УДК 519.681.3

И. С. Ануреев

Институт систем информатики им. А. П. Ершова СО РАН
пр. Академика Лаврентьева, 6, Новосибирск, 630090, Россия
E-mail: anureev@iis.nsk.su

ЯЗЫК ОПИСАНИЯ ОНТОЛОГИЧЕСКИХ СИСТЕМ ПЕРЕХОДОВ OTSL КАК СРЕДСТВО ФОРМАЛЬНОЙ СПЕЦИФИКАЦИИ ПРОГРАММНЫХ СИСТЕМ *

Онтологические системы переходов – формализм, предназначенный для спецификации программных систем. Они объединяют концептуальный подход к статической семантики систем, основанный на онтологиях, с операционным подходом к описанию динамики систем, базирующемся на системах переходов. В работе представлен язык описания онтологических систем переходов OTSL и определена формальная семантика этого языка. Примеры спецификаций на языке OTSL типовых задач, решаемых информационной системой с расширяемой онтологией, иллюстрируют выразительную силу этого языка.

Ключевые слова: онтологическая система переходов, программная система, онтология, система переходов, операционно-онтологическая семантика, OTSL, операционная семантика

Введение

Формальная спецификация программной системы является основой полного и непротиворечивого документирования этой системы, использования развитых математических методов для ее тестирования и верификации. Специфика программных систем заключается в том, что они обладают как развитой концептуальной структурой, так и сложным поведением. Поэтому разработка языков формальной спецификации программных систем, определяющих в едином унифицированном формате статическую и динамическую семантику программных систем, – актуальная задача современной теории и практики программирования.

Логико-алгебраический подход к решению этой задачи, основанный на машинах абстрактных состояний (далее АС-машины), был предложен Гуревичем [2004]. АС-машины – специальный вид систем переходов, состояниями в которых являются алгебраические системы. Выбор подходящей сигнатуры алгебраической системы позволяет приблизить формальное описание программной системы к ее естественной концептуальной структуре. Динамическая семантика программной системы определяется отношением перехода на состояниях. Примеры приложений этого формализма могут быть найдены в [Huggins]. Этот подход реализован в языках ASML [AsmL] и XASM [XasM].

В настоящее время развивается онтологический подход к спецификации программных систем (прежде всего информационных систем). Онтологии применяются в искусственном интеллекте, семантической паутине (SemanticWeb) и информационных технологиях как форма представления знаний. Они являются более естественным средством представления концептуальной структуры программных систем по сравнению с алгебраическими системами. Однако в отличие от АС-машин онтологии не описывают динамическую семантику программных систем.

Предлагаемый в статье операционно-онтологический подход к спецификации программных систем объединяет достоинства вышеописанных подходов. Он основан на онтологических системах переходов [Anureev, 2007] (далее ОТ-системах), которые являются «гибридом» онтологий и систем переходов. ОТ-системы задают: а) множество объектов программной системы; б) концептуальную структуру программной системы в виде онтологии; в) значение этой концептуальной структуры (онтологическую модель); г) переходы как действия, которые изменяют значение концептуальной структуры и/или саму концептуальную структуру.

* Работа выполнена при частичной финансовой поддержке РФФИ (проекты № 06-01-00464а и 08-01-00899а) в рамках интеграционного проекта СО РАН № 14.9.

© И. С. Ануреев, 2008

В данной работе представлен язык описания ОТ-систем OTSL, определена его формальная операционная семантика и рассмотрены примеры спецификации на языке OTSL типовых задач, решаемых информационной системой с расширяемой онтологией.

Предварительные сведения

Соглашения и обозначения. Условимся начинать обозначения множеств со строчной буквы, а их элементы обозначать так же, как множества, но начиная с прописной буквы и, возможно, добавляя индексы, штрихи и т. п. Например, `st` – множество состояний; `St`, `St'`, `St1` – конкретные состояния из множества `st`.

Пусть `x → y`, `x × y`, `bool` обозначают «множество тотальных функций из `x` в `y`», «декартово произведение множеств `x` и `y`» и «булевское множество `{true, false}`» соответственно. Запись `x → y → z` является сокращением для `x → (y → z)`.

Пусть `[x → A]` обозначает функцию `f` с областью определения `x` такую, что `f(X) = A` для каждого `X`. Пусть `upd(f, (`A_1`, ..., `A_n`), e)` обозначает функцию `f'` такую, что `f'(`A_1`)...(`A_n`) = e` и `f'(`B_1`)...(`B_n`) = f(`B_1`)...(`B_n`)` в случае, если `(`B_1`, ..., `B_n`) ≠ (`A_1`, ..., `A_n`)`.

ОТ-система `Ots` определяется как шестерка `(ob, st, |.|`$_{op}$`, |.|`$_0$`, |.|`$_1$`, |.|`$_2$`)`. ОТ-системы используются для спецификации сложных динамических систем (программных систем, информационных систем, виртуальных машин, задающих семантику языков программирования и т. д.), поэтому в пояснениях к компонентам ОТ-системы `Ots` будем ссылаться на динамическую систему, которая специфицируется `Ots`, как на `DSys`.

