УДК 519.681.3

ОНТОЛОГИЧЕСКИЕ СИСТЕМЫ ПЕРЕХОДОВ И ИХ ПРИМЕНЕНИЕ К СЕМАНТИКЕ КОМПЬЮТЕРНЫХ ЯЗЫКОВ

И.С. Ануреев

Институт систем информатики имени А.П. Ершова, г. Новосибирск
E-mail: anureev@iis.nsk.su

Операционно-онтологический подход к формальной спецификации компьютерных языков был разработан автором как методология создания операционной семантики компьютерного языка на базе его онтологии – так называемой операционно-онтологической семантики. В статье предлагается формализм для описания операционно-онтологических семантик компьютерных языков – онтологические системы переходов, и проводится сравнение онтологического описания компьютерных языков с традиционным грамматическим описанием.

Ключевые слова:
Спецификация, операционно-онтологическая семантика, компьютерные языки, онтологии, онтологические системы переходов.

Key words:
Specification, operational ontological semantics, computer languages, ontologies, ontological transition systems.

Введение

Традиционный подход к формальной спецификации компьютерных языков (КЯ) обычно базируется на операционной семантике языка, описываемой в терминах помеченной системы переходов, в которой применимость правил перехода (правил операционной семантики) к некоторой конструкции языка определяется посредством синтаксического сопоставления с образцом на уровне грамматики этого языка.

Ю. Гуревичем был предложен логико-алгебраический подход к решению этой задачи, основанный на машинах абстрактных состояний (abstract state machines) [1]. Машины абстрактных состояний – специальный вид систем переходов, состояниями в которых являются алгебраические системы. Выбор подходящей сигнатуры алгебраической системы позволяет приблизить формальную спецификацию КЯ к его естественной концептуальной структуре. Примеры приложений этого формализма могут быть найдены в [2]. Этот подход реализован в языках ASML [3] и XASM [4].

Операционно-онтологический подход к формальной спецификации КЯ был предложен автором в [5] как методология создания операционной семантики КЯ на базе его онтологии – так называемой операционно-онтологической семантики. В отличие от обычной операционной семантики, которая не накладывает никаких ограничений на состояния, в операционно-онтологической семантике состояния определяются как онтологические модели (множества экземпляров понятий онтологии) КЯ.

В статье для описания операционно-онтологической семантики КЯ предлагается специальный вид помеченных систем переходов – онтологические системы переходов (ОСП), в которых онтологические модели определяются как алгебраические системы специального вида. Использование алгебраических систем в качестве состояний позволяет также рассматривать ОСП как специальный случай машин абстрактных состояний.

Соглашения и обозначения

Будем использовать для представления выражений, обозначающих данные или действия, списковую нотацию $(A_1...A_n)$, в которой подвыражения A_i разделены пробелами. Подвыражение A_i, которое не является списком, будем называть атомом. Например, применение функции f к аргументам $v_1,...,v_n$ запишется в этой нотации как $(f\ v_1...v_n)$, а кортеж из элементов $v_1,...,v_n$ как $(v_1...v_n)$.

Пусть *bool* обозначает множество {*true*, *false*}, а *undef* – отсутствие значения у некоторого выражения.

Атрибутной структурой *as* называется конечное множество пар $\{(a_1v_1),..., (a_nv_n)\}$ таких, что $(a_i{\neq}a_j)$ при $(i{\neq}j)$. Объект a_i называется атрибутом структуры *as*, а объект v_i – значением атрибута a_i структуры *as*. Атрибутная структура *as* имеет упрощенную запись $(a_1{:}v_1,..., a_n{:}v_n)$. На атрибутных структурах определены операции доступа (.) и модификации (*upd*): $(as.a_i{=}v_i)$ и $((upd\ as\ a\ v){=}aas)$, где атрибутная структура *aas* может отличаться от *as* только значением атрибута *a* и $(aas.a{=}v)$. Эти операции расширяются на случаи последовательных и одновременных доступов и модификации с помощью модификаторов #*s* и #*p*, соответственно:

