Российская академия наук
Сибирское отделение
Институт систем информатики
им. А. П. Ершова

МОЛОДАЯ ИНФОРМАТИКА

СБОРНИК ТРУДОВ
АСПИРАНТОВ И МОЛОДЫХ УЧЕНЫХ

Под редакцией
к.ф.-м.н. И. С. Ануреева

Новосибирск 2005

Сборник содержит статьи, представленные аспирантами и молодыми сотрудниками ИСИ СО РАН по следующим направлениям: теоретические аспекты программирования, информационные технологии и информационные системы, системное программное обеспечение, прикладное программное обеспечение.

© Институт систем информатики им А. П. Ершова СО РАН, 2005

Siberian Division of the Russian Academy of Sciences
A. P. Ershov Institute of Informatics Systems

YOUNG INFORMATICS

COLLECTION OF PAPERS
OF GRADUATE STUDENTS
AND YOUNG SCIENTISTS

Novosibirsk 2005

The volume contains the papers presented by post-graduates and young researchers of A.P. Ershov Institute of Informatics Systems which concern the following research areas: theoretical aspects of programming, informational technologies and information systems, system software and applied software.

© A. P. Ershov Institute of Informatics Systems, 2005

ПРЕДИСЛОВИЕ

Цель сборника — стимулирование научной деятельности аспирантов и молодых сотрудников (до 35 лет) Института систем информатики СО РАН и их обучение качественному представлению научных работ. При обучении использовалось двухэтапное рецензированием работ, и в сборник включались те статьи, которые были доработаны с учетом рецензий. Работы принимались в рамках тематики института по следующим направлениям: теоретические аспекты программирования, информационные технологии и информационные системы, системное программное обеспечение, прикладное программное обеспечение.

В работе «Система знаний информационного интернет-портала по научной тематике» предложен подход к организации такой системы знаний на примере портала по археологии. Система знаний портала представляется как совокупность взаимосвязанных онтологий, описывающих научную деятельность и ее участников, а также специфику определенной научной дисциплины. Выделение онтологии науки, независимой от предметной области, позволит использовать предложенную схему для расширения и настройки портала на другие направления научной деятельности.

В работе «Интегрированная среда визуального функционального программирования SFP» кратко описываются возможности и архитектура такой интегрированной среды. Данная среда предназначена для создания и отладки функциональных программ, предназначенных для выполнения на компьютерах с параллельной архитектурой, на персональных компьютерах.

В работе «Элиминация механизма исключений при переводе из языка C#-light в язык C#-kernel» рассматривается алгоритм элиминации механизма исключений в контексте перевода из языка C#-light в язык C#-kernel. Для полноты изложения дано краткое описание языков C#-light, C#-kernel, а также метапеременных и имен функций языка аннотаций.

Эффективный алгоритм, реализующий замкнутый набор булевых операций над полигональными областями, разработанный в ИСИ СО РАН, при вычислениях с ограниченной точностью имеет ряд проблем, связанных с численной устойчивостью. Использование целочисленного представления координат для решения этих проблем приводит к возможности образования в некоторых случаях вырожденных рёбер, приводящих к неправильной работе алгоритма. В работе «Реализация замкнутого набора булевых операций над полигональными областями на дискретной сетке» предлагается модификация этого алгоритма, решающая проблему вырожденных рёбер и обладающая несколько большей эффективностью. Предлагаемая модифи-

кация позволяет эффективно выполнять объединение, пересечение, разность и симметрическую разность над множествами многоугольников, гранично заданных на дискретной сетке координат. При этом сохраняется замкнутость набора операций, т.е. результат работы алгоритма удовлетворяет требованиям к его входным данным.

Цель работы «Наглядное представление работы алгоритмов глобальной интервальной оптимизации» — реализация пакета программ, позволяющего решать интервальные задачи глобальной оптимизации функции двух переменных, наглядно демонстрировать процесс работы различных алгоритмов, отображать и сравнивать результаты. Некоторые из используемых алгоритмов разработаны в рамках данной работы.

Автоматический перевод спецификаций выполнимых протоколов в формальные модели объединяет выразительность языков выполнимых спецификаций и удобство анализа и верификации формальных моделей. В работе «Трансляция SDL-спецификаций с динамическими конструкциями в раскрашенные сети Йенсена» описывается автоматическая трансляция SDL-спецификаций в иерархические раскрашенные сети, предложенные Йенсеном и реализованные в системе CPN Tools. Моделируются динамические конструкции SDL. Для анализа графов достижимости сетевых моделей может использоваться как система CPN Tools, так и другие средства автоматической верификации методом проверки моделей.

В настоящее время онтологии широко используются при разработке информационных систем для описания данных используемых семантических сетей. В работе «Обзор методов применения схем данных и онтологий для объединения распределенных гетерогенных информационных ресурсов» рассматривается применение схем данных и онтологий для объединения распределенных гетерогенных источников информации. На основе проведенных в этой области исследований рассматривается зависимость методов и архитектуры объединения информационных ресурсов от выбранного способа построения общей онтологии, описывающей информационные ресурсы результирующей системы. Также уделяется внимание методам поддержки эволюции и версионности программных систем, которые предоставляются онтологиями, а также дальнейшим исследованиям в этой области.

В работе «Расширение возможностей системы ФинПлан на основе структурных моделей» рассказывается о структурных моделях как средстве представления недоопределенных вычислительных моделей, а также о разработанной на их основе схеме представления и взаимодействия недоопределенных вычислительных моделей в системе ФинПлан.

Работа «Об анализе тестовой эквивалентности дискретно-временных сетей Петри» посвящена исследованию эквивалентностных понятий моделей параллельных систем, функционирующих в режиме реального времени. В рамках работы вводятся и исследуются понятия временных тестовых эквивалентностей в модели дискретно-временных сетей Петри. Дается альтернативная характеризация данных эквивалентностей, позволяющая разработать алгоритмы их распознавания.

В работе «Методы интеллектуальной обработки документов, основанные на экспертных знаниях» представлена методика интеллектуальной обработки документов, которая решает две основные задачи: автоматическая индексация и адресация деловых документов. Предлагаемое решение ориентировано на понимание основного содержания документов и базируется на интеграции знаний о языке и жанре документов и знаний о предметной области.

В работе «Графический метаязык для описания транслятора» описывается наглядное графическое представление алгоритма трансляции, позволяющее генерировать детерминированный распознаватель. Также вводится модель упрощенного магазинного автомата, описывающая класс таких распознавателей.

В работе «Системы переписывания графов: выбор лидера и распознавание топологии в анонимных сетях» рассматривается основанное на базе систем переписывания графов построение алгоритмов выбора лидера в анонимных сетях с топологией хордального или слабо хордального графа, а также алгоритмы распознавания принадлежности топологии сети классам хордальных и слабо хордальных графов. Предложенные алгоритмы используют память в вершинах, не зависящую от размера всей сети, что является улучшением по сравнению с существующими универсальными алгоритмами. Приведен результат о невозможности построения алгоритмов распознавания, имеющих более простые правила переписывания.

PREFACE

The purpose of the volume is to stimulate research activity of post-graduates and young researchers of A.P. Ershov Institute of Informatics Systems and to train them in qualitative presentation of scientific papers. Training has been based on two-stage paper reviewing, and the volume contains only those papers which were improved according to reviewers' remarks. To be accepted, the paper should cover one of the research topics of the Institute, such as: theoretical aspects of programming, information technologies and information systems, system software and application software.

The paper "The knowledge system of informational Internet-portal on scientific subjects" presents an approach to organization of such a system by the example of an archeological portal. The knowledge system of the portal is represented as a collection of related ontologies which describe research activity and its participants, as well as the specific features of a particular branch of science. Construction of domain-independent ontology of science makes the knowledge portal easily adjustable to a given field of science.

The paper "Integrated environment SFP for visual functional programming" briefly describes the functionality and architecture of SFP. This environment is intended for creation and debugging of functional programs oriented to computers with parallel architecture and personal computers.

The paper "Exception handling elimination while translating from C#-light into C#-kernel" considers an algorithm of exception handling elimination used in translation from C#-light into C#-kernel. For completeness of presentation, the C#-light and C#-kernel languages, as well as metavariables and the names of functions of the annotation language are briefly described.

An efficient algorithm, developed in the Institute of Informatics Systems for implementation of a closed set of Boolean operations on polygonal regions, has numerical stability problems with finite precision numeric computations. The use of integer representation of coordinates intended to fix these problems may give rise to new degenerate edges and failure of the algorithm. The paper "Implementation of a closed set of Boolean operations over polygonal regions on a discrete grid" presents a modification of an algorithm that solves the problem of edge degeneration and increases the algorithm's efficiency. The presented algorithm performs union, intersection, difference and symmetrical difference on polygonal regions represented by their boundaries on a discrete grid. The improved algo-

rithm provides a closed set of Boolean operations, i.e. the output of the algorithm satisfies its input restrictions.

The paper "Visual representation of running the algorithms of global interval optimization" describes implementation of a software package intended to solve interval problems of global optimization for a two-variable function, to visualize the process of execution of different algorithms, to show and compare their results. Some of these algorithms have been elaborated within this project.

The automatic translation of Formal definition technique specifications to a formal model combines expressiveness of FDT-languages with convenience of analysis and verification of formal models. The paper "Translation of SDL-specifications with dynamic constructions to Coloured Petri Nets by Jensen" describes the automatic translation of SDL-specifications to hierarchical coloured nets, proposed by K. Jensen and implemented in the CPN Tools. Dynamic SDL constructions are modelled. For reachability graph analysis, CPN Tools or other automatic verification tools may be used.

At present, ontology is widely used in the development of large scalable information systems for description of semantic web data.

The paper "Survey of methods of application of ontology and data schemes to integration of distributed heterogeneous information resources" discusses the use of data schemes and ontologies in integration of distributed heterogeneous information resources. Based on research conducted in this area, it is considered how the integration solution methods and architecture depend on the way of creating a common ontology which describes information resources of the whole system. Additionally, the ontology-based methods for support of evolution and versioning of software systems are taken into account. Possible directions of future research in this area are also discussed.

The paper "Extension of capabilities of the FinPlan system using structure models" presents an approach to representation of subdefinite computing models by structure models and a scheme of representation and cooperation of subdefinite computing models in the FinPlan system using structure models.

The paper "Analysis of timed Petri Nets based on testing equivalences" is devoted to the development and analysis of equivalence notions of concurrent and real-time systems. In particular, testing equivalences and preorders are defined and studied for timed Petri Nets. We also present algorithms for deciding the equivalences.

The paper "Document processing methods based on expert knowledge" describes the methods of intellectual processing of documents. We consider two main problems: automatic indexing and addressing of documents. Our solution is directed to understanding of document content and is based on integration of

knowledge about the language and genre of the documents and knowledge about the subject domain.

The paper “Graphical metalanguage for translator specification” describes an intuitive graphical representation of the translation algorithm which allows us to generate a deterministic recognizer. A model of a simplified stack automaton that describes a class of such recognizers is also introduced.

The paper “Graph rewriting systems: leader selection and topology recognition in anonymous nets” considers the algorithms for leader selection in an anonymous net with the topology of a chordal or weakly chordal graph. In addition, the algorithms for recognition of chordality and a weak chordality of the net's graph are presented. The algorithms are based on the graph rewriting systems. They utilize memory of the graph nodes which is independent of the net size, and this is an improved result as compared to other universal algorithms. The result about impossibility of simplification of rewriting rules for topology recognition is also presented.

О.И. Боровикова, С.В. Булгаков, Е.А. Сидорова[1]

СИСТЕМА ЗНАНИЙ ИНФОРМАЦИОННОГО ИНТЕРНЕТ-ПОРТАЛА ПО НАУЧНОЙ ТЕМАТИКЕ

1. ВВЕДЕНИЕ

В настоящее время огромный объем научных знаний представлен на различных информационных ресурсах (Интернет-сайтах, электронных библиотеках и архивах). Однако доступ к этим знаниям сильно затруднен. Основными причинами этого являются несистематизированность и слабая структурированность ресурсов, распределенность информации по различным Интернет-сайтам, а также недостаточная формализованность самой области знаний.

Для решения данных проблем существует ряд подходов, одни из которых направлены непосредственно на унификацию или реорганизацию данных [1], другие ориентированы на унификацию средств доступа к ним [2].

Предлагаемый нами подход направлен на построение специализированных Интернет-порталов знаний [3], ориентированных на работу с множеством разнородных ресурсов или источников данных по определенной научной тематике. Информационную основу портала знаний составляет онтология, применение которой позволяет сочетать принципы вышеперечисленных подходов: на ее основе обеспечивается как сведение ресурсов в единое информационное пространство, так и содержательный доступ к ним через Интернет.

Портал знаний предоставляет возможность поиска информации по совокупности различных аспектов научной деятельности (например, поиск информации об ученых, занимающихся определенной научной деятельностью). А использование онтологии для построения системы знаний портала позволяет не только целостно представить проблемную область, но и учитывать предпочтения пользователя. Пользователю предоставляется возможность настраивать интерфейс портала, скрывая не нужные понятия и связи, существующие в онтологии.

[1] Работа выполняется при финансовой поддержке РФФИ (проект № 04-01-00884а) и СО РАН (Междисциплинарный интеграционный проект № 149)."

Практическим приложением этого подхода является создание специализированного Интернет-портала знаний по археологии и этнографии.

2. ПОРТАЛ ЗНАНИЙ

С системной точки зрения портал знаний представляет собой специализированную информационную систему, снабженную эргономичным пользовательским web-интерфейсом.

С точки зрения пользователя портал является тематическим Интернет-ресурсом, обеспечивающим возможность поиска и просмотра информации в рамках заданной предметной области (археология и этнография). При разработке архитектуры портала учитывались недостатки и преимущества существующих порталов [4, 5], а также такие требования со стороны пользователей, как его полнота с профессиональной точки зрения, удобство и простота использования.

Как информационный ресурс портал выполняет следующие функции:

- обеспечивает доступ к информации по различным аспектам и участникам научной деятельности, таким как: составляющие научной дисциплины (подразделы дисциплины, методы и техники исследования, используемые термины и понятия), персоналии исследователей, информация по группам, сообществам, организациям, включенным в процесс исследования;
- позволяет интегрировать в единое информационное пространство близкие по тематике ресурсы, представленные в сети Интернет (реляционные базы данных, XML- и HTML-ресурсы, новостные каналы и т.п.);
- предоставляет средства поиска интересующей пользователя информации в рамках всего информационного пространства портала;
- обеспечивает информационную поддержку пользователей ресурса (например, анонсирование разного рода событий и мероприятий);
- поддерживает гибкий пользовательский интерфейс, позволяющий учитывать предпочтения пользователя по работе с ресурсом и предоставляемыми сервисами.

3. СИСТЕМА ЗНАНИЙ ПОРТАЛА

Основу системы знаний портала составляет онтология и соотнесенное с ней описание соответствующих сетевых ресурсов. Онтология описывает

структуру проблемной области и включает множество классов понятий, связывающих понятия отношений, а также ограничений, определяющих интерпретацию используемых понятий. Использование в качестве основы портала набора онтологий делает систему знаний портала легко расширяемой и настраиваемой — в нее могут интегрироваться как новые знания (например, о новых направлениях той или иной науки), так и новые типы информационных ресурсов.

3.1. Компоненты системы знаний

На рис. 1. представлена иерархия знаний, используемых в предлагаемом подходе.

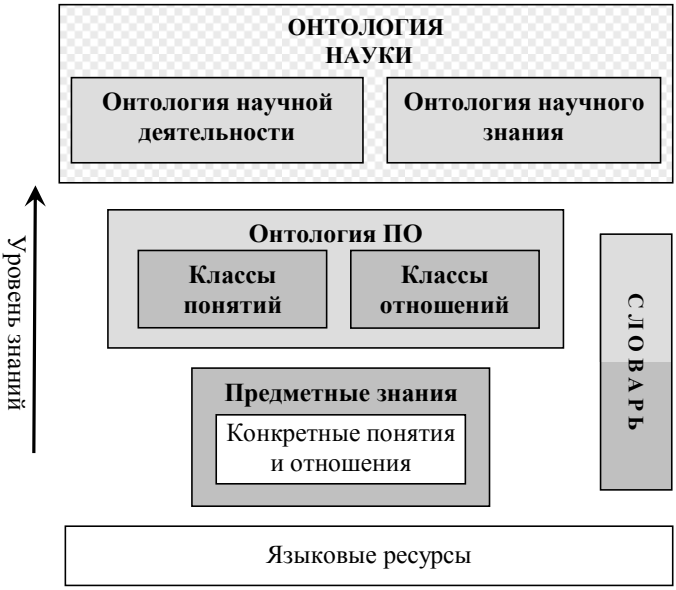


Рис. 1. Система знаний портала

Рассматриваются следующие компоненты системы знаний.

- **Онтология науки** представляет собой ядро системы знаний и отражает сущность решаемых задач данной предметной области (ПО). Данная онтология фиксирует те базовые содержательные структуры, которые используются для построения онтологий бо-

лее низкого уровня (онтологии ПО) и определяют структуру информационной базы портала. Онтология науки условно разделена на онтологию научной деятельности и онтологию научного знания;

- **Онтология научной деятельности** включает общие классы понятий, относящиеся к организации научной деятельности.
  - *Ученые.* К этому классу относятся понятия, связанные с субъектами научной деятельности: исследователями, сотрудниками и членами организаций, исторически-значимыми персонажами и другими людьми.
  - *Организации.* Понятия этого класса описывают различные организации, научные сообщества и ассоциации, институты, исследовательские группы и другие объединения.
  - *События.* В этот класс входят понятия, описывающие научно-организационную или научно-исследовательскую деятельность — научные мероприятия, конференции, исследовательские поездки, проекты, программы и т.п.
  - *Публикации.* Этот класс служит для описания различного рода публикаций и материалов, представленных в печатном или электронном форматах (монографии, статьи, отчеты, труды конференций, периодические издания, фото- и видеоматериалы и др.).
  - *Информационный ресурс.* Этот класс служит для описания информационных ресурсов, представленных в сети Интернет.
- **Онтология научного знания** содержит метапонятия, задающие структуры для описания рассматриваемой предметной области.
  - *Раздел науки*. Этот класс позволяет структурировать науку, выделять в ней значимые разделы и подразделы.
  - *Метод исследования* и *Объект исследования*. Понятия этих классов задают типизацию методов и объектов исследования и структуры для их описания.
  - *Научный результат*. К этому классу относятся такие понятия, как открытия, новые законы, теории и методы исследования. Обычно, именно научные результаты находят свое отражение в публикациях, размещаемых на различных информационных ресурсах.
- **Онтология ПО (археологии)** отражает общие знания о ПО, такие как иерархия классов понятий, семантические отношения на этих классах.

- **Предметные знания** — выделенная часть знаний об археологии, которая содержит только частные понятия и конкретные отношения данной ПО.
- **Онтология языка документов (словарь)** — это система языковых средств выражения онтологии ПО. Лингвистическая информация представлена в словаре с помощью функциональных групп лексических единиц, выделенных классов понятий и набора дополнительных атрибутов, отражающих специфику выражений: синонимы, омонимы, составные понятия и т.п.
- **Языковые ресурсы —** исходные данные для системы знаний, характеризующие предметную область пользователя. Обеспечить автоматическое извлечение знаний из этих данных является главной задачей эксперта при наполнении и настройке системы знаний портала.

3.2. Онтология портала

Таким образом, разработанная нами система знаний включает в себя следующие относительно независимые онтологии:

1) онтология научной деятельности,
2) онтология научного знания,
3) онтология предметной области, описывающая конкретную научную дисциплину.

Такое структурирование системы знаний в виде онтологий, большая часть которых является предметно-независимыми, значительно упрощает настройку портала на выбранную область научных знаний.

3.2.1. Схема онтологии портала

На рис. 2 представлен эскиз общей схемы онтологии портала знаний для такой научной дисциплины, как археология. Он включает онтологии науки и научного знания, а также соотнесенный с ними фрагмент онтологии археологии. На данной упрощенной схеме показаны не все связи, существующие между изображенными на ней понятиями. Но в целом схема отражает основные понятия онтологии портала и связи между ними и является основой для построения полной модели.

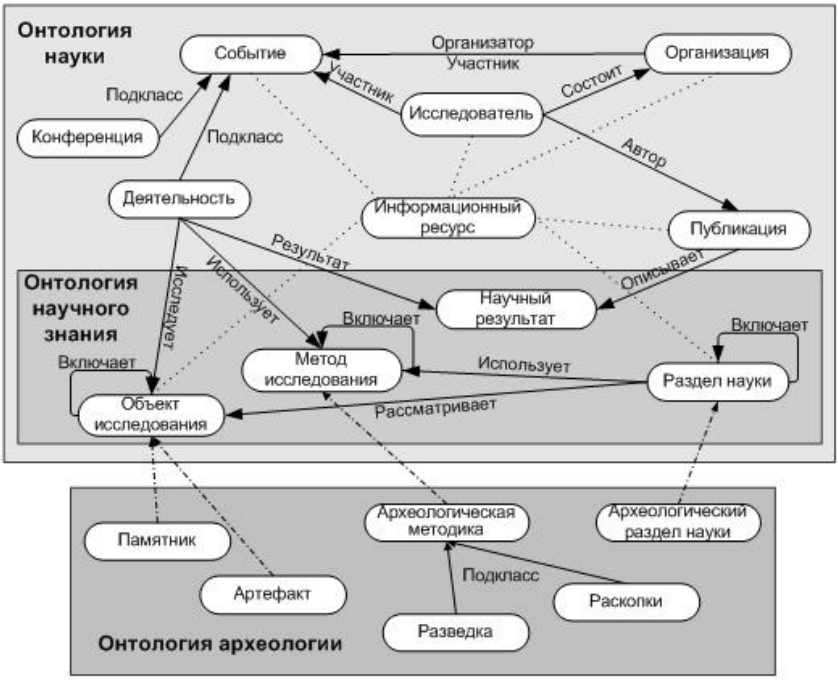


Рис. 2. Упрощенная схема онтологии портала

3.2.2. Онтология ПО археологии

Онтология предметной области описывает научную дисциплину в целом как раздел науки и включает формальное и неформальное описания понятий и отношений между ними. Эти понятия являются реализациями метапонятий онтологии научного знания и могут быть упорядочены в иерархию общее-частное и часть-целое. Для гуманитарных наук очень важны методы исследования и объекты исследования. В частности, в археологии *Методам исследования* будут соответствовать такие понятия, как методика раскопки, методика археологической разведки, а в качестве *Объектов исследования* будут выступать культуры, памятники, артефакты.

