения семантики функции преобразования данных

Предусловием (PRE) операции является конъюнкция правильной типизации аргументов. Тело операции составляет подстановка неограниченного выбора ANY, в которой определяются все используемые в функции вспомогательные переменные (*theta*, *th*, *dd*, *da*), задаются ограничения на их значения.

Так, если значение переменной в функции задается одним оператором присваивания (Листинг 2), например,

```
th=theta*math.pi/180.0
```

то его семантика выражается конъюнкцией предикатов типизации и равенства (Листинг 3):

```
th: REAL & th = theta * pi/180.0
```

Если значение переменной в функции задается несколькими операторами присваивания (как, например, в случае переменной *theta* в Листинге 2), то для каждого оператора присваивания в подстановке ANY определяется отдельная переменная (например, *theta* и *theta1*) и отдельный предикат равенства (Листинг 3).

Семантика условного оператора *if* выражается здесь логическими импликациями. Так, например, семантика условного оператора (Листинг 2)

```
if theta<0:
  theta=360.0+theta
```

выражается конъюнкцией импликаций (Листинг 3):

```
(theta < 0.0 => theta1 = 360.0 + theta) &
(theta >= 0.0 => theta1 = theta)
```

C. Особенности определения семантики библиотечных строковых функций

Семантика библиотечных строковых функций выражается в AMN при помощи операций отдельной спецификации *StringType* вида MACHINE.

При определении семантики библиотечных строковых функций, также, как и при определении семантики библиотечных математических функций, можно абстрагироваться от конкретных реализаций этих функций и определить их семантику с использованием тех видов подстановок, которые могут использоваться в спецификациях вида MACHINE [9] (в частности, предусловия PRE, условной подстановки IF-THEN-ELSE, подстановки неограниченного выбора ANY).

Ниже определение семантики библиотечных строковых функций в AMN будет иллюстрироваться на примере ограниченного варианта функции *split*, используемого в программах интеграции данных в базе данных двойных звезд (Листинг 4). Фактически, используется *бинарное разбиение* строк по разделителю. В случае, если разбиение не является бинарным, выбрасывается ошибка.

```
res=str.split(char)
if len(res)!=2:
  raise_error
```

Листинг 4. Фрагмент программы интеграции данных с использованием библиотечной строковой функции

Семантика функции бинарного разбиения строки представляется в AMN операцией *binary _split* (Листинг 5).

```
res <-- binary_split(str, char) =
PRE str: seq(CHAR) & char: CHAR &
  res: seq(seq(CHAR)) & str: FIN(str) &
  ( size(str) = 0 or size(str) > 0 &
   card({ind | ind: INT & ind >= 1 &
     ind <= size(str) & str(ind) = char }) <= 1 )
THEN
  IF card({ind | ind: INT & ind >= 1 &
      ind <= size(str) & str(ind) = char }) = 0
  THEN
   res:= [str]
```

```
  ELSE
   ANY ind WHERE
     ind: INT & ind >= 1 & ind <= size(str) &
     str(ind) = char
   THEN
     res:= [str /|\ (ind - 1), str \|/ ind]
   END
  END
END
```

Листинг 5. Пример определения семантики строковой функции

Строки типизируются как последовательности (seq) символов (CHAR).

Семантика выбрасывания ошибки по условию выражается в AMN с использованием добавления соответствующего предиката в предусловие операции. Так, условие небинарного разбиения строки (Листинг 4)

```
len(res)!=2
```

эквивалентно вхождению в строку разделяющего символа в количестве большем 1. В этом случае к предусловию конъюнктивно добавляется предикат, утверждающий обратное (Листинг 5):

```
card({ind | ind: INT & ind >= 1 &
  ind <= size(str) & str(ind) = char }) <= 1
```

Основное тело операции составляет условная подстановка IF с двумя ветвями. В первой ветви при условии, что разделительный символ в строке не встречается, операция возвращает последовательность [*str*] из одного элемента (входной строки *str*). Во второй ветви, отвечающей условию, что разделительный символ встречается ровно один раз, операция возвращает последовательность из двух подстрок, разделяемых разделяющим символом во входной строке. При этом операция *str* /|\ *n* обозначает операцию обрезания хвоста последовательности, начиная с индекса *n*, а операция *str* \|/ *m* обозначает обрезание *m* первых элементов последовательности [9].