- $(a.(\#s\ b_1b_2...b_n){=}a.b_1.(\#s\ b_2,...,b_n))$;
- $(a.(\#s){=}a)$;
- $(a.(\#p\ b_1...b_n){=}(a.b_1...a.b_n))$;
- $((upd\ a\ (\#s\ b_1...b_n)\ e){=}(upd\ a.b_1\ (\#s\ b_2,...,b_n)))$;
- $((upd\ a\ (\#s)\ e){=}a)$;
- $((upd\ a\ (\#p\ b_1...b_n)\ (\#p\ e_1...e_n)){=}(upd\ (upd\ a\ b_1\ e_1)\ (\#p\ b_2...b_n)\ (\#p\ e_2...e_n)))$;
- $((upd\ a\ (\#p){=}(\#p))\ a)$.

Операционная семантика компьютерных языков

Операционная семантика КЯ обычно определяется как помеченная система переходов.

Помеченная система переходов *lts* – это тройка (*sts labs tr*), где *sts* и *labs* – множества, элементы которых называются состояниями и метками, соответственно, функция $tr{\in}(sts{\times}labs{\times}sts){\to}bool$ назы-

вается функцией перехода. Говорят, что из состояния *s* можно перейти в состояние *ss* по метке *lab*, если (*tr s lab ss*).

Пусть *lan* – некоторый КЯ. Конструкции (операторы, выражения, классы, интерфейсы и т. д.) компьютерного языка *lan* будем называть *lan*-выражениями. Пусть (*alph lan*) – обозначение для алфавита языка *lan*. Пусть *exps* обозначает множество конечных последовательностей *lan*-выражений.

При описании операционной семантики *lan* метку *lab* можно представить атрибутной структурой с обязательными атрибутами *in* и *out* такими, что *lab.in*∈*exps* и *lab.out*∈*exps*. Говорят, что выполнение *lab.in* в *s* сводится к выполнению *lab.out* в *ss*, если (*tr s lab ss*)=*true*. Случай, когда *tr* определяется так, что результирующая последовательность всегда пуста, называется операционной семантикой входа–выхода.

В зависимости от выбранного описания операционной семантики языка *lan* метка *lab* может иметь те или иные дополнительные атрибуты. Например, дополнительным атрибутом может быть атрибут, значением которого является значение, возвращаемое при выполнении *lab.in*.

Конфигурацией называется кортеж (*e s*), где *e*∈*exps* и *s*∈*sts*. Говорят, что *lts* переходит из конфигурации (*e s*) в конфигурацию (*ee ss*), если (*tr s lab ss*)=*true* для некоторой метки *lab* такой, что *lab.in*=*e* и *lab.out*=*ee*.

Отношение перехода *tr* определяется как объединение отношений перехода (*tr es*) для отдельных классов *es* последовательностей выражений *lab.in*. Каждое отношение (*tr es*) определяется множеством правил перехода (правил операционной семантики). Правило перехода *r* является атрибутной структурой с атрибутами *cond* и *res*, определяющими условие и результат применения правила *r*, соответственно.

Условие применения *r.cond* правила *r* определяет *es* с помощью логической функции *match* следующим образом: *e*∈*es* в состоянии *s* тогда и только тогда, когда существует *mr* такой, что (*match e r.cond mr s*)=*true*. Объект *mr* называется результатом сопоставления *e* с правилом *r* и описывает информацию, полученную при этом сопоставлении.

Множество результирующих последовательностей выражений *ees* для исходной последовательности *e* в состоянии *s* определяется с помощью логической функции *transform* следующим образом: *ee*∈*ees* в *s* тогда и только тогда, когда (*transform e ee r.res mr s ss*)=*true* для некоторого состояния *ss*∈*sts*. Таким образом, *r.res* содержит информацию о том, что получается из *e*, а функция *transform* на основе этой информации выделяет класс *ees* результирующих выражений, к выполнению которых сводится выполнение *e*.