Множество `ob` определяет объекты, которые можно выделить в системе `DSys`. Элементы множества `ob` называются объектами.

Множество `st` определяет состояния, в которых может находиться система `DSys`. Элементы множества `st` называются состояниями.

Остальные компоненты ОТ-системы `Ots` определяют различные семантики, приписываемые объектам.

Функция `|.|`$_{op}$ ∈ `ob → st × st → bool` называется операционной семантикой объектов. Эта функция специфицирует динамическую семантику системы `DSys`. Она связывает объекты, рассматриваемые как знаки, с множествами пар состояний, задавая переходы между состояниями, которые возможны в системе `DSys`. Таким образом, эта интепретация объектов рассматривает их как переходы между состояниями. Функция `|Ob|`$_{op}$ называется операционным значением объекта `Ob`. Свойство `|Ob|`$_{op}$`(St, St')` означает, что имеется переход с именем `Ob` из состояния `St` в состояние `St'`. Пара состояний `(St, St')` называется экземпляром перехода `Ob` в состоянии `St`, если `|Ob|`$_{op}$`(St, St') = true`. Функция `|.|`$_{op}$ также определяет, какие переходы выполнимы в текущем состоянии. Переход `Ob` выполним в состоянии `St`, если `|Ob|`$_{op}$ ≠ `[st × st → false]`.

ОТ-системы предназначены также для описания онтологий и онтологических моделей динамических систем. Онтология обычно включает понятия, атрибуты, отношения и, возможно, экземпляры (или индивиды). Онтологическая модель обеспечивает значения для этих составляющих онтологии. Следующие три компоненты ОТ-системы `Ots` специфицируют онтологию и онтологическую модель системы `DSys`.

Функция `|.|`$_0$ ∈ ob → st → ob называется онтологической семантикой объектов уровня `0`. Она связывает объекты, рассматриваемые как знаки, с объектами, рассматриваемыми как индивиды. При такой интерпретации объектов, они служат обозначениями для индивидов. Объект (индивид) `|Ob|`$_0$`(St)` называется онтологическим `0`-значением объекта `Ob` в состоянии `St`.

Функция `|.|`$_1$ ∈ `ob → st → ob → bool` называется онтологической семантикой уровня `1`. Она связывает объекты, рассматриваемые как знаки, с множествами объектов, задавая понятия на объектах. Функция `|Ob|`$_1$`(St)` называется онтологическим `1`-значением объекта `Ob` в состоянии `St`. Объект `Ob` интерпретируется как понятие на объектах, а его

1-значение – как содержимое (значение) этого понятия. Заметим, что содержимое понятия может меняться при переходе из состояния в состояние, что позволяет задавать с помощью ОТ-систем динамические (изменяющиеся от состояния к состоянию) онтологические модели. Если выделить специальное множество объектов, рассматривая понятия, имеющие это множество в качестве значения, как несуществующие, то с помощью ОТ-систем можно задавать и динамические онтологии. Объект Ob′ называется экземпляром понятия Ob в состоянии St, если $|Ob|_1(St)(Ob') = true$.

Функция $|.|_2 \in ob \to st \to bool$ называется онтологической семантикой уровня 2. Она связывает объекты, рассматриваемые как знаки, с множествами состояний, задавая понятия на состояниях. Функция $|Ob|_2$ называется онтологическим 2-значением объекта Ob. Объект Ob интепретируется как понятие на состояниях, а его 2-значение – как содержимое (значение) этого понятия.

Как будет показано далее при описании языка OTSL, допуская в качестве объектов последовательности объектов, с помощью ОТ-систем можно также моделировать такие составляющие онтологии, как атрибуты и отношения.

Объекты в языке OTSL

Множество объектов ob языка OTSL в точности совпадает с множеством синтаксических конструкций языка OTSL, т. е. любой объект имеет синтаксическое представление в языке OTSL, и любая синтаксическая конструкция языка OTSL является объектом.

Атомы – наименьшие синтаксические элементы языка OTSL, из которых строятся все синтаксические конструкции языка OTSL. Обозначим через at множество атомов языка OTSL. Конкретное наполнение множества at определяется реализацией языка OTSL. Оно может быть множеством строк Unicode, множеством слов алфавита a..zA..Z и т. д.

Множество объектов ob определяется следующим образом:

1) $At \in ob$;
2) $\{\}, [], \langle\rangle, () \in ob$;
3) $\{Ob\}, [Ob], \langle Ob\rangle, (Ob) \in ob$;
4) $Ob\ Ob' \in ob$.