Поскольку портал по археологии ориентирован на разные по профессиональному уровни группы пользователей, то было предложено использо-

вать две классификации археологической науки, которые должны быть связаны для корректной работы портала.

- *Системная классификация,* предложенная Ю.П. Холюшкиным и Е.Д. Гражданниковым [6], в большей степени ориентирована на подготовленного пользователя. Классификация имеет многомерную сложную структуру, которая, с одной стороны, достаточно полно и профессионально описывает ПО и позволяет при поиске лучше отразить содержательную часть запроса, с другой стороны, она усложняет доступ к информационным ресурсам, например, при навигации по порталу.
- *Упрощенная классификация,* принятая на большинстве Интернет-ресурсов по археологии, позволяет пользователю легко ориентироваться на сайте, но не достаточно глубоко и профессионально отражает и использует представленные знания.

3.3. Информационный ресурс

Важным компонентом информационного наполнения портала является описание информационных ресурсов. Описание информационного Интернет-ресурса включает специфические атрибуты, а также связи, определяющие его взаимоотношения с элементами онтологии. Набор атрибутов и связей основан на стандарте Dublin Core [7].

Онтология портала, с одной стороны, выступает основой интегрирования различных ресурсов в рамках портала, с другой — является основой для информационного поиска по этим ресурсам [8].

При добавлении ресурсов устанавливается соответствие между онтологией портала и системой терминов ресурса.

В информационном пространстве портала интегрируются следующие типы ресурсов:

- неструктурированные ресурсы — текстовое представление данных;
- слабоструктурированные ресурсы — например, xml-документы;
- структурированные источники данных (СИД) — внешние базы данных, к которым есть права доступа.

При добавлении ресурсов они содержательно (с точки зрения нашей онтологии) индексируются, и их индексы заносятся во внутреннюю БД, по которым в дальнейшем и ведется поиск. Индекс ресурса включает специфические атрибуты и связи, определяющие взаимоотношения с элементами онтологии:

class ИнформационныйРесурс
Название: строка;
Описание: строка;
Тип: строка(сайт, страница, портал, исполняемый файл, графика, мультимедиа, ссылка);
Формат: строка; // html, xml, doc, txt, rdf, архив, …
Язык: строка; //ISO-стандарт
Тип_доступа: строка(платный, бесплатный);
Права_доступа: строка(свободный, регистрация);
URL: строка;
Дата_создания: Дата;
Дата_обновления: Дата;
Число посетителей: Число;
relations:
Ресурс-Человека, Ресурс-Организации, Ресурс-События, Ресурс-НаучногоМероприятия, Ресурс-Публикации;
ТематикаРесурса, Ресурс-ОбъектаИсследования, Ресурс-МетодаИсследования;
end;

При работе с СИД мы можем не копировать информацию во внутреннюю БД (как это происходит при индексировании), а динамически подключать их к порталу. Для этого во внутренней БД в унифицированном виде будут храниться описания схем данных внешних источников и их соответствия элементам онтологии портала. Установление таких соответствий выполняется экспертом-настройщиком.

При поиске информации пользователю предоставляется возможность задания запроса не только и не столько по ключевым словам, сколько в терминах ПО. В такой поисковый запрос входят следующие элементы.

- *Понятия*, являющиеся элементами онтологии.
- *Ограничения*, которым должны удовлетворять найденные данные. Ограничения могут быть заданы в виде выражений над значениями атрибутов понятий ПО.

Сформулированный таким образом запрос представляется как фрагмент онтологии с дополнительными ограничениями. Этот запрос преобразуется в один или несколько запросов к внутренней базе данных портала и/или к подключенным к порталу внешним СИД.

4. ЗАКЛЮЧЕНИЕ

Использование в качестве основы портала совокупности онтологий и выделение предметно-независимой онтологии науки делает систему знаний портала легко расширяемой и настраиваемой на различные научные дисциплины. Еще одним важным достоинством предложенного подхода к организации тематического информационного портала является возможность содержательного доступа не только к собственным информационным ресурсам, но и к внешним структурированным источникам данных по близкой тематике.

К настоящему времени разработана архитектура портала по археологии [9], формально описаны онтологии науки и научного знания. Завершается разработка начальной версии онтологии археологии. Реализованы пользовательский интерфейс системной классификации [10], а также интерфейс администратора для наполнения БД-портала.

СПИСОК ЛИТЕРАТУРЫ

1. **Data** Warehousing Technology. — http://www.kenorrinst.com/dwpaper.html
2. **ANSI/NISO** z39.50-2003. Information Retrieval (z39.50): Application Service Definition and Protocol Specification. — Approved November 27, 2002 by the American National Standards Institute — NISO Press, Bethesda, Maryland, U.S.A., 2003. — 276 p.
3. **Боровикова О.И., Загорулько Ю. А.** Организация порталов знаний на основе онтологий // Труды междунар. семинара Диалог'2002 по компьютерной лингвистике и ее приложениям. — Т.2. — Протвино, 2002. — С. 76-82.
4. **Портал** “Археология России”. — www.archeologia.ru
5. **Портал** “Социально-гуманитарное и политологическое образование”. — www.humanities.ru
6. **Холюшкин Ю.П., Гражданников Е.Д.** Системная классификация археологической науки (элементарное введение в археологическое науковедение). — Новосибирск: Изд-во ИДМИ Минобразования, 2000. — 58 с.
7. **Using** Dublin Core. — http://dublincore.org/documents/usageguide/
8. **Булгаков С.В.** Подход к построению мультиагентной системы содержательного поиска во множестве разнородных структурированных источников данных // Труды IX конференции по искусственному интеллекту КИИ-2004. — М.: Физматлит, 2004. –Т.2. — С.706–714.
9. **Боровикова О.И., и др.** Концепция интеллектуального Интернет-портала знаний для доступа к информационным ресурсам по археологии и этнографии // Тр.

VI-й междунар. конф. «Проблемы управления и моделирования в сложных системах». — Самара: Самарский Научный Центр РАН, 2004. — С. 215–220.

10. **Сектор** археологической теории и информатики ИАЭТ СО РАН. — http://www.sati.archaeology.nsc.ru/sibirica/encycs.html

М. П. Глуханков

ИНТЕГРИРОВАННАЯ СРЕДА ВИЗУАЛЬНОГО ФУНКЦИОНАЛЬНОГО ПРОГРАММИРОВАНИЯ SFP[1]

ВВЕДЕНИЕ

Для выполнения сложных научных расчётов может потребоваться использование суперкомпьютеров, имеющих параллельную архитектуру. При этом вполне естественно выглядит желание создавать и отлаживать необходимые программы на персональных компьютерах, которые не обладают такой архитектурой.

С целью решения этой задачи был разработан язык SISAL [1–9]. Он позволяет создавать программы, эффективно исполняющиеся как на машинах с последовательной, так и с параллельной архитектурой. SISAL является функциональным языком и обладает следующими важными семантическими качествами: он математически правильный (т. е. его функции отображают входы в выходы без побочных эффектов), имена в нём прозрачны для ссылок (т. е. они олицетворяют значения, а не ячейки памяти), это язык однократного присваивания. Результаты исполнения программ всегда детерминированы, несмотря на параллельную архитектуру. Пользователь освобождён от большинства сложностей, связанных с параллельным программированием. Ему не нужно заботиться о синхронизации потоков, исполняемых в реальном времени.

Программы на языке SISAL удобно представлять в виде графов потоков данных. На них основаны представления программ IF1, IF2 и IR1 [10–13]. Такие представления не только наглядны для пользователя, но и удобны для преобразований и интерпретации программ. Их можно применять для представления программ и на других языках [11], есть реализация транслятора с языка Fortran в IF1 [18].

Существуют реализации трансляторов различных версий языка SISAL, среды программирования на нём, а также ведутся исследования в области оптимизации SISAL-программ [8,16].

[1] Работа выполнена при финансовой поддержке Научной программы “Университеты России” (грант УР.04.01.027).

Разрабатываемая в лаборатории конструирования и оптимизации программ ИСИ СО РАН система SFP предназначена для создания и отладки параллельных программ, предназначенных для исполнения на суперкомпьютерах, на персональных ЭВМ. Предполагается создать среду программирования, позволяющую транслировать SISAL-программы в промежуточные представления (IR1, IR2), на которых можно будет производить различные оптимизации и отлаживать программы, а также транслировать внутренние представления на другие языки программирования с настройкой на целевую архитектуру суперкомпьютеров. Важной особенностью среды является её способность к расширению, притом что её компоненты являются частью единой визуальной интерактивной среды. Интерактивность заключается в том, что оптимизаторы, отладчики и другие компоненты системы могут взаимодействовать с пользователем через визуализаторы промежуточных представлений программ и их исходных текстов. Например, пользователь может указать, на каких элементах внутреннего представления следует выполнить некоторое преобразование или по какой ветви следует продолжить исполнение программы после достижения заданной точки программы при её интерпретации.

ИНТЕГРИРОВАННАЯ СРЕДА ПРОГРАММИРОВАНИЯ

Интегрированная среда — это набор программных средств, позволяющий решать определённые задачи. Интегрированная среда программирования как минимум предполагает наличие текстового редактора программ и транслятора или интерпретатора. Кроме этого, среда может иметь различные другие редакторы, оптимизирующие трансляторы, интерпретаторы и отладчики, визуализаторы или редакторы представлений программ, поддержку работы с модулями программы, а также средства взаимодействия этих сервисов внутри среды и с пользователем. Пользовательский интерфейс под операционной системой Windows [17] обычно представляет собой одно главное окно с меню, панелями инструментов, панелями управления и рабочей областью. На панелях инструментов в основном располагаются кнопки с пиктограммами, панели управления могут отображать другие элементы управления и отображать дополнительную информацию, в основной рабочей области располагаются различные редакторы либо на вкладках, либо в отдельных окнах.

РАСШИРЯЕМОСТЬ

Имеется большое поле для исследований в области параллельных функциональных языков программирования [21-24]. Необходимо иметь систему, которая станет настоящим “полигоном” для испытаний трансляторов, оптимизаторов и других средств программирования. Поэтому предполагается, что со временем функциональность системы будет расти. Например, могут быть добавлены новые возможности визуализации, трансляции и многое другое. Нельзя заранее сказать, какие средства будут добавляться. Поэтому нет возможности заложить всю функциональность системы в её исходный код. Кроме того, разработчикам дополнительной функциональности не всегда хочется разбираться во всех исходных текстах системы. Работа с исходными кодами всей системы требует от её разработчиков большой согласованности в работе. Интерактивность, расширяемость и визуальность среды вместе сильно усложняют задачу её разработки и делают её сложной системой, которая должна развиваться от простой к сложной.

Модель компонентных объектов COM [14] и платформа .NET [15] избавляют разработчиков от необходимости работы с чужими исходными текстами программ. Необходимо лишь использовать описания программных интерфейсов, а также типов данных и констант, используемых в этих интерфейсах. Компоненты распространяются в виде динамически подсоединяемых библиотек или сборок. У каждого компонента можно запросить, какую функциональность поддерживает его данная реализация путём запроса интерфейсов. Версии компонентов могут различаться набором поддерживаемых интерфейсов. Но во время исполнения компоненты могут определять поддерживаемую другими компонентами функциональность и корректно работать.

Современные методологии программирования рекомендуют разбивать сложные системы на модули, классы и объекты [19-20]. Рост системы может быть сопровожден ростом зависимостей её модулей. Число зависимостей может расти очень быстро. Наихудший вариант, это когда каждый модуль будет зависеть от каждого. В этом случае расширение и изменение системы в определённый момент станет очень сложной задачей. Система должна иметь средства решения данной проблемы. В SFP применён один из вариантов шаблона проектирования “издатель-подписчик” со слабыми ссылками. В качестве слабых ссылок используются события .NET. Под слабой ссылкой здесь понимается ссылка, не влияющая на время жизни связываемых объектов, т. е. она не учитывается счётчиками ссылок и сборщиком мусора. Объекты публикуют свои события с помощью специального объек-

та — менеджера событий. Менеджер событий содержит информацию обо всех источниках событий. Любой объект (издатель) может зарегистрировать у менеджера событий свои события. Когда появляется новый объект, ему сообщается информация обо всех имеющихся событиях, и в зависимости от их типа объект решает, подписываться на них или нет.

Первоначально модули среды разрабатывались с использованием технологии COM. Но из-за того что в .NET есть мощные средства по динамическому получению информации о типах и того что программирование на .NET упрощает создание компонентов и пользовательского интерфейса по сравнению с COM, а также позволяет разрабатывать модули на различных языках программирования и использовать модули COM, было принято решение вести разработку системы на платформе .NET с использованием написанных ранее COM-компонентов.

Базовая структура среды позволяет добавлять к ней модули в виде готовых сборок .NET. Она автоматически загружает эти сборки, инициализирует их объекты и устанавливает между ними связи, а также позволяет модулям создавать и использовать основные элементы пользовательского интерфейса, такие как окна, меню и панели. Эта основа разрабатывается независимо и фактически представляет собой элемент технологии построения визуальных интегрированных сред.

ВОЗМОЖНОСТИ СРЕДЫ SFP

Разрабатываемая среда программирования предполагает возможность включения средств программирования на различных параллельных функциональных языках. На данный момент реализуется транслятор с языка Sisal 3.0 [1,5] в графовое представление IR1 [10]. Среда содержит текстовый редактор программ, транслятор с Sisal 3.0 в IR1, визуализатор IR1, интерпретатор-отладчик и средство для создания программ из нескольких модулей. Интерпретатор-отладчик исполняет программу по её IR1. Визуализатор IR1 позволяет просматривать IR1, позволяет другим компонентам обрабатывать действия пользователя и показывать отдельные части представления. Например, интерпретатор-отладчик может исполнять программу по шагам, делая обход графа представления программы, при этом пользователю будут показываться вершины графа, а пользователь указывает дуги графа, по которым следует двигаться. Кроме этого, пользователь может перейти к просмотру соответствующих строк исходного текста программы.

Разрабатываются средства для совместной работы модулей над промежуточным представлением программ и визуализации их состояний. В частности, компоненты могут помечать вершины графа пометками. Визуализаторы могут отображать эти пометки пиктограммами. Например, отладчик может пометить вершину точкой останова в ответ на действие пользователя и отобразить соответствующую пиктограмму поверх вершины в визуализаторе представления программы. Для этого создаётся специальный компонент дополнительной информации, предоставляющий ассоциативный массив. В этом массиве ключами являются ссылки на объекты, а значениями — ссылки на объекты информации.

Планируется создать компоненты, оптимизирующие программы. Они будут преобразовывать представления программ. В SFP предусмотрена возможность управления процессом не только интерпретации программ, но и их оптимизации. Пользователь сможет указывать прямо на изображении графа, к каким вершинам нужно применить то или иное преобразование или как должен оптимизатор использовать определённые части графа. Такие возможности делают среду разработки интерактивной.

ТИПЫ КОМПОНЕНТОВ

Перечислим основные типы компонентов, интеграция которых в среду предполагается. Заметим, что архитектура не запрещает расширить этот список.

- *Ядро* — основной компонент системы, призванный упростить взаимодействие других компонентов и визуальной оболочки.
- *Компоненты дополнительной информации* — хранилища информации о текущем состоянии работы системы.
- *Внутренние представления* — представления программ (IR).
- *Компоненты сохранения и загрузки внутреннего представления* — обеспечивают поддержку сохранения и загрузки внутренних представлений в файлы различных форматов.
- *Визуализаторы* — позволяют отображать на экране различную информацию и обеспечивают взаимодействие системы с пользователем. Они могут быть различных типов: интерактивные, редакторы, информационные. В частности, в систему могут быть добавлены компоненты, рисующие графы промежуточных представлений программ.

- *Отображения* — компоненты, управляющие содержимым, отображаемым визуализаторами. Они могут строиться со знанием структур представления программы.
- *Интерпретаторы* — компоненты, позволяющие исполнять программу в среде.
- *Трансляторы* — предназначены для трансляции текста программ во внутреннее представление.
- *Оптимизаторы* — производят различные преобразования над внутренними представлениями, могут также по одному представлению, строить другое.
- *Конверторы* — строят по внутреннему представлению текст программы на каком-либо языке, включая машинный.
- *Ретрансляторы* — разновидность конвертеров, строят по внутреннему представлению программу на исходном языке.
- *Компиляторы* — позволяют пользователю создавать и выполнять сценарии обработки программ. Сценарии описывают последовательность выполнения команд других компонент.

БАЗОВАЯ СТРУКТУРА СРЕДЫ

Базовая часть среды состоит из визуального каркаса и загружаемого ядра. Каркас предоставляет базовый интерфейс пользователя: главное окно, меню, строку состояния, панели управления и инструментов, дочерние окна документов и другие стандартные элементы пользовательского интерфейса. Функциональность этого интерфейса пользователя зависит от загруженных компонентов и их состояния. Например, пункты меню могут меняться динамически. При своей инициализации компоненты получают ссылку на интерфейс ядра, а у него можно запросить ссылку на интерфейс каркаса.

Каркас позволяет модулям создавать окна, меню и элементы управления. Кроме того, каркас может сам создавать элементы пользовательского интерфейса и предоставлять программный интерфейс модулям для работы с этим интерфейсом. В задачи каркаса входит также создание в отдельном окне нужного элемента управления. Элементами управления могут являться, например, визуализаторы.

Общая схема связей компонентов изображена на рис. 1.

Визуализаторы позволяют отображать на экране информацию в виде графических изображений. Например, может быть визуализатор, рисующий граф представления программы. Способ задания графа визуализатору мо-

жет быть любым и не обязан зависеть от представления программы, т.е. визуализатор может просто уметь рисовать графы некоторого типа. О том, как отобразить представление программы с помощью визуализатора, знает отображение.

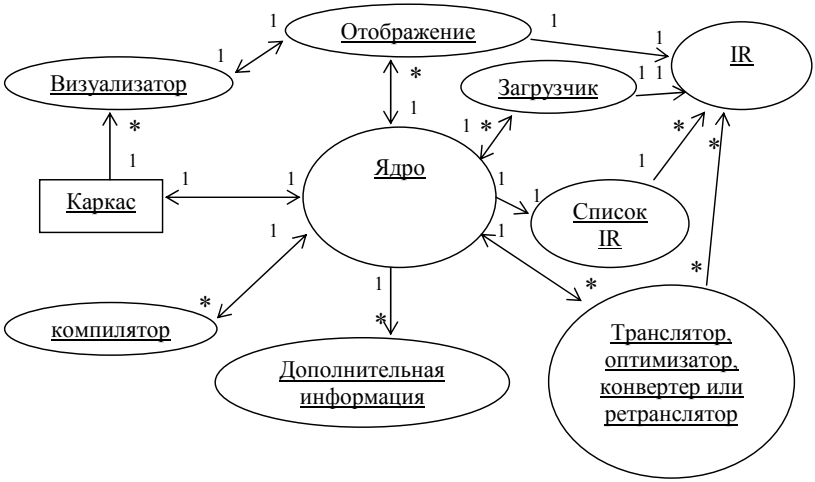


Рис. 1

Отображения могут являться не отдельными компонентами, а лишь интерфейсами визуализаторов. Каждое отображение предназначено для связи внутреннего представления программы и визуализатора. Компоненты представления и визуализации не обязаны знать о существовании друг друга, всё их взаимодействие осуществляется через компоненты отображений. Использование компонентов отображений позволяет увеличить повторное использование компонентов визуализации.

Компоненты загрузки и сохранения позволяют хранить внутренние представления в файловом виде. Форматы хранения могут быть текстовыми или бинарными. Текстовые форматы могут быть полезны для ручного редактирования.

Компоненты взаимодействуют друг с другом и пользователем при помощи событий. Пользователь генерирует события при помощи вызова команд пользовательского интерфейса. Под событием понимается вызов одного компонента другим. Как уже было сказано, события реализуются при

помощи событий .NET. Вызывающий компонент имеет информацию только о сигнатуре вызова. Вызывающий компонент называется источником события, а вызываемый — подписчиком. Если компонент может обрабатывать события определённого типа, то он подписывается ко всем источникам данного события. О появлении нового источника оповещаются все компоненты, что позволяет организовать подписку. На одно событие могут подписаться сразу несколько компонентов (это не относится к пользовательским командам). Использование механизма событий позволяет упростить связи между модулями, а также интеграцию новых модулей, так как источники и подписчики могут работать независимо друг от друга.