Функция *match* в типичных операционных семантиках для КЯ выполняет сопоставление с образцом либо на уровне строки исходной последовательности выражений, либо на уровне последовательности синтаксических деревьев этих выражений. В дальнейшем будем представлять деревья в виде термов.

В первом случае *r.cond* определяется как атрибутная структура с атрибутами *samp* и *var*, где *r.cond.samp* – последовательность строк символов алфавита (*union* (*alph lan*) *vs*), называемая образцом правила *r*; *r.cond.var* – атрибутная структура, атрибутами которой являются элементы, называемые переменными образца *r.cond.samp*, а значениями этих атрибутов – элементы множества {*str*, *seq*}, называемые типами переменных образца; *vs* – множество переменных образца. Результат сопоставления *mr* есть подстановка на множестве переменных *r.cond.var* (функция, сопоставляющая переменным образца их значения). Значениями переменных образца типа *str* и *seq* являются строки и последовательности строк, соответственно. Функция *match* определяется следующим образом: (*match e r.cond mr s*)=*true* тогда и только тогда, когда (*str-subst r.cond.samp mr*)=*e*, где *str-subst* – функция, выполняющая подстановку на строках (заменяющая переменные образца в каждой из строк последовательности *r.cond.samp* на их значения).

Во втором случае *r.cond* определяется как атрибутная структура с атрибутами *samp* и *var*, где *r.cond.samp* – последовательность термов, называемая образцом правила *r*; *r.cond.var* – атрибутная структура, атрибутами которой являются элементы, называемые переменными образца *r.samp*, а значениями этих атрибутов – элементы множества {*term*, *seq*}, называемые типами переменных образца. Результат сопоставления *mr* есть подстановка на множестве переменных *r.cond.var*. Значениями переменных образца типа *term* и *seq* являются термы и последовательности термов, соответственно. Функция *match* определяется следующим образом: (*match e r.cond mr s*)=*true* тогда и только тогда, когда (*subst r.cond.samp mr*)=*e*, где *subst* – функция, выполняющая подстановку на термах (заменяющая переменные образца в каждом из термов последовательности *r.cond.samp* на значения соответствующих переменных образца).

Рассмотрим в качестве примера правило операционной семантики для оператора присваивания (*x*:=*u*), где *x* – целочисленная переменная, *u* – целое число. Пусть ((x_1 v_1)... (x_n v_n)) обозначает подстановку *sub* на переменных x_i такую, что (*sub* x_i)=v_i. Пусть состояниями являются атрибутные структуры, где атрибуты – целочисленные переменные, а значения атрибутов – значения этих переменных. Тогда правило перехода *r* для оператора присваивания имеет вид (*cond:*(*samp:*(*a*:=*b*) *c*, *var:*(*a: term*, *b: term*, *c: seq*)), *res:*(*upd s a b*)), а функции *match* и *transform* принимают для этого правила следующий вид:

- (*match e r.cond mr s*)=*true* тогда и только тогда, когда *e* имеет вид (*set aa bb*) *cc* и *mr*=((*a aa*) (*b bb*) (*c cc*));
- (*transform e ee r.res mr s ss*)=*true* тогда и только тогда, когда *ss*=(*upd s aa bb*) и *ee*=*cc*.

Для современных КЯ помимо выражений, описывающих инструкции КЯ, требуется определить операционную семантику выражений, описывающих концепции КЯ, такие как, например, просачивание исключений, разрешение перегрузки операций, взаимодействие приложения с операционным окружением, нахождение динамического типа объекта в объектно-ориентированных КЯ и т. д.

При типичном грамматическом определении операционной семантики КЯ мы получаем громоздкие и трудные для понимания спецификации для современных концептуально-сложных КЯ. Естественный способ преодолеть эту сложность состоит в переходе от описаний, ориентированных на грамматику КЯ, к концептуальным описаниям, использующим онтологию КЯ. В качестве формализма, обеспечивающего концептуальные описания, предлагается использовать специальный вид помеченных систем переходов – онтологические системы переходов.