D. Особенности определения семантики оператора присваивания и вызова вспомогательных функций

В данном подразделе рассмотрены виды присваивания значений вспомогательным переменным, используемые в программах интеграции данных в базу данных двойных звезд.

Следует отметить, что семантика вспомогательных переменных выражается в AMN с использованием одноименных абстрактных переменных соответствующей спецификации. Переменные должны быть явно определены в разделе ABSTRACT_VARIABLES и явно типизированы в разделе INVARIANT (Листинг 6).

```
ABSTRACT_VARIABLES
  pmC, pmRA, compA, RA1, RA2, DE2
INVARIANT
  pmC: REAL &
  pmRA: REAL &
  compA: seq(CHAR)
  RA1: REAL &
  RA2: REAL &
  DE1: REAL &
  DE2: REAL &
  theta: REAL &
  rho: REAL
```

Листинг 6. Определение и типизация вспомогательных переменных

Семантика присваивания переменной значения встроенного типа, арифметического выражения или встроенной функции, семантика которой может быть выражена при помощи встроенной функции AMN (Листинг 7) выражается подстановкой «becomes equal to» (:=) (Листинг 8).

```
pmC=1.0
pmRA=pmC*fld13
compA=fs[0]
```

Листинг 7. Пример операторов присваивания

```
pmC:= 1.0
pmRA:= pmC*real(fld13)
compA:= fs(1)
```

Листинг 8. Семантика операторов присваивания

В качестве примера встроенной функции выше приведено обращение к элементу массива. Семантика массивов выражается в AMN последовательностями, нумерация элементов которых начинается с 1 (поэтому индексу массива 0 соответствует индекс последовательности 1).

Семантика присваивания переменной значения функции преобразования данных

```
RA1=hms2deg(float(fld20),float(fld21),fld22)
```

выражается подстановкой вызова операции (<--), выражающей семантику этой функции преобразования данных:

```
RA1 <-- hms2deg(real(fld20), real(fld21), fld22)
```

В случае, если в параметрах вызова функции (например, *rel2abs*) используется другая функция преобразования данных (например, *asec2deg*)

```
(RA2,DE2)=rel2abs(RA1,DE1,theta,asec2deg(rho))
```

для сохранения значения вложенного функционального терма используется подстановка локальной переменной VAR (Листинг 9).

```
VAR rho1 IN
  rho1 <-- asec2deg(rho);
  RA2, DE2 <-- rel2abs(RA1, DE1, theta, rho1)
END
```

Листинг 9. Семантика присваивания с использованием вложенного функционального терма

E. Особенности определения семантики последовательной композиции операторов

Семантика последовательной композиции операторов императивного языка может быть выражена в AMN следующими способами:

1) с использованием последовательной подстановки «;»;
2) с использованием параллельной подстановки «||»;
3) с использованием последовательного вызова операций AMN.

В случае, если набор операторов, соединяемых в последовательную композицию, достаточно *прост*, т.е. содержит не более двух вызовов функций, семантика которых выражается операциями AMN (Листинг 10), то его семантика может быть выражена как операцией с использованием последовательной подстановки

(Листинг 11), так и операцией с использованием параллельной подстановки (Листинг 12).

```
compA="A"
compB="B"
```

Листинг 10. Пример последовательной композиции операторов

```
pre_iteration_CompInit =
BEGIN
  compA:= [CHAR_A];
  compB:= [CHAR_B]
END
```

Листинг 11. Семантика последовательной композиции с использованием последовательной подстановки

```
pre_iteration_CompInit =
BEGIN
  compA:= [CHAR_A]
  ||
  compB:= [CHAR_B]
END
```

Листинг 12. Семантика последовательной композиции с использованием параллельной подстановки

Заметим, что последовательная композиция может быть реализована с использованием параллельной подстановки только в том случае, если порядок исполнения операторов не влияет на результат программы (например, если в наборе нет двух операторов, изменяющих одну и ту же переменную).

