УДК 004.8:681.3.

А.Ф. Кургаев, С.Н. Григорьев

Анализ доминирующих моделей представления и использования знаний

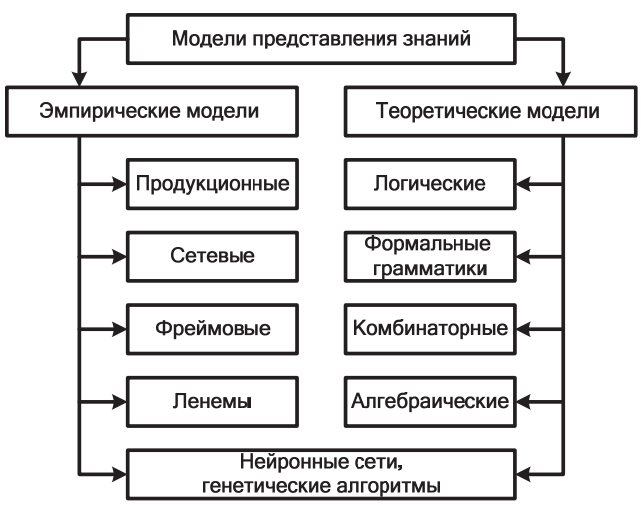
Выполнен анализ специфики, достоинств и недостатков эвристических моделей представления знаний (продукционных, сетевых, фреймовых), а также использования различных вариантов метаязыкового представления. Показано, что ни одна из известных моделей представления знаний не имеет решающих преимуществ – у каждой свои существенные недостатки. Сделано заключение о целесообразности развития метаязыковых моделей представления знаний.

Analysis of the specific character, advantages and disadvantages of the heuristic models of knowledge representation (production, network, frame-based models) and applying the different variants of meta-language representation models is performed. It is shown that none of the known knowledge representation models have any decisive superiority over the other models, each model has its own disadvantages. It is advisable to focus on the development of the different meta-language knowledge representation models.

Виконано аналіз специфіки, переваг і недоліків евристичних моделей подання знань (продукційних, мережевих, фреймових), а також використання різних варіантів метамовного подання. Показано, що жодна з відомих моделей подання знань не має вирішальних переваг, – у кожної свої істотні недоліки. Зроблено висновок про доцільність розвитку метамовних моделей представлення знань.

Введение. Развитие языков и систем программирования связывают с представлением знаний в интеллектуальных информационных системах и компьютерах.

Известно множество моделей представления знаний [1]. Основываясь на этих моделях и их определенных сочетаниях, создано и используется много языков представления знаний. Большинство моделей реальных систем – гибридные. В методических целях известнейшие модели классифицируют в рамках схемы (рисунок).



Ключевые слова: модели представления знаний, продукционные, сетевые, фреймовые системы, метаязыки.

Первый подход, называемый эмпирическим, основан на понимании организации человеческой памяти и моделировании механизмов решения задач. Согласно этому подходу разработаны и получили наибольшую популярность:

- продукционные модели, основанные на правилах в виде предложений типа: «ЕСЛИ условие, ТО действие;
- сетевые модели (или семантические сети) – в инженерии знаний их понимают как граф, отображающий содержание целостного образа. Узлы графа соответствуют понятиям и объектам, а дуги – отношениям между объектами;
- фреймовая модель основывается на понятии фрейма (*frame* – рамка, каркас). Фрейм – структура данных для представления некоторого концептуального объекта. Информация, принадлежащая фрейму, содержится в его слотах. Последние могут быть терминальными или сами быть фреймами, образуя таким образом единую иерархическую сеть.

В [2] и ряде других публикаций уже исследована задача анализа моделей представления знаний. Цель исследований в данной статье – уточнение известных результатов и расширение области анализа металингвистическими моделями представления знаний.