ЗАКЛЮЧЕНИЕ

На данный момент система находится на стадии разработки. Имеются рабочие версии следующих модулей системы: модуль построения представления программ IR1, транслятор с языка Sisal 3.0 в IR1, интерпретатор IR1, визуализатор IR1, ретранслятор из IR1 в Sisal 3.0, модуль сохранения/восстановления IR1 в файл.

Использование готового средства визуализации графов HIGRES [25] не позволило реализовать интерактивность среды. В результате попыток поддержать независимую разработку модулей несколькими разработчиками и объединить их в единую интерактивную систему была разработана технология построения визуальных интегрированных сред. С её помощью объединяются имеющиеся модули, и будут добавляться новые.

СПИСОК ЛИТЕРАТУРЫ

1. **Касьянов В.Н, Бирюкова Ю.В., Евстигнеев В.А.** Функциональный язык Sisal 3.0 // Поддержка супервычислений и интернет-ориентированные технологии. — Новосибирск, 2001. — С. 54–67.
2. **Bohm A. P. W., Oldenhoeft R. R., Cann D. C., Feo J. T.** The SISAL 2.0 Reference Manual. — Livermore, CA, 1991. — (Prepr. / Lawrence Livermore Nat. Lab.; UCRL-MA-109098, LLNL).
3. **Евстигнеев В. А., Городняя Л. В., Густокашина Ю. В.** Язык функционального программирования SISAL // Интеллектуализация и качество программного обеспечения. — Новосибирск, 1994. — С. 21–42.
4. **Feo D. T., Miller P. J., Skedzielewski S. K., Denton S. M.** Sisal 90 User's Guide / Lawrence Livermore Nat. Lab. Draft 0.96. — Livermore, CA, 1995.

5. **Бирюкова Ю. В.,** SISAL 90 руководство пользователя. — Новосибирск, 2000. — 84с. — (Препр. / Сиб. Отд-е. РАН ИСИ; № 72).
6. **Feo J. T.** Sisal. — Livermore, CA, 1992. — (Prepr. / Lawrence Livermore Nat. Lab.; UCRL-JC-110915, LLNL).
7. **Feo J. T., Cann D.C, Oldehoeft R.R.** A report on the Sisal language project // J. on Parallel and Distributed Computing. — 1990. — Vol. 10. — P. 349–366.
8. **Gaudiot J., DeBoni T., Feo J., Bohm W., Najjar W., Miller P.** The Sisil Project: Real Word Functional Programming // Compiler optimizations for scalable parallel systems. — // Lect. Notes Comput. Sci. — 2001. — Vol. 1808. — P. 45–72.
9. **McGraw J. R.** Parallel functional programming in Sisal: fictions, facts, and future. — Livermore, CA, 1993. — (Prepr. / Lawrence Livermore Nat. Lab.; LLNL).
10. **Стасенко А.** П., Внутреннее представление системы функционального программирования SISAL 3.0. — Новосибирск, 2004. — (Препр. / Сиб. Отд-е. РАН ИСИ; №110).
11. **Skedzielewski S. K., Glauert J.** IF1 — An intermediate form for applicative languages. — Livermore, CA, 1985. — (Lawrence Livermore National Lab.; Manual M-170).
12. **Густокашина Ю.В., Евстигнеев В.А.** IF1 — промежуточное представление Sisal-программ // Проблемы конструирования эффективных и надежных программ. — Новосибирск, 1995. — С. 70–78.
13. **Welcome M., Skedzielewski S. Yates R.K., Ranelletti J.** IF2 — An applicative language intermediate form with explicit memory management. — Livermore, CA, 1986. — (Lawrence Livermore Nat. Lab.; Manual M-195).
14. **Box D.** Essential COM — 1998. — Addison Wesley Longman, Inc.
15. **Оберг Р.Д., Торстейнсон П.** Архитектура .NET и программирование с помощью Visual C++.: Пер. с англ. — М.: Издательский дом “Вильямс”, 2002.
16. **Attali I., Caromel D., Guider R., Wendelborn A. L.** Optimizing Sisal programs: a formal approach // Lect. Notes Comput. Sci. — 1999. — Vol. 1123. — P. 137–144.
17. **Richter J.** Advanced Windows, Third Edition. — Channel Trading Ltd., 1997.
18. **Lachanas A., Evripidou P.** Exploiting course grain parallelism from Fortran by mapping it to IF1 // Lect. Notes Comput. Sci. — 1998. — Vol. 1470.
19. **Буч Г.** Объектно-ориентированное проектирование с примерами применения. / Пер. с англ. — М.: Конкорд, 1992. — 519 с., ил.
20. **Буч Г.** Объектно-ориентированный анализ и проектирование с примерами приложений на C++, 2-е изд. / Пер. с англ. — М.: «Издательства Бином», СПб: «Невский диалект», 1998 г. — 560 с., ил.
21. **Казаков Ф.А., Кузьмин Д.А., Легалов А.И.** Параллельный язык управления потоков данных // Математическое обеспечение и архитектура ЭВМ: Сб. научных работ. —Красноярск: КГТУ, 1997. — Вып. 2. — С. 105–113.
22. **Легалов А.И.** Управление в вычислительных системах и языках программирования // Материалы конф. "Проблемы информатизации города". — Красноярск, 1994. — С.198–202.

23. **Легалов А.И., Казаков Ф.А., Кузьмин Д.А.** Потоковая модель параллельных вычислений // Вестник Красноярского государственного технического университета. — Красноярск, 1996. — Вып. 6. — С. 60–67.
24. **Легалов А.И., Казаков Ф.А., Кузьмин Д.А. Водяхо А.И.** Модель параллельных вычислений функционального языка // Известия ГЭТУ. Структуры и математическое обеспечение специализированных средств. — С.-Петербург, 1996. — Вып. 500. — С. 56–63.
25. **Лисицын И.А.** Применение системы HIGRES для визуальной обработки иерархических графовых моделей // Проблемы систем информатики и программирования. — Новосибирск, 1999. — С. 64–77.

И.В. Дубрановский

ЭЛИМИНАЦИЯ МЕХАНИЗМА ИСКЛЮЧЕНИЙ ПРИ ПЕРЕВОДЕ ИЗ ЯЗЫКА C#-LIGHT В ЯЗЫК C#-KERNEL[1]

1. ВВЕДЕНИЕ

Верификация программ, написанных на таких широко распространенных объектно-ориентированных языках программирования, как C++, C#, Java [5], представляет в настоящее время все больший интерес. Отчасти этому способствует применение нового многоуровневого подхода к верификации [2, 3, 6, 7], совмещающего операционную и аксиоматическую семантику, что существенно упрощает доказательство условий корректности. Одним из ключевых этапов многоуровневого подхода является перевод из входного языка в некий внутренний язык с прозрачной аксиоматической семантикой.

В работе [3] для языка C этап перевода имеет вид локальных трансформаций фрагментов программы, что позволяет проводить формальное обоснование его корректности локально и без особых усилий. Сложности перевода C# в работе [1] связаны с такими особенностями этого языка, как объектная ориентированность, более высокий уровень абстракции, наличие механизма исключений.

В этой работе мы описываем алгоритм элиминации механизма исключений при переводе из языка C#-light в язык C#-kernel [6]. Этот алгоритм отличается от остальных преобразований, так как его применение требует расширения операционной семантики C#-light.

Работа состоит из трех разделов. Краткое описание языков C#-light и C#-kernel приводится в разд. 2. Элиминация механизма исключений рассматривается в разд. 3. Результаты и перспективы дальнейшей работы обсуждаются в разд. 4.

2. ЯЗЫКИ C#-LIGHT И C#-KERNEL

Язык C#-light. Язык C#-light является подмножеством языка C#. Это язык последовательных программ, т.е. в нем запрещено использование опе-

[1] Эта работа частично поддержана грантом РФФИ 04-01-00114а.

ратора `lock` и классов, связанных с созданием и управлением потоками. В C#-light не поддерживаются атрибуты и деструкторы. Еще одним ограничением является отсутствие оператора использования ресурса `using`. Также не поддерживаются операторы и выражения `checked` и `unchecked`. Следствием этого ограничения является отсутствие в C#-light возможности выбора контекста вычисления выражений (`checked` или `unchecked`). По определению, в C#-light всегда используется проверяемый (checked) контекст вычислений. Наконец, в C#-light запрещено использование небезопасного кода и директив препроцессора.

Язык C#-kernel. C#-kernel — это объектно-ориентированный язык, основанный на синтаксисе языка C#-light. В отличие от C#-light, в C#-kernel:

- запрещены пространства имен и using-директивы;
- из всех операторов разрешены только операторы объявления локальной переменной и константы, оператор-выражение, блок, помеченный оператор, операторы `if` и `goto`;
- в операторах `if` в качестве условного выражения могут использоваться только локальные переменные типа boolean;
- вызов статических функциональных членов может производиться только в тех местах в программе, в которых выполнена статическая инициализация соответствующих классов или структур;
- все метки, имена локальных переменных и имена локальных констант должны быть уникальны;
- множества меток, имен типов, имен локальных переменных и имен локальных констант не пересекаются;
- все комментарии являются аннотациями;
- добавляются метаинструкции, модифицированные операторы-выражения и модифицированные декларации классов и структур.

Метаинструкции используются для работы с метапеременными (переменными абстрактной машины [6]).

Метапеременные. Модификация переменных в программе осуществляется через следующие метапеременные:

1) `MeM` — переменная типа `Names` ∪ `TypeSpecs` → `Locations`, т. е. управление памятью (**Me**mory **M**anagement);
2) `MD` — переменная типа `Locations` → `CT#` ∪ `Locations`, т. е. дамп памяти (**M**emory **D**ump);
3) `TP` — переменная типа `Names` ∪ `Locations` ∪ `CT#` → `TypeSpecs`; задает типы идентификаторов, ячеек памяти и констант типов C#-light;

4) `UT` — переменная типа `TypeSpecs` → `TypeSpecs`, т. е. **U**nderlying **T**ype; для каждого типа перечисления задает его базовый тип,

где `CT#` — объединение всех допустимых типов C#-light, `Names` — множество идентификаторов объектов в программе, `Locations` — множество ячеек памяти, `TypeSpecs` — множество абстрактных имен типов. Тип `Locations` содержит специальные ячейки `Val` и `Exc`, используемые для хранения значения соответственно последнего вычисленного (под)выражения и брошенного исключения.

Метаинструкции. В C#-kernel существует пять метаинструкций:

1) `x := E` присваивает метапеременной `x` значение выражения `E`, которое записывается на языке аннотаций [6];
2) `new_instance(x)` ассоциирует идентификатор `x` с новой ячейкой памяти;
3) `Init(C)` выполняет статическую инициализацию класса `C`, если этот класс не инициализирован; иначе, ничего не делает;
4) `catch(T, x)` возвращает `true`, если в ячейке `Exc` лежит значение типа `T`; иначе, возвращает `false`. Если в `Exc` лежит значение типа `T`, объект-исключение, находящийся в `Exc`, записывается в переменную `x`, а затем удаляется из `Exc`, чтобы показать, что исключение перехвачено. Эта метаинструкция может использоваться только как условное выражение в операторе `if`;
5) `catch(x)` возвращает `true`, если значение ячейки `Exc` определено; иначе, возвращает `false`. Так же, как в случае с `catch(T, x)`, объект-исключение, находящийся в `Exc`, записывается в переменную `x`, а затем удаляется из `Exc`. Эта метаинструкция может использоваться только как условное выражение в операторе `if`, в котором отсутствует ветвь `else`, и моделирует общую `catch`-секцию оператора `try`.

Из множества имен функций и предикатов языка аннотаций мы выделяем `upd`, `member`, γ, `add`, `can_cast`, каждое из которых имеет фиксированную интерпретацию. `upd` — функция замены значения со следующей интерпретацией. Если `s` — выражение типа $T \to T'$, а выражения e_1, e_2 имеют типы `T` и `T'` соответственно, то терм `upd(s, e1, e2)` является выражением `s'` типа $T \to T'$, которое совпадает с `s` на всех аргументах, кроме, возможно e_1, и `s'(e1)` = e_2. Интерпретацию остальных символов можно найти в [1].

Оператор-выражение. *Оператор-выражение* в C#-kernel — это *нормализованное выражение* или метаинструкция, за которой следует точка с запятой. *Нормализованные выражения* определяются с помощью следующих ограничений на выражения языка C#-light:

- нормализованное выражение имеет вид `x.y(`z_1`, …, `z_n`)` или `y(`z_1`, …, `z_n`)`, где `x`, `y` — имена, z_1, …, z_n — имена или литералы;
- в нормализованных выражениях, функциональные члены могут быть вызваны только в нормальной форме [4];
- в нормализованных выражениях логические операции `||` и `&&`, условная операция `?:`, операция `new` и все операции присваивания запрещены.

Описание модифицированных деклараций классов и структур можно найти в [1].

3. ЭЛИМИНАЦИЯ МЕХАНИЗМА ИСКЛЮЧЕНИЙ

В этом разделе мы детально рассмотрим подход к элиминации механизма исключений. Полностью перевод из C#-light в C#-kernel описан в работе [1].

С элиминацией механизма исключений связано две проблемы. Во-первых, невозможно локализовать преобразуемый фрагмент программы, в отличие, например, от элиминации циклов. Во-вторых, прежде чем удалять `try`, необходимо корректно преобразовать (нормализовать) те операторы `goto`, `goto case` и `goto default`, которые передают управление за пределы операторов `try`, имеющих finally-блоки. Для решения этих проблем разработан алгоритм, состоящий из нескольких шагов, каждый из которых сохраняет семантическую корректность и функциональную эквивалентность программы. Тем не менее, применение такого алгоритма требует введения на промежуточных шагах расширенного оператора `try`, т. е. оператора `try` с помеченным finally-блоком.

Упомянутый алгоритм использует правило **NSS5** нормализации меток оператора `switch`, описанное ниже.

Правило NSS5. Фрагмент вида
switch-labels *statement-list*

где *statement-list* не начинается с помеченного оператора вида *identifier*`:;`, порожденного данным правилом, заменяется фрагментом
switch-labels `L:;` *statement-list*

где `L` — новая уникальная метка.

Алгоритм 1. Для краткости, рассмотрим случай нормализации операторов `goto case` и `goto default`. Множество всех меток верхнего уровня в

некотором блоке или операторе B обозначим через $Lab(B)$. Элементы множества $Lab(B)$ обозначим $\mathtt{L}_1(B)$, …, $\mathtt{L}_K(B)$. Пусть V — множество всех значений константных выражений, допустимых в switch-операторах.

1. Все switch-операторы в программе пронумеровываются в текстуальном порядке.
2. Применяется правило **NSS5**.
3. Пусть Lab^{new} — множество всех меток, порожденных правилом **NSS5** на этапе нормализации операторов switch, а $Lab^{new}(S) = \{\mathtt{L}_1(S), \ldots, \mathtt{L}_{n(S)}(S)\}$ — множество таких меток в switch-операторе S. Определим соответствия $CaseL$: $V \times \mathtt{N} \to Lab$ и $DefL$: $\mathtt{N} \to Lab$ между метками switch-операторов и новыми метками следующим образом. Для всех меток вида `case E:`, входящих в *switch-labels* оператора S_i, $CaseL(\mathtt{E}, i)$ `:=` `L`, где `L` — метка, порожденная правилом **NSS5** для switch-секции, содержащей `case E:`. Для меток вида `default:`, входящих в *switch-labels* оператора S_i, $DefL(i)$ `:=` `L`.
4. В перечисление `GT_LABELS` добавляются новые различные элементы $\mathtt{l}_1$, …, $\mathtt{l}_N$, $N = |Lab^{new}|$. Пусть функция v: $Lab^{new} \to$ `GT_LABELS` устанавливает взаимнооднозначное соответствие между элементами множества Lab^{new} и $\mathtt{l}_1$, …, $\mathtt{l}_N$.
5. Для всех switch-операторов S, для всех блоков B в операторе S, кроме тел вложенных switch-операторов, для всех finally-блоков в блоке B на верхнем уровне, если finally-блок `finally {A}` является самым верхним в S из всех finally-блоков, то он заменяется фрагментом

```
finally(f0) {
    A
    if (GT.gt == GT_LABELS.v(L1(S))) goto L1(S);
    else if (GT.gt == GT_LABELS.v(L2(S))) goto L2(S);
    …
    else if (GT.gt == GT_LABELS.v(Ln(S)(S))) goto Ln(S)(S);
}
```

   иначе, он заменяется фрагментом

```
finally(fj+1) {
    A
    if (GT.gt != GT_LABELS.none) goto fj;
}
```

   где $\mathtt{f}_j$ — новые различные метки, $j \geq 0$. Перечисление по блокам B проводится рекурсивно, начиная с тела оператора S.
6. Для всех switch-операторов S_i, для всех операторов `goto case E;` в операторе S_i, если `goto case E;` передает управление из try-

оператора с finally-блоком, помеченным меткой f_j, то `goto case E;` заменяется фрагментом `GT.gt = GT_LABELS.`v(*CaseL*(`E`, i))`; goto` f_j`;`. Иначе, `goto case E;` заменяется фрагментом `goto` *CaseL*(`E`, i)`;`.

7. Для всех switch-операторов S_i, для всех операторов `goto default;` в операторе S_i, если `goto default;` передает управление из try-оператора с finally-блоком, помеченным меткой f_j, то `goto default;` заменяется фрагментом `GT.gt = GT_LABELS.`v(*DefL*(i))`; goto` f_j`;`. Иначе, `goto default;` заменяется фрагментом `goto` *DefL*(i)`;`.

Заметим, что в пунктах 5–7 стратегия применения кванторов всеобщности по блокам имеет вид рекурсивного обхода блоков от объемлющего к вложенному. Однако операторы `goto case` и `goto default` обрабатываются в текстуальном порядке сверху вниз.

Таким образом, алгоритм 1 преобразует операторы `goto case` и `goto default` в обычный оператор `goto`. В дополнение к этому, в полученной на выходе программе `goto` не передают управление за пределы `try`. В программе, обладающей таким свойством, элиминация `try` является законной. Удаление операторов `try` — это уже локальное преобразование, реализуемое набором недетерминировано применяемых правил. Приведем в качестве примера два правила элиминации расширенного оператора try.

Правило ETRY1. Фрагмент вида

```
try {A} finally(L) {B}
```

заменяется фрагментом

```
{A} L:;
    if (catch(x))
    {
        B
        MD := upd(MD, Exc, MD(MeM(x)));
        goto M;
    }
    {B}
    M:;
```

где `M` — новая метка.

Правило ETRY2. Фрагмент вида

```
try {A} catch (T1 x)  {C1}
        catch (T2 x)  {C2}
        …
        catch (Tn x)  {Cn}
        catch         {Cg}
```

заменяется фрагментом

```
{A} if      (catch(T1, x)) { T1 eSaved; eSaved = x; C1 }
    else if (catch(T2, x)) { T2 eSaved; eSaved = x; C2 }
    …
    else if (catch(Tn, x)) { Tn eSaved; eSaved = x; Cn }
    else if (catch(eSaved)) { Cg }
```

Остальные правила аналогичны **ETRY1**, **ETRY2** и описаны в [1].

4. ЗАКЛЮЧЕНИЕ

В данной статье представлен алгоритм элиминации механизма исключений при переводе из языка C#-light в язык C#-kernel. Этот алгоритм отличается от остальных преобразований, так как его применение требует расширения операционной семантики C#-light.

Следующим закономерным шагом является формальное обоснование корректности перевода. В частности, предстоит провести доказательство функциональной эквивалентности исходной и преобразованной алгоритмом 1 программы.

СПИСОК ЛИТЕРАТУРЫ

1. **Дубрановский И.В.** Верификация C#-программ: перевод из языка C#-light в язык C#-kernel. — Новосибирск, 2004. — 65с. — (Препр. / СО РАН. Ин-т систем информатики; № 120).
2. **Непомнящий В.А., Анureев И.С., Михайлов И.Н., Промский А.В.** На пути к верификации C-программ. Язык C-light и его формальная семантика // Программирование. — 2002. — № 6. — С. 19–30.
3. **Непомнящий В.А., Анureев И.С., Михайлов И.Н., Промский А.В.** На пути к верификации C-программ. Часть 3. Перевод из языка C-light в язык C-light-kernel и его формальное обоснование. — Новосибирск, 2002. — 84с. — (Препр. / СО РАН. Ин-т систем информатики; № 97).

4. **C#** Language Specification: ECMA-334, http://www.ecma.ch, Appr. Dec. 2001. — 382 p.
5. **Huisman M., Jacobs B.** Java program verification via a Hoare logic with abrupt termination // Proc. / FASE 2000. — Berlin etc., 2000. — P. 284–303. — (Lect. Notes Comput. Sci.; N 1783).
6. **Nepomniaschy V.A., Anureev I.S., Dubranovsky I.V., Promsky A.V.** A three-level approach to C# program verification // J. Bull. of NCC and IIS. — 2004. — Vol. 20 — P. 61–86.
7. **Nepomniaschy V.A., Anureev I.S., Promsky A.V.** Verification-oriented language C-light and its structural operational semantics (extended abstract) // PSI: Proc. / Fifth Internat. Conf. on Perspectives of System Informatics, Novosibirsk, Russia, July 2003. — Berlin etc., 2003. — P. 103–111. — (Lect. Notes Comput. Sci.; N 2890).

