АСПЕКТЫ НЕКЛАССИЧЕСКОЙ ТЕОРИИ НОМИНАЦИИ И ИХ ИСПОЛЬЗОВАНИЕ В ФОРМАЛЬНЫХ ЯЗЫКАХ

Россада Татьяна

***Аннотация**: В сообщении рассматривается ряд аспектов именования, которые не вкладываются в рамки классической теории имен Фреге-Рассела.*

***Ключевые слова**: именование, формальный язык, аспекты именования.*

***АСМ классификация ключевых слов**: I.6.4 Model Validation and Analysis; ; I.1.3 Computing Methodologies – Symbolic Algebraic Manipulation - Languages and Systems, H.3.1 Database management – information storage and retrieval; H.3.1 Database management – Content Analysis and Indexing*

Вступление

На данном этапе развития информационных технологий есть возможность распределять разработку программного обеспечения, следствием чего является стилистическая неоднородность результирующего кода, в основе которой, в частности, – отсутствие *единого стандарта именования*: несмотря на четко определенные правила и грамматику программного инструментария, стандарты именования до сих пор являются локальными (в пределах одной команды), неформальными (C-стиль, CamelStyle, Pascal-стиль, Венгерский стиль и т.д.), нечетко определенными, а в некоторых случаях даже несовместимыми в рамках одного проекта. В результате имеем ряд реально существующих на сегодняшний день проблем, требующих решения. Одна из них – риск возникновения *коллизии имен* в программных системах. Для подчеркивания важности исследования и решения этой проблемы вспомним пресловутый аппарат лучевой терапии Therac-25, неправильная работа которого привела к смерти минимум шести человек: одной из ошибок в программном обеспечении было использование одинакового имени переменных в разных фрагментах программы, введенных для совершенно разных задач.

На сегодняшний день проблема риска возникновения коллизии имен частично решается явным или неявным (если программный инструментарий не поддерживает такие возможности) введением пространства имен. Последнее, при правильном применении, действительно решает проблему коллизии, но в итоге получаем в основном запутанный громоздкий код, неприменимый к автодокументации, а в некоторых случаях даже невыполнимый (сложные операции запросов над группами пространства имен выполняются не оптимально или не выполняются вообще).

Еще одна проблема, которая является результатом отсутствия единого стандарта именования в формальных языках – непригодный к совершенствованию код, которым его делают сложные, запутанные имена, понятные только их автору, довольно часто образованные транслитерацией по языку программиста ("nazvanie"), огромные по объему ("GetIndexOfBookWithCondition") или, наоборот, короткие и непонятные идентификаторы ("a", "abc", "ghti", ...) и т.д. Интересен тот факт, что сложные для понимания имена (например, f34_7dde) существенно уменьшают вероятность возникновения коллизии, в отличие от понятных и четко определенных имен (например state, studentID), для которых вероятность того, что другой программист использует это имя в другой части программы для других задач, существенно возрастает. Собственно, понимая это, некоторые опытные программисты, чтобы избежать такие ситуации, используют весьма оригинальную методику: добавляют к имени переменной прозвище своего домашнего животного, или для каждого модуля используют разную «тему» имен: цветы, животные,

природа и т.д. [Селко, 2006]. Хотя это и позволяет снизить риск возникновения коллизии имен, но значительно ухудшает читаемость кода результирующей программной системы.

Введение единого стандарта именования в теории могло бы решить указанную проблему, но, как показывает практика, все попытки объединить существующие системы, правила, направления и т.д., приводят лишь к возникновению еще одной системы, правила, направления и т.д., обычно не хуже и не лучше предыдущих.