Эмпирические модели

Продукционная система включает в себя : базу правил *IF-THEN*, указывающих, какие выводы в соответствующих ситуациях должны быть сделаны или нет, глобальную базу данных и интерпретатор правил. База правил – область памяти, содержащая базу знаний в форме последовательности правил вида:

$$i; Q; P; A \Rightarrow B; N.$$

Здесь элемент i – *имя продукции*, выделяющее эту продукцию из базы знаний. Элемент Q характеризует *сферу проблемной области применения продукции*. Основной элемент продукции – ее *ядро*: $A \Rightarrow B$. Интерпретация ядра продукции может быть разной и зависит от условия A, слева от знака *секвенции* $\Rightarrow$. Ядро продукции имеет смысл: **ЕСЛИ** A, **ТО** B. Более сложные конструкции ядра допускают в правой части альтернативный выбор, например, **ЕСЛИ** A, **ТО** B_1, **ИНАЧЕ** B_2. Секвенция может толковаться в обычном логическом смысле, как знак логического следования B из истинного A (если A ложно, то о B ничего сказать нельзя). Возможны и другие интерпретации ядра продукции, например A описывает некоторое условие, необходимое, чтобы совершить действие B.

Элемент P – это *внешнее условие применимости* ядра продукции в форме логического выражения (предикат). Ядро продукции активизируется только тогда, когда P – истинно. Элемент N описывает *постусловие* продукции, указывающее изменения в базе знаний после выполнения продукции. Оно актуализируется, лишь если реализовалось ядро продукции. Если в памяти системы хранится некоторый набор продукций, они образуют систему продукций. В системе продукций должны быть заданы специальные процедуры управления продукциями, с использованием которых происходит актуализация и выбор для выполнения той или иной продукции из числа актуализированных.

Глобальная база данных – это область памяти, содержащая факты (описание начальных, текущих данных и состояний системы). Базы данных имеют разную форму, но могут быть описаны как массив, содержащий имя данных, атрибуты и значения атрибутов.

Интерпретатор реализует вывод заключений, пользуясь базой правил и данных. Механизм вывода включает в себя поиск в базе знаний, выполнение операций над знаниями и формирование выводов; обычно реализуется определенное сочетание прямых (восходящих, управляемых данными) и обратных (нисходящих или ориентированных на цель) выводов.

Представление знаний правилами и вывод на них легко понимаемы, поскольку близки силлогизмам, а однообразие формы обеспечивает легкую модификацию системы продукций введением и/или удалением одних правил независимо от содержания других.

Эти свойства – достоинство при постановке простых, однородных по смыслу задач (в форме переходов между состояниями), но ведут к падению эффективности решения проблем в составе разнородных задач. Даже для простой проблемы трудно построить систему управления знаниями как единым целым, поэтому весь процесс управления должен контролироваться человеком.

Широкое применение систем, основанных на правилах, обусловлено следующим.

• *Модульная организация*. Благодаря этому упрощается представление знаний и расширение системы методом инкрементной разработки.

• *Наличие средств объяснения*. Поскольку антецеденты точно указывают условия активизации правил, с их помощью легко создавать средства объяснения в форме восстановления процесса рассуждений для получения заключения.

• *Аналогия с процессом познания человека*. Правила – естественный образ моделирования процесса решения задач, упрощающий формализацию знаний экспертов.

Правила принадлежат к типу продукций, идея которых начата в работах 1940-х гг. В символьной логике продукционные системы впервые использовал Е. Пост, доказавший, что любая система математики или логики может быть оформлена в виде системы продукционных правил определенного типа. В отличие от обычного языка программирования, такого как *C* или *C++*,

порядок, в котором записаны правила, не имеет значения.

Основным ограничением продукционных правил Е. Поста относительно программирования есть отсутствие стратегии управления упорядочением вызова правил. Система Е. Поста позволяет применять правила к строкам любой формы.

Одной из лучших реализаций продукционных систем на сегодня считается инструментальная среда *CLIPS* (*C Language Integrated Production System* – язык Си, интегрированный с продукционными системами), предназначенная для разработки баз знаний и экспертных систем (ЭС) [3]. Сейчас *CLIPS* свободно распространяется через интернет (*http*:// *www.ghg.net*/*clips*/*CLIPS.html*), в частности для преподавания предмета «Экспертные системы» в сотнях университетов всего мира:

- наиболее широко используется (в том числе в государственных организациях и учебных заведениях) в качестве инструментальной среды для разработки ЭС, благодаря скорости, эффективности и бесплатности;
- исходный код программного пакета *CLIPS* распространяется свободно, его можно установить на любой платформе, поддерживающей стандартный компилятор языка Си. Существующую версию можно эксплуатировать на платформах *UNIX*, *DOS*, *Windows* и *Macintosh*;
- хотя теперь и стала общественным достоянием, *CLIPS* до сих пор поддерживается ее автором, Гери Райли (*Gary Riley*);
- содержит полноценный объектно-ориентированный язык *COOL* для написания ЭС;
- хотя *CLIPS* и написана на языке Си, ее интерфейс намного ближе к языку программирования *LISP*;
- во многом схожа с языками, созданными на базе *LISP*, в частности *OPS5* и *ART*;
- имеет четко сформулированный синтаксис;
- включает в себя множество апробированных конструкций из других инструментальных средств;
- допускает вызов внешних функций, написанных на других языках программирования, а модули, написанные на *CLIPS*, могут быть вызваны программами, написанными на других языках;
- может быть обеспечена возможность работы в реальном масштабе времени, когда реакция системы на возмущение не должна превышать нескольких миллисекунд. *CLIPS* –сегодня лучший выбор для работы в реальном времени.

Основные недостатки систем продукций:

- трудности составления продукционного правила, адекватного элементу знания, при наличии ограничения возможностей выразить сложные правила;
- невозможность вызова одного правила из другого (связь между правилами косвенная, только через данные);
- отсутствие внутренней структуры, ведущее к неразрешимости проблемы непротиворечивости базы знаний;
- отсутствие зависимости шагов вывода от стратегии выбора, что усложняет их интерпретацию.

Следствие. Продукционным системам не хватает строгой теории. Пока у них властвует эвристика. При заданной модели проблемной области в виде совокупности продукций нельзя быть уверенным в ее полноте и непротиворечивости. Причина неудач создания теории кроется в расплывчатости понятия продукции, в той интерпретации, которая приписывается ядру, а также в разных способах управления системой продукций.

Семантические сети – это классический образ представления пропозициональной информации в искусственном интеллекте (пропозициональным утверждением, высказыванием может быть предложение истинное или ложное). Высказывания имеют форму декларативных знаний, поскольку в них утверждаются факты. С формальной точки зрения семантическая сеть – это отмеченный ориентированный граф.

Семантические сети впервые разработал в 1968 г. Квиллиан для исследований в области искусственного интеллекта как образ описания человеческой памяти и языка. С того времени семантические сети успешно применя-

лись для решения многих задач представления знаний. Понимание смысла с помощью семантических сетей позволяет выйти за пределы возможностей программного обеспечения простых экспертных систем или искусственного интеллекта.

Связи в семантической сети представляют отношения, а узлы – объекты, концепты или ситуации. Для семантических сетей отношения имеют исключительное значение, поскольку придают знаниям базовую структуру, позволяющую выводить новые знания.

Семантические сети называют ассоциативными сетями, поскольку одни узлы в таких сетях ассоциированы (связаны) с другими. В оригинальной работе Квиллиана человеческая память моделировалась ассоциативной сетью, в которой, если при чтении слов предложения стимулируется один узел, то активизируются его связи с другими, и такая активность распространяется по сети.

К двум наиболее распространенным связкам принадлежат связки *is-a* (экземпляр) и *a-kind-of* (АКО – подмножество, подкласс). При этом более общий класс, на который указывает стрелка АКО, называется суперклассом. Если суперкласс имеет связь АКО, указывающую на другой узел, то он одновременно есть классом суперкласса, на который указывает стрелка АКО.

Все объекты класса должны иметь некоторые общие атрибуты, каждый из которых приобретает значение. Комбинация атрибута и значения называется *свойством*. В семантических сетях можно также найти связи других типов.

Повторение характеристик узла у его потомков называется *наследованием*. Если не утверждается обратное, то считается, что все элементы некоторого класса наследуют все его свойства. Наследование – полезное средство в представлении знаний, поскольку позволяет не указывать повторно общие характеристики. Связи и наследование есть основой эффективных способов представления знаний, давая возможность показывать сложные отношения с помощью нескольких узлов и связок.