Таким образом, объекты строятся из атомов с помощью последовательной композиции и четырех видов структурных скобок, соответствующих четырем видам семантики. С помощью фигурных скобок строятся объекты, которые интерпретируются переходами из состояния в состояние и для которых определена операционная семантика. С помощью квадратных скобок строятся объекты, которые интерпретируются индивидами и для которых определена онтологическая семантика уровня 0. С помощью угловых скобок строятся объекты, которые интерпретируются понятиями на объектах и для которых определена онтологическая семантика уровня 1. С помощью круглых скобок строятся объекты, называемые формулами, которые интерпретируются понятиями на состояниях и для которых определена онтологическая семантика уровня 2. Определение каждой из семантик расширяется на все объекты в силу тотальности функций |.|, задающих семантики.

Объект () называется пустым объектом. Равенство = на объектах определяется как синтаксическое совпадение объектов с точностью до пустого объекта: $Ob\ () = () \ Ob = Ob$.

Объект любого из видов, описываемых в пунктах 1 – 3 определения множества объектов, называется унарным объектом.

Определим подстановку как функцию вида $at \to ob$. Пусть sub обозначает множество всех подстановок. Область определения подстановки расширяется на объекты следующим образом:

- $Sub[At] = At$ для связанного вхождения At (атомы связываются в кванторных формулах и кванторных переходах, рассмотренных ниже);
- $Sub[At] = Sub(At)$ для свободного (не связанного) вхождения At;
- $Sub[()] = (), Sub[[]] = [], Sub[\{\}] = \{\}, Sub[\langle\rangle] = \langle\rangle$;
- $Sub[Ob\ Ob'] = Sub[Ob]\ Sub[Ob']$;

- `Sub[(Ob)] = (Sub[Ob)], Sub[[Ob]] = [Sub[Ob]], Sub[{Ob}] = {Sub[Ob]}, Sub[<Ob>] = <Sub[Ob]>`.

Пусть $[Y_1 \leftarrow Z_1, \ldots, Y_n \leftarrow Z_n]$ обозначает подстановку Sub такую, что $Sub(Y_i) = Z_i$ и $Sub(X) = X$, если $X \neq Y_i$ для всех $1 \leq i \leq n$.

Объект Ob называется примером объекта Ob′, если существует подстановка Sub такая, что Ob = Sub(Ob′). Объект Ob′ в этом случае называется образцом для объекта Ob. Сопоставление с образцом (pattern matching) является одной из характерных особенностей языка OTSL. В этом аспекте язык OTSL похож на язык Refal [Содружество «РЕФАЛ/Суперкомпиляция»], но в отличие от последнего допускает несколько видов структурных скобок.

Состояния в языке OTSL

Состояние в языке OTSL определяется как набор функций, с помощью которых задаются операционная и онтологические семантики объектов. Перечислим имена и сигнатуры функций, входящих в этот набор, а также опишем их связи с семантиками. Функции сгруппированы в соответствии с определяемыми ими видами семантик.

Операционная семантика. Для определения операционной семантики объектов $|.|_{op}$ используется функция $OpSem \in ob \to ob \to bool$:

$|\{Ob\}|_{op}(St, St') = true$ тогда и только тогда, когда существуют Ob', Ob'' и Sub такие, что $OpSem(Ob')(Ob'') = true$, $Sub(Ob') = |Ob|_0(St)$ и $|Sub(Ob'')|_{op}(St, St') = true$.

Таким образом, функция OpSem хранит пары вида (Ob′, Ob′′), где Ob′ – образец, которому должен удовлетворять целевой объект Ob (быть примером этого образца относительно некоторой подстановки Sub), а Ob′′ интерпретируется как переход, конкретизация которого относительно подстановки Sub задает частичную операционную семантику объекта Ob. Полная операционная семантика объекта Ob определяется совокупностью всех пар, описываемых (характеристической) функцией OpSem. Условием использования функции OpSem являются фигурные скобки вокруг объекта Ob.

Изменение функции OpSem задается переходом {+{} Ob | Ob′}, добавляющим пару в множество, описываемое функцией OpSem, и переходом {-{} Ob | Ob′}, удаляющим пару из этого множества:

- $|\{+\{\}\ Ob\ |\ Ob'\}|_{op}(St, St') = true$ тогда и только тогда, когда $St' = upd(St, (OpSem, |Ob|_0(St), |Ob'|_0(St)), true)$;
- $|\{-\{\}\ Ob\ |\ Ob'\}|_{op}(St, St') = true$ тогда и только тогда, когда $St' = upd(St, (OpSem, |Ob|_0(St), |Ob'|_0(St)), false)$.

Онтологическая семантика уровня 0. Для определения онтологической семантики $|.|_0$ уровня 0 используется функция $Sem0 \in ob \to ob$:

$|[Ob]|_0(St) = Sem0(Ob)$.

Таким образом, функция Sem0 хранит индивиды для объектов. Условием использования этой функции являются квадратные скобки вокруг целевого объекта Ob.

Изменение функции Sem0 задается переходом {Ob := Ob′}, который устанавливает значение индивида, обозначаемого объектом Ob, равным индивиду, обозначаемому объектом Ob′:

$|\{Ob := Ob'\}|_{op}(St, St') = true$ тогда и только тогда, когда

$St' = upd(St, (Sem0, |Ob|_0(St)), |Ob'|_0(St))$.