В случае, если набор операторов содержит более двух вызовов функций, семантика которых выражается операциями AMN (Листинг 13), то его семантика выражается с использованием последовательного вызова операций AMN (Листинг 14).

```
if fld18.find("B")>=0:
  band="blue"
if fld18.find("K")>=0:
  band="ir"
if fld18.find("R")>=0:
  band="red"
```

Листинг 13. Пример последовательности операторов с более чем двумя вызовами функций

```
pre_iteration_band_blue =
SELECT
  state = COMP_AB_UPDATE
THEN
  VAR nn IN
    nn <-- find(fld18, CHAR_B);
    IF nn >= 0 THEN
     spBand:= [CHAR_b, CHAR_l, CHAR_u, CHAR_e]
    END
  END;
  state:= FLD18_FIND_B
END;

pre_iteration_band_ir =
SELECT
  state = FLD18_FIND_B
THEN
  VAR nn IN
    nn <-- find(fld18(iteration_num), CHAR_K);
    IF nn >= 0 THEN
      spBand:= [CHAR_i, CHAR_r]
    END
  END;
  state:= FLD18_FIND_K
END;

pre_iteration_band_red =
SELECT
  state = FLD18_FIND_K
THEN
  VAR nn IN
    nn <-- find(fld18(iteration_num), CHAR_R);
    IF nn >= 0 THEN
      spBand:= [CHAR_r, CHAR_e, CHAR_d]
    END
  END;
  state:= FLD18_FIND_R
END;
```

Листинг 14. Пример определения семантики последовательности операторов с более чем двумя вызовами функций

В данном случае в программе интеграции производится вызов функции *find(str, char)*, возвращающей первую позицию, на которой в строке *str* находится символ *char* (Листинг 13).

Последовательная передача управления между операциями осуществляется при помощи переменной состояния *state*. Например, после исполнения первой операции *pre_iteration_band_blue* переменная *state* принимает значение FLD18_FIND_B. Вторая операция *pre_iteration_band_ir* может быть исполнена ровно в этом же случае, об этом говорит условие в секции SELECT (Листинг 14):

```
state = FLD18_FIND_B
```

Препятствием к объединению всех трех операций в одну и замену последовательного вызова операций на последовательную или параллельную подстановку являются их свойства в языке AMN. Дело в том, что сложность теорем корректности отдельной операции AMN экспоненциально растет от количества операций, определенных в других спецификациях, и вызываемых в ней. Сложность теоремы корректности для операции можно оценить при помощи количества автоматически генерируемых лемм (proof obligations). Так, для всех трех операций, приведенных выше, генерируется по 7 лемм. Если объединить две операции в одну, то для нее будет сгенерировано 25 лемм. Если же объединить все три операции в одну, то будет сгенерировано 85 лемм.

Отметим, что количество лемм не меняется от замены последовательной подстановки на параллельную. А поскольку применение параллельной подстановки связано с дополнительным анализом независимости результата от порядка исполнения операторов, то в простом случае достаточно использовать только последовательную подстановку.

F. Особенности определения семантики условного оператора

Семантика условного оператора с произвольным количеством ветвей, условия которых не содержат вызовов функций преобразования данных (Листинг 15) естественным образом выражается с использованием условной подстановки (Листинг 16).

```
if fld7 >=0 and fld9 >= 0:
  theta=fld7
  rho=fld9
  year=fld4
elif fld6 >=0 and fld8 >=0:
  theta=fld6
```

```
  rho=fld8
  year=fld3
else:
  theta=0
  rho=0
  year=fld[4]
```

Листинг 15. Пример условного оператора, не содержащего вызовы функций преобразования данных

```
pre_iteration_Angles =
BEGIN
  IF fld7 >= 0 & fld9 >= 0.0
  THEN
    theta:= real(fld7);
    rho:= fld9;
    year:= fld4
  ELSIF fld6 >= 0 & fld8 >= 0.0
  THEN
    theta:= real(fld6);
    rho:= fld8;
    year:= fld3
  ELSE
    theta:= 0.0;
    rho:= 0.0;
    year:= fld4
  END
END
```

Листинг 16. Пример семантики условного оператора, не содержащего вызовы функций преобразования данных