Стоит заметить, что проблема риска коллизии имен возникает уже на первых этапах перехода к языкам программирования высокого уровня: использование имен позволило программистам подавать команды на языке близком к человеческому (первые имена-команды представляли собой сокращение английских слов), но в то же время заставило неявно отказаться от *уникальности*, присущей только адресам. Неявно – потому что этот аспект именования в языках программирования и спецификаций не был закреплен формально со времен определения принципов именования Фреге-Рассела [Фреге, 2000], основанных на теории имен Аристотеля, но во время разработки программного обеспечения в рамках одной команды все же пытались его придерживаться. Оказалось, что при разработке программных систем командами разработчиков, не контактирующих между собой в процессе создания продукта по поводу именования тех или иных объектов, переменных, классов и т.д., контролировать выполнение принципа однозначности гораздо труднее или вовсе невозможно, учитывая тот факт, что использование одного имени для различных переменных не всегда отрицательно влияет на корректность работы программы (например, если это касается локальных переменных). В то же время, неформальные стандарты именования поддерживают идею уникальности имен в пределах одной модели, системы, базы данных.

Если обеспечить уникальность имен в пределах одной системы, имеем следующие преимущества:

• исключение возможности возникновения коллизии имен за счет представления на транслятор кода с уникальными именами;

• возможность введения таких аспектов номинации, присущих естественном языку более, чем формальному, как многозначность, синонимия, косвенное именование, др.;

• возможность использовать теоретические достижения школы программирования В.М. Глушкова, рассматривая множество имен как своеобразные адреса.

Цель работы – исследование и формализация нетрадиционных аспектов именования, таких как иерархичность имен, уникальность именования, его неоднозначность и синонимичность, свойства непрямого (косвенного) именования.

Работа имеет следующую структуру: в следующем разделе будет дано определение иерархических данных, которые является основной моделью данных, построенных на основе отношения именования. Затем будут рассмотрены неклассические аспекты именования, в заключении будут сформулированы основные результаты.

Формальное представление иерархических данных

Практика работы с традиционными моделями данных показывает, что строгая структуризация последних значительно ограничивает их выразительность. Более гибкими и выразительно способными, хотя и более сложными для обработки, являются слабоструктурированные (иерархические) данные [Cooper, 2001], [McHugh, 2012]: данные с нерегулярной и нечеткой структурой, которая может изменяться во время обработки. Такие данные обычно представляют в виде древовидной структуры, ветви которой обозначены именами, а листья – базовыми значениями (рис. 1), т.е. аналогично именному (номинативному) данному [Nikitchenko, 1998]. Работа с такими данными усложнена отсутствием в них

строгой структуры, неоднозначности имен, синонимии и тому подобное. Это требует исследований таких данных и разработки эффективных алгоритмов работы с ними.

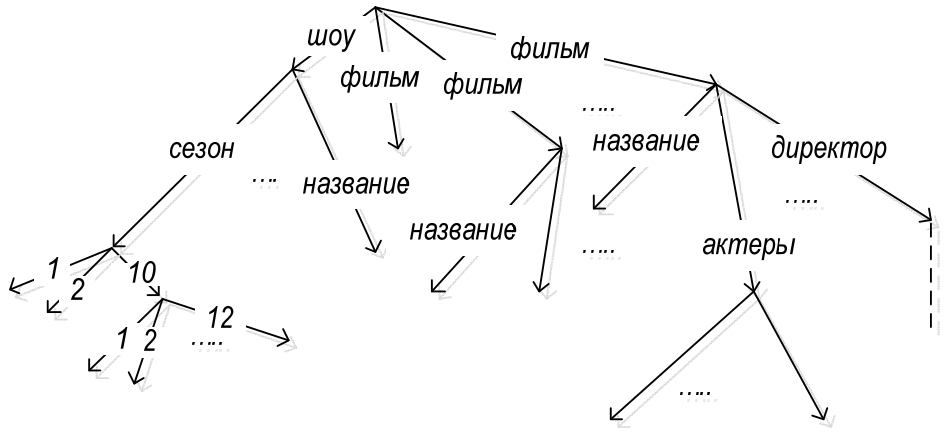


Рис. 1. Пример иерархического данного

Иерархические данные представляют собой множество пар *имя → значение*, где значение может быть как *простым* (определяется абстрактно, как некоторый элемент из множества возможных значений: число, строка, таблица, фильм, дата и т.д., или их идентификаторов: номерной знак автомобиля, идентификационный код человека и т.д.), так и *сложенным* (другим иерархическим данным). Ранг иерархического данного определяет длину максимального пути от корня иерархии до листка. Иерархическое данное ранга 1 являет собой номинативное множество.