Онтологическая семантика уровня 1. Для определения онтологической семантики уровня 1 используются функции $DefSem1 \in ob \to ob \to bool$, $PSem1 \in ob \to ob \to bool$ и $BaseCo \in ob \to ob \to bool$:

$|<Ob>|_1(St)(Ob') = true$ тогда и только тогда, когда

- существует Ob′′ такой, что $DefSem1(|Ob|_0(St))(Ob'') = true$ и $|Ob''|_1(St)(Ob') = true$ или

- существует Ob'' такой, что BaseCo(|Ob|$_0$(St))(Ob'') = true, |Ob''|$_1$(St)(Ob') = true и PSem1(|Ob|$_0$(St))(Ob') = true.

Таким образом, функция DefSem1 хранит пары (Ob$_1$, Ob$_2$), где Ob$_1$ – имя понятия на объектах, а Ob$_2$ интерпретируется как понятие, которое является определением для понятия Ob$_1$. Заметим, что согласно описанию функции DefSem1 понятие может иметь несколько определений, задаваемых этой функцией. В этом случае достаточно, чтобы объект, проверяемый на принадлежность понятию Ob$_1$, принадлежал хотя бы одному из определяющих понятий. Такое решение позволяет разрешать перегрузку имен понятий, что часто встречается на практике. Функция PSem1 выполняет частичное означивание понятий. Она хранит пары (Ob$_1$, Ob$_2$), где Ob$_1$ – имя понятия на объектах, а Ob$_2$ интерпретируется как экземпляр этого понятия. Функция BaseCo хранит пары (Ob$_1$, Ob$_2$), где Ob$_1$ – имя понятия на объектах, а Ob$_2$ интерпретируется как базовое понятие для понятия Ob$_1$, т. е. экземпляры понятия Ob$_1$ должны также быть экземплярами понятия Ob$_2$. Согласно описанию функции BaseCo допускается множественность базовых понятий, т. е. для одного и того же понятия может быть несколько базовых понятий.

Изменение функции DefSem1 задается переходом {Ob =+ Ob'}, добавляющим пару в множество, задаваемое (характеристической) функцией DefSem1, переходом {Ob =- Ob'}, удаляющим пару из этого множества и переходом {Ob =-*}, удаляющим все определения для понятия, имя которого является индивидом, обозначаемым объектом Ob:

- |{Ob =+ Ob'}|$_{op}$(St, St') = true тогда и только тогда, когда St' = upd(St, (DefSem1, |Ob|$_0$(St), |Ob'|$_0$(St)), true);
- |{Ob =- Ob'}|$_{op}$(St, St') = true тогда и только тогда, когда St' = upd(St, (DefSem1, |Ob|$_0$(St), |Ob'|$_0$(St)), false);
- |{Ob =-*}|$_{op}$(St, St') = true тогда и только тогда, когда St' = upd(St, (DefSem1, |Ob|$_0$(St)), [ob → false]).

Изменение функции PSem1 задается переходом {Ob +. Ob'}, добавляющим пару в множество, задаваемое (характеристической) функцией PSem1, переходом {Ob -. Ob'}, удаляющим пару из этого множества, и переходом {Ob -.*}, удаляющим все экземпляры для понятия, имя которого является индивидом, обозначаемым объектом Ob:

- |{Ob +. Ob'}|$_{op}$(St, St') = true тогда и только тогда, когда St' = upd(St, (PSem1, |Ob|$_0$(St), |Ob'|$_0$(St)), true);
- |{Ob -. Ob'}|$_{op}$(St, St') = true тогда и только тогда, когда St' = upd(St, (PSem1, |Ob|$_0$(St), |Ob'|$_0$(St)), false);
- |{Ob -.*}|$_{op}$(St, St') = true тогда и только тогда, когда St' = upd(St, (PSem1, |Ob|$_0$(St)), [ob → false]).

Изменение функции BaseCo задается переходом {+ Ob | Ob'}, добавляющим пару в множество, задаваемое (характеристической) функцией BaseCo, переходом {- Ob | Ob'}, удаляющим пару из этого множества, и переходом {-* Ob}, удаляющим все базовые понятия для понятия, имя которого является индивидом, обозначаемым объектом Ob:

- |{+<> Ob | Ob'}|$_{op}$(St, St') = true тогда и только тогда, когда St' = upd(St, (BaseCo, |Ob|$_0$(St),|Ob'|$_0$(St)), true);
- |{-<> Ob | Ob'}|$_{op}$(St, St') = true тогда и только тогда, когда St' = upd(St, (BaseCo, |Ob|$_0$(St),|Ob'|$_0$(St)), true);
- |{-<>* Ob}|$_{op}$(St, St') = true тогда и только тогда, когда St' = upd(St, (BaseCo, |Ob|$_0$(St)), [ob → false]).