Если условие содержит небольшое число (1 или 2) вызовов функций преобразования данных, например

```
if rho>=0:
  (RA2,DE2)=rel2abs(RA1,DE1,theta,asec2deg(rho))
```

то семантика такого оператора также может быть выражена в одной операции при помощи условной подстановки IF и подстановки локальной переменной VAR (Листинг 17).

```
pre_iteration_rel2abs =
BEGIN
  IF rho >= 0.0 THEN
    VAR rho1 IN
      rho1 <-- asec2deg(rho);
      RA2, DE2 <-- rel2abs(RA1, DE1, theta, rho1)
    END
  END
END
```

Листинг 17. Пример семантики условного оператора, содержащего небольшое количество вызовов функций преобразования данных

Если же условие содержит большее количество вызовов функций преобразования данных (Листинг 18), то семантика такого условного оператора выражается с использованием последовательного вызова операций и вспомогательных переменных, используемых для хранения результатов вызова операций, выражающих семантику функций преобразования данных (Листинг 19).

```
if fld18.find("S")>=0 or
   fld18.find("U")>=0 or
   fld18.find("Y")>=0:
  otype="Opt"
```

Листинг 18. Пример условного оператора, содержащего более чем два вызова функций преобразования данных

```
pre_iteration_otype_S =
SELECT
  iteration_state = FLD18_FIND_P
THEN
  S_count <-- find(fld18, CHAR_S);
  iteration_state:= FLD18_FIND_S
END;

pre_iteration_otype_U =
SELECT
  iteration_state = FLD18_FIND_S
THEN
  U_count <-- find(fld18, CHAR_U);
  iteration_state:= FLD18_FIND_U
END;

pre_iteration_otype_Y =
SELECT
  iteration_state = FLD18_FIND_U
THEN
  Y_count <-- find(fld18, CHAR_Y);
  IF S_count >= 0 or U_count >= 0 or Y_count >= 0
  THEN
    otype:= [CHAR_O, CHAR_p, CHAR_t]
  END;
  iteration_state:= FLD18_FIND_Y
END;
```

Листинг 19. Пример семантики условного оператора, содержащего более чем два вызова функций преобразования данных

Семантика вызовов функций преобразования данных (в данном случае *find*) выражается при помощи отдельных операций AMN. Условная подстановка включается в последнюю из операций.

G. Особенности определения семантики циклов

В данном подразделе рассматривается два вида циклов: *while* и *for*, общий вид которых изображен на Листинге 20.

```
// while loop
iter = 1
while iter <= ITER_COUNT:
  data = transform_data(iter)
  iter += 1
// for loop
for rec in records
  data = transform_data(rec)
```

Листинг 20. Общий вид операторов циклов

Основная идея определения семантики циклов в AMN состоит в следующем. Цикл выражается двумя операциями *init* и *iterate*. Операция *init* переводит систему из состояния (задаваемого переменной *iteration_state*) LOOP_INITIAL в состояние LOOP_ITERATION. Операция *iterate* вызывает операцию трансформации данных, составляющую тело цикла, и изменяет счетчик цикла. Если условие продолжения цикла не выполняется, то операция переводит систему в состояние LOOP_COMPLETED.

Спецификация, выражающая семантику цикла *while* приведена в Листинге 21.

```
SETS
  ITERATION_STATE = {
    LOOP_INITIAL, LOOP_ITERATION, LOOP_COMPLETED }

ABSTRACT_VARIABLES
  iteration_state,
  iter,
  data
```

```
INVARIANT
  iteration_state: ITERATION_STATE &
  iter: INT &
  data: DATA &
  records: FIN(DATA) &
  recs: FIN(DATA)

OPERATIONS
init_while =
SELECT
  iteration_state = LOOP_INITIAL
THEN
  iter:= 1;
  iteration_state:= LOOP_ITERATION
END;

iterate_while =
SELECT
  iteration_state = LOOP_ITERATION
THEN
  IF iter <= ITER_COUNT THEN
    data <-- transform_data_for(iter);
    iter:= iter + 1
  ELSE
    iteration_state:= LOOP_COMPLETED
  END
END;
```