Сами данные можно подавать в текстовом (линейном) или в графическом виде. Возьмем, например, такую текстовую запись: $d = [A \longmapsto [B \longmapsto [D \longmapsto 1,\ C \longmapsto [E \longmapsto 2]],\ F \longmapsto 3],\ B \longmapsto 4]$. Ей соответствует определенное ориентированное дерево, с дугами, размечены именами, и конечными вершинами (листьями), размеченными значениями. На рис. 2 представлены различные его формы: а) традиционная и б) схемная.

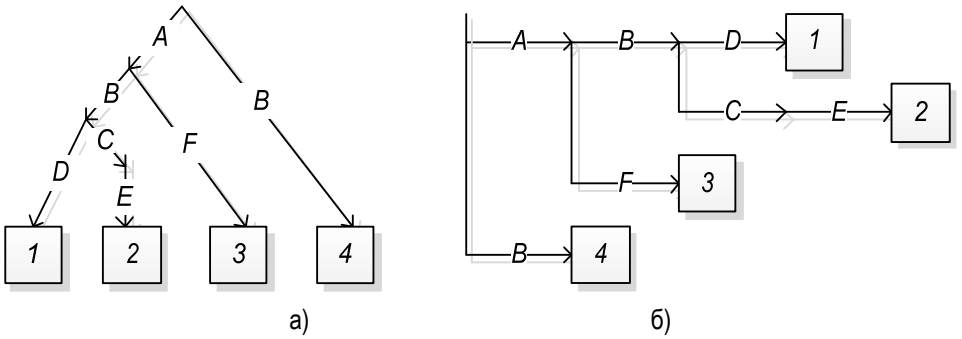


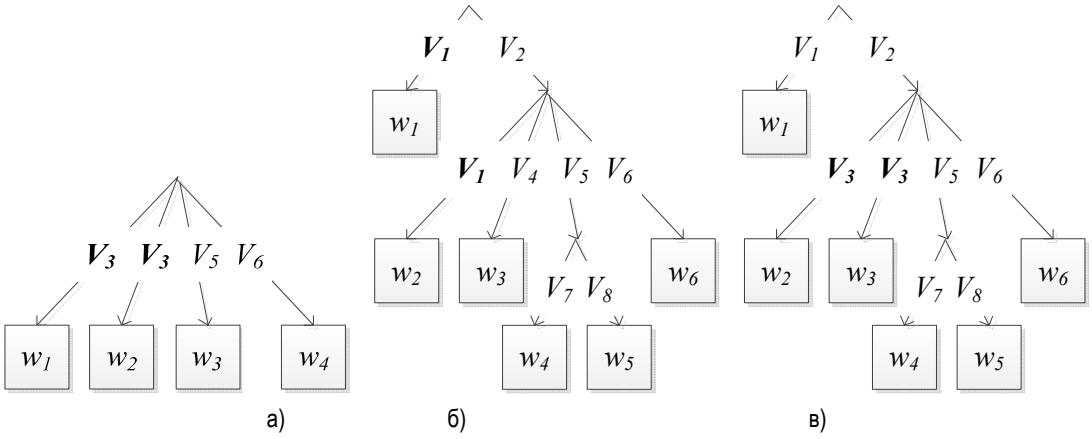
Рис. 2. Графические представления иерархического данного: а) традиционная форма; б) схемная форма.