Онтологическая семантика уровня 2. Для определения онтологической семантики уровня 2 используется функция Sem2 ∈ ob → ob → bool:

|Ob|$_2$(St) = true тогда и только тогда, когда существуют Ob', Ob'' и Sub такие, что Sem2(Ob')(Ob'') = true, Sub(Ob') = |Ob|$_0$(St) и |Sub(Ob'')|$_2$(St) = true.

Таким образом, функция Sem2 хранит пары вида (Ob′, Ob′′), где Ob′ – образец, которому должен удовлетворять целевой объект Ob (быть примером этого образца относительно некоторой подстановки Sub), а Ob′′ интерпретируется как формула, конкретизация которой относительно подстановки Sub задает частично 2-значение объекта Ob. Полное 2-значение определяется совокупностью всех пар, описываемых (характеристической) функцией Sem2.

Изменение функции Sem2 задается переходом {+() Ob | Ob′}, добавляющим пару в множество, задаваемое функцией Sem2, и переходом {-() Ob | Ob′}, удаляющим пару из этого множества:

- $|\{+()\ Ob\ |\ Ob'\}|_{op}(St, St') = true$ тогда и только тогда, когда $St' = upd(St, (Sem2, |Ob|_0(St), |Ob'|_0(St)), true)$;
- $|\{-()\ Ob\ |\ Ob'\}|_{op}(St, St') = true$ тогда и только тогда, когда $St' = upd(St, (Sem2, |Ob|_0(St), |Ob'|_0(St)), false)$.

Объекты с предопределенной семантикой

Помимо семантики, вычисляемой в соответствии с общими правилами, описанными выше, объекты могут иметь предопределенную семантику. Значения объектов, обеспечиваемые предопределенной семантикой имеют приоритет над значениями, вычисляемыми в соответствии с общими правилами. Множество объектов с предопределенной семантикой является расширяемым. Реализации языка могут обеспечивать механизм добавления новых объектов с предопределенной семантикой. Перечислим объекты, которые имеют предопределенную семантику в текущей версии языка OTSL, сгруппировав их по видам семантики. В текущей версии языка OTSL объекты с предопределенной семантикой уровня 0 отсутствуют.

Операционная семантика. Объекты языка OTSL с предопределенной операционной семантикой реализуют общезначимые механизмы управления обработкой данных в современных языках программирования – такие, как, например, охранные условия, последовательные композиции, ветвления, только в некоторой более абстрактной форме. Использование абстрактной формы позволило ограничиться небольшим набором таких объектов, обеспечивая при этом достаточную для практики выразительную силу. Перечислим эти объекты и определим их семантику.

Охранное условие (Ob) задает переход, который выполним только при истинности формулы Ob в текущем состоянии:

$|(Ob)|_{op}(St, St') = true$ тогда и только тогда,
когда $|(Ob)|_2(St) = true$ и $St = St'$.

Охранное условие true задает переход, который всегда выполним:

$|true|_{op}(St, St') = true$ тогда и только тогда, когда $St = St'$.

Охранное условие false задает переход, который никогда не выполним:

$|false|_{op}(St, St') = false$.

Композиция переходов Ob Ob′ задает последовательное выполнение переходов Ob и Ob′: $|Ob\ Ob'|_{op}(St, St') = true$ тогда и только тогда, когда найдется состояние St′′ такое, что $|Ob|_{op}(St, St'') = true$ и $|Ob'|_{op}(St'', St') = true$.

Ветвление $\{Ca_1 \ldots Ca_n\}$ задает выбор из нескольких вариантов Ca_i. Варианты делятся на варианты недетерминированного выбора @ Ob и варианты альтернативного выбора # Ob. Условие выполнимости Cond(* Ob) варианта * Ob, где * ∈ {@, #}, определяется как переход Ob′ {#}, если Ob = Ob′ {#} Ob′′ для некоторого Ob′′ и Ob′ не содержит {#} на верхнем уровне композиции.

Переход {#} называется меткой завершения условия выполнимости варианта. В случае, если эта метка отсутствует, условие выполнимости варианта определяется как Ob. Выполнение метки {#} не меняет состояние ОТ-системы: $|\{\#\}|_{op}(St, St') = true$ тогда и только тогда, когда St′ = St. Вариант Ca выполним в состоянии St, если существует состояние St′ такое, что $|Cond(Ca)|_{op}(St, St') = true$.

Семантика ветвления определяется тремя правилами. Правило для пустого ветвления {} имеет вид $|\{\}|_{op}(St, St') = false$. Таким образом, пустое ветвление никогда не выполняется.

Пусть CaSeq – последовательность вариантов, которая либо начинается с альтернативного варианта, либо является пустой. Правило для вариантов недетерминированного выбора имеет вид: $\{@\ Ob_1\ \ldots\ @\ Ob_m\ CaSeq\}(St, St') = true$ тогда и только тогда, когда существует $1 \le i \le m$ такое, что $|Ob_i|_{op}(St, St')$ или варианты $@Ob_1$, ..., $@Ob_m$ невыполнимы и $|\{CaSeq\}|_{op}(St, St') = true$. Оно утверждает, что выполняется любой из выполнимых вариантов недетерминированного выбора.