Листинг 21. Семантика цикла *while*

Спецификация, выражающая семантику цикла *for* приведена в Листинге 22.

```
ABSTRACT_VARIABLES
  records,
  recs

INVARIANT
  records: FIN(DATA) &
  recs: FIN(DATA)

OPERATIONS
init_for =
SELECT
  iteration_state = LOOP_INITIAL
THEN
  recs:= records;
  iteration_state:= LOOP_ITERATION
END;

iterate_for =
SELECT
  iteration_state = LOOP_ITERATION
THEN
  IF recs /= {} THEN
    ANY rcrd WHERE rcrd: recs
    THEN
      data <-- transform_data_while(rcrd);
      recs:= recs - {rcrd}
    END
  ELSE
    iteration_state:= LOOP_COMPLETED
  END
END
```

Листинг 22. Семантика цикла *for*

III. Применение формальной семантики для верификации программ интеграции данных

Общие принципы применения формальной семантики императивного языка программирования для верификации программ интеграции данных состоят в следующем.

Для программы интеграции данных, подлежащей верификации, определяется ее формальная семантика в языке AMN в соответствии с правилами, изложенными в предыдущем разделе. При этом для каждого библиотечного модуля и каждого модуля функций преобразования данных создается отдельная спецификация вида MACHINE. Для выражения семантики основной программы интеграции создается спецификация вида REFINEMENT. Свойства корректности программы интеграции данных, подлежащие верификации, определяются экспертом вручную в виде формул, конъюнктивно присоединяемых в раздел INVARIANT спецификации (примеры этих свойств приведены, в частности, в работах [16] [17]). Спецификации загружаются в проект инструментария Atelier B [11], проводится проверка из синтаксической корректности, проверка соответствия типов, автоматически генерируются теоремы корректности спецификаций. Для доказательства теорем применяются средства автоматического и интерактивного доказательства. Примеры статистики доказательства теорем для конкретных программ интеграции данных приведены в работах [16] [17]. В случае, если все теоремы удалось доказать, интеграция данных считается корректной. Если же какие-то из теорем не удалось доказать, производится выяснение причин этого, то есть выявление ошибок в программах интеграции данных. Затем процесс определения семантики и доказательства корректности повторяется итеративно для исправленных программ интеграции данных.

IV. Заключение

В статье выделены особенности и суммирован опыт по определению формальной семантики императивных языков для верификации программ интеграции данных в базе данных двойных звезд, в интегрированной системе баз данных по свойствам неорганических веществ и материалов, в системе «Умное землепользование». Денотационная семантика подмножества конструкций императивного языка, используемого для создания программ интеграции данных, определена с использованием формального языка AMN, основанного на логике первого порядка и теории множеств. Такое определение семантики позволяет применять для доказательства корректности программ интеграции данных технологию и автоматизированные средства доказательства Atelier B, имеющие более чем двадцатилетний мировой опыт применения при разработке промышленных программных систем.

Благодарности

Автор выражает благодарность научному сотруднику ИНАСАН Павлу Кайгородову за содержательные консультации по текстам программ интеграции данных в базу данных двойных звезд.

Работа выполнена в рамках НИР 121041900216-9 «Конвергентные когнитивно-информационные технологии, интеллектуальные инструменты, сервисы и

ресурсы безопасных распределенных информационно-вычислительных инфраструктур в науке, образовании, социуме».

Библиография

[1] M. Masmoudi, S. Ben Abdallah Ben Lamine, M. H. Karray, B. Archimede, and H. Baazaoui Zghal, “Semantic data integration and querying: A survey and challenges,” ACM Comput. Surv., vol. 56, no. 8, pp. 1–35, 2024.

[2] D. Kovaleva, P. Kaygorodov, O. Malkov, B. Debray, and E. Oblak, “Binary star DataBase BDB development: Structure, algorithms, and VO standards implementation,” Astron. Comput., vol. 11, pp. 119–125, 2015.

[3] P. Kaygorodov, N. Skvortsov, D. Kovaleva, and O. Malkov, “A new version of the binary star database BDB: Challenges and directions,” Open Astron., vol. 32, no. 1, 2023.