Типизация семантических сетей обусловливается смысловым содержанием образующих их отношений. Например, если дуги сети выражают родовидовые отношения, то такая сеть определяет классификацию объектов предметной области. Аналогично, наличие в сети причинно-следственных (каузальных) отношений позволяет интерпретировать ее как сценарий. Построение сети на ассоциативных отношениях формирует ассоциативную структуру понятий рассматриваемого фрагмента предметной области. При практическом использовании семантических сетей для представления знаний решающее значение имеют унификация типов объектов и выделение базовых видов отношений между ними.

Очевидные достоинства сетевой модели – ее высокая общность, наглядность отображения системы знаний, а также понятность. В то же время в семантической сети наблюдается смешение групп знаний, относящихся к различным ситуациям при назначении дуг между вершинами, что усложняет интерпретацию знаний. Другая проблема сетевой модели состоит в трудности унификации процедур вывода и механизмов управления ими на сети [4].

Признаны следующие недостатки семантических сетей.

• Одна из проблем, связанных с применением семантических сетей, заключается в том, что не предусмотрены стандартные определения для имен связок. Например, нередко связки *is-a* используют для представления и индивидуальных, и общих отношений, т.е., кроме ее значения, ей придают и значение связки АКО.

• Отсутствует однозначное определение семантической сети.

Перспективным направлением повышения эффективности сетевого представления, развиваемым в настоящее время, считается использование онтологии и параллельных выводов на семантических сетях.

Основные аспекты специфики онтологического подхода. Это представление, во-первых, содержит как формальные, так и описательные (выражаемые на естественном языке) компоненты.

Если первое важно для логического вывода (новых знаний, решений, рекомендаций, оценок

и пр.), то второе – для представления человеку смысла, стоящего за каждым действием интеллектуальной системы. Во-вторых, для отражения семантики определяются все используемые термины, а это требует спецификации общих терминов в рамках онтологии верхнего уровня. Фактически наблюдается иерархия онтологий: онтологии общих знаний на вершине, затем предметные онтологии и онтологии задач. В-третьих, онтологический подход, как правило, предполагает общение интеллектуальной системы с пользователями на языках, близких к естественным (формальные языки применяются программистами, реализующими оболочки для работы с онтологиями) [4].

Фреймовая система – модель представления знаний, основанная на теории М. Минского, как один из подходов к описанию знаний, пригодный для понимания сцен и языка. В фреймовой системе единица представления информации – объект, называемый фреймом. Он есть формой представления некоторой ситуации, описываемой совокупностью понятий и данных.

Каждый фрейм имеет имя, единое во фреймовой системе, и определенную внутреннюю структуру на множестве именованных слотов, которые также имеют определенную структуру данных. Однажды установленную структуру фрейма можно менять лишь в деталях.

Эта модель основана на свойстве концептов иметь аналогии и иерархические структуры отношений типа «абстрактное–конкретное». Ее применение ограничено случаями четкой иерархии между фрагментами данных или знаний. Примерами могут быть классификации растений, животных, неисправностей аппаратуры, заболеваний людей и др.

Характеристики фрейма:

- имя – символ, уникальный в данной системе;
- положение в иерархической структуре задается указателями на родительский фрейм и список дочерних фреймов;
- информация относительно фрейма содержится в слотах, каждый слот – это атом или список, первый элемент которого – всегда ключ (имя слота);
- присоединенные процедуры – служебные программы как значение слотов. Их запуск – по сообщениям из других фреймов (аналоги методов в объектно-ориентированном программировании – ООП).

Итак, фрейм – это форма описания знаний (в текущей ситуации при решении данной задачи) фрагмента предметной области. В процессе адаптации обобщенного фрейма к конкретным условиям выполняется уточнение значений слотов, изначально определенных по умолчанию. При поиске в памяти фрейма, релевантного некоторому образу, на первом этапе проверяется совпадение всех существенных слотов фрейма–кандидата с соответствующими составляющими образа. На втором этапе значения слотов, заданные по умолчанию, согласуются с прочими аспектами образа. Механизм подобного согласования предусматривает выявление качественной сопоставимости компонентов фрейма и образа (например, принадлежность атрибутов образа интервалам, определенным в слотах по умолчанию), после чего значения по умолчанию конкретизируются.