Пусть CaSeq – произвольная последовательность вариантов. Правило для вариантов альтернативного выбора имеет вид:

$\{\#\ Ob\ CaSeq\}(St, St') = true$ тогда и только тогда,

когда $|Ob|_{op}(St, St')$ или вариант $@Ob$ невыполним

и $|\{CaSeq\}|_{op}(St, St') = true$.

Оно утверждает, что альтернативный вариант выполняется в том случае, если не выполняется ни один из предшествующих вариантов ветвления.

Следующую группу переходов составляют кванторные переходы. Экзистенциальный кванторный переход $\{?\ At\ :\ Ob\ |\ Ob'\}$ позволяет моделировать недетерминированный выбор из множества переходов, которые получаются заменой в переходе Ob′ атома At на экземпляр понятия Ob в текущем состоянии St:

$|\{?\ At\ :\ Ob\ |\ Ob'\}|_{op}(St, St') = true$ тогда и только тогда,

когда найдется объект Ob такой, что $|Ob|_1(St)(Ob'') = true$

и $|[At \leftarrow Ob'']Ob')|_{op}(St, St') = true$.

Универсальный кванторный переход $\{!\ At\ :\ Ob\ |\ Ob'\}$ позволяет моделировать инвариантность выбора из множества переходов, которые получаются заменой в переходе Ob′ атома At на экземпляр понятия Ob в текущем состоянии St, т. е. при любом выборе гарантируется одно и то же множество результатов:

$|\{!\ At\ :\ Ob\ |\ Ob'\}|_{op}(St, St') = true$ тогда и только тогда,

когда для любого объекта Ob такого, что $|Ob|_1(St)(Ob'') = true$

выполнено $|[At \leftarrow Ob'']Ob')|_{op}(St, St') = true$.

Чтобы обеспечить тотальность функции $|.|_{op}$, доопределим ее для объектов, не попавших ни в одну из рассмотренных групп и неопределяемых с помощью функции OpSem из определения состояния:

- $|\{\}|_{op}(St, St') = true$ тогда и только тогда, когда $St = St'$;
- $|<Ob>|_{op}(St, St') = false$;
- $|<>|_{op}(St, St') = false$;
- $|[Ob]|_{op}(St, St') = true$ тогда и только тогда, когда $|Sem1(Ob)|_{op}(St, St') = true$;
- $|[]|_{op}(St, St') = false$;
- $|()|_{op}(St, St') = false$.

Онтологическая семантика уровня 1. Объекты языка OTSL с предопределенной онтологической семантикой уровня 1 включают предопределенные понятия, описывающие такие базовые синтаксические категории, как атом, объект, унарный объект, и конструктор множеств, определенный на объектах. Предопределенные понятия увеличивают выразительные возможности механизма сопоставления с образцом, а конструктор множеств обеспечивает широкие возможности при построении новых понятий. Определим синтаксис и семантику этих объектов.

Язык OTSL включает следующие объекты с предопределенной семантикой уровня 1.

Предопределенное понятие at определяет множество атомов, т. е.

$|at|_1(St)(Ob) = true$ тогда и только тогда, когда Ob – атом.

Предопределенное понятие ob определяет множество всех объектов, т. е. $|ob|_1(St)(Ob) = true$ для каждого объекта Ob.

Предопределенное понятие unob определяет множество всех унарных объектов, т. е. |unob|$_1$(St)(Ob) = true тогда и только тогда, когда Ob – унарный объект.

Конструктор понятий <Ob | Ob′> задает множество всех примеров образца |Ob|$_0$(St) таких, что конкретизация объекта |Ob′|$_0$(St) относительно этих примеров дает формулу, истинную в текущем состоянии St:

|<Ob | Ob′>|$_1$(St)(Ob′′) = true тогда и только тогда,

когда существует Sub такая, что Sub(|Ob|$_0$(St)) = Ob′′

и |Sub(|Ob′|$_0$(St))|$_2$(St) = true.

Онтологическая семантика уровня 2. Объекты языка OTSL с предопределенной онтологической семантикой уровня 2 реализуют «джентльменский набор» логических средств задания формул: булевские константы, пропозициональные связки, кванторы, равенство и слабое равенство, принадлежность индивида понятию, модальности динамической логики. Определим синтаксис и семантику этих объектов.

Булевские константы true и false имеют обычную семантику:

- |true|$_2$(St) = true;
- |false|$_2$(St) = false.

Формула (Ob | Ob′) определяет принадлежность индивида |Ob′|$_1$(St), обозначенного объектом Ob′, содержимому понятия Ob в состоянии St:

|(Ob | Ob′)|$_2$(St) = true тогда и только тогда,

когда |Ob|$_1$(St)(|Ob′|$_0$(St)) = true.