С.С. Канюс, А.Г. Никитин

РЕАЛИЗАЦИЯ ЗАМКНУТОГО НАБОРА БУЛЕВЫХ ОПЕРАЦИЙ НАД ПОЛИГОНАЛЬНЫМИ ОБЛАСТЯМИ НА ДИСКРЕТНОЙ СЕТКЕ

ВВЕДЕНИЕ

Алгоритмы нахождения объединения, пересечения, разности и симметрической разности многоугольников широко используются в САПР, геоинформационных системах, компьютерной графике и во многих других сферах применения вычислительной геометрии. К основным требованиям, предъявляемым к таким алгоритмам, относятся быстродействие, численная устойчивость и замкнутость набора операций (т.е. возможность повторного выполнения операций над результатом).

В работе [1] был предложен простой и эффективный алгоритм, реализующий замкнутый набор булевых операций над множествами граничнозаданных многоугольников (над так называемыми регионами). Однако попытка перенести это решение на регионы, заданные на дискретной сетке координат (когда координаты вершин являются целыми числами [4]), выявила ряд недостатков, связанных с численной устойчивостью [2], т. е. из-за округления координат точек пересечения алгоритм в некоторых случаях не может выполнить требуемые операции.

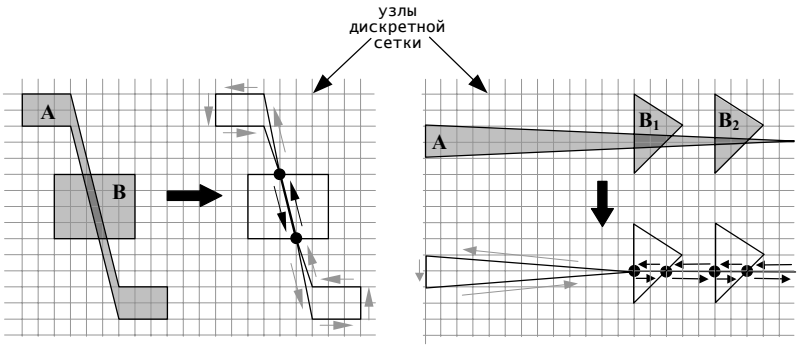
Предметом данной статьи является нахождение способа повышения численной устойчивости базового алгоритма [1] на дискретной сетке и повышение его эффективности. Предлагается эффективный алгоритм, реализующий замкнутый набор булевых операций над полигональными областями и обладающий численной устойчивостью. Сохраняя все достоинства своего предшественника, новый алгоритм позволяет более эффективно решать поставленные задачи — в частности, существенно расширить круг обрабатываемых случаев.

ПОСТАНОВКА ЗАДАЧИ

Отличительной чертой базового алгоритма является его устойчивость к вырожденным случаям типа самокасаний и кратных пересечений. Это, наряду с возможностью обработки многоугольников с отверстиями, обеспечивает замкнутость набора операций {*объединение, пересечение, разность, симметрическая разность*} при использовании вещественнозначной вычислительной модели с неограниченной точностью. Однако практическое применение описанного в [1] алгоритма в рамках вычислительной модели с использованием округлённой арифметики выявило в некоторых случаях его некорректную работу, связанную с ошибками округления на этапе пересечения рёбер исходных многоугольников. Остановимся на этом немного подробнее.

Для того чтобы в результате ошибок округления положение вершин-пересечений не изменяло топологию контуров, приводя к противоречиям с аксиоматикой алгоритма, было предложено применить в нём дискретную сетку [2, 4]. Это означает, что координаты вершин «внутри» алгоритма принадлежат дискретному множеству, представляющему собой узлы целочисленной координатной сетки на плоскости. Проблема возникает, когда при пересечении рёбер две или более вершины одного и того же многоугольника в результате округления попадают в один узел сетки, образуя *вырожденные* (геометрически совпадающие, но противоположные по направлению) рёбра. Некоторые случаи такого вырождения показаны на рис. 1. Существующая на настоящий момент структура алгоритма, описанная в [1], не предусматривает корректную обработку таких случаев. А именно, в алгоритме предполагается, что область, ограничиваемая контуром, является односвязной ([1], стр. 7, Опр. 2) и что в окрестности любой вершины контура имеются точки этой области ([1], стр. 8, Опр. 3). Первое условие нарушается в случае, изображённом на рис. 1а, второе — на рис. 1б. При обработке такого рода случаев алгоритм получает неправильный результат.

Таким образом, возникает необходимость модифицировать базовый алгоритм так, чтобы он допускал корректную обработку регионов с вырожденными рёбрами.



(а) (б)

Рис. 1. Образование сдвоенных рёбер, нарушающее аксиоматику алгоритма Леонова и Никитина. Стрелками показаны направления обхода контуров. (а) Область, ограниченная контуром — несвязна. (б) В контуре присутствуют вершины, окрестности которых не содержат точек области, ограниченной этим контуром

ПОСТРОЕНИЕ ГРАФА И МАРКИРОВКА РЁБЕР

Принцип данного алгоритма, как и принцип его предшественника [1], состоит в построении на исходных контурах графа, вершинами которого являются вершины исходных контуров и точки пересечения рёбер исходных контуров (так называемое *линейно-узловое разбиение* [3] исходных контуров). Регион-результат собирается обходом рёбер построенного графа в определённом порядке, который зависит от конкретной булевой операции.

Итак, если исходные контуры пересекаются, то их рёбра оказываются связаны через свои точки пересечения (посредством вышеупомянутого графа). Это означает, что в процессе обхода рёбер первого контура можно «перепрыгнуть» в точке пересечения на второй контур и продолжать обход уже по рёбрам второго контура. Правильно выбирая следующее ребро в каждой точке пересечения, можно обойти исходные рёбра и «собрать» таким образом результирующий контур с нужными свойствами. Например, для сборки минимального контура достаточно брать в точках пересечения следующее ребро, ближайшее к текущему ребру в направлении по часовой

стрелке. Остаётся вопрос о правильном «маршруте» этого обхода применительно к булевым операциям, т. е., фактически, необходимо сформулировать условия включения ребра в результат.

Условия включения ребра в результат можно продемонстрировать следующими рассуждениями на примере пересечения двух регионов (рис.2). Ребро входит в пересечение двух регионов, если оно не вырождено, принадлежит границе только одного из регионов и лежит внутри области, описываемой другим регионом. Ребро также входит в результат, если оно не вырождено, используется в двух регионах, и оба лежат слева от этого ребра (считаем, что область, описываемая контуром, лежит слева от его ребер). Аналогичным образом могут быть получены условия включения ребра в объединение, разность и симметрическую разность регионов. Итак, для определения принадлежности ребра результату нам нужна информация о том, является ли ребро региона вырожденным, и если нет, то лежит ли оно снаружи, внутри или на границе другого региона, и если на границе, то совпадает ли по направлению с ребром из другого региона или противоположно ему. Всю эту информацию можно отразить в так называемой *метке ребра*, которая может принимать следующие значения (рис. 2):

1) *Inside* — ребро одного региона лежит внутри второго региона;
2) *Outside* — ребро одного региона лежит снаружи второго региона;
3) *Shared1*(*s*) — ребро одного региона совпадает с ребром *s* другого региона геометрически и по направлению;
4) *Shared2*(*s*) — ребро одного региона совпадает с ребром *s* другого региона геометрически и противоположно ему по направлению;
5) *Shared3* — ребро вырождено либо дублирует граничное (разъяснение ниже).

Особо следует сказать о метке *Shared3*. Фактически, ей помечаются рёбра, которые заведомо ни при каких условиях не должны войти в результат. Во-первых, это вырожденные рёбра. Кроме того, заметим, что среди граничных рёбер региона могут существовать «двойники» — совпадающие геометрически и по направлению рёбра, принадлежащие одному региону. Ясно, что из таких рёбер в результат может войти лишь одно. Поэтому будем помечать все рёбра-двойники, кроме одного, как *Shared3*. Впоследствии, оставшееся ребро будет помечено какой-нибудь из четырёх других меток, на основании которой будет принято решение о его включении в результат.

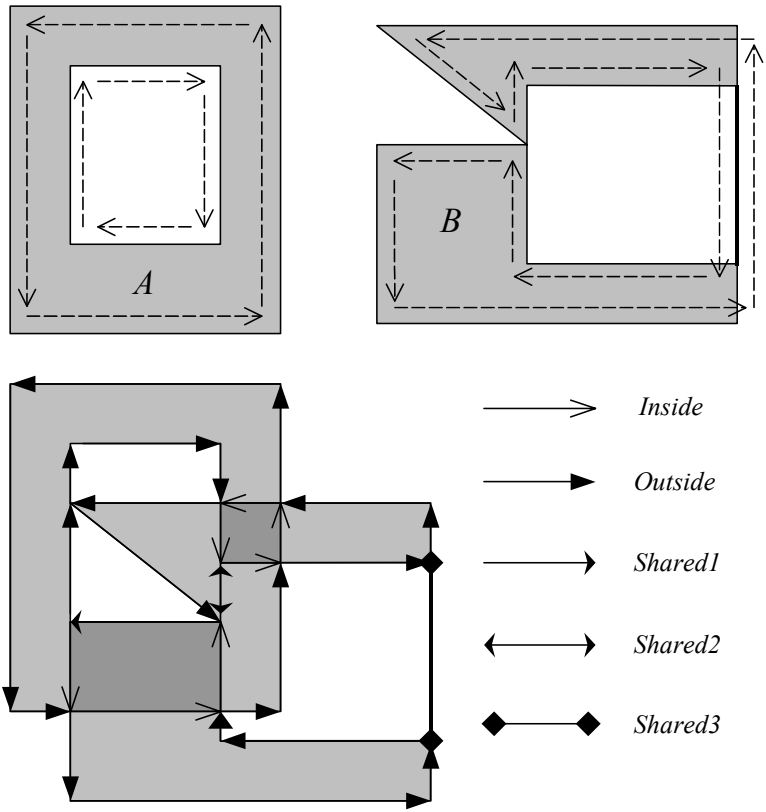


Рис. 2. Метки рёбер и результат маркировки регионов *A* и *B*

Итак, меткой *Shared3* рёбра помечаются в первую очередь. После этого непомеченными являются лишь несовпадающие граничные рёбра, которые, фактически по остаточному принципу, помечаются остальными метками. Процесс маркировки рёбер метками *Inside*, *Outside*, *Shared1*(*s*) и *Shared2*(*s*) в целом не отличается от приведённого в [1] и подробно описан в [3].

Рёбра контура, который ни с чем не пересекается, помечать неэффективно, так как любой такой контур целиком либо входит, либо не входит в результат, в зависимости от того, лежит он снаружи или внутри того региона-операнда, которому он не принадлежит. Как и в случае рёбер, эту информацию удобно хранить в виде *меток контуров*:

1) *Inside* — контур одного региона лежит внутри второго региона;

2) *Outside* — контур одного региона лежит снаружи второго региона;
3) *Intersected* — контур содержит точки пересечения с рёбрами других контуров.

Таким образом, рёбра контуров, помеченных как *Inside* и *Outside*, не помечаются.

СБОРКА РЕЗУЛЬТАТА

Итак, пометив рёбра и контуры, мы имеем возможность для любого ребра контура, помеченного как *Intersected,* и для любого контура, помеченного как *Inside* или *Outside,* узнать, входят они в результат или нет (конкретные условия включения ребра в результат для различных операций приведены в [1, 3]). Обратим внимание, что в зависимости от операции и метки ребро может входить в результат со своим или противоположным направлением, а контур — со своей или противоположной ориентацией.

Теперь необходимо собрать результирующий регион. Для *Intersected*-контуров достаточно каким-то образом находить первое ребро, входящее в результат, после чего, начиная с этого ребра, двигаться вдоль цепочек контуров от одной точки пересечения к другой, пока контур не замкнётся. Остаётся два вопроса — где брать первое ребро и куда идти дальше, когда мы находимся в точке пересечения.

В алгоритме, описанном в [1], первое ребро, входящее в результат, находится простым линейным поиском по контуру. Однако существует способ более эффективного нахождения первого ребра. Заметим, что любая цепочка рёбер, находящаяся между двумя какими-нибудь точками пересечения целиком входит в результат, если её крайние рёбра входят в результат. Поэтому достаточно из всех ещё необработанных рёбер, связанных с какой-либо точкой пересечения, выбрать рёбро, удовлетворяющее условию включения в результат, и, начиная с него, произвести сборку результирующего контура до его замыкания. И повторять это, пока не будут обработаны все рёбра, связанные с какой-либо точкой пересечения. Так как таких рёбер в большинстве случаев гораздо меньше общего количества рёбер, то такая схема сборки более эффективна.

После того как собраны все ограничивающие контуры результирующего региона, требуется правильным образом скомпоновать внешние и внутренние контуры в полигоны. Внешнему контуру соответствует новый полигон, а для внутреннего должен быть найден соответствующий полигон. Таким образом, необходимо для каждого внутреннего контура определить

минимальный внешний контур, в который попадает данный внутренний, и вставить его в соответствующий полигон. После того как это будет сделано, поставленная задача решена.

КЛЮЧЕВЫЕ ОТЛИЧИЯ ОТ БАЗОВОГО АЛГОРИТМА

Итак, при сборке результата в каждой точке пересечения выбирается направление дальнейшего движения, т.е. среди рёбер, связанных с данной точкой пересечения, осуществляется поиск ребра с нужными свойствами. Для того чтобы такой поиск был эффективен, в базовом алгоритме все рёбра, связанные с каждой точкой пересечения, упорядочиваются по углам вхождения в точку пересечения (строятся так называемые *списки связности* [1, 3]).

Основное отличие обсуждаемого в данной статье алгоритма от своего предшественника состоит в способности принимать регионы, содержащие вырожденные рёбра. Это несколько усложняет механизм отбора рёбер для включения в результат. Для реализации этого механизма потребовалось введение новой метки (*Shared3*) и модификация способа упорядочивания рёбер в точках пересечения. В базовом алгоритме порядок рёбер с одинаковым углом в точке пересечения не определён, в то время как в модифицированном алгоритме порядок следования рёбер с одинаковым углом определяется их принадлежностью контурам одного или другого региона с учётом направления рёбер (входящие или выходящие). При предложенном способе упорядочивания появляется возможность эффективно выявлять рёбра как вырожденные, так и дублирующие граничные, каждое из которых помечается как *Shared3* (соответствующий алгоритм можно найти в [3]). И это гарантирует, что рёбра, оставшиеся непомеченными, являются уникальными и граничными. На следующем этапе маркируются оставшиеся рёбра.

С точки зрения эффективности алгоритма важным усовершенствованием является модификация способа сборки результирующих контуров, о чём упоминается в предыдущем разделе. Так как разработанный алгоритм использует более эффективный, чем у базового алгоритма, способ сборки, то на данном этапе его вычислительная трудоемкость несколько ниже, чем у предшественника.

ЗАКЛЮЧЕНИЕ

Предложенная в данной статье модификация базового алгоритма [1] позволяет эффективно реализовать замкнутый набор булевых операций над регионами, допускающими вырожденные рёбра. Это существенно расширяет круг обрабатываемых алгоритмом случаев при сохранении достоинств своего предшественника.

Обсуждаемые в данной статье решения теоретически обосновываются в работе [3], где также приводится подробное описание алгоритма и показывается, что фактически речь идёт об обобщении базового алгоритма, позволяющем подавать на вход регионы с гораздо менее строгими ограничениями (так называемые обобщённые регионы [3]), чем у оригинального алгоритма. Следует отметить, что в результате решения этой задачи, несмотря на существенное расширение области определения входных данных, область определения выходных данных осталась прежней (так называемые простые регионы [3]).

Предложенная в статье модификация базового алгоритма обладает несколько более высокой производительностью.

Авторская реализация алгоритма с применением обсуждаемых в данной статье решений занимает около 3000 строк на языке C#.

СПИСОК ЛИТЕРАТУРЫ

1. **Леонов М.В., Никитин А.Г.** Эффективный алгоритм, реализующий замкнутый набор булевых операций над множествами многоугольников на плоскости. — Новосибирск, 1997. — 24 с. — (Препр. / СО РАН. Ин-т систем информатики им. А.Н. Ершова).
2. **Леонов М.В.** Реализация булевых операций над множествами многоугольников на плоскости: Дипломная работа: 01.06.1998 — Новосибирск: НГУ, 1998. — 26 с.
3. **Канюс С.С.** Эффективный численно устойчивый алгоритм, реализующий замкнутый набор булевых операций над полигональными областями: Маг. дис: 10.06.2004 — Новосибирск: НГУ, 2004. — 36 с.
4. **Hobby J.** Practical segment intersection with finite precision output. —Manuscript. 1994.
5. **Foley J. D., Dam A. Van, Feigner S. K., Hughes J. F.** Computer Graphics: Principles and Practice. — Addison-Wesley, Reading, MA, 1990.
6. **Кнут Д.** Искусство Программирования, 3-е изд., 1т. — М., 2000
7. **Ласло М.** Вычислительная геометрия и компьютерная графика на C++. — М.: Бином, 1997.

8. **Bentley J. L., Ottman T. A.** Algorithms for reporting and counting geometric intersections // IEEE Trans. Comput, 1979. — Vol. C-28. — P.643–647.
9. **Finke U., Hinrichs K.H.** Overlaying simply connected planar subdivisions in linear time // Proc. 11th Annual ACM Sympos. On Computational geometry. — 1995. — P.119–126.

В.В. Колдаков, Н.В. Панов, С.П. Шарый

НАГЛЯДНОЕ ПРЕДСТАВЛЕНИЕ РАБОТЫ АЛГОРИТМОВ ГЛОБАЛЬНОЙ ИНТЕРВАЛЬНОЙ ОПТИМИЗАЦИИ

1. ВВЕДЕНИЕ

Последние десятилетия ушедшего века стали свидетелями быстрого и успешного развития нового направления математики — интервального анализа, существенно расширившего возможности математического моделирования и позволяющего решать некоторые традиционные постановки задач быстрее и точнее классических методов.

Однако возможности интервального анализа используются всё ещё недостаточно широко, и происходит это не потому, что его модели и методы не востребованы. Наоборот, интервальные вычисления недостаточно популярны лишь по причине того, что их возможности плохо известны широкому кругу специалистов (впрочем, как видимо, и само существование интервального анализа). Во многом это обусловлено тем, что программ, позволяющих проводить интервальные вычисления, еще крайне мало, в особенности лёгких демонстрационных пакетов. Данная работа способствует заполнению этого пробела.

2. ПОСТАНОВКА ЗАДАЧИ

Работа посвящена разработке и реализации пакета программ, позволяющих ставить и решать задачу глобальной оптимизации функции двух переменных интервальными методами, а также наглядно демонстрировать процесс и результаты работы различных алгоритмов.

Реализованный программный комплекс основан на клиент-серверной архитектуре и включает в себя следующие компоненты:

1) мультиплатформенная клиентская часть со встроенным собственным трехмерным ядром;
2) серверная часть, объединяющая собственную программу-сервер;
3) решатель, в свою очередь, состоящий из интервального ядра, библиотеки алгоритмов и символьного интерпретатора.

Представляемая система призвана способствовать популяризации интервальных методов, в частности методов глобальной оптимизации, и создавалась с рекламно-демонстрационной целью, но может рассматриваться и как полноценный инструмент для решения задач глобальной оптимизации в трехмерном пространстве.

3. СПЕЦИФИКА ЗАДАЧИ

В настоящее время математических пакетов, поддерживающих в том или ином виде методы интервального анализа, всё еще немного, из них общедоступные не очень удобны и имеют ряд ощутимых недостатков. При проектировании нашей системы, мы старались избежать основных недостатков существующих решений, мешающих программам такого рода завоевать массового пользователя:

1) “ресурсоемкость” (требуют от пользователя большой мощности процессора, много места на диске и оперативной памяти);
2) платформеннозависимость.

Кроме того, в нашем случае одним из основных проектных требований было наложение системой как можно меньших ограничений на потенциального пользователя и его рабочее место.

Вся работа осуществляется через сеть Internet при помощи любого Веббраузера с поддержкой языка Java (Internet Explorer, Netscape Navigator, Mozilla, Opera), но без дополнительных модулей (которые обычно требуются, например, для поддержки трехмерной графики).

Так как системе предстоит работать с трехмерной графикой, но у пользователя может не быть графической подсистемы DirectX или OpenGL, встала необходимость создания собственного трехмерного ядра для клиентской части, потому что мы не вправе требовать их предустановки на компьютер пользователя.

4. КРАТКИЙ ПЕРЕЧЕНЬ ТРЕБОВАНИЙ, ПРЕДЪЯВЛЯЕМЫЙ К СОЗДАВАЕМОМУ ПРОГРАММНОМУ ОБЕСПЕЧЕНИЮ

Разрабатываемое программное обеспечение должно, прежде всего, позволять пользователю ставить задачу глобальной оптимизации. Далее, поставленная задача должна быть решена за конечное время, и полученный ответ должен быть гарантированным: пропуск какого-либо решения задачи недопустим. Создаваемое ПО должно обеспечивать наглядную демонстра-

цию работы интервальных алгоритмов для решения задачи глобальной оптимизации. Результаты работы должны быть также представлены в наглядном виде. Наконец, система должна накладывать как можно меньше ограничений на потенциального пользователя и его рабочее место.

5. СТРУКТУРА КОМПЛЕКСА

Использование клиент-серверной архитектуры позволяет разгрузить клиентскую машину, “унеся” все тяжелые вычисления на серверную сторону. Это не только практически снимает ограничение на мощность пользовательской машины, но и позволяет повысить скорость и точность результатов за счет использования мощных серверных станций. Выбор Java в качестве языка реализации визуализационной части гарантирует мультиплатформенность. Математическая часть выполнена на языке C++, более подходящем для этих целей.