Онтологические системы переходов

Состояниями в ОСП являются алгебраические системы специального вида. Пусть *alph* – некоторое множество символов, не содержащее символа _. Состояния над *alph* определяются как атрибутные структуры с атрибутами *sig*, *val*, *int* и *arg*. Пусть *A* – алгебраическая структура. Множество *A.sig* называется сигнатурой *A*, а элементы *A.sig* – составными символами *A*. Составной символ определяется как список, элементы которого принадлежат $alph \cup \{\_\}$, а *i*-е вхождение символа _ в составной символ называется *i*-м аргументом этого составного символа. Число аргументов в составном символе называется его местностью. Множество *A.val* называется носителем *A*. Функция *A.int*, определенная на *A.sig*, называется интерпретацией символов *A*. Для нее выполнено свойство: если $B \in A.sig$ и n – местность B, то $(A.int\ B) \in A.val^n \to A.val$. Например, для операции равенства соответствующий составной символ будет иметь вид (_=_). Функция *A.arg*, определенная на *A.sig*, называется аргументной интерпретацией символов *A*. Для нее выполнено следующее свойство: если $B \in A.sig$ и n – местность B, то (*A.arg B*) – список длины n, и любой элемент этого списка принадлежит множеству {*val*, *itself*}.

Говорят, что выражение *E* является примером составного символа *B* местности *n* относительно списка $(v_1 \ldots v_n)$, если *E* получается из *B* последовательной заменой вхождений символа _ слева направо на $v_1, \ldots, v_n$, соответственно.

Значение (*val E A*) выражения в алгебраической системе A определяется следующим образом:

- если *E* – атом, то (*val E A*)=*E*;
- если *E* – список, *E* – пример $B \in A.sig$ местности *n* относительно $(v_1 \ldots v_n)$, то $(val\ E\ A) = A.int\ (vv_1 \ldots vv_n)$;
- если *E* – список и *E* – пример более одного символа из *A.sig*, или *E* не является примером ни одного символа из *A.sig*, то (*val E A*)=*undef*.

Объекты vv_i в вышеприведенном определении задаются следующим образом. Если *val* – *i*-й элемент списка (*A.arg B*), то $vv_i = (val\ v_i\ A)$. Если *itself* – *i*-й элемент списка (*A.arg B*), то $vv_i = v_i$.

Онтологической сигнатурой *o-sig* будем называть множество составных символов над *alph*, специфицирующее элементы онтологии (понятия, атрибуты и т. п.). Будем называть элементы множества *o-sig* онтологическими символами.

Рассмотрим пример онтологической сигнатуры, одновременно последовательно помечая аргументы буквами *a*, *b*, *c* и определяя неформальный смысл символов, входящих в нее:

- (*a is concept*) означает, что *a* – понятие;
- (*a is attribute of b*) означает, что *a* – атрибут понятия *b*;
- (*a is attribute of b of type c*) означает, что *a* – атрибут понятия *b* типа *c*.

Сигнатурой экземпляризации *i-sig* будем называть множество составных символов над *alph*, специфицирующее связи элементов онтологии с экземплярами. Будем называть элементы множества *i-sig* символами экземпляризации.

Рассмотрим пример сигнатуры экземпляризации, одновременно последовательно помечая аргументы буквами *a*, *b*, *c* и определяя неформальный смысл символов, входящих в нее:

- (*a is b*) означает, что *a* – экземпляр понятия *b*;
- (*value of attribute a of b*) означает значение атрибута *a* экземпляра *b* некоторого понятия. Для этого составного символа также используется упрощенная запись *b.a*;
- (*value of attribute a of instance b of c*) означает значение атрибута *a* экземпляра *b* понятия *c*. Этот символ используется, чтобы разрешить конфликт в случае, когда *b* является экземпляром нескольких понятий, имеющих атрибут *a*.