Организация вывода во фреймовой системе базируется на обмене сообщениями между фреймами, на активации и выполнении присоединенных процедур. Отражение в иерархии фреймов родовидовых отношений обеспечивает возможность реализации в рамках фреймовой модели операции наследования, позволяющей приписывать фреймам нижних уровней иерархии свойства, присущие фреймам вышележащих уровней. Аналогичные механизмы наследования используются в настоящее время в объектно-ориентированном программировании.

С созданием теории фреймов появились и языки (*FRL*, *KRL*, *RLL*, *FMS*, *KEE*, *KRINE*, *LOOPS* и др.), описывающие формальные процессы в виде программ действий, выполняемых для каждого объектного мира (фрейма). Программы вызываются из соответствующего фрейма, а при общении между фреймами осуществляется межфреймовый обмен информацией или передача управления.

К *недостаткам* фреймовых систем относят следующие.

• Поскольку знания во фреймовой системе описываются в процедурной форме, это представление, по-сути, есть расширением обычных систем процедурного типа. Поэтому здесь сложнее (чем в других моделях) приобретение и изменение знаний.

• Использование фреймов допустимо для сравнительно небольших проблем, а с ростом сложности проблемы возрастают трудности понимания описания, описание и управление становятся более сложными, чем в традиционных процедурных системах.

• Во фреймовой системе не решена проблема выявления семантических противоречий.

При переходе к сложным комплексным предметным областям, предполагающим знания различных типов, возникает потребность совмещения в одном языке представления знаний различных концепций. К моделям подобного типа относится двухуровневая схема в составе *L-языка спецификации знаний* и *базовой формальной системы* (БФС) [5].

БФС – язык представления знаний, объединяющий разные концепции обработки знаний: семантическую сеть, фрейм, систему продукций. Семантика *L*-языка полностью описывается в терминах БФС. *L*-язык как бы надстраивается над БФС путем ввода терминальных выражений и специальных конструкций. Единица *L*-языка – *ленема* – конструкция, задающая схему описания понятия и по форме напоминающая фрейм. Спецификация модели прикладной области и конкретных фактов в форме выражений *L*-языка существует только на внешнем, пользовательском уровне, а собственно хранение и обработка информации осуществляются на уровне БФС.

К числу достоинств этой модели относят: сочетание существенных для разработчика простоты и однородности языка с удобством для пользователя (инженера знаний) развитых средств поддержки процессов создания моделей прикладных областей.

Основной недостаток – отсутствие единой теории, разнородность концепций обработки знаний не соответствует принципу концептуального единства Ф. Брукса, приоритет которого утверждается в [6]: «концептуальное единство является самым важным соображением при проектировании системы».

Таким образом, ни одна из эвристических моделей представления знаний не удовлетворяет ни содержательным, ни формальным требованиям представления произвольных знаний. На этом фоне остается единственный вариант – перейти на более высокий уровень абстракции, на уровень метаязыков.

Теоретические модели

Второй подход (объединяющий теоретические модели на рисунке) считают теоретически обоснованным, гарантирующим правильность решений. Он в основном представлен моделями, основанными на формальной логике (исчисление высказываний и предикатов), формальных грамматиках, комбинаторными моделями, в частности моделями конечных проективных геометрий, теории графов, тензорными и алгебраическими моделями. Признано [1], что в рамках этого подхода до сих пор удавалось решать *лишь сравнительно простые задачи из узкой предметной области*.

Среди известных теоретических моделей представления знаний лишь генеративная грамматика целенаправлена на формализацию знаний, теории произвольной языковой компетенции как сущностной основы языковой деятельности.

Для строгого и точного описания языков программирования используют специальные *метаязыки* (языки для описания других языков) [7]. Наиболее распространенными метаязыками стали *металингвистические формулы Бекуса–Наура* (БНФ) и расширенные Бекусо–Науровы формы (РБНФ).

Метаязыком БНФ представляют спецификацию произвольного языка в виде системы взаимосвязанных формул, похожих на математические. Для каждого понятия языка существует единая метаформула (нормальная форма). Она составляется из левой и правой частей. В левой части указывается определяемое понятие, нетерминал, а в правой – множество допустимых конструкций языка, которые объединяются в это понятие.