[4] The Binary Star Database. 2025. URL: https://bdb.inasan.ru/

[5] Интегрированная система баз данных по свойствам неорганических веществ и материалов. ИМЕТ РАН. 2025. URL: https://imet-db.ru/

[6] N. N. Kiselyova, V. A. Dudarev, and A. V. Stolyarenko, “Integrated system of databases on the properties of inorganic substances and materials,” High Temp., vol. 54, no. 2, pp. 215–222, 2016.

[7] Крупный научный проект фундаментальных исследований “Актуальные научные задачи стратегии адаптации потенциала землепользования России в современных условиях беспрецедентных вызовов (экономический кризис, изменения климата, кризис глобальных тенденций природопользования)”. – Почвенный институт имени В.В. Докучаева. 2023. URL: https://www.esoil.ru/activities/projects_programs/minobr/knp_2020/

[8] V. Nundloll, R. Lamb, B. Hankin, and G. Blair, “A semantic approach to enable data integration for the domain of flood risk management,” Environmental Challenges, vol. 3, no. 100064, p. 100064, 2021.

[9] J.-R. Abrial, The B-book: Assigning programs to meanings. Cambridge, England: Cambridge University Press, 2011.

[10] B Language Reference Manual. Version 1.8.10. ClearSy, 2025.

[11] Atelier B, the industrial tool to efficiently deploy the B Method. 2025. URL: http://www.atelierb.eu/

[12] N. White, S. Matthews, and R. Chapman, “Formal verification: will the seedling ever flower?,” Philos. Trans. A Math. Phys. Eng. Sci., vol. 375, no. 2104, p. 20150402, 2017.

[13] M. Butler et al., “The first twenty-five years of industrial use of the B-method,” in Lecture Notes in Computer Science, Cham: Springer International Publishing, 2020, pp. 189–209.

[14] S. Stupnikov, “Semantics and verification of entity resolution and data fusion operations via transformation into a formal notation,” in Communications in Computer and Information Science, Cham: Springer International Publishing, 2017, pp. 145–162.

[15] S. Stupnikov, “Rule-based specification and implementation of multimodel data integration,” in Communications in Computer and Information Science 822, Cham: Springer International Publishing, 2018, pp. 198–212.

[16] S.A. Stupnikov, “Formal Semantics and Verification of Procedural SQL Programs Implementing Materialized Data Integration,” Lobachevskii J Math, 2025. In print.

[17] С.А. Ступников, “Верификация интеграции данных в интегрированной системе баз данных по свойствам неорганических веществ и материалов,” Ученые записки Казанского университета. Серия Физико-математические науки, 167(2), 2025. В печати.

[18] P. D. Mosses, “Formal Semantics of Programming Languages: An Overview,” Electronic Notes in Theoretical Computer Science, 148(1), pp. 41-73, 2006.

[19] D. Guth, “A formal semantics of Python 3.3,” Doctoral dissertation, University of Illinois at Urbana-Champaign, 2013.

[20] M. A. Köhl, “An executable structural operational formal semantics for Python,” Master Thesis, Saarland University, 2020. URL: https://embedded.cs.uni-saarland.de/publications/theses/thesis_cs_msc_Koehl_Maximilian.pdf

On the Features of a Formal Semantics of an Imperative Language for the Data Integration Program Verification

Sergey Stupnikov

***Abstract*—Increasing number of heterogeneous data sources in science and industry leads to the need for their integration. Specialized data integration systems are being developed in various subject domains. Due to the complexity of data integration programs, formal verification of their correctness becomes an important issue. The correctness of a data integration program is expert-defined property binding the state of a set of data sources and the state of the target integrating database, expressed as a logical formula. The formal verification process can be difficult, but the cost of correcting an error after the system is released into production exceeds the cost of correcting an error at the system development stage by tens or hundreds of times. In practice, data integration programs are often implemented using imperative languages. The paper highlights the features and summarizes the experience in definition the formal semantics of imperative languages for data integration programs verification in the domains of astronomy, materials science, and land use management. The features of defining the semantics of library mathematical and string functions, data transformation functions, assignment statement and auxiliary function invocation, sequential composition of statements, conditional statement, and loop statements are considered. The general principles of application of the formal semantics of imperative programming languages for data integration programs verification using automated proof tools are presented.**

***Keywords*—formal semantics of programs, program verification, data integration.**

REFERENCES

[1] M. Masmoudi, S. Ben Abdallah Ben Lamine, M. H. Karray, B. Archimede, and H. Baazaoui Zghal, "Semantic data integration and querying: A survey and challenges," ACM Comput. Surv., vol. 56, no. 8, pp. 1–35, 2024.