Пусть *val* – некоторое множество элементов, называемое носителем. Оно описывает элементы, над которыми определяется онтологическая структура.

Пусть *c-sig* – сигнатура над *alph*, называемая константной сигнатурой. Она включает символы, семантика (интерпретация) которых не меняется при переходе из состояния в состояние в ОСП. Пусть *c-int* – (постоянная) интерпретация этих символов.

Пусть *h-sig* – бесконечная сигнатура составных символов вида (*f*), где $f \in alph$, называемая сигнатурой истории перехода. Она включает символы, семантика (интерпретация) которых в конкретном состоянии *s* содержит информацию о состояниях, предшествующих *s* относительно *tr*, т. е. историю перехода в *s*.

Пусть *sig*=(*o-sig i-sig c-sig h-sig*), где сигнатуры *o-sig*, *i-sig*, *c-sig* и *h-sig* попарно не пересекаются.

Онтологическая система переходов относительно (*alph sig val c-int*) – это помеченная система переходов (*sts labs tr*), где *sts* – множество состояний над *alph* с носителем *val*, такое, что

- $s.sig \setminus (o\text{-}sig \cup i\text{-}sig \cup c\text{-}sig)$ конечно для любого $s \in sts$;
- $(s.sig \setminus (o\text{-}sig \cup i\text{-}sig \cup c\text{-}sig)) \subseteq h\text{-}sig$ для любого $s \in sts$;
- $(s.int\ f) = c\text{-}int$ для любых $s \in sts$ и $f \in c\text{-}sig$;

- последовательности *lab.in* и *lab.out* состоят из экземпляров понятий и метавыражений. Метавыражения позволяют напрямую оперировать состояниями.
- если (*tr s lab ss*), то (*s.sig*∩*h-sig*)⊆(*ss.sig*∩*h-sig*).

Метавыражения бывают двух видов и имеют следующую семантику:

- если *lab.in* имеет вид ((*a*$_1$...*a*$_n$)::=*b*) *eee*, то (*tr s lab ss*)=*true* тогда и только тогда, когда *lab.out*=*eee*, *ss*=(*upd s* (#*s int* ((*val aa*$_1$ *s.int*)... (*val aa*$_n$ *s.int*))) (*val b s.int*)). Метавыражение такого вида называется метаприсваиванием;
- если *lab.in* имеет вид (*assume a*) *eee*, то (*tr s lab ss*)=*true* тогда и только тогда, когда *lab.out*=*eee*, *ss*=*s* и (*val a s.int*)=*true*. Метавыражение такого вида называется метаусловием продолжения.

Часто используемые метаприсваивания (*a*::=*true*) и (*a*::=*undef*) имеют упрощенный синтаксис (*a*::=*t*) и (*a*::=*u*), соответственно.

ОСП сохраняет онтологию, если для любых *s*∈*sts*, *ss*∈*sts* и *lab*∈*labs* из (*tr s lab ss*)=*true* следует (*s.int f*)=(*ss.int f*) для любого символа *f*∈*o-sig*.

Операционно-онтологическая семантика компьютерных языков

Операционно-онтологическая семантика (ООС) КЯ является видом операционной семантики, при котором в качестве помеченных систем переходов используются ОСП.

Формально ООС языка *lan* есть кортеж (*o-ots e-ots alph sig val c-int*), где *o-ots* – ОСП относительно (*alph sig val c-int*), определяющая онтологию *lan*, *e-ots* – ОСП относительно (*alph sig val c-int*), сохраняющая онтологию и определяющая семантику *lan*-выражений.

Исходная последовательность *e* в конфигурации (*e s*) состоит из экземпляров понятий онтологии языка *lan* и метавыражений. Семантика метавыражений определена выше, а семантика экземпляров понятий задается с помощью правил.

Пусть *eee* – последовательность выражений.

Правило ООС *r* для экземпляров понятий имеет вид: (*if a hvar b then c*), где *a* – *lan*-выражение, называемое условием применимости правила *r*; *b* – последовательность промежуточных переменных, разделенных запятыми; *c* – последовательность *lan*-выражений и метавыражений.