Аспекты неклассической теории номинации и их формализация

Любую систему данных можно представить в виде иерархического данного с уникальными именами из множества *V*. Доступ к таким данным можно ускорить с помощью индексации. Это было исследовано в работе [Россада, 2011], основным результатом которой является **теорема о стабильности программы при изменении структуры данных**: для любой программы *Pr* и для любого уникальноименного данного *d* выполняется следующее: если $Pr_u(d)\downarrow=r$, то $Pr_i(ind(d))\downarrow=r'$, причем $cnt(r')=r$. Суть теоремы в следующем:

для любой интерпретации Pr_u семантического терма Pr в алгебре уникальноименных данных, его интерпретация Pr_i в алгебре строгоиндексированных данных (которая имеет лучшие характеристики доступа к компонентам данных) будет равнозначной Pr_u. Используя особенности такой системы, можно подавать расширенные структуры данных: с многозначным именем, косвенной адресацией, синонимией и т.д..

На номинативном множестве (иерархическом данном ранга 1) полисемия (многозначное именование) определяется тривиально (Рис. 3 а)): одно имя может иметь несколько значений, эквивалентных в смысле определенного отношения, но не обязательно равных [Россада, 2012].



а) б) в)

Рис. 3. Примеры номинативных данных с полисемией
а) номинативное множество с полисемией; б) иерархическое данное с одноранговой полисемией;
в) иерархическое данное с разноранговой полисемией

Полисемию для иерархических данных можно определять как простую (одноранговую) и разноранговую. Простая (Рис. 3 б)) определяется по аналогии с полисемией определенной для номинативного множества. В случае разноранговой полисемии (Рис. 3 в)) данное изначально должно быть уникальноименным [Россада, 2011а]. Для простой полисемии на номинативном множестве верны следующие результаты.

Теорема 1. Программы языка SICON над номинативными множествами с полисемией мультимонотонны относительно отношения включения данных $\subseteq$ [Россада, 2011].

Теорема 2. Для $\forall d, d' \in RNS$ таких, что $d \sim d'$ и для $\forall f \in SF_R$ выполняется следующее: для $\forall r, r' \in RNS$ таких, что $f(d) = r$ і $f(d') = r'$, значения r і r' будут эквивалентными [Россада, 2012].

Следствие из теоремы 2. Для $\forall d \in RNS$ такого, что $d \sim d$ и для $\forall f \in SF_R$ выполняется следующее: для $\forall r, r' \in RNS$ таких, что $f(d) = r$ і $f(d) = r'$, значения r и r' будут эквивалентными..

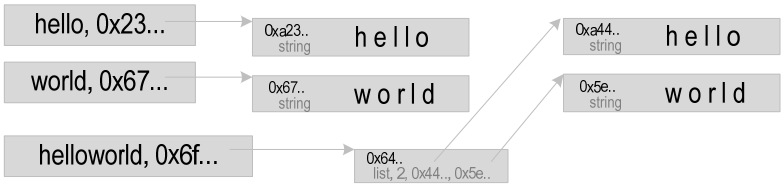
Эти же результаты действительны и для иерархических данных с одноранговой полисемией, которые строятся на основе номинативного множества с полисемией; они так же верны для иерархических данных с разноранговой полисемией, поскольку последние строятся на основе уникальноименных [Россада, 2011] данных, и потому, используя предыдущие наработки, их можно свести к номинативному множеству с полисемией, которое является частным случаем иерархических данных с одноранговой полисемией.

Синонимия является одним из фундаментальных понятий лингвистики (лексическая синонимия и синонимия языковых единиц), логики (бинарное отношение, в котором находятся любые два равнозначные, но не тождественные выражения), логической семантики и семиотики. В логике и логической семантике, зависимости от отношения к общему денотату или сигнификату, различают,

соответственно, экстенсиональную (например, «5» и «2+3») и интенсиональную (например, «5» и «отлично») синонимию.

В формальных языках возможно использование как интенсиональной синонимии так и экстенсиональной [Шрейдер, 1974], но для этого должно быть определено хотя бы одно нетривиальное отношение эквивалентности или равенства. Например, имена корреляций (псевдонимы, alias) в SQL представляют собой местоимения и играют ту же роль, что и местоимения в обычной речи: делают предложения более короткими и удобными для чтения [Селко, 2006]; Synonym (SQL) в Oracle используется для альтернативного именования объектов, что дает возможность по-разному интерпретировать одни и те же данные (аналог номинативных определение), или для упрощения записи длинных имен объектов баз данных, особенно полных имен, состоящие из четырех частей (например ServerName.DatabaseName.OwnerName.ObjectName); синонимы в результатах – функция в поиске Google которая выполняется в процессе обработки запроса путем ответа на вопрос «содержит страница синонимы слов запроса». Примером интенсиональной синонимии в языках программирования является использование переменных, которые ссылаются на одну и ту же область памяти (например, в Fortran таким образом повторно использовуются участки памяти) [Себеста, 2001].