Формула (Ob = Ob′) задает равенство индивидов, обозначенных объектами Ob и Ob′:

|(Ob = Ob′)|$_2$(St) = true тогда и только тогда,

когда индивиды |Ob|$_0$(St) и |Ob′|$_0$(St) совпадают.

Формула (Ob ~ Ob′) задает слабое равенство индивидов, обозначенных объектами Ob и Ob′:

|(Ob ~ Ob′)|$_2$(St) = true тогда и только тогда,

когда индивиды |Ob|$_0$(St) и |Ob′|$_0$(St) совпадают с точностью до перестановки унарных объектов, входящих в последовательную композицию верхнего уровня.

Группа пропозициональных формул включает отрицание (not Ob), конъюнкцию (Ob and Ob′), дизъюнкцию (Ob or Ob′), импликацию (Ob implies Ob′) и эквивалентность (Ob iff Ob′), которые определяются стандартным образом. Например, |(not Ob)|$_2$(St) = true тогда и только тогда, когда |Ob|$_2$(St) = false.

Группа кванторных формул, включающая экзистенциальные кванторные формулы (? At : Ob | Ob′) и универсальные кванторные формулы (! At : Ob | Ob′), определяется следующим образом:

- |(? At : Ob | Ob′)|$_2$(St) = true тогда и только тогда, когда существует Ob′′ такой, что |Ob|$_1$(St)(Ob′′) = true и |[At ← Ob′′](Ob′)|$_2$(St) = true;
- |(! At : Ob | Ob′)|$_2$(St) = true тогда и только тогда, когда для любого Ob′′ такого, что |Ob|$_1$(St)(Ob′′) = true, выполнено |[At ← Ob′′](Ob′)|$_2$(St) = true.

Группа динамических формул, включающая экзистенциальные динамические формулы (? Ob | Ob′) и универсальные динамические формулы (! Ob | Ob′) определяется так же, как соответствующие модальности в динамической логике:

- |(? Ob | Ob′)|$_2$(St) = true тогда и только тогда, когда существует состояние St′ такое, что |Ob|$_{op}$(St, St′) = true и |Ob′|$_2$(St′) = true;
- |(! Ob | Ob′)|$_2$(St) = true тогда и только тогда, когда для любого состояния St′ такого, что |Ob|$_{op}$(St, St′) = true, выполнено |Ob′|$_2$(St′) = true.

Примеры спецификации типовых задач, решаемых информационной системой с расширяемой онтологией

Рассмотрим примеры спецификации на языке OTSL некоторых типовых задач, решаемых информационной системой (далее ИС) с расширяемой онтологией.

Задача расширения онтологии специфицируется переходами, которые добавляют новые понятия, отношения и атрибуты. Например, переходы

```
{+<> статья | at} {+<> персона | at} {+<> журнал | at}
{+<> автор <A B | ((статья | A) and (at | B))>}
{автор =+ <X | ((персона | X) and (? Y : статья | (автор | Y X)))>}
{+<> опубликована в | <A B | ((статья | A) and (журнал | B))>}
{+<> ссылается на | <A B | ((статья | A) and (статья | B))>}
```

добавляют понятия `статья`, `персона`, `журнал`, `автор` и атрибуты `автор`, `опубликована в` и `ссылается на` понятия `статья` в онтологию.

Задача построения онтологической модели («означивания» онтологии) специфицируется переходами, которые описывают операции изменения содержимого понятий. Онтологическая модель определяет базу знаний ИС. Например, переходы

```
{статья +. СТ1} {статья +. СТ2} {статья +. СТ3}
{персона +. А1} {персона +. А2} {автор +. СТ1 А1}
{автор +. СТ2 А1} {автор +. СТ2 А2} {автор +. СТ3 А2}
{журнал +. Ж1} {журнал +. Ж2} {опубликована в +. СТ1 Ж1}
{опубликована в +. СТ2 Ж1} {опубликована в +. СТ3 Ж2}
{ссылается на +. СТ3 СТ2} {ссылается на +. СТ2 СТ1}
```

специфицируют содержимое введенных понятий и атрибутов. Все понятия и атрибуты, кроме понятия `автор`, определяются перечислением входящих в них экземпляров, а понятие `автор` задается определением, которое говорит, что автором является персона, имеющая хотя бы одну статью.

Другие задачи обновления базы знаний ИС (редактирования и удаления экземпляров понятий, отношений и атрибутов, удаление понятий, отношений и атрибутов) аналогичным образом специфицируются с помощью переходов, определяющих операции над понятиями.