[2] D. Kovaleva, P. Kaygorodov, O. Malkov, B. Debray, and E. Oblak, "Binary star DataBase BDB development: Structure, algorithms, and VO standards implementation," Astron. Comput., vol. 11, pp. 119–125, 2015.

[3] P. Kaygorodov, N. Skvortsov, D. Kovaleva, and O. Malkov, "A new version of the binary star database BDB: Challenges and directions," Open Astron., vol. 32, no. 1, 2023.

[4] The Binary Star Database. 2025. URL: https://bdb.inasan.ru/

[5] Integrated system of databases on the properties of inorganic substances and materials. Baikov Institute of Metallurgy and Materials Science, Russian Academy of Science. 2025. URL: https://imet-db.ru/

[6] N. N. Kiselyova, V. A. Dudarev, and A. V. Stolyarenko, "Integrated system of databases on the properties of inorganic substances and materials," High Temp., vol. 54, no. 2, pp. 215–222, 2016.

[7] A major scientific project "Current scientific objectives of the strategy for adapting Russia's land use potential in modern conditions of unprecedented challenges (economic crisis, climate change, crisis of global environmental trends)". – V.V. Dokuchaev Soil Science Institute, 2023. URL: https://www.esoil.ru/activities/projects_programs/minobr/knp_2020/

[8] V. Nundloll, R. Lamb, B. Hankin, and G. Blair, "A semantic approach to enable data integration for the domain of flood risk management," Environmental Challenges, vol. 3, no. 100064, p. 100064, 2021.

[9] J.-R. Abrial, The B-book: Assigning programs to meanings. Cambridge, England: Cambridge University Press, 2011.

[10] B Language Reference Manual. Version 1.8.10. ClearSy, 2025.

[11] Atelier B, the industrial tool to efficiently deploy the B Method. 2025. URL: http://www.atelierb.eu/

[12] N. White, S. Matthews, and R. Chapman, "Formal verification: will the seedling ever flower?," Philos. Trans. A Math. Phys. Eng. Sci., vol. 375, no. 2104, p. 20150402, 2017.

[13] M. Butler et al., "The first twenty-five years of industrial use of the B-method," in Lecture Notes in Computer Science, Cham: Springer International Publishing, 2020, pp. 189–209.

[14] S. Stupnikov, "Semantics and verification of entity resolution and data fusion operations via transformation into a formal notation," in Communications in Computer and Information Science, Cham: Springer International Publishing, 2017, pp. 145–162.

[15] S. Stupnikov, "Rule-based specification and implementation of multimodel data integration," in Communications in Computer and Information Science 822, Cham: Springer International Publishing, 2018, pp. 198–212.

[16] S.A. Stupnikov, "Formal Semantics and Verification of Procedural SQL Programs Implementing Materialized Data Integration," Lobachevskii J Math, 2025. In print.

[17] S.A. Stupnikov, "Verification of data integration in the integrated system of databases on the properties of inorganic substances and materials," Uchenye Zapiski Kazanskogo Universiteta. Seriya Fiziko-Matematicheskie Nauki, 167(2), 2025. In print.

[18] P. D. Mosses, "Formal Semantics of Programming Languages: An Overview," Electronic Notes in Theoretical Computer Science, 148(1), pp. 41-73, 2006.

[19] D. Guth, "A formal semantics of Python 3.3," Doctoral dissertation, University of Illinois at Urbana-Champaign, 2013.

[20] M. A. Köhl, "An executable structural operational formal semantics for Python," Master Thesis, Saarland University, 2020. URL: https://embedded.cs.uni-saarland.de/publications/theses/thesis_cs_msc_Koehl_Maximilian.pdf

Stupnikov S.A., Ph. D., Research Scientist, Lomonosov Moscow State University and Federal Research Center "Computer Science and Control" of the Russian Academy of Sciences, Moscow, Russia (e-mail: sstupnikov@ipiran.ru)