В формуле используют специальные метасимволы:

- символ «::=» – разделяет левую и правую части формулы, его смысл эквивалентен словам «есть по определению»;
- символы «< >» – угловые скобки выделяют нетерминалы (представленные произвольной символьной строкой) – определяемые понятия языка;
- символ «|» – «или» разграничивает альтернативные варианты определений в правой части формул;
- терминальные символы записываются как есть, без использования каких-то метасимволов.

Метаязык БНФ впервые был применен для описания Алгола-60, а также использован Н. Виртом при описании языка Паскаль. БНФ и других метаязыков (Хомского, Хомского – Щутценберже) достаточно для описания синтаксиса произвольных языков. Но отсутствие в нотации метаязыков средств явного задания повторений создает ряд трудностей. Во-первых, определения оказываются сложными для понимания, недостаточно наглядными из-за насыщенности рекурсиями. Во-вторых, возникают проблемы с тем, что грамматики, дающие подходящие семантические деревья, оказываются леворекурсивными.

Для удобства и компактности описания в метаязык вводят дополнительные конструкции. В частности, специальные метасимволы были разработаны для описания необязательных цепочек, повторяемых цепочек, обязательных альтернативных цепочек. Существуют разные расширения формы метаязыков, незначительно отличающихся между собой. Их разнообразие чаще всего объясняется желанием разработчиков языков программирования по-своему описать создаваемый язык. К примерам таких широко известных метаязыков можно отнести: метаязык Вирта [8], использованный при описании Модулы-2, метаязык Кернигана–Ритчи, описывающий Си, и др.

Главные модификации БНФ касаются введения скобок для повторений вхождения цепочек терминалов и нетерминалов в правые части формул. Соглашения относительно обозначений терминалов и нетерминалов также изменены.

Метаязыковым БНФ (или РБНФ) описанием служит набор правил, определяющих отношение между терминалами и нетерминалами. Терминалы – те элементы структуры, что собственной структуры не имеют; это идентификаторы (имена, которые полагают заданными для данного описания) или цепочки – последовательности символов в кавычках или апострофах, определенные вне метаязыкового описания. Нетерминалы – элементы структуры, имеющие собственные имена и структуру. Каждый из них определяется правилами, фиксирующими его зависимость от одного или более терминалов и/или нетерминалов.

Семантика правила РБНФ: нетерминал, заданный идентификатором слева от знака «=», определяется некоторым отношением терминалов и нетерминалов. Полное описание структуры есть набор правил, определяющих все нетерминалы так, что каждый из них может быть сведен к комбинации терминалов путем последовательного (рекурсивного) применения правил.

Набор основных конструкций РБНФ: конкатенация, альтернативный выбор и итерация, а дополнительных, стилистических – отношение необязательности (необязательный элемент выражения выделяют квадратными скобками) и структурные круглые скобки (употребляются для группирования элементов при формировании сложных выражений).

Конкатенация определяется последовательной записью символов выражения, разделяемых одним или более пробельными символами. Правило вида **A = B C.** обозначает, что нетерминал **A** состоит из двух символов – **B** и **C**.

Альтернативный выбор обозначается вертикальной чертой. Правило **A = B|C|D.** обозначает, что нетерминал **A** может состоять или из **B**, или из **C**, или из **D**.

Итерацию – конкатенацию любого числа (включая нуль) элементов обозначают фигурными скобками, выделяющими итерируемые элементы. Правило вида **A = {B}.** обозначает, что **A** – или пустой, или есть конкатенацией некоторого числа символов **B**. В варианте БНФ

отношение итерации описывается с помощью рекурсии.

Метаязык РБНФ пригоден для описания практически значимых языков. В том числе средствами РБНФ определяют и собственно метаязык РБНФ [8]:

```
описание = определение { определение }.
определение = имя_понятия тело_определения тчк.
имя_понятия = идентификатор.
идентификатор = буква { буква | цифра }.
тело_определения = есть_структура выражение.
выражение = элемент {отношение_ИЛИ
    элемент | отношение_И элемент}.
элемент = имя_понятия | строка | "(" выражение ")" |
    "[" выражение "]" | "{" выражение "}".
строка = кавычка {знак} кавычка.
```

В этом описании термины "тчк", "буква", "цифра", "есть_структура", "кавычка", "знак" – терминальные.