Пусть *b*=*x*$_1$... *x*$_n$ и *c*=*c*$_1$... *c*$_m$. Пусть *e*=*e*$_1$ *eee*, где *e*$_1$ – экземпляр некоторого понятия. Пусть *v*$_j$∈*val*, *f*$_j$∈*v-sig* и *f*$_j$∉*s.sig* для всех *j*∈{1,...,*n*}.

Пусть *sss*=(*upd s* (#*p sig int*) (#*p* (*s.sig*∪{*f*$_1$,..., *f*$_n$}) (*upd s.int* (#*p* (*f*$_1$)...(*f*$_n$)) (#*p* *v*$_1$... *v*$_n$)))).

Пусть *sub*=((# *e*$_1$) (*x*$_1$ *f*$_1$())... (*x*$_n$ *f*$_n$ ()). Символ # в условии применимости *a* обозначает первый элемент исходной выполняемой последовательности *e* в случае, если он является экземпляром некоторого понятия языка *lan*.

Пусть *cc*=*cc*$_1$... *cc*$_m$, где *cc*$_i$ определяются следующим образом:

- если *c*$_i$ – *lan*-выражение, то *cc*$_i$=(*val* (*subst c*$_i$ *sub*) *sss.int*);
- если *c*$_i$ – метавыражение, то *cc*$_i$=(*subst c*$_i$ *sub*).

Атрибуты *cond* и *res* и функции *match* и *transform* определяются для правила *r* следующим образом:

- *r.cond*=*a* и *r.res*=(*b c*);
- (*match e r.cond mr s*)=*true* тогда и только тогда, когда (*s.int* (*subst a* ((# *e*$_1$))))=*true* и *mr*=*e*$_1$;
- (*transform e ee r.res mr s ss*)=*true* тогда и только тогда, когда *ss*=*sss* и *ee*=(*del-stop c cc e*), где
  - если *c*$_m$=*stop*, то (*del-stop c cc e*)=*cc*$_1$... *cc*$_{m-1}$;
  - в противном случае (*del-stop c cc e*)=*cc eee*.

Объект *stop* служит признаком исключения хвоста *eee* исходной последовательности *e* из результирующей последовательности *lab.out*.

В качестве примера рассмотрим ООС некоторых простых операторов.

Онтология оператора присваивания (*x*:=*u*), где *x* – имя переменной, *u* – либо переменная, либо целое число, а значения переменных ограничены множеством целых чисел, задается метаприсваиваниями ((*assignment-statement is concept*)::=*t*), ((*left-side is attribute of assignment-statement*)::=*t*) и ((*right-side is attribute of assignment-statement*)::=*t*).

Первое метаприсваивание определяет новое понятие *assignment-statement*, экземплярами которого являются операторы присваивания. Два последних метаприсваивания определяют атрибуты *left-side* и *right-side* для операторов присваивания, значением которых являются левая и правая части оператора присваивания, соответственно.

Семантика оператора присваивания задается правилом (*if* (# *is assignment-statement*) *hvar w then* (*assume* (*w*=#.*left-side*)) ((*w is variable*)::=*t*) #.*right-side* (*w.value*::=(*val*)).

Правило для оператора присваивания использует понятие *variable*, добавляемое в онтологию *o-ots* посредством метаприсваиваний ((*variable is concept*)::=*t*) и ((*value is attribute of variable*)::=*t*). В случае, если бы КЯ помимо целочисленного типа включал переменные других типов, к понятию *variable* был бы добавлен атрибут *type*, определяемый метаприсваиванием ((*type is attribute of variable*)::=*t*).