В связи с тем, что мы не можем требовать от пользователя наличие каких-либо средств для работы с трехмерной графикой, встала необходимость создания собственного трехмерного ядра для клиентской части. В его задачи входит не только отрисовка трехмерных объектов средствами двумерной графики, но и немалая математическая часть: вся рутина по вычислению проекций, пересчет координат при повороте и прочие матричные преобразования; расчет затененных областей и удаления невидимых граней.

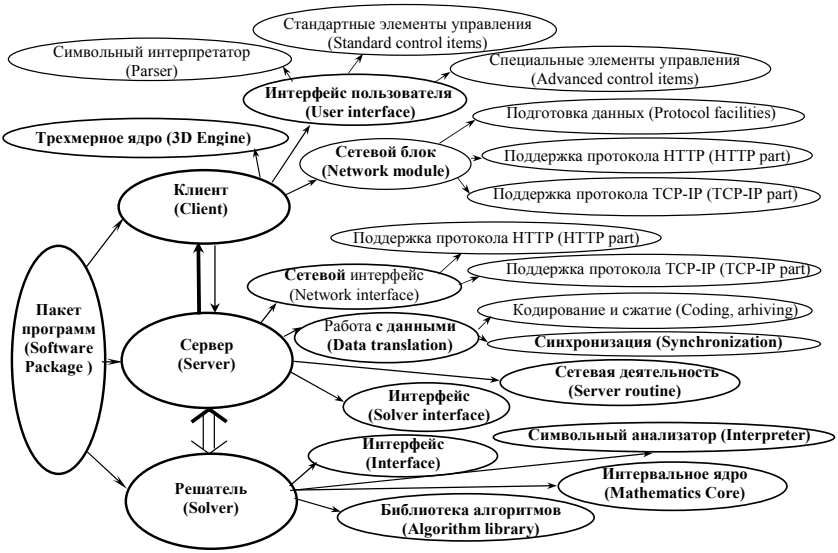
В задачи сервера входят непосредственно отслеживание и поддержание соединений, аутентификация пользователей и разделение прав, перекодировка и передача информации между клиентом и системой решения, запуск решателя и постановка задачи для него.

Особенность серверной части в том, что она должна сочетать в себе модули, написанные на языках Java и C++. Серверная часть реализована в двух вариантах — в виде сервлета (Java Servlet) и в виде самостоятельного сервера. В силу большой специфичности круга задач, решаемых сервером, и особенностей архитектуры, программа-сервер создавалась "с нуля", без использования готовых основ.

Математический решатель (Solver) состоит из интервального ядра, библиотеки алгоритмов, модуля обеспечения и символьного анализатора.

Интервальное ядро реализует представление типа данных «интервал» и набор базовых функций. Библиотека функций — это расширяемый набор интервальных алгоритмов, а модуль обеспечения отвечает за ввод-вывод,

синхронизацию, целостность системы и взаимодействие с сервером (пользователем).



6. ОПИСАНИЕ АЛГОРИТМОВ ТРЕХМЕРНОЙ ГРАФИКИ

Здесь не приводится. Заинтересованный читатель может изучить работы [2, 6], подробно рассматривающие данный вопрос.

7. ИНТЕРВАЛЬНЫЕ АЛГОРИТМЫ ГЛОБАЛЬНОЙ ОПТИМИЗАЦИИ ФУНКЦИЙ НЕСКОЛЬКИХ ПЕРЕМЕННЫХ

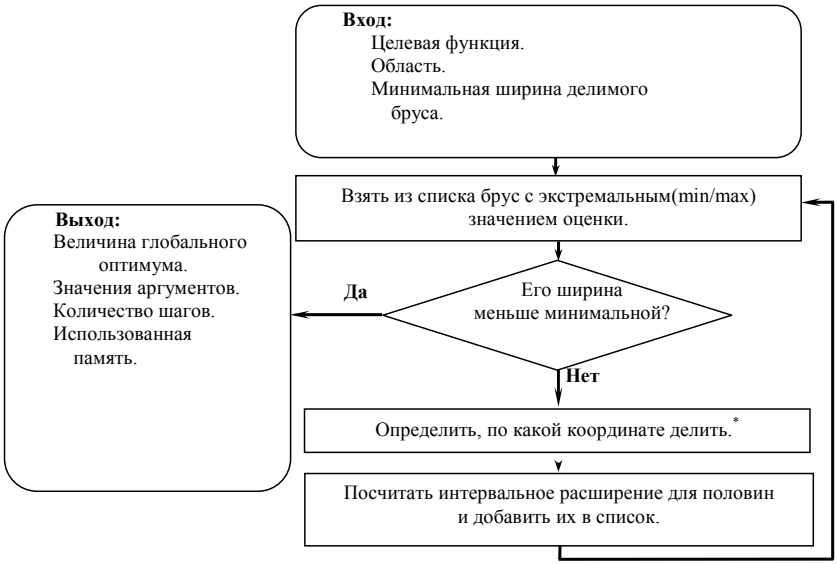
В статье весьма кратко описываются только два из использованных алгоритмов — базовый алгоритм Мура—Скелбоу и интервальный метод симулированного отжига. Оба они применяют для решения задачи глобальной оптимизации интервальные методы, основанные на адаптивном дроблении области определения в сочетании с оцениванием области значений по получающимся подобластям. Методы этого типа хорошо работают для функций сложного рельефа, находя гарантированные оценки как для глобального оптимума, так и (после небольшой модификации) для достав-

ляющих его значений аргументов. Формат статьи не позволяет здесь рассмотреть другие интересные алгоритмы и даже остановиться более подробно на этих двух рассматриваемых. За подробной информацией читателю следует обратиться к работам [1, 3, 4, 5].

7.1. Алгоритм Мура—Скелбоу

Алгоритм Мура—Скелбоу — это классический метод, который на каждой итерации выбирает брус, доставляющий наихудшую оценку искомого глобального экстремума, и делит его пополам.

Работа алгоритма проиллюстрирована на рис. 1.



*) Деление может осуществляться несколькими способами.

Рис 1. Блок-схема алгоритма Мура—Скелбоу

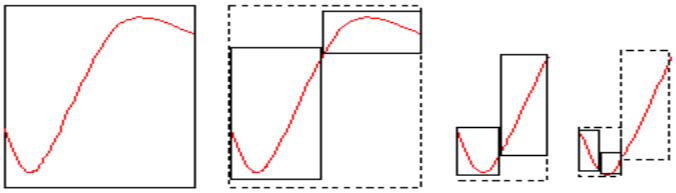


Рис. 2. Поиск минимума алгоритмом Мура—Скелбоу

- **Естественное дробление (Дробление большой стороны).**
  За одну итерацию алгоритма выполняется одно деление. Делится более широкая сторона бруса. Это связано с тем, что точность интервальной оценки зависит от ширины интервала, в котором меняются аргументы, и от зависимости целевой функции по каждому из аргументов. Не делая никаких предположений о конфигурации функции, алгоритм последовательно уточняет интервальное расширение, деля наибольшую и, вероятно, дающую наименее точную оценку сторону. Например, брус такого вида (рис. 2а) будет поделен таким (рис. 2б) образом.
- **Тотальное дробление (Дробление всех сторон).**
  За одну итерацию алгоритма брус делится на восемь частей. Пример дробления по всем сторонам на рис. 3.
- **Дробление с памятью (Эвристическое дробление).**
  За одну итерацию алгоритма выполняется одно деление. Сторона для деления выбирается исходя из предыдущих итераций, используется вместе с алгоритмом естественного деления (деление большой стороны). Каждый раз, производя деление очередного бруса, этот алгоритм анализирует, насколько улучшилась интервальная оценка при делении именно по этой координате. В существующей реализации алгоритм надежд не оправдал.

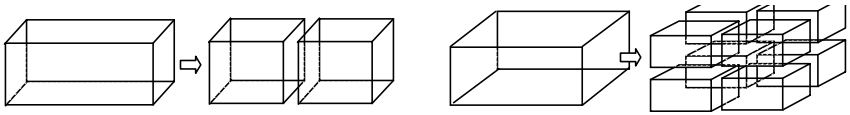


Рис 3. а), б) — деление по большей стороне, в), г) — тотальное деление

Алгоритм Мура—Скелбоу является, фактически, соединением техники интервального оценивания значений целевой функции по подобластям со стратегией дробления, заимствованной из известного в комбинаторной оптимизации метода «ветвей и границ». Достоинства алгоритма Мура—Скелбоу — простота реализации и широкая сфера применимости. В рамках данной работы был разработан ряд алгоритмов, также основанных на адаптивном дроблении.

7.2. Вероятностный алгоритм

Другим популярным подходом к решению задач глобальной оптимизации является так называемый метод *«симулированного отжига»* (известный еще как *алгоритм Метрополиса*). Это вероятностный алгоритм, моделирующий одноименный физический процесс, и его имеет смысл применять в тех ситуациях, где доминирующими требованиями является желание получить в сложной задаче хоть какие-то результаты за заданное ограниченное время. Разработанная С.П. Шарым и реализованная в рамках данной работы интервальная версия алгоритма «симулированного отжига» для глобальной оптимизации функций сочетает в себе гарантированность интервальных методов с неприхотливостью вероятностного подхода.

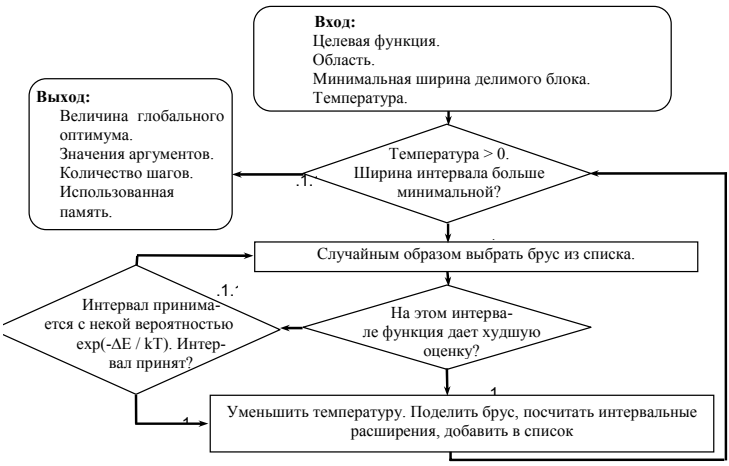


Рис 4. Блок-схема алгоритма интервального симулированного отжига для глобальной оптимизации

8. РЕАЛИЗАЦИЯ

8.1. Решатель

Решатель реализован на языке C++ и платформеннозависим. Существующая реализация позволяет осуществлять переход между операционными системами Windows (поддерживается вся линейка, начиная с Windows 95) и клонами Linux без изменения кода. Поддержки графического интерфейса пользователя нет. Самодостаточен. Интервальное ядро реализует представление типа интервал и базовые функции. Может выступать в роли самостоятельного продукта. В библиотеке алгоритмов реализованны функции высокого уровня, алгоритмы глобальной оптимизации (в том числе алгоритм Мура—Скелбоу и метод «симулированного отжига», рассмотренные в параграфе 7) и методы построения графиков функций (поверхностей). Символьный анализатор позволяет ставить задачу, используя командную строку или файл инициализации, позволяет определять константы, функции, переменные, допускаются данные интервального типа, возможен ввод дробных чисел и чисел в экспоненциальном формате.

Рис 5. Структурная схема решателя

8.2. Сервер

Сервер реализован на языке Java в двух вариантах — в виде самостоятельной java-программы и с использованием технологии Servlet.

8.3. Реализация клиентской части

Клиентская часть реализована на языке Java в виде аплета, предоставляющего графический интерфейс пользователя со встроенным трехмерным ядром, собственной системой меню, диалогов и командной строкой. Позволяет ставить задачу, отображать работу одновременно нескольких алгоритмов, в том числе в пошаговом режиме в полноценном трехмерном пространстве.

СПИСОК ЛИТЕРАТУРЫ

1. **Калмыков С.А., Шокин Ю.И., Юлдашев З.Х**. Методы интервального анализа. — Новосибирск, Наука, 1986.
2. **Фоли Дж., Дэм А.** Машинная графика. 2т. — М.: Мир, 1988.
3. **Шарый С.П.** Интервальные алгебраические задачи. Теория, приложения и численное решение. — Новосибирск, 2004. — `http://www.ict.nsc.ru/lab1.2/books.html`
4. **Hansen E.R.** Global optimization using interval analysis. — Marcel Dekker, 1992.
5. **Kearfott R.B.** Rigorous Global Search: Continuous Problems. — Dordrecht, Kluwer, 1996.
6. **`www.enlight.ru/faq3d/main.html`** – demo.design 3D. Programming FAQ.

М. Ю.Машуков

ТРАНСЛЯЦИЯ SDL-СПЕЦИФИКАЦИЙ С ДИНАМИЧЕСКИМИ КОНСТРУКЦИЯМИ В РАСКРАШЕННЫЕ СЕТИ ЙЕНСЕНА[1]

1. ВВЕДЕНИЕ

Верификация распределенных систем, таких как коммуникационные протоколы — сложная и актуальная проблема современного программирования. На практике используются несколько подходов к этой проблеме, один из них базируется на использовании таких моделей, как конечные автоматы, сети Петри и их обобщения, и заключается в моделировании коммуникационных протоколов и верификации полученных моделей. Такие модели удобны для анализа и верификации, но, как правило, моделирование каждого протокола необходимо выполнять отдельно. Задача верификации процесса моделирования сравнима по сложности с задачей верификации исходного протокола. Автоматический перевод спецификаций выполнимых протоколов в формальные модели, для которых существуют эффективные методы анализа и автоматические средства верификации, объединяет преимущества обоих подходов.

В данной работе описывается реализация автоматической трансляции SDL-спецификаций в иерархические раскрашенные сети, предложенные Йенсеном [1], и эксперименты по верификации коммуникационных протоколов ATMR [2] и InRes [3, 4]. Транслируемые SDL-спецификации могут содержать динамические конструкции SDL. Для работы с получаемыми сетями используется система CPN Tools [5].

2. ЯЗЫК SDL

Язык SDL (Specification and Description Language) — язык спецификации и описаний, предназначен для описания структуры и функционирования систем реального времени, в частности, сетей связи. Система, описываемая на SDL, состоит из блоков и каналов, соединяющих эти блоки меж-

[1] Работа частично поддержана грантом РФФИ 03-07-90331в.

ду собой и с окружающей средой. Каждый канал передает сигналы определенного типа. В блоках находятся процессы, соединенные маршрутами. Процессы общаются посылкой/приемкой сигналов между собой.

Процесс представляет собой расширенный конечный автомат. Переход из одного состояния в другое совершается под влиянием входных сигналов. В процессе совершения перехода могут выполняться стандартные операторы процедурных языков программирования (операторы присваивания, перехода на метку, вызова процедуры и условный оператор), посылаться сигналы, выдаваться запросы на порождение/уничтожение процессов и использоваться таймеры.

Описание процесса представляет собой шаблон, по которому может быть создано несколько экземпляров, имеющих одно и то же имя, одни и те же входные/выходные маршруты и различающихся только своими личными идентификаторами. Процессы могут отправлять сигналы как отдельным экземплярам процессов, так и всем процессам, существующим в системе.

В процессе работы системы, описываемой на SDL, количество активных экземпляров процессов может меняться. Для этого служат операторы динамического порождения и удаления экземпляров процессов.

3. СЕТЕВАЯ МОДЕЛЬ

SDL-спецификации транслируются в сети, являющиеся модификацией известной модели раскрашенных сетей, предложенной Йенсеном. Раскрашенные сети дополнены концепцией времени.

Структура сети представляет собой направленный двудольный граф с двумя типами вершин — местами и переходами. Места имеют разметку, ненулевая разметка означает наличие в месте одной или нескольких фишек. Раскраска места означает тип фишки, например, целый или список записей (в терминах языка Паскаль).

Переходы являются активными компонентами сетей. Они срабатывают, забирая фишки из своих входных мест и помещая в выходные. Переход может сработать, если каждой переменной, используемой в выражениях на входных дугах перехода, можно сопоставить значение такое, что в каждом входном месте перехода имеется необходимое множество фишек и условие срабатывания перехода истинно.

Иерархическая сеть представляет собой множество неиерархических сетей, называемых страницами. Страницы могут содержать вершины специального типа, называемые модулями. Каждому модулю должна соответст-

вовать своя страница. Места, связанные дугами с модулями, называются гнёздами. Каждое такое место должно быть связано с местом на странице, соответствующей модулю. Такие места на странице модуля называются портами. Поведение иерархической сети определяется эквивалентной ей неиерархической сетью, получающейся при замещении всех модулей страницами, которые они представляют, и слиянии мест-гнёзд и мест-портов.

Декларации сети состоят из описания типов (множеств цветов в терминологии Йенсена), объявления переменных и, возможно, определения функций. Допускаются следующие типы: целый, вещественный, строковый, перечислимый, запись и список.

В сетевой модели Йенсена имеется понятие глобальных часов, посредством которых представляется текущее модельное время. Множество цветов может иметь признак *timed*. Фишка, принимающая значения из такого множества, дополнительно несет значение, называемое временным штампом. Оно определяет момент времени, раньше которого фишка не может использоваться при срабатывании перехода. При срабатывании перехода фишки, помещаемые в выходные места, получают временные штампы, равные $Now+T_п+T_д$, где Now — текущее время в сети, $T_п$ — временная пометка перехода, $T_д$ — временная пометка на выходной дуге.

4. ТРАНСЛЯЦИЯ

Трансляция SDL-спецификации в раскрашенные сети проходит в несколько этапов. На первом этапе создаётся синтаксическое дерево разбора SDL-спецификации, на втором — осуществляется генерация сетевой модели во внутреннем представлении и, наконец, на третьем этапе — генерация файла, содержащего описание модели в формате CPN Tools.

Синтаксический анализатор языка SDL построен с помощью программного средства Bison [6]. Анализатор осуществляет лексическую свертку текстового файла, содержащего SDL-спецификацию, его синтаксический разбор и построение внутреннего представления спецификации. В случае отсутствия ошибок запускается модуль генерации сетевой модели.

Генерация сетевой модели во внутреннем представлении осуществляется по шагам. На первом шаге создаётся страница, соответствующая всей спецификации. Она содержит модули, соответствующие блокам в SDL-спецификации, и места, соответствующие внешним каналам системы. Эти модули и места соединяются дугами в соответствии с описанием системы в спецификации. Места, соответствующие каналам, содержат фишки-списки

записей. Каждая такая запись соответствует одному сигналу, который может передавать данный канал.

На втором шаге для каждого блока спецификации строятся модули, соответствующие процессам в спецификации, и места, соответствующие маршрутам. На третьем шаге на страницах, соответствующих процессам, создаются места, соответствующие переменным и таймерам процессов, служебные места и модули, соответствующие переходам процессов.

На следующем шаге создаются подсети, моделирующие логику SDL-переходов. Каждому действию в SDL-переходе соответствует один или более переходов в сети.

Для каждого описания процесса, экземпляры которого будут создаваться только динамически, строится сеть-шаблон, в которой все места изначально не будут иметь никаких фишек. Для моделирования динамического поведения SDL-спецификаций экземпляр процесса в сетевой модели представляется набором фишек-записей, первое поле в которых соответствует личному идентификатору процесса. Фишки появятся в местах, после того как в сети сработает переход, моделирующий оператор порождения экземпляра процесса.

Помимо моделирования динамического порождения и удаления экземпляров процессов, нетривиальной оказалась задача моделирования таймеров с параметрами. По семантике SDL-таймер, имеющий один или несколько параметров, подобно блоку или процессу, представляет собой шаблон, по которому может быть создано несколько экземпляров. Если к некоторому моменту уже был выполнен оператор установки таймера и если таймер ещё не сработал (т.е. время, указанное в операторе установки, ещё не наступило в системе), то при выполнении нового оператора установки возможны два варианта. Если новая установка таймера совершается с теми же параметрами, то предыдущая установка отменяется. В противном случае таймер начинает ожидать два момента времени. При наступлении каждого из них процессу посылается сигнал от таймера с соответствующими параметрами.

Для таймеров с параметрами был выбран следующий способ моделирования в сетевой модели: такой таймер отображается местом, в котором фишка содержит список наборов параметров и моментов времени, на которые установлен таймер. Записи в этом списке отсортированы по времени, а временной штамп фишки всегда соответствует первому моменту времени, в который таймер должен сработать.

5. ОПТИМИЗАЦИЯ СЕТИ

Автоматическая верификация сетевых моделей, в основном, осуществляется посредством построения графов достижимости. Основная проблема при работе с такими графами — их большой размер. Частично эта проблема решена реализацией автоматической оптимизации сети — её сокращения, не меняющего поведения модели. Оптимизация позволяет значительно уменьшить время, необходимое для построения и анализа графов и, в некоторых случаях делает возможной анализ моделей, для которых мощности имеющегося аппаратного и программного обеспечения недостаточно в случае отсутствия оптимизации.

Оптимизация производится путём слияния объектов сети, соответствующих последовательным действиям в спецификации, не имеющих зависимостей по переменным, и удаления неиспользуемых объектов сети. Цепочка переход1—место—переход2 может быть сокращена до одного перехода (рис. 1, 2). Это происходит, если переход2 не содержит условия срабатывания и не использует переменных, значение которых меняется переходом1. Цепочка место1—переход—место2 может быть сокращена до одного места, если у данного перехода нет других входных/выходных дуг.

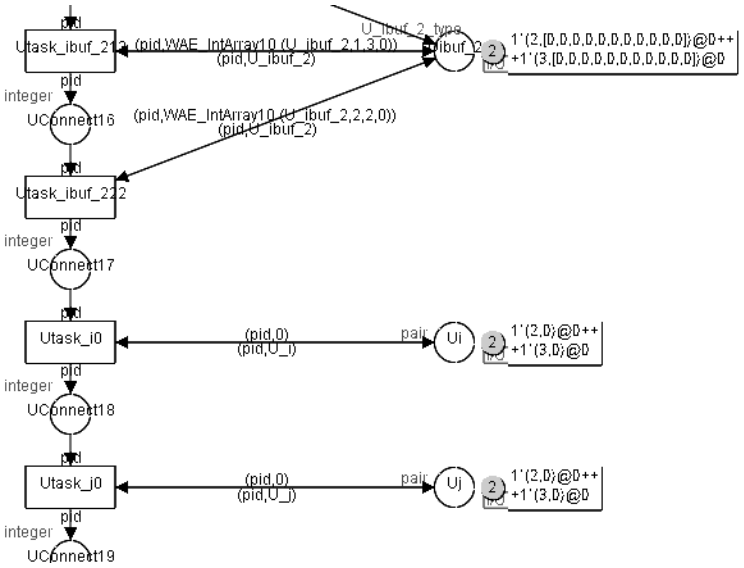


Рис. 1. Фрагмент сети, соответствующей SDL-переходу

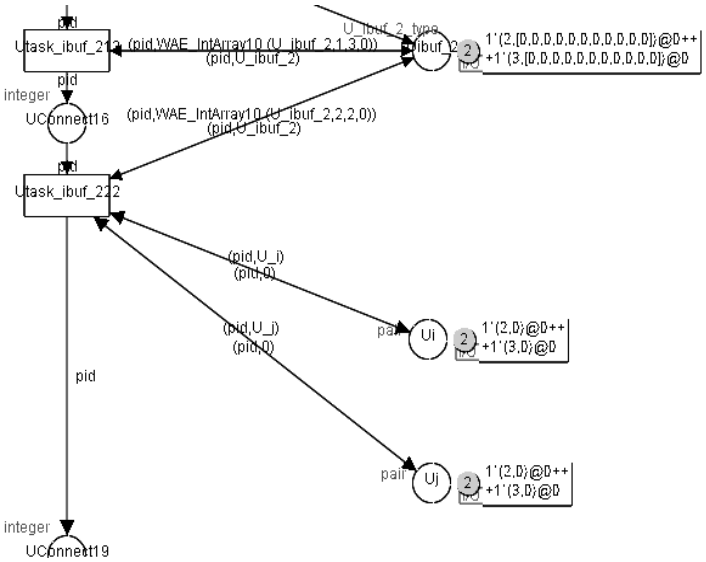


Рис. 2. Фрагмент сети после оптимизации

Оптимизация осуществляется отдельно для каждой страницы сети. После оптимизации одному переходу в сети может соответствовать несколько действий в исходной SDL-спецификации.

6. ЭКСПЕРИМЕНТЫ

Проведены эксперименты по трансляции и анализу кольцевого (RE) протокола [7], InRes-протокола [3, 4], i-протокола [8], ATMR-протокола [2] и протокола с выборочным повтором Таненбаума [9]. Объёмы SDL-спецификаций протоколов — порядка 200−400 строк. Генерация сетевой модели для каждой из спецификаций занимает порядка секунды на компьютере Athlon XP-1600. Описанная выше оптимизация позволила уменьшить количество переходов в сетях на 20−40 %.

Загрузка в систему CPN Tools и синтаксическая проверка сетевых моделей требует нескольких минут процессорного времени, инициализация модуля построения графов достижимости — от 10 минут до часа. Построение графов занимает от нескольких секунд (при размерах графа порядка тысячи вершин) до десяти часов (для графов, содержащих 200000 вершин). При

наличии 512 МБ оперативной памяти CPN Tools позволяет работать с графами, размером до 200000–400000 вершин.

Для анализа графов достижимости сетевых моделей может использоваться система CPN Tools, однако средств анализа, имеющихся в CPN Tools на данный момент, оказалось недостаточно, и для того чтобы использовать внешние средства анализа, был реализован модуль экспорта графа достижимости в текстовый файл. Модуль реализован на встроенном языке системы CPN Tools ML.

Наиболее интересны эксперименты по анализу протоколов ATMR и InRes, SDL-спецификации которых содержат динамические конструкции SDL.

6.1. ATMR-протокол

Модель ATMR-протокола [2] содержит несколько экземпляров процесса-станции Unit. Каждая станция получает идентификатор экземпляра процесса, соответствующего станции, следующей далее по кругу. Станция всегда отправляет сообщения с этим идентификатором в качестве процесса-получателя, и таким образом станции образуют кольцо. Экземпляры процесса Unit создаются динамически отдельным процессом Creator.

Был проведён эксперимент с тремя станциями, которые отправляли 3, 4 и 5 блоков данных соответственно, и надёжной средой передачи данных. Содержимое блока данных моделировалось одним целым числом. Сетевая модель содержала 72 перехода и 73 места в 10 модулях. Был получен полный граф достижимости, содержащий 74643 вершины. Суммарное время генерации сетевой модели, построения графа и вывода графа в файл составило менее двух часов на машине с процессором Pentium 4-1600 и 512 МБ ОЗУ.

6.2. Inres-протокол

Система Inres [3, 4] не является реальной системой, но содержит многие базисные концепции модели взаимодействия открытых систем и удобна для иллюстрации. Протокол описывает взаимодействия между двумя протокольными объектами: Initiator и Responder. Связь между ними происходит через ненадежную среду передачи, которая может терять сообщения, хотя не может их копировать или переупорядочивать. Каждый сеанс связи между объектами Initiator и Responder разбит на фазу установления связи и фазу передачи данных. Обмен данными в системе реализован через дополни-

тельные процессы. SDL-спецификация протокола была расширена процессами User_Ini и User_Res, моделирующими пользователей протокола.

Целью эксперимента с протоколом InRes была проверка корректности передачи данных. Проверка осуществлялась для одного сеанса связи, возможности рассоединения и потери сообщений были исключены. Для работы со многими пользователями в спецификацию был добавлен процесс, который во время функционирования по истечении времени, установленном в таймере, создаёт экземпляры всех остальных процессов. Каждая пара пользователей обслуживается одним набором протокольных объектов. Все сигналы в системе несут личные идентификаторы процесса-получателя и процесса-отправителя.

Объём сетевой модели спецификации составил 33 модуля, 225 переходов и 222 места. Построение сетевой модели заняло около секунды, загрузка полученного текстового представления в CPN Tools и инициализация модуля симуляции — около 10 минут.

Во время эксперимента с полученной сетевой моделью была подтверждена неэффективность протокола, описанная в работе [3] и состоящая в дублировании данных, передаваемых между протокольными объектами. С помощью пошаговой симуляции было выяснено, что дублирование данных происходит в случае, когда Initiator не успевает получить подтверждение и отправляет дубль сообщения, которое уже было получено процессом Responder. Эта ситуация возникает в том случае, если установленное в таймере время недостаточно велико, и может быть исключена путём установки больших значений задержек таймеров, достаточных для того, чтобы Initiator и Responder успели обменяться сообщениями. После изменения временных задержек поведение сети соответствовало ожидаемому.

ЗАКЛЮЧЕНИЕ

Автоматическая генерация сетевых моделей коммуникационных протоколов существенно сокращает трудоёмкость проведения экспериментов по их анализу и верификации, а использование принципа иерархии — поуровнего создания сети — делает возможным построение сетевых моделей для систем реальной сложности.

При помощи описанного транслятора были получены сетевые модели кольцевого (RE) протокола, InRes-протокола, i-протокола, ATMR-протокола и протокола с выборочным повтором Таненбаума. В ходе экспе-

риментов были подтверждены семантические ошибки в кольцевом и i-протоколе.

Список литературы

1. **Jensen K.** Coloured Petri nets: Basic concepts, analysis methods and practical use. — Springer-Verlag, 1996. — Vol. 1–3.
2. **ISO**. Specification of the Asynchronous Transfer Mode Ring (ATMR) protocol, 2.0 edition, 1993.
3. **Fisher J., Dimitrov E.** Verification of SDL'92 specifications using extended Petri nets // Proc. IFIP 15th Intern. Symp. on Protocol Specification, Testing and Verification. — Warsaw, Poland, 1995. — P. 455–458.
4. **Непомнящий В. А., Алексеев Г. И., Быстров А. В. и др.** Верификация Estelle-спецификаций распределенных систем посредством раскрашенных сетей Петри. — Новосибирск, 1998. — С. 103–112.
5. **Ratzer A.V. et al.** CPN Tools for editing, simulating, and analysis Coloured Petri Nets // Proc. ICATPN 2003. — Lect. Notes Comput. Sci. — 2003. — Vol. 2679. — P. 450–462.
6. `http://www.gnu.org/software/bison/bison.html`
7. **Cohen R., Segall A.** An efficient reliable ring protocol. — IEEE Transact. Communs. 1991. — Vol. 39, N. 11. — P. 1616-1624.
8. **Dong Y. et al** Fighting livelock in i-protocol: a comparative study of verification tools // Proc. TACASS'99. — Lect. Notes Comput. Sci. — 1999. — Vol. 1579. — P. 74–88.
9. **Таненбаум Э.** Компьютерные сети. — СПб.: Питер, 2003.

В.П. Петров

ОБЗОР МЕТОДОВ ПРИМЕНЕНИЯ СХЕМ ДАННЫХ И ОНТОЛОГИЙ ДЛЯ ОБЪЕДИНЕНИЯ РАСПРЕДЕЛЕННЫХ, ГЕТЕРОГЕННЫХ, ИНФОРМАЦИОННЫХ РЕСУРСОВ

ВВЕДЕНИЕ

Оптимальное использование информационных ресурсов предполагает возможность доступа к существующей информации, а также ее обработки. В реальных задачах, при работе с информационными ресурсами, мы имеем дело с распределенными системами, у которых отсутствует стандарт на доступ к информации. Для того чтобы разрешить задачу эффективного обмена и использования информационных ресурсов, требуется решить ряд технических проблем. Во-первых, это проблема поиска и фильтрации информационных ресурсов [1, 16]. Во-вторых, организация непосредственного доступа к данным и объединение ресурсов нескольких источников. Если проблема поиска решается довольно эффективно при помощи использования существующих поисковых систем и технологий и подробно описана рядом исследователей [16–18], то для решения задач непосредственного доступа до сих пор не существует универсального подхода. При работе с распределенными и в тоже время гетерогенными, т. е. разнородными и несовместимыми вычислительными системами, возникает задача объединения всех информационных ресурсов. Решение задачи объединения предполагает возможность взаимодействия отдельных узлов друг с другом либо выделение отдельного сервиса, самостоятельно взаимодействующего с каждым узлом системы. Если рассмотреть подробнее взаимодействие между элементами распределенной системы, можно выделить следующие уровни [5, 6].

1. *Уровень сетевых протоколов.* Как правило, реализуется с использованием одной из стандартных технологий клиент-серверного взаимодействия. На практике, исследователями, при разработке распределенных приложений для поддержки семантических сетей, достаточно часто используется технология Web-Services [10, 14]. Также применимы другие клиент-серверные технологии DCOM, CORBA, RMI [5, 14, 22].

2. *Уровень структурного взаимодействия.* Структура данных в разных системах может различаться и требует дополнительного преобразования информационных потоков к общему виду. Преобразование можно выполнять на уровне клиент-серверного взаимодействия, что поддерживает, например, технология CORBA, а также при помощи специальных преобразователей, или медиаторов [5, 6, 22].
3. *Уровень информационного взаимодействия.* Взаимодействие на уровне данных предполагает не столько возможность доступа к информации, сколько возможность обработки и использования данных прикладными системами. Также может различаться информационный смысл самих данных. Для взаимодействия системы должны располагать совпадающим набором сущностей в описаниях схем данных или в онтологии [5, 6].

Несовместимость семантики информационных ресурсов может возникать в некоторых случаях. Можно выделить следующие [3, 4, 27]:

а) информационные сущности или объекты обладают одинаковым значением, но используются в разных контекстах,

б) несовпадение размерности, например, в разных контекстах или в разных ресурсах может использоваться разная система измерений,

в) конфликты имен — сущности могут обладать одинаковыми названиями, но в действительности абсолютно различаться.

В этой работе мы рассмотрим некоторые методы применения онтологий (схем данных) в распределенных, гетерогенных системах.

СХЕМА ДАННЫХ И ОНТОЛОГИЯ

Онтология определяет основные правила описания и представления пространства знаний некоторой предметной области. Онтология используется для явного описания семантики информационных ресурсов [7, 20, 21, 23, 27]. Схемы данных, описывающие структуры XML или RDF документов, представляют простейшие примеры использования онтологий [20, 27, 28]. Можно выделить следующие базовые способы описания онтологии, пользующиеся популярностью у исследователей в последнее время [20, 23–28]: UML, RDFS, и его развитие — OWL.

При объединении нескольких информационных ресурсов, можно выделить три основных способа интеграции с использованием схемы данных [2, 25, 26]:

- объединение ресурсов с выделением единой онтологии, или в простом случае — общей схемы данных,
- объединение ресурсов с отдельной онтологией для каждого,
- объединение ресурсов с отдельной онтологией для каждого, но с общим базисом основных примитивов.

Очевидно, что объединение ресурсов с выделением единой онтологии является частным случаем двух других подходов.

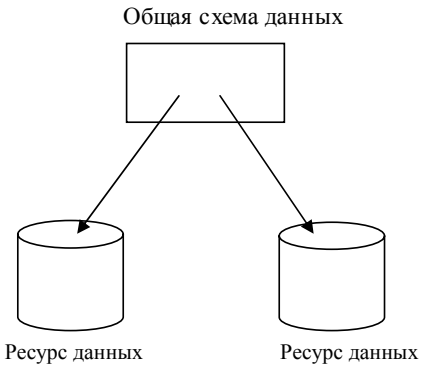


Рис. 1

Общая схема данных (единая онтология).

Самый простой способ (рис. 1). При таком способе интеграции данных используется единая онтология, содержащая общую схему данных для описания семантики подключенных информационных ресурсов. Если первоначально семантика каждого информационного ресурса описывалась собственной схемой данных, можно построить глобальную схему данных, являющейся комбинацией всех частных схем данных. Методика модульного импорта (или включения) онтологии успешно используется исследователями для некоторых реализаций общих схем данных в распределенных системах с единой онтологией [7, 20]. Возможность объединения нескольких онтологий в одну — общую для всей системы — определяется тем, каким образом отличаются их контексты, существуют ли пересечения в описаниях схем данных и совпадает ли базовый набор примитивов, использованных в описаниях этих онтологий. Очевидно, что построение единой онтологии на базе нескольких онтологий из различных предметных областей вполне

возможно, так как отсутствуют пересечения в контекстах применяемых терминов. Также объединение двух онтологий, каждая из которых содержит непересекающиеся расширения (или дополнения) некоторой базовой схемы данных, будет простой задачей. Но объединение двух онтологий из одинаковой предметной области с разным набором сущностей либо с одинаковыми сущностями, но основанными на несовпадающих наборах базовых примитивов, будет нетривиальной задачей [8]. Такой способ интеграции хорошо подходит при объединении информационных ресурсов, семантика которых описана слабо различающимися схемами, с преобладающими общими типами, без пересечений в дополнениях.

Если рассматривать эволюцию распределенной системы в процессе постепенного изменения информационных ресурсов, входящих в ее состав, можно выяснить, что изменения в ресурсах данных сильно влияют на единую онтологию и могут затрагивать все подключенные ресурсы. В зависимости от качественных характеристик изменений, они могут потребовать изменений в общей схеме данных и даже отразиться на схемах других информационных ресурсов.

Недостатки данного метода интеграции отсутствуют у решений с раздельными схемами данных.

Раздельная схема данных (множественная онтология).

Для способа интеграции с множественной онтологией, каждому информационному ресурсу соответствует собственное семантическое описание [9] (рис. 2). Возможно также, что каждая онтология является некоторой комбинацией других, но не предполагается возможность выделять общую базовую онтологию, или некоторый словарь примитивов, общий для всех частных схем данных.

Такой способ интеграции нескольких информационных ресурсов предполагает минимальную зависимость каждого ресурса и его схемы данных от других, что существенно облегчает модификацию информационной структуры данных каждого источника по отдельности, а также упрощает добавление новых или удаление существующих информационных ресурсов [8].

С другой стороны, задача программной поддержки усложняется из-за отсутствия точек пересечения между разными онтологиями. Недостаток общих базовых примитивов усложняет методы доступа, получения и сравнения информации из нескольких информационных источников одновременно [20, 27]. Очевидно, что отсутствие минимального базиса общих примитивов, потребует создания некоторого формализма, устанавливающего

соответствия между отдельными схемами данных для возможности сопоставления данных в информационных ресурсах, описанных разными онтологиями. Существуют примеры формализации создания соответствий между отдельными онтологиями [5, 9].

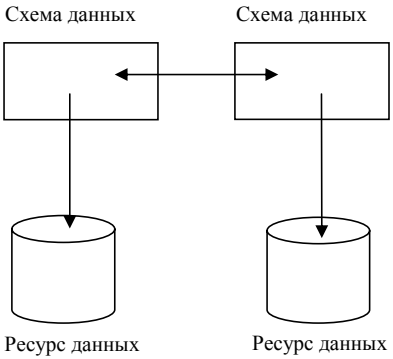


Рис. 2

Раздельная схема данных с общим базисом основных примитивов (гибридная онтология).

Этот способ является комбинацией двух предыдущих. Для того чтобы обойти недостатки предыдущих подходов, требуется оставить описание схемы отдельно для семантики каждого из источников данных, но для возможности взаимодействия и сопоставления данных из разных ресурсов преобразовать схемы данных таким образом, чтобы они использовали общий базис основных примитивов [5, 6] (рис. 3).

Общий базис содержит основные термы (или примитивы), одинаковые для схем данных семантики информационных ресурсов. Для того чтобы выстраивать более сложные термы схем существующих источников информации, исходные примитивы общего словаря объединяются посредством комбинирующих операторов, описывающих онтологию. Так как любой терм схем источников информации определяется из набора существующих примитивов общего базиса, возможность обработки и сопоставления данных упрощается [5].

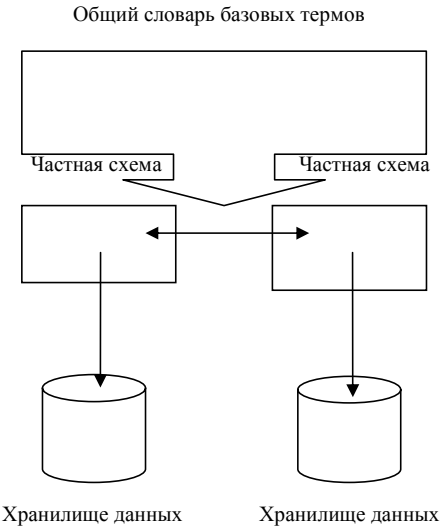


Рис. 3

Вместо базиса также можно выделить более сложную сущность, например, выделенную базовую схему, общую для всех частных схем данных. В отличие от способа интеграции с применением единой онтологии, при таком способе не возникает проблем в задаче объединения двух разных ресурсов одинаковой предметной области при частичном несовпадении схем данных.

Еще одно качество такого подхода заключается в том, что есть возможность добавления новых информационных ресурсов без внесения изменений в существующие схемы и, возможно, без изменения общего базиса [27]. Это качество облегчает задачу поддержки последовательных изменений информационных ресурсов в процессе эволюции программной системы.

Несмотря на описанные преимущества, недостаток подхода заключается в том, что не все существующие схемы могут быть легко переиспользованы или преобразованы к новому виду с выделением базовой схемы или набора базовых примитивов. Как пример, можно привести две различные онтологии, описывающие одну и ту же предметную область, но различающиеся по общим базовым примитивам. В этом случае, схемы потребуют

разработки с самого начала, так как все участвующие во взаимодействии источники информации должны описываться схемами, основанными на едином базисе основных примитивов, описанных в общем словаре [7, 20, 27].

ГЛОБАЛЬНАЯ МОДЕЛЬ ЗАПРОСОВ

Кроме непосредственного описания данных, схема данных (или онтология) может свободно применяться как глобальная модель запросов для используемых источников информации [14, 15]. Совершенно очевидно, что управление большим числом информационных ресурсов, семантика которых описывается сложными схемами данных, не тривиально при использовании низкоуровневого интерфейса RDF/RDFS. Преимущество реляционных баз данных перед файловыми архивами заключается в возможности декларативного доступа и в разделении логической и физической структуры данных. Под разделением логической и физической структуры данных информационных ресурсов следует понимать возможность явного указания только требуемых ресурсов или данных. Задача эффективного хранения и доступа возлагается на дополнительную программную прослойку [15]. Из существующих стандартов стоит отметить RDQL и RQL [24].

Возможность использования онтологии как модели для построения запросов основывается на том, что структура запроса и результата должна отвечать требованиям и возможностям предметной области информационных ресурсов.

Для программных реализаций можно выделить следующие моменты: поступающий запрос, основанный на глобальной онтологии, разбивается на подзапросы к отдельным ресурсам, результаты исполнения которых, сопоставляются и комбинируются в общий результат. Декомпозиция порождающего запроса на подзапросы должна выполняться с учетом того, что результаты обработки отдельного подзапроса должны будут объединиться со всеми остальными результатами для создания общего и не противоречивого в рамках глобальной онтологии результата [12, 29].

ПОДДЕРЖКА ЭВОЛЮЦИИ ИНФОРМАЦИОННЫХ СИСТЕМ

В процессе эксплуатации программных систем возникают ситуации, требующие модификации информационных ресурсов. В зависимости от выбранной модели объединения нескольких ресурсов, модификация от-

дельного ресурса может в разной степени повлиять как на остальные ресурсы, так и на общую информационную среду системы. Можно выделить следующие виды возможных изменений системы, использующей онтологию или схему данных для описания информационных ресурсов:

- непосредственная модификация информации,
- изменение явной спецификации онтологии,
- модификация описания и представления пространства знаний,
- изменение предметной области.

Модификация информации в данном случае означает изменение существующих записей как, например, экземпляров некоторого класса, или — если брать для примера реляционную модель — записей в некоторой таблице реляционной базы данных. Также возможно добавление новых записей и удаление существующих. В некоторых случаях, подобная модификация может потребовать дополнительной проверки целостности и достоверности данных. Такая потребность может возникать, например, при наличии описанных в схеме данных, ограничений на количество возможных связей между элементами семантической сети или ограничений на возможные значения свойств элементов.

Изменение явной спецификации онтологии (или схемы данных).

Самый простой пример подобного изменения — это переход от одного способа описания онтологии к другому. Способы описания могут отличаться не только синтаксисом (это в основном характерно для языковых методов описания схем данных), но и своей выразительностью и семантикой.

Модификация описания и представления пространства знаний.

Изменения в описании схем данных могут быть результатом изменений информационных потребностей потребителей. Как пример, изменение набора свойств у отдельных сущностей информационного пространства. На примере реляционных баз данных это будет выглядеть как изменение в наборе колонок некоторой таблицы.

Изменение предметной области.

Результат изменения базового набора основных примитивов. Изменение в предметной области можно сравнить с изменением в схеме данных реляционных баз данных. Подобное может быть, например, в случае добавления новых фундаментальных сущностей.

Возможности поддержки эволюции и версионности.

Изменение схем данных и онтологий может привести к тому, что уже существующие данные потеряют целостность и достоверность в рамках новых схем данных [11, 13, 32]. Из-за такой возможности иногда необходимы методы, поддерживающие существующие данные в достоверном состоянии при возможных изменениях схем. Самый простой пример — методы поддержки версий, доступные в RDF(S) и OWL [11, 31 – 33]. Кроме методов поддержки актуальности данных существуют технологии преобразования данных при помощи промежуточных медиаторов, в некоторых случаях с использованием специализированных языков описания и поддержки изменений в схемах данных [30].

ЗАКЛЮЧЕНИЕ

В этой работе мы рассмотрели основные методы применения схем данных и онтологий для объединения распределенных, гетерогенных, информационных ресурсов. Описанные способы объединения информационных ресурсов могут быть использованы в качестве основы для создания более сложных архитектурных комбинаций. С ростом популярности распределенных информационных систем проблема эффективной поддержки эволюции и версионности онтологии становится более актуальной. Можно выделить следующие предметы для исследования эволюции онтологии и методов поддержки версий, требующие более глубокого исследования:

- одновременная поддержка нескольких версий онтологии и данных, описанных разными версиями;
- возможность поиска различий между версиями при отсутствии информации о непосредственной трансформации;
- определить необходимый и достаточный набор операций трансформации онтологии и исследовать влияние этих операций на разные виды онтологии;
- необходимость поддержки целостности и достоверности информации при изменении в онтологии.

СПИСОК ЛИТЕРАТУРЫ:

1. **Kleinberg J.M.** Authoritative Sources in a Hyperlinked Environment // Proc. of the Annual ACM-SIAM Symposium on Discrete Algorithms, New York. — ACM, 1998. — P. 668–677.

2. **Calvanese D., De Giacomo G., Lenzerini M.** Description logics for information integration // Computational Logic: From Logic Programming into the Future. — Springer-Verlag, 2001.
3. **Bressan S., Goh C.H.** A Procedure for Mediation of Queries to Sources in Disparate Contexts. — 1997.
4. **Bressan S., Goh C.H.** Answering Queries in Context. — 1998.
5. **Visser U.** Stuckenschmidt H. Enabling Technologies for Interoperability. — 2000.
6. **Stuckenschmidt H.** Ontology-Based Information Sharing in Weakly Structured Environments. — 2002.
7. **Gruber T.** A translation approach to portable ontology specifications // Knowledge Acquisition. — 1993. —5(2). — P. 199–220.
8. **Gruber T.** Toward principles for the design of ontologies used for knowledge sharing. — 1993.
9. **Mena E., Kashyap V.** OBSERVER: An Approach for Query Processing in Global Information Systems based on Interoperation across Pre-existing Ontologies. — 1996.
10. **Sollazzo T., et al.** Semantic Web Service Architecture — EvolvingWeb Service Standards toward the Semantic Web // Proc. of the 15th Internat. FLAIRS Conf., Pensacola, Florida, 2002. — AAAI Press, 2002.
11. **Heflin J., Hendler J.** Semantic interoperability on the web // Extreme Markup Languages 2000. — 2000.
12. **Decker S., Brickley D.** A Query and Inference Service for RDF. — 1998.
13. **Fensel D., Bussler C**. Semantic Web Application Areas. — 2002.
14. **Visser U., Schuster G.** Finding and Integration of Information A Practical Solution for the Semantic Web // Proc. of ECAI 02, Workshop on Ontologies and Semantic Interoperability, Lyon, France, 2002. — P. 73-78
15. **Karvounarakis G., Alexaki S.** RQL: A Declarative Query Language for RDF. — 2002.
16. **Wang M.** A Significant Improvement to Clever Algorithm in Hyperlinked Environment. — 2002.
17. **Gibson D., Kleinberg J.** Inferring Web Communities from Link Topology. — 1998.
18. **Brin S.** The Anatomy of a Large-Scale Hypertextual Web Search Engine. — 1998.
19. **Pan J., Horrocks I.** Metamodeling Architecture ofWeb Ontology Languages. — 2001.
20. **Eberhart A.** Survey of RDF data on the Web. — 2002.
21. **Haustein S.** Semantic Web Languages: RDF vs. SOAP Serialisation. — 2001.
22. **Doerr M., Hunter J.** Towards a Core Ontology for Information Integration. — 2002.
23. **Wilkinson K., Sayers C.** Efficient RDF Storage and Retrieval in Jena2. — 2003.
24. **Cai M.** RDFPeers: A Scalable Distributed RDF Repository. — 2004.
25. Wilkinson1 k., Sayers C. Supporting Scalable, Persistent Semantic Web Applications // Bull. of the IEEE Computer Society Technical Committee on Data Engineering. — 2003.

26. **Wache H., Visser U., Scholz Th.** Ontology construction — An iterative and dynamic task // Proc. of the Florida Artificial Intelligence Research Society Conf. (FLAIRS), Pensacola, FL, USA, 2002. — P. 445-449.
27. **Cranefield S.** Networked Knowledge Representation and Exchange using UML and RDF // J. of Digital Information. — 2001. — Vol. 1, Iss. 8, No. 44.
28. **Miller L., Seaborne A.** Three Implementations of SquishQL, a Simple RDF Query Language. — 2002.
29. **Sindt T.** Formal Operations for Ontology Evolution // Proc. of the Int'l Conf. on Emerging Technologies', 2003.
30. **Zhang Z., Zhang L.** Data Migration for Ontology Evolution. — `http://apex.sjtu.edu.cn/docs/DataMigration_final.pdf`
31. **Heflin J., Hendler J.** Dynamic Ontologies on the Web. — 2000.
32. **Klein M., Fensel D.** Ontology versioning and change detection on the Web. — 2002.

Е.А. Плавенчук

РАСШИРЕНИЕ ВОЗМОЖНОСТЕЙ СИСТЕМЫ ФИНПЛАН НА ОСНОВЕ СТРУКТУРНЫХ МОДЕЛЕЙ

В статье рассказывается о структурных моделях — средстве представления недоопределенных вычислительных моделей и разработанной на их основе схеме представления и взаимодействия недоопределенных вычислительных моделей в системе ФинПлан.

ВВЕДЕНИЕ

Аппарат недоопределённых вычислительных моделей (Н-моделей) [1], относящийся к направлению *Программирование в ограничениях (constraint programming),* одному из наиболее перспективных в области ИТ, предоставляет уникальные возможности для решения расчётных задач с неполными и неточно определёнными исходными данными.

К числу наиболее развитых инструментов недоопределённых вычислений относится решатель UniCalc [2, 3]. Решатель используется как в качестве самостоятельного приложения, так и в виде вычислительного ядра ряда прикладных систем, в частности системы ФинПлан [4, 5], позволяющей работать с моделями и задачами, представленными в виде недоопределённых (интервальных) электронных таблиц.

Применение Н-моделей для решения практических задач в таких областях, как экономика, ресурсно-календарное планирование, инженерные расчёты, связано с построением моделей объектов, имеющих сложную внутреннюю структуру, для описания которых требуется большое количество параметров и ограничений. Этим обстоятельством объясняется актуальность создания таких средств разработки Н-моделей, которые обеспечивают следующие возможности:

- представление структуры объектов моделирования;
- наглядное представление Н-моделей с большим количеством ограничений и параметров;
- многократное использование типовых фрагментов модели;
- раздельная разработка и отладка фрагментов моделей.

Первостепенную роль в обеспечении таких возможностей играет представление Н-моделей в структурированном виде. С этой целью в РосНИИ ИИ разрабатывается аппарат структурных недоопределённых моделей. В этой статье рассматривается возможность использования этого аппарата в разрабатываемой в настоящее время новой версии системы ФинПлан, которая должна поддерживать эффективную работу с системой взаимосвязанных недоопределённых электронных таблиц.

СТРУКТУРНЫЕ НЕДООПРЕДЕЛЕННЫЕ МОДЕЛИ

Структурные недоопределённые модели (СМ) представляют собой расширение аппарата Н-моделей средствами их структурирования. Структурирование модели обеспечивается разделением её на множество модулей, каждый из которых имеет уникальный идентификатор и содержит некоторое подмножество параметров модели и ограничений на них. Модули связаны деревом транзитивных отношений вложенности. Таким образом, каждый модуль может включать в себя произвольное число других модулей, но при этом каждый модуль может быть непосредственно вложен только в один модуль. При этом вложенный модуль, в свою очередь, выступает в роли сложного ограничения на параметры охватывающего и связывается с внешней по отношению к нему частью Н-модели через выделенные в нем внешние переменные, которые включаются в ограничения вместе с переменными охватывающего модуля.

Рассмотрим два примера использования структурных моделей. На рис. 1 представлена СМ, набор ограничений которой включает уравнение второй степени с недоопределёнными коэффициентами. Данное уравнение, его коэффициенты и переменная содержатся в главном модуле модели Parabola. Коэффициенты могут быть определены с помощью других систем ограничений, в которые также входят дополнительные переменные. Эти системы ограничений описаны во вложенных модулях Ac и Ab. Модули связаны через общие параметры, благодаря чему их наборы ограничений образуют единую вычислительную модель.

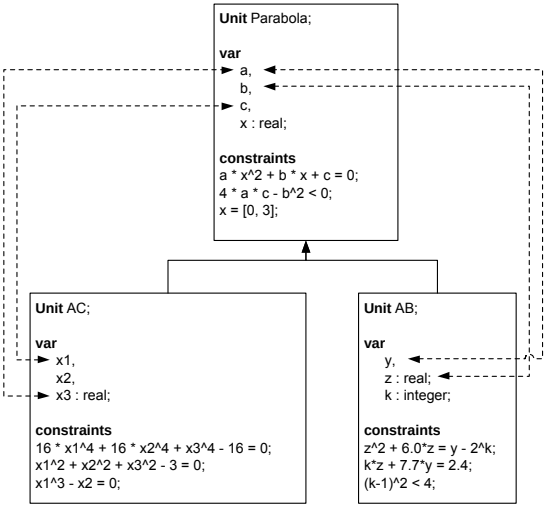


Рис.1. Структурная модель параболы с недоопределёнными коэффициентами, доопределяемыми с помощью систем дополнительных ограничений

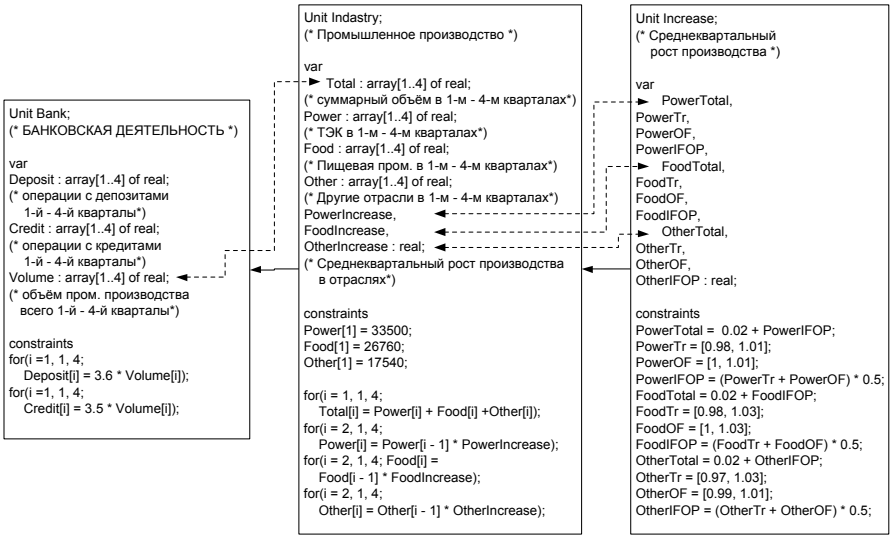


Рис. 2. Структурная модель экономики региона

На рис. 2 представлена структурная H-модель экономики региона. В модели отражены две сферы экономики — банковская деятельность и промышленное производство. В отдельные модули выделены группы параметров и ограничений, характеризующие объём различных видов банковских операций, объём производства в различных отраслях промышленности и показатели среднеквартального роста производства.

Объём банковских операций с кредитами и депозитами связан с суммарным объёмом промышленного производства. Для задания этого отношения в модуль Bank, содержащий подмодель банковской деятельности, вложен модуль Industry, содержащий подмодель промышленности. Модули связаны через массив внешних параметров Total, представляющих объём производства в 1–4 кварталах.

Объём производства в разных отраслях промышленности во 2–4 кварталах определяется через известный объём производства в первом квартале и показатели среднеквартального роста производства. Эти параметры и ограничения, определяющие их значения, сгруппированы в модуле Increase. Данный модуль вложен в модуль Industry и связан с ним через внешние параметры PowerTotal, FoodTotal, OtherTotal, представляющие обобщающие показатели роста производства в топливно-энергетическом комплексе, пищевой промышленности и других отраслях, соответственно.

Приведённые примеры показывают, что предложенная структура H-модели позволяет:

- улучшить обозримость модели за счёт представления сложных систем ограничений на разном уровне обобщения;
- реализовать средства навигации по модели;
- обеспечить возможность автономной разработки и отладки фрагментов модели, состоящих из отдельных модулей или поддеревьев.

Кроме того, на основе модульной структуры могут быть реализованы такие дополнительные возможности разработки моделей и организации процесса вычислений, как применение типовых модулей и представление модуля как логического выражения. Рассмотрим эти возможности более подробно.

Типовой модуль предназначен для представления набора ограничений, который может применяться неоднократно в разных структурных компонентах H-модели по отношению к различным наборам переменных. Так же как и обычные модули, типовой модуль содержит наборы ограничений и параметров, некоторые из которых выделяются как внешние. Такой модуль служит шаблоном, по которому может быть порождено любое количество модулей-экземпляров, которые, в свою очередь, включаются в модульную

структуру модели и служат в качестве ограничений для конкретных переменных. Применение типовых модулей может существенно упростить процесс конструирования моделей, повысить их компактность и наглядность.

Поскольку каждый модуль можно рассматривать как логическое выражение — конъюнкцию его ограничений, то имя модуля может выступать в модели как логическая переменная, значением которой является это логическое выражение. Благодаря определению логического значения модуля появляется возможность обрабатывать ситуации, когда ограничения модуля **A** не удовлетворяются **(A = ЛОЖЬ) → B**. Обработка такой ситуации может включать изменение модели и продолжение вычислений с другой системой ограничений, которая, возможно, будет совместной.

Модульная структура позволяет также повысить эффективность элементов динамики базовых Н-моделей, которые представлены условными ограничениями **A → B**. Такие ограничения позволяют расширять активную часть Н-модели, подключая к ней отношения **B** при выполнении условий **A**. Как уже было отмечено выше, модуль структурной модели также представляет собой отношение, поэтому в правой части условного ограничения может быть записано имя модуля.

Таким образом, предложенный аппарат структурных моделей может служить основой для создания средств разработки Н-моделей, отвечающих требованиям, сформулированным во введении.

СТРУКТУРНОЕ ПРЕДСТАВЛЕНИЕ МОДЕЛЕЙ В СИСТЕМЕ ФИНПЛАН

Электронная таблица в технологии ФинПлан представляет собой совокупность ячеек-параметров, каждой из которых может быть сопоставлена локальная Н-модель, задающая ограничения на значение данной ячейки. Ограничения могут связывать данную ячейку с другими ячейками, а также глобальными переменными, не входящими в табличную структуру. Наряду с локальными моделями ячеек таблица может также содержать глобальную Н-модель, включающую ограничения, не привязанные к конкретным ячейкам таблицы. Глобальные и локальные ограничения таблицы, ячейки, глобальные переменные и их значения вместе образуют Н-модель таблицы, которая вычисляется решателем UniCalc.

Увеличение размера таких таблиц, естественно, вызывает необходимость разделения их на взаимосвязанные модули (подтаблицы), что позволит улучшить обозримость таблицы, выявить её содержательную структу-

ру, обеспечить возможность конструирования и отладки таблицы по частям и использования готовых фрагментов.

Структурные модели послужили основой для разработки новой схемы взаимодействия между таблицами в системе ФинПлан. Модели в терминах СМ представляются в виде модулей и разбиваются на более мелкие модули. Конечными элементами такого разбиения являются таблицы ФинПлана.

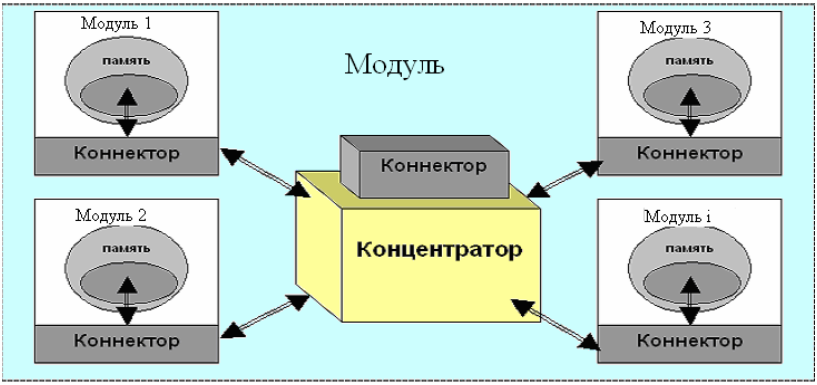


Рис. 3. Модульное представление модели

Каждая таблица снабжается табличным коннектором — внешним интерфейсом таблицы, обеспечивающим ее взаимодействие с другими элементами охватывающего модуля и с самим охватывающим модулем. Связующим звеном между коннекторами является еще одна введенная сущность — концентратор. Концентраторы позволяют связать данные, предоставляемые коннекторами, в общую вычислительную модель. Сами концентраторы также обладают коннекторами и могут экспортировать данные. Совокупность коннекторов и концентраторов, связанных вычислительными моделями концентраторов, и есть модуль. Его внешними переменными будут переменные любого коннектора, входящего в модуль (рис. 3).

Коннекторы служат для обмена данными между таблицами ФинПлана, концентраторами и любыми другими источниками данных, имеющими коннекторы. Коннекторы для разных сущностей выглядят одинаково. Коннектор представляет собой список переменных, которые он экспортирует и/или импортирует. О каждой переменной известно ее имя, тип, текущее значение и тип доступа к ней. Коннектор не обязан представлять сведения об источнике данных. Данные из коннекторов могут быть взяты концентра-

тором и использованы в его модели, которая может связывать общими ограничениями данные из разных коннекторов.

Концентратор связывает данные из разных коннекторов в единую вычислительную модель. Помимо этого концентратор обладает собственным коннектором и может экспортировать/импортировать данные (рис. 4). Каждому концентратору можно задать конфигурацию вычисления, т.е. модели концентраторов могут вычисляться как независимо, так и совместно с моделями других концентраторов и таблиц.

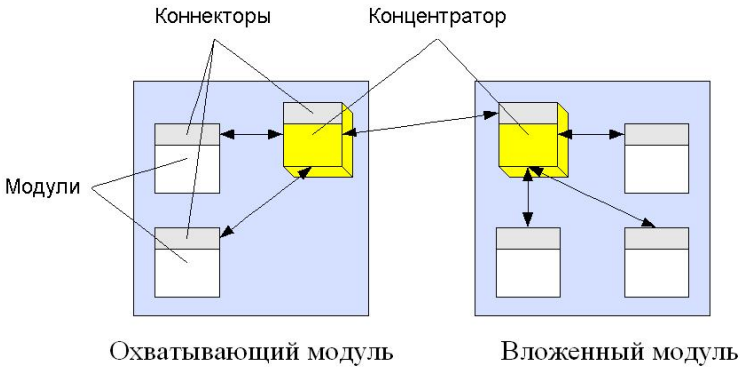


Рис. 4. Схема представления модели

Предложенное в этом разделе представление Н-модели позволяет выделять ее части и использовать в других моделях. Таким образом, может быть сформирована библиотека типовых модулей (в терминах СМ), которые могут быть использованы в других моделях.

Такое представление вычислительной модели позволяет распределять ее модули между рабочими станциями, тем самым обеспечивая ее совместное использование различными пользователями. При этом коннекторы четко определяют доступ к данным, а модели концентраторов могут представлять собой те части вычислительной модели, изменение которых пользователями нежелательно. Концентраторы могут инициировать автоматический пересчет данных в случае изменения данных, импортируемых из коннекторов.

ЗАКЛЮЧЕНИЕ

Предлагаемая концепция структурного представления таблиц в системе ФинПлан позволяет значительно упростить процесс создания сложных моделей. Она обеспечивает удобную навигацию по модели и ее отладку, делает возможным совместное вычисление нескольких таблиц, обеспечивает защищенность данных в таблице, позволяет создавать многопользовательские модели и библиотеки типовых модулей для повторного использования. В настоящее время по данной концепции ведется реализация прототипа.

СПИСОК ЛИТЕРАТУРЫ

1. **Нариньяни А.С.** Недоопределенность в системах представления и обработки знаний // Изв. АН СССР. Техн. кибернетика. — 1986. — № 5. — С. 3–28.
2. **Дмитриев В.Е.** UNICALC — интеллектуальный решатель систем алгебраических уравнений и неравенств // Тр. 12 Всесоюзной конф. «Искусственный интеллект-90».— Минск, 1990. — С. 89–913.
3. **Костов Ю.В., Липовой Д.А., Мамонтов П.Г., Петров Е.С.** Новая версия универсального решателя UNICALC: возможности и перспективы развития // Тр. VI-й междунар. конф. CSCMP-2004. — Самара, 2004.
4. **Нариньяни А.С., Корниенко В.В., Прейс С.В., Швецов И.Е.** ФинПлан: новая технология финансово-экономического планирования в условиях неполноты информации // Информационные технологии. — 1998. — № 11. — С. 10–17.
5. **Загорулько Г.Б., Нариньяни А.С.** Интеллектуальные таблицы: новые возможности в решении сложных задач // Материалы Междунар. научно-практич. конф. «Информационные технологии, информационные измерительные системы и приборы в исследовании сельскохозяйственных процессов». Ч. 1. — Новосибирск, 2003. — С. 240–242.

Н. М. Ринская

ОБ АНАЛИЗЕ ТЕСТОВОЙ ЭКВИВАЛЕНТНОСТИ ДИСКРЕТНО-ВРЕМЕННЫХ СЕТЕЙ ПЕТРИ

ВВЕДЕНИЕ

Понятие эквивалентности является важнейшим для любой теории систем. Поведенческие эквивалентности обычно используются при спецификации и верификации для сравнения поведения систем, а также упрощения их структуры. В настоящее время для параллельных/распределенных систем существует большое разнообразие эквивалентностных понятий, взаимосвязи между которыми хорошо изучены. Наиболее известными являются два подхода — бисимуляционный [12, 11] и тестовый [11]. Две системы считаются бисимуляционно эквивалентными, если внешний наблюдатель не может обнаружить различий в поведении этих систем. При тестовом подходе поведение системы исследуется посредством набора тестов. Два процесса считаются тестово эквивалентными, если они могут или должны проходить один и тот же набор тестов. Разрешимость тестовой эквивалентности обычно достигается сведением ее к бисимуляционной [9].

В последнее десятилетие резко возрос интерес к разработке и исследованию распределенных систем, функционирующих в режиме реального времени. Поэтому в литературе были сделаны попытки ввести понятие времени в эквивалентностные отношения, что позволяет исследовать временные аспекты поведения систем. Разрешимость временной бисимуляции для автоматных моделей с непрерывным временем была показана с использованием техники регионов [5, 8] и техники зон [15]. Проблема распознавания временной тестовой эквивалентности для автоматов с дискретным и непрерывным временем была исследована в работах [10] и [14] соответственно. В работе [2] введено и изучено понятие тестовой эквивалентности для моделей структур событий с дискретным и непрерывным временем и разрешена проблема распознавания данного вида эквивалентности.

В этой работе определено в контексте более широкой модели — модели временных сетей Петри с дискретным временем — понятие временных тестовых эквивалентностей и разработан алгоритм распознавания данных эквивалентностей. Проблема распознавания разрешена путем сведения вве-

денных эквивалентностей к модификации бисимуляционной эквивалентности, для которой известны алгоритмы распознавания.

ДИСКРЕТНО-ВРЕМЕННЫЕ СЕТИ ПЕТРИ

Пусть *Act* — конечное множество действий и τ — невидимое действие, причем $\tau \notin Act$. Тогда $Act_\tau = Act \cup \tau$. Пусть $\boldsymbol{N}$ — множество натуральных чисел, N_0 — множество натуральных чисел с нулем. Определим множество временных интервалов $Interv(N_0) = \{[d_1, d_2] \subset N_0 \mid d_1, d_2 \in N_0\}$. Теперь мы можем ввести понятие дискретно-временной сети Петри.

Определение 2.1. Дискретно-временная сеть Петри (ДВСП), помеченная над Act_τ, — это шестерка $DN = (P, T, F, l, D, m_{DN})$, где P — конечное множество мест, T — конечное множество переходов ($P \cap T = \varnothing$), $F \subseteq (P \times T) \cup (T \times P)$ — отношение инцидентности, $l : T \to Act_\tau$ — помечающая функция, сопоставляющая каждому переходу из T действие из Act_τ, $D : T \to Interv(N_0)$ — временная функция, сопоставляющая каждому переходу из T временной интервал из $Interv(N_0)$, $m_{DN} \subseteq P$ — начальная разметка.

Разметкой m сети Петри *DN* называется любое подмножество множества мест *P*. Для разметки m определим множество *Enable(m)* переходов, все входные места которых содержат фишки. Введем $\Gamma(DN) = [T \to N_0]$ — множество временных функций, сопоставляющих текущие временные значения переходам из *T*. *Состоянием M* ДВСП *DN* называется пара $\langle m, \nu \rangle$, где *m* — разметка, $\nu \in \Gamma(DN)$. *Начальным состоянием DN* назовем пару $\langle m_{DN}, \nu_{DN} \rangle$, где $\nu_{DN}(t) = 0$ для всех $t \in T$. Изменение состояния сети Петри *DN* осуществляется либо посредством срабатывания перехода, либо посредством истечения некоторого дискретного промежутка времени. Переход $t \in T$ *может сработать* в состоянии $M = \langle m, \nu \rangle$ ($M \xrightarrow{t}$), если $t \in Enable(m)$ и $\nu(t) \in D(t)$. Будем писать $M \xrightarrow{a}$, если $M \xrightarrow{t}$ и $l(t) = a$. Состояние *M переходит* в состояние $M' = \langle m', \nu' \rangle$ ($M \xrightarrow{t} M'$) посредством срабатывания перехода *t*, если $m' = (m \setminus {}^\bullet t) \cup t^\bullet$, где ${}^\bullet t$ и $t^\bullet$ — множество входных и выходных мест перехода *t* соответственно, и

$$\nu'(t') = \begin{cases} 0, & \text{если } t' \in Enable(m') \setminus Enable(m) \\ \nu'(t') & \text{иначе.} \end{cases}$$

Время $d \in \mathbf{N}$ *может пройти* в состоянии M, если $Enable(m) \neq \varnothing$ и $\forall t \in Enable(m)\ \exists d' \geq d$: $\nu(t)+d' \in D(t)$. Состояние M *переходит* в состояние $M'=<m',\nu'>$ посредством истечения времени $d \in \mathbf{N}$ ($M \xrightarrow{d}$), если $m' = m$ и $\nu'(t) = \nu(t)+d$ для всех $t \in T$. Состояние M сети Петри DN называется *достижимым*, если M — начальное или существует достижимое состояние M' такое, что $M' \xrightarrow{x} M$ для некоторого $x \in T \cup \mathbf{N}$. Обозначим через $RS(DN)$ множество всех достижимых состояний сети Петри DN.

Будем называть сеть Петри DN *однобезопасной*, если для каждого достижимого состояния $<m,\nu> \in RS(DN)$ в каждом месте находится не более одной фишки ($\forall <m,\nu> \in RS(DN)$ и $\forall\ t \in Enable(m)$ выполнено: $m \cap t^{\bullet} = \varnothing$). В дальнейшем всегда будем рассматривать помеченные дискретно-временные однобезопасные сети Петри, удовлетворяющие условию возрастания времени (не существует циклов без хотя бы одного перехода по времени).

Слабое отношение перехода на состояниях в DN определяется как отношение $\Rightarrow$ такое, что $\overset{\varepsilon}{\Rightarrow} \Leftrightarrow \overset{\tau}{\rightarrow}{}^{*}$ и $\overset{x}{\Rightarrow} \Leftrightarrow \overset{\varepsilon}{\Rightarrow} \overset{x}{\rightarrow} \overset{\varepsilon}{\Rightarrow}$, где $x \in Act \cup \mathbf{N}$ и $\overset{\tau}{\rightarrow}{}^{*}$ — рефлексивное транзитивное замыкание отношения $\overset{\tau}{\rightarrow}$.

Также нам понадобится понятие языка сети Петри DN, который неформально можно определить как множество возможных последовательностей действий и временных промежутков (множество слов) при переходах от одного достижимого состояния к другому. Заметим, что язык ДВСП с циклами может быть бесконечным.

ВРЕМЕННЫЕ ТЕСТОВЫЕ ЭКВИВАЛЕНТНОСТИ И ИХ ХАРАКТЕРИЗАЦИИ

Введем понятия временных тестовых предпорядков и эквивалентностей в контексте рассматриваемой модели ДВСП.

Пусть $\omega \notin Act_{\tau}$ и $Act_{\tau,\omega} = Act_{\tau} \cup \{\omega\}$. Тогда $\mathbf{DTN}_{\tau,\omega}$ будет обозначать множество тестов — множество ДВСП без циклов с помечающей функцией над $Act_{\tau,\omega}$. Для теста $TDN \in \mathbf{DTN}_{\tau,\omega}$ через T_{TDN} будем обозначать его начальное состояние. Далее, пусть x с индексом и без него — элемент множества $Act \cup \mathbf{N}$, а y с индексом и без него — элемент множества $Act_{\tau} \cup N$.

Определение 3.1. Пусть DN ∈ $\mathbf{DTN}_{\tau}$, TDN ∈ $\mathbf{DTN}_{\tau,\omega}$ и M ∈ RS(DN), T ∈ RS(TDN). Тогда

- *$M||T$ — тестовое состояние. Тестовое состояние является успешным, если переход, помеченный символом ω, может сработать в T ;*
- *если* $M \xrightarrow{x} M'$ *и* $T \xrightarrow{x} T'$, *то* $M \| T \xrightarrow{x} M' \| T'$;

  если $M \xrightarrow{\tau} M'$, *то* $M \| T \xrightarrow{\tau} M' \| T$;

  если $T \xrightarrow{\tau} T'$, *то* $M \| T \xrightarrow{\tau} M \| T'$;
- *конечная максимальная последовательность*

  $$M_0 \| T_0 \xrightarrow{y_0} M_1 \| T_1 \ldots \xrightarrow{y_n} M_n \| T_n \quad (n \geq 0)$$

  называется вычислением, начинающимся тестовым состоянием $M_0 \| T_0$. *Вычисление является успешным, если* $\exists 0 \leq i \leq n : M_i \| T_i$ — *успешное. Множество всех вычислений, начинающихся тестовым состоянием M||T, будем обозначать через Comp(M||T);*
- *M may T, если существует успешное вычисление* $\nu \in Comp(M||T)$.
  DN may TDN, если M_{DN} *may* T_{TDN} ;
- *M must T, если каждое вычисление* $\nu \in Comp(M||T)$ *успешно*.
  DN must TDN, если M_{DN} *must* T_{TDN} .

Слабое отношение перехода на тестовых состояниях определяется так же, как и на состояниях дискретно-временной сети Петри.

Определим понятия временных тестовых предпорядков и эквивалентностей.

Определение 3.2.

- $DN \leq_{\alpha} DN' \Leftrightarrow \forall TDN \in \mathbf{DTN}_{\tau,\omega}$. $DN \; \alpha \; TDN \Rightarrow DN' \; \alpha \; TDN$,

  где $\alpha \in \{may, must\}$,
- $DN \leq_{test} DN' \Leftrightarrow DN \leq_{may} DN' \& DN \leq_{must} DN'$,
- $DN \approx_{\alpha} DN' \Leftrightarrow DN \leq_{\alpha} DN' \& DN' \leq_{\alpha} DN$, *где* $\alpha \in \{may, must, test\}$.

На рис. 1 приведен пример тестово эквивалентных сетей Петри DN_1 и DN_2 .

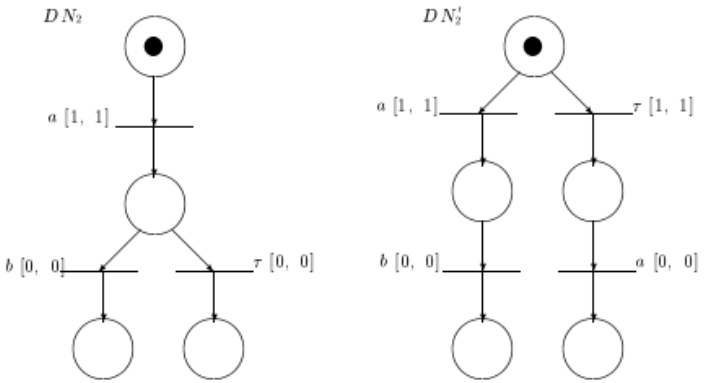


Рис. 1

Известно, что можно дать альтернативную характеризацию временных тестовых предпорядков, получив связь между свойствами языков ДВСП и отношениями $\leq_{may}$ и $\leq_{must}$.

БИСИМУЛЯЦИЯ, ПРЕБИСИМУЛЯЦИЯ И ИХ ХАРАКТЕРИЗАЦИИ

Прежде чем перейти к алгоритму распознавания тестовой эквивалентности, рассмотрим понятие графа обобщенных состояний.

Легко видеть, что число состояний ДВСП с циклами может быть бесконечным. Чтобы получить конечное пространство состояний, вводится понятие обобщенного состояния. Обобщенное состояние представляет собой множество эквивалентных состояний сети Петри — состояний с одинаковой разметкой и согласованными локальными временами переходов (временные функции соответствующих переходов либо совпадают, либо одновременно выходят за верхнюю границу *D(t)*). Ясно, что число обобщенных состояний для ДВСП будет конечным.

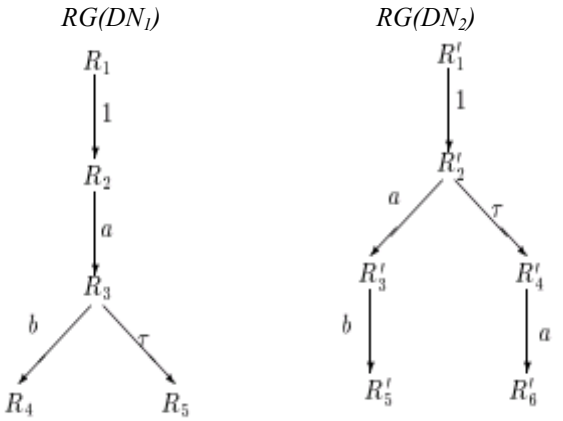


Рис. 2

Графом обобщенных состояний для ДВСП *DN* называют ориентированный размеченный граф *RG(DN)*, множество вершин которого — это множество обобщенных состояний для сети *DN*. Множество дуг определяется отношением перехода на обобщенных состояниях, помечающая функция определяется функцией *l(t)*. На рис. 2 изображены графы обобщенных состояний для ДВСП с рис. 1.

Как несложно заметить, граф обобщенных состояний является недетерминированным, т. е. из каждой его вершины может исходить несколько одинаково помеченных дуг, кроме того пометка τ считается так называемой "пустой" пометкой. Введем понятие *класса* (τ-замыкания обобщенного состояния) и *графа классов CG(DN)*, который строится по графу обобщенных состояний *RG(DN)* с помощью известного алгоритма построения детерминированного графа по недетрминированному [1]. Кроме того, каждой вершине графа классов ставится в соответствие некоторое информационное поле, формальное определение которого опущено в связи с ограничениями на объем работы. На рис. 3 приведены графы классов для ДВСП DN_1 и DN_2.

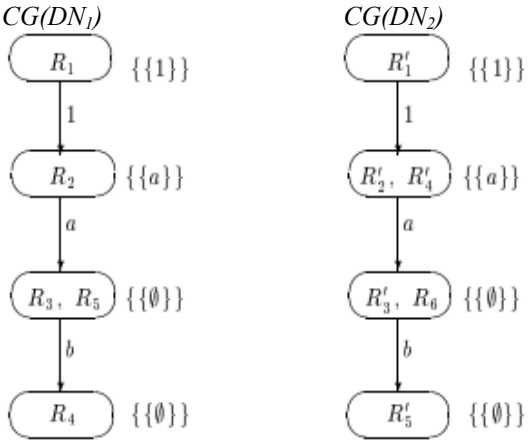


Рис. 3

Введем неформальное определение бисимуляции между графами классов. Неформально два графа классов бисимулятивны, если вершины одного графа могут быть сопоставлены вершинам другого следующим образом:

1) если две вершины сопоставлены друг другу, то содержимое их информационных полей должно быть «сравнимо»;
2) если две вершины сопоставлены друг другу, то переходу по z из одной из рассматриваемых вершин должен соответствовать переход по z из другой;
3) начальные вершины графов классов сопоставлены друг другу.

Вводя различные порядки на информационных полях вершин графа классов, можно получить различные виды бисимуляции. Мы будем рассматривать U-бисимуляцию (любые два поля считаются сравнимыми) и Π_1-бисимуляцию, которая накладывает более строгие ограничения на сравнимость информационных полей. Например, графы классов на рис. 3 U-бисимулятивны и Π_1-бисимулятивны:

$$(CG(DN_1) \sim_U CG(DN_2), CG(DN_1) \sim_{\Pi_1} CG(DN_2)).$$

РАСПОЗНАВАНИЕ ВРЕМЕННЫХ ТЕСТОВЫХ ЭКВИВАЛЕНТНОСТЕЙ И ПРЕДПОРЯДКОВ

Теперь можно сформулировать теорему, выявляющую связь между отношением временных тестовых эквивалентностей ДВСП и отношением бисимуляции соответствующих графов классов.

Теорема 5.1.

(а) $DN \approx_{may} DN' \Leftrightarrow CG(DN) \sim_{U} CG(DN')$,

(б) $DN \approx_{must} DN' \Leftrightarrow CG(DN) \sim_{\Pi_1} CG(DN')$,

(в) $DN \approx_{test} DN' \Leftrightarrow CG(DN) \sim_{\Pi_1} CG(DN')$.

Таким образом, проблема распознавания временных тестовых отношений сводится к проблеме распознавания отношений бисимуляции и пребисимуляции, алгоритмы решения которой хорошо изучены [13, 7]. Алгоритм распознавания разбивается на следующие шаги.

1. Построение графов обобщенных состояний для исходных ДВСП.
2. Построение графов классов. Алгоритм построения графа классов является модификацией известного алгоритма построения детерминированного графа по данному недетерминированному (по графу обобщенных состояний) [1].
3. Обход графов классов и сравнение пометок дуг и информационных полей вершин. В зависимости от результата сравнения делается вывод, являются ли исходные временные сети Петри тестово эквивалентными или нет.

Приведем теоретическую оценку сложности алгоритма анализа тестовой эквиваленности. Для этого нам понадобится оценка для числа обобщенных состояний временной сети Петри *DN*.

Лемма 5.1. *Сложность алгоритма построения графа обобщенных состояний в наихудшем случае оценивается величиной* $O\left[\left(d_{DN}+2\right)^{|T|}\cdot\left(|T|+1\right)\cdot 2^{|P|}\right]$, *где* $d_{DN}=\max\limits_{t\in T}\sup\left(D(t)\right)$.

Заметим, что сложность упоминаемого алгоритма распознавания бисимуляции — $O(k*l)$, где k — сумма мощностей множеств вершин графов классов, а l — сумма мощностей множеств дуг графов классов.

Сложность алгоритма построения графа классов экспоненциальна, так как теоретически для каждого подмножества множества вершин графа

обобщенных состояний может существовать вершина в графе классов. Таким образом, алгоритм распознавания временной тестовой эквивалентности в наихудшем случае обладает двойной экспоненциальной сложностью по размерности исходных ДВСП.

Теорема 5.2.

(а) Проблема распознавания, верно ли для дискретно-временных сетей Петри DN и DN', что $DN \approx_{may} DN'$*, разрешима.*

(б) Проблема распознавания, верно ли для дискретно-временных сетей Петри DN и DN', что $DN \approx_{must} DN'$*, разрешима.*

(в) Проблема распознавания, верно ли для дискретно-временных сетей Петри DN и DN', что $DN \approx_{test} DN'$*