Метаязык (тот или иной) используется для конструирования спецификации объектного языка программирования, согласно которой в языке реализации разрабатывается программа распознавателя – анализа и построения дерева вывода (разбора) утверждений объектного языка программирования. Построение дерева разбора подтверждает принадлежность входной цепочки символов данному языку [7–9].

В настоящее время как развитие классических инструментов *YACC* и *LEX* созданы и используются многие специализированные средства разработки синтаксических анализаторов, различающихся конкретной реализацией некоторой версии метаязыка РБНФ, алгоритмами разбора, объектными языками и др. По результатам анализа возможностей современных инструментов в [10] сделано заключение, что в настоящее время нет инструмента для создания анализаторов, превосходящего другие генераторы по всем показателям. На практике наиболее удобны инструменты, использующие *LL*-алгоритм, среди которых предпочтительнее других выглядят *ANTLR*, *JavaCC* и *Coco*/*R*, порождающие анализаторы, реализованные по методу рекурсивного спуска, что облегчает чтение и отладку сгенерированного кода.

Однако в качестве языка представления знаний метаязык РБНФ (и все другие известные метаязыки) имеет определенные недостатки.

- Созданный первично для узкоспециальных целей и такой, что полностью их обеспечивает, метаязык РБНФ не есть функционально полным языком и потому не пригоден для представления знаний произвольных прикладных областей.
- Семантический разрыв [7, 11] между формальным (в метаязыке) описанием языков программирования и методами реализации трансляторов этих языков приводит к необходимости преодолевать ряд проблем при разработке трансляторов:
  - ♦ первая из них – необходимость преобразования модели формальной грамматики в автоматную модель распознавателя. Между этими моделями существует несколько противоречий:
    - ✓ противоречие между иерархической структурой исходного описания языка с использованием формальной грамматики и одноуровневой табличной моделью автомата, реализующего синтаксический разбор;
    - ✓ противоречие между высокоуровневыми средствами описания языков и ограничениями на грамматики, используемые для перехода к автоматной модели распознавателя;
    - ✓ формальное описание языка в итоге должно быть преобразовано в модель автомата, осуществляющего разбор. Это преобразование основывается на эвристических алгоритмах, каждый из которых ориентирован на конкретный класс грамматик и автоматов. В результате необходимости применения данного шага теряется связь между исходным описанием языка и его реализацией.
  - ♦ вторая проблема – это необходимость преобразования грамматики. Для преобразования исходного описания языка в соответствующую автоматную модель необходимо, чтобы грамматика принадлежала некоторо-

му конкретному классу. Если это условие не выполняется, то автомат не может быть реализован, что и вынуждает преобразовать грамматику для реализации автомата. Однако по ряду причин желательно избегать таких преобразований: во-первых, трудно выбрать вид преобразования, а во-вторых, невозможно гарантировать, что они не изменят создаваемый язык.

Принцип синтаксически управляемой обработки данных в таких приложениях, как трансляция языков программирования, давно и успешно используется в форме управления процессом трансляции согласно синтаксической структуре предложений входного языка. Более изощренное применение этого принципа – непосредственное задание структуры решения некоторой задачи.

В *SYNTAX*-технологии для спецификации трансляций используется метаязык *TSL* (*КС*-уровня), который есть входным языком технологического комплекса *SYNTAX*, предназначенного для разработки средств синтаксически управляемой обработки данных. В применении к реализации языков программирования *TSL* служит для описания синтаксиса и семантики языков в форме трансляционных грамматик [12].

Трансляционная грамматика состоит из *управляющей грамматики* и *описания операционной среды*. Управляющая грамматика – это контекстно свободная грамматика с правилами, в которых помимо нетерминалов и терминалов можно использовать дополнительные *семантические* и *резольверные* символы. Описание операционной среды определяет ее как некоторое пространство данных – элементов операционной среды, а интерпретацию дополнительных символов – как множество преобразований и предикатов над текущим состоянием операционной среды.

Это означает, что программа состоит из коллекции объектов, организованных для конкретного применения посредством некоторой трансляционной грамматики. Объекты представляют собой данные и методы их обработки, а управляющая грамматика определяет возможные последовательности вызовов этих методов. Грамматика порождает некоторый класс вычислений в форме множества цепочек терминал–действий и семантик, реализуемых посредством методов. Конкретное вычисление выбирается из этого множества цепочек в зависимости от текущего состояния операционной среды.