Рассмотрим на примере. Пусть имеем некоторое данное *[hello ⟼ "hello", world ⟼ "world", helloworld ⟼ [0 ⟼ "hello", 1 ⟼ "world"]]*:



На данном этапе имеем разные имена (*hello* и *helloworld.0*, *world* и *helloworld.1*), которые именуют одинаковые значения (то есть, интенсиональные синонимы). Но это неявная синонимия: её невозможно использовать или учитывать при работе программы. Покажем это. Рассмотрим функции f_1 и f_2, которые выводят на экран значения имен *hello* и *world*:

```
def f1():
      print hello
      print world
```

```
def f2():
      for i in range(len(helloworld))
            print helloworld[i]
```

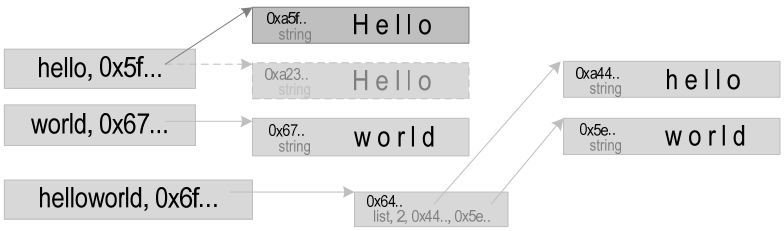
,

и функцию *f*, которая именует значение аргумента *hello*:

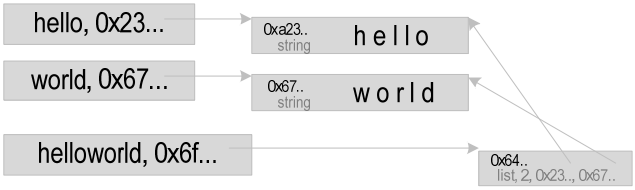
```
def f():
      hello="Hello"
```

.

На заданном входном данном функции f_1 и f_2 возвращают одинаковые результаты, но при выполнении функций $f \bullet f_1$ и $f \bullet f_2$, где f меняет значение одной из переменных, результаты будут отличаться:

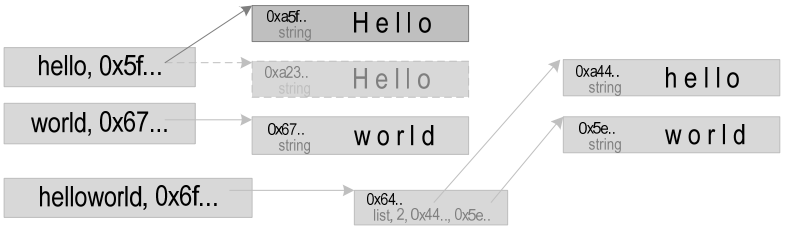


Другое дело, если указанные переменные ссылаются на одну и ту же самую область памяти. В этом случае для большинства языков программирования, которые разрешают создавать имена-синонимы в указанном смысле (например, с помощью указателей) значения функций $f \bullet f_1$ и $f \bullet f_2$ будут совпадать:



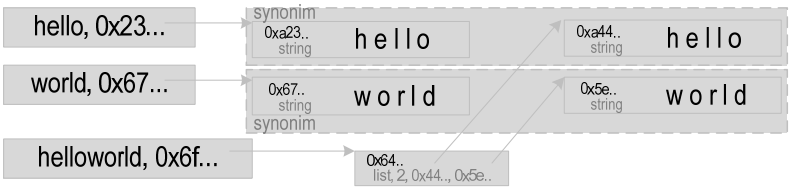
,

но в общем случае синонимия сохраняться не будет: например, одно из свойств языков программирования с динамической типизацией (например, python) – при изменении значения неизменного аргумента (числа, списки) создается новый объект, а не изменяется значение по старому адресу. А это означает, что результаты выполнения функций не будут совпадать:

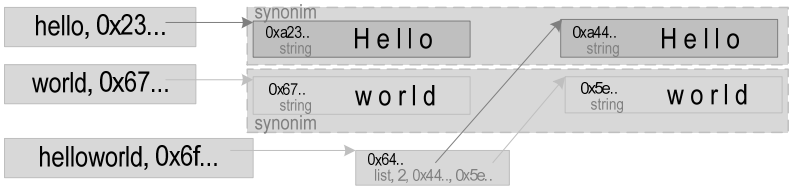


.

Совсем другое дело, если синонимия используется явно, заданием соответствующего отношения на множестве имен переменных:



В таком случае результаты выполнения функций $f \bullet f_1$ и $f \bullet f_2$ будут совпадать:



Синонимия задается введением соответствующего отношения на множестве имен данного [Россада, 2011a]. Синонимия также может быть одноранговой (рис. 4 а), б)) и разноранговой (рис. 4 в)). Основной результат исследования синонимии следующий:

Теорема 3. Для $\forall s_R, s'_R \in NS$ таких, что $s_R \sim_R s'_R$, для некоторого отношения синонимичной эквивалентности R и для $\forall f \in F$: $f(s_R)\downarrow \Leftrightarrow f(s'_R)\downarrow$ и $f(s_R) \sim_R f(s'_R)$ [Россада, 2011a].

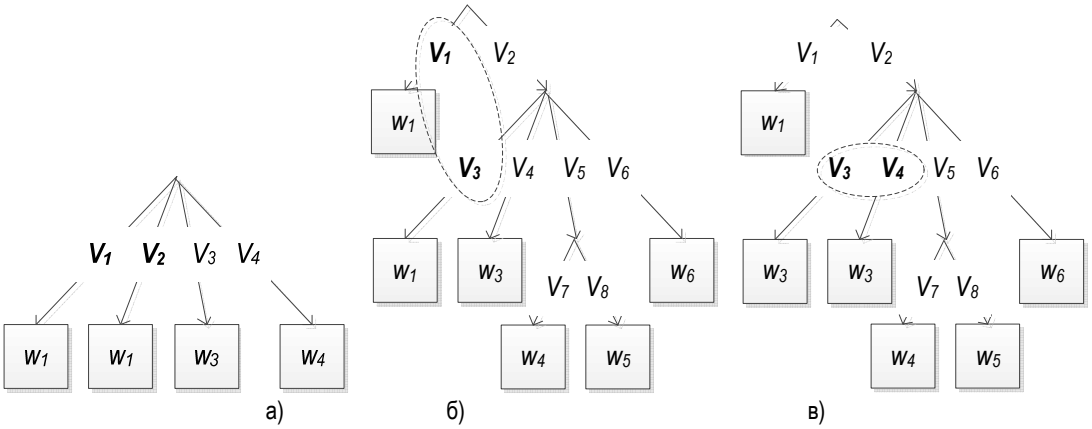


Рис. 4. Примеры номинативных данных с синонимией
а). Номинативное множество с синонимией; б) иерархическое данное с одноранговой синонимией; в) иерархическое данное с разноранговой синонимией

Результаты исследования показывают, что использование аспектов неклассической теории номинации в формальных языках целесообразно для повышения его выразительности при этом без корректность не нарушается.

Заключение

В статье рассмотрены аспекты именования, которые не входят в рамки классической теории именования, более того: противоречат ей. Результаты исследований формальных представлений данных аспектов показывают, что их использование в формальных языках расширяет их выразительность, не нарушая при этом формальность и корректность. Таким образом, несмотря на внешнюю противоречивость основным принципам, предложенная расширенная теория дополняет строгую классическую, подобно тому, как теория комплексных чисел дополняет теорию натуральных и целых чисел.