Задача построения запроса к ИС специфицируется обновлением содержимого понятия `result`, в котором хранится результат запроса. Например, запрос «найти журналы, в которых есть ссылки на статьи автора `А1` в журнале `Ж1`» специфицируется следующим образом:

```
{result -.*}
{result =+ <X | ((журнал | X) and
 (? Y : статья ? Z : статья | (опубликована в | Y X) and
 (ссылается | Y Z) and (опубликована в | Z Ж1) and
 (автор | Z А1)))>}
```

Задача добавления функциональности объектам ИС специфицируется объектами с предопределенной операционной семантикой, определяющими базовую функциональность, и объектами с определяемой операционной семантикой. Например, функциональность, которая выдает при клике на автора все статьи этого автора, определяется следующим образом. В качестве объекта с предопределенной операционной семантикой выступает объект `{print, Co}`, который выводит значение понятия `Co` в виде web-страницы. Тогда требуемая функциональность определяется объектом со следующим операционным определением:

```
{+{} X | (клик | X) (автор | X) {result -.*}
 {result =+ <Y | (статья | Y) and (автор | Y X)>}
 {print result}}
```

Согласно этому определению выполняется следующая последовательность шагов: выбирается объект `X` (`+{} X {...}`), проверяется, что он является автором (`(автор | X)`) и был «кликнут» (`(клик | X)`), обнуляется текущее содержимое понятия `result` (`{result =-*}`), содержимым отношения `result` становится множество статей автора `X` (`{result =+ <...>}`), выполняется вывод результатов `{print result}`.

Заключение

В работе представлен язык описания ОТ-систем OTSL. Достоинствами этого языка являются: наличие формальной семантики; возможность описания концептуальной структуры

программных систем и ограничений на нее; возможность задания сложных, основанных на онтологии, запросов к программной системе; развитые средства описания динамики систем; возможность изменения онтологии при функционировании программной системы.

В то время как языки ASML и XASM изначально проектировались как выполнимые языки спецификаций, нацеленные на тестирование и верификацию времени исполнения (run-time verification) программных систем, основные две характеристики языка OTSL можно определить следующим образом: OTSL – язык нотаций, и OTSL – язык, ориентированный на классическую дедуктивную верификацию и, возможно, на проверку на моделях (model checking).

Как язык нотаций OTSL может использоваться на ранних стадиях проектирования программной системы для создания спецификаций на основе требований к системе и как средство задания унифицированных канонических спецификаций, например формальных спецификаций для языков программирования. В частности, в рамках проекта по классификации языков программирования язык OTSL используется для описания общезначимых конструкций и механизмов языков программирования [Ануреев, 2008а; 2008б].

Ориентированность на нотационный аспект привела к тому, что для повышения выразительной силы языка в OTSL включены такие неэффективные с точки зрения выполнения переборные конструкции, как логические кванторы, модальности динамической логики и кванторные переходы. Поэтому рассмотрение языка OTSL как языка выполнимых спецификаций потребует наложения ограничений на эти конструкции.

Как язык, ориентированный на дедуктивную верификацию, OTSL более приближен к логическому базису по сравнению с языками ASML и XASM, представляющими объектно-ориентированную реализацию АС-машин.

Текущая версия языка OTSL ограничивается спецификацией централизованных программных систем. Поскольку ОТ-системы позволяют распределять информацию о концептуальной структуре программной системы по атомам и назначать им действия, в дальнейшем предполагается расширить язык OTSL на распределенные системы.

Список литературы

Ануреев И. С. Операционно-онтологическая семантика обработки исключений // Тезисы докладов международной конференции «Космос, астрономия и программирование» (Лавровские чтения). СПб., 2008а. С. 15–22.

Ануреев И. С. Операционно-онтологическая семантика операторов безусловной передачи управления в языке C# // Тезисы докладов международной конференции «Космос, астрономия и программирование» (Лавровские чтения). СПб., 2008б. С. 259–266.

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.

Содружество «РЕФАЛ / Суперкомпиляция». [Электронный ресурс]. Режим доступа: http://refal.net

Anureev I. S. Ontological Transition Systems // Joint NCC&IIS Bull. Series: Computer Science. 2007. Iss. 26. P. 1–17.

AsmL: The Abstract State Machine Language. – Reference Manual. 2002. [Электронный ресурс]. Режим доступа: http://research.microsoft.com/fse/asml/doc/AsmL2_Reference.doc.

Huggins J. Abstract State Machines Web Page. http://www.eecs.umich.edu/gasm.

XasM An Extensible, Component-Based Abstract State Machines Language. [Электронный ресурс]. Режим доступа: http://xasm.sourceforge.net/XasmAnl00/XasmAnl00.html

Материал поступил в редколлегию 26.08.2008

I. S. Anureev

A Language of Description of Ontological Transition Systems OTSL as a Tool for Formal Specification of Program Systems

Ontological transition systems are a formalism for specification of program systems. They combine a conceptual approach to static semantics of these systems, based on ontologies, with an operational approach to description of dynamics of the systems, based on transition systems.

In this paper the language of description of ontological transition systems OTSL is presented and formal semantics of this language is defined. Examples of OTSL specifications of routine problems, which are solved by an information system with open ontology, illustrate expressive power of the language.

Keywords: ontological transition system, program system, ontology, transition system, operational-ontological semantics, OTSL, operational semantics.