Достоинства такой архитектуры:

- разделение синтаксического и семантического уровня спецификации и реализации трансляции, связь между которыми реализуется посредством контекстных символов;
- инвариантность программы относительно любых преобразований ее структуры управления; действительно, реализация такой программы представляется фиксированной процедурой, функционирующей под управлением структуры, специфицированной исходной грамматикой.

Этот подход поддерживает *парадигму программирования*, согласно которой каждое правило грамматики определяет некоторую структуру программ. При этом любая конструкция продолжает вычислительный процесс, уже выполненный ее составляющими, и требуется лишь использовать предшествующие результаты для выработки своего собственного. Поэтому при написании правила грамматики для некоторой конструкции, достаточно определить, какие действия над результатами ее подконструкций должна совершить данная конструкция. Благодаря такой *концептуальной модульности*, воплощаемой в правилах грамматики, значительно облегчается разработка программы.

К числу недостатков *SYNTAX*-технологии [12] можно отнести:

- деформацию исходной синтаксической структуры (*КС*-уровня) до автоматного уровня рабочей (регуляризированной) грамматики;
- ее исследовательский характер, неразработанность относительно классов решаемых задач, ограничения применений, а также незавершенность практической реализации.

Заключение. Таким образом, ни одна из известных моделей представления знаний не имеет решающих преимуществ перед другими, – у каждой свои существенные недостатки.

Целесообразна разработка иерархической системы моделей представления знаний, соот-

ветствующей принципу концептуального единства Ф. Брукса и удовлетворяющей требованиям универсальности – представлению разнородных знаний. Один из первых претендентов на ядро такой системы – развитие метаязыковых формализмов, а в качестве универсальной формы представления разнородных знаний – структура научной теории.

1. *Классификация* моделей представления знаний. – *http://www.aiportal.ru/articles/knowledge-models/classification.html*
2. *Кургаев А.Ф.* Анализ моделей представления знаний // Нові комп'ютерні засоби, обчислювальні машини та мережі. – К.: Ін-т кібернетики ім. В.М. Глушкова НАН України, 2001. – Т. 1. – С. 129–135.
3. *Chapter* 7. Introduction to CLIPS. – *http://ir.nuk.edu*.tw:8080/*ir*/*bitstream*/310360000*Q*/11342/2/*CLIPS_Intro-CLIPS.pdf*
4. *Башмаков А.И.*, *Башмаков И.А.* Интеллектуальные информационные технологии: Учеб. пособие. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2005. – 304 с.
5. *Искусственный* интеллект: В 3-х кн. Кн.2. Модели и методы: Справочник / Под ред. Д.А. Поспелова. – М.: Радио и связь, 1990. – 304 с.
6. *Брукс Ф.П.* Как проектируются и создаются программные комплексы. Мифический человеко-месяц: Очерки по системному программированию. – М.: Наука, 1979. – 152 с.
7. *Softcraft* разноликое программирование: Основы разработки трансляторов. – *http://sl-ur.narod*2.*ru/studentu/vtoroi_semestr*_2010_*g/pyavu/Osnovy_razrabotki translyatorov_.rar*
8. *Вирт Никлаус*. Построение компиляторов. – М.: ДМК Пресс, 2010. – 192 с.
9. *Грогоно П.* Программирование на языке Паскаль. – М.: Мир, 1982. – 384 с.
10. *Чемоданов И.С.*, *Дубчук Н.П.* Обзор современных средств автоматизации создания синтаксических анализаторов // Системное программирование. – 2006. – Т. 2, **1**. – С. 268–296.
12. *Майерс Г.* Архитектура современных ЭВМ: В 2-х кн. Кн. 1. – М.: Мир, 1985. – 364 с.
13. *Мартыненко Б.К.* Синтаксически управляемая обработка данных. – СПб: Изд-во СПбГУ, 2004. – 316 с.

Поступила 16.01.2014

Тел. для справок: +38 050 881-6218 (Киев)
E-mail: *afkurgaev@ukr.net*

●