Благодарности

Автор высказывает благодарность за конструктивные замечания научному руководителю проф. Никитченко Н.С. Работа опубликована при финансовой поддержке проекта **ITHEA XXI** Института информационных теорий и приложений FOI ITHEA Болгария www.ithea.org и Ассоциации создателей и пользователей интеллектуальных систем ADUIS Украина www.aduis.com.ua.

Литература

[Cooper, 2001] Cooper B. A fast index for semistructured data / B. F. Cooper, N. Sample, J. F. Michael, G. R. Hjaltason, M. Shadmon // VLDB. – 2001. – P. 341-350.

[McHugh , 2012] McHugh J. Indexing Semistructured Data [електронный ресурс] / J. McHugh, J. Widom, S. Abiteboul, Q. Luo, A. Rajaraman // Technical Report, January 1998. Режим доступу: http://infolab.stanford.edu/lore/pubs/semiindexing98.pdf. Дата звертання: 01.05.2012.

[Nikitchenko, 1998] Nikitchenko N.S. A Composition Nominative Approach to Program Semantics // Technical Report IT-TR: 1998-020. – Technical University of Denmark.– 1998.– 103 p.

[Бирюков, 2000] Бирюков Б.В. Готтлоб Фреге: современный взгляд / Б.В. Бирюков // Г. Фреге. Логика и логическая семантика. Сборник трудов – М., 2000. – С. 33.

[Гнеденко, 1961] Гнеденко Б.В. Элементы программирования / Гнеденко Б.В., Королюк В.С., Ющенко Е.Л. – М.: Физматгиз, 1961. – 348 с.

[Нікітченко, 2009] Нікітченко М.С., Іванов Є.В. Властивості композиційно-номінативних мов програм з асоціативним розіменуванням / М.С. Нікітченко, Є.В. Іванов // Матеріали XVI Всеукраїнської наукової конференції "Сучасні проблеми прикладної математики та інформатики" – Львів, 2009. – С. 157-158.

[Селко, 2006] Джо Селко. Стиль программирования Джо Селко на SQL / Санкт Петергбург : Русская редакция, 2006. - 196 с. - ISBN 5-469-01396-0.

[Россада, 2011] Россада Т.В. Класи номінативних даних із швидким доступом до компонентів / Т.В. Россада // Тези конференції «Шевченківська весна». – Київ, 2011 – С. 49-52.

[Россада, 2011а] Россада Т.В. Моделювання та дослідження синонімії у формальних мовах / Т.В. Россада // Тези конференції DSMSI – Київ 2011 – ст. 404.

[Россада, 2012] Россада Т.В. Формалізація багатозначного іменування у мовах програм над номінативними даними / Т.В. Россада // Вісник Київського національного університету імені Тараса Шевченка. Сер.: фізико-математичні науки. – 2011 – вип. 1. – С.217-222.

[Россада, 2012а] Россада Т.В., Скляр А.В. Властивості слабкоструктурованих даних з багатозначним іменуванням та їх використання / Т.В. Россада, А.В. Скляр // Вісник Київського національного університету імені Тараса Шевченка. Сер.: фізико-математичні науки. – 2012 – вип. 2.

[Себеста, 2001] Себеста Р. Основные концепции языков программирования / Себеста Р. – М.: Вильямс, 2001. – 659 с.

[Фреге, 2000] Фреге Г. О смысле и значении / Г. Фреге // Логика и логическая семантика: Сборник трудов. – М.: Аспект Пресс, 2000. – С. 230-247.

[Шрейдер, 1974] Шрейдер Ю. А. Логика знаковых систем / Шрейдер Ю. А. – М.: Едиториал УРСС, 1974. – 64 с.

Информация об авторе

Россада Т.В. *– КНУ имени Тараса Шевченко, факультет кибернетики, аспирант; e-mail: trossada@gmail.com*

Основные области научных исследований: теоретическое программирование, формальные языки, программные алгебры