Кроме того, правило для оператора присваивания вычисляет правую часть оператора присваивания, выполняя выражение #.*right-side*. Поскольку правая часть оператора присваивания является либо переменной, либо целым числом, вычисление правой части задается двумя правилами (*if* (# *is variable*) *then* ((*val*)::=#.*value*)) и (*if* (# *is integer*) *then* ((*val*)::=#)). Во втором правиле используется предопределенное понятие *integer*, экземплярами которого являются целые числа. Множество экземпляров предопределенного понятия нельзя изменять с помощью метаприсваивания.

Онтология оператора удаления переменной (*del x*) задается метаприсваиваниями ((*delvar-statement is concept*)::=*t*) и ((*variable is attribute of delvar-statement*)::=*t*). Семантика этого оператора задается правилом (*if* (# *is del-statement*) *hvar w then* (*assume* (*w*=#.*variable*)) (*w.value*::=*u*) ((*w is variable*)::=*u*)).

Онтология условного оператора (*if f then b else c*) задается метаприсваиваниями ((*if-statement is concept*)::=*t*), ((*condition is attribute of if-statement*)::=*t*),

((*then is attribute of if-statement*)::=*t*) и ((*else is attribute of if-statement*)::=*t*).

Заметим, что онтологические символы позволяют типизировать атрибуты. Например, если необходимо учитывать тип атрибута *then*, то метаприсваивание для этого атрибута следует переписать к виду ((*then is attribute of if-statement of type statement*)::=*t*).

Семантика условного оператора задается правилами (*if* (*# is if-statement*) *then #.condition* (*assume* ((*val*)=*true*)) *#.then*) и (*if* (*# is if-statement*) *then #.condition* (*assume* ((*val*)=*false*)) *#.else*).

В качестве иллюстрации применения правил рассмотрим выполнение следующей исходной последовательности операторов: (*x*:=3)(*y*:=*x*)(*del x*)(*x*).

При выполнении этой последовательности используются правила для оператора присваивания, оператора удаления переменной, вычисления переменной и целого числа. Опишем результат применения этих правил в виде последовательностей метавыражений, соответствующих этим операторам:

(*assume* ((*f*$_1$)=*x*)) (((*f*$_1$) *is variable*)::=*t*) ((*val*)::=3) ((*f*$_1$).*value*::=3)
(*assume* ((*f*$_2$)=*y*)) (((*f*$_2$) *is variable*)::=*t*) ((*val*)::=*x.value*) ((*f*$_2$).*value*::=*3)*
(*assume* ((*f*$_3$)=*x*)) ((*f*$_3$).*value*::=*u*) (((*f*$_3$) *is variable*)::=*u*) ((*val*)::=*u*)

Выводы

Предложен новый подход к формальной спецификации КЯ, основанный на специальном виде помеченных систем переходов – онтологических системах перехода и модифицированном понятии операционно-онтологической семантики. Проведено его сравнение с традиционным подходом, основанным на помеченных системах переходов.

Работа выполнена при финансовой поддержке гранта РФФИ № 11-01-00028-а и междисциплинарного интеграционного проекта СО РАН № 3 «Принципы построения онтологии на основе концептуализаций средствами логических дескриптивных языков».

СПИСОК ЛИТЕРАТУРЫ

1. Гуревич Ю. Последовательные машины абстрактных состояний // Формальные методы и модели информатики: Сб. науч. тр. Серия «Системная информатика». – Новосибирск: Изд-во СО РАН, 2004. – Вып. 9. – С. 7–50.
2. Huggins J. Abstract State Machines Web Page. URL: http://www.eecs.umich.edu/gasm (дата обращения: 20.04.2012).
3. AsmL: The Abstract State Machine Language. URL: http://research.microsoft.com/en-us/projects/asml/ (дата обращения: 20.04.2012).
4. XasM: An Extensible, Component-Based Abstract State Machines Language. URL: http://xasm.sourceforge.net/XasmAnl00/XasmAnl00.html (дата обращения: 20.04.2012).
5. Ануреев И.С. Операционно-онтологический подход к формальной спецификации языков программирования // Программирование. – 2009. – № 1. – С. 1–11.

Поступила 30.09.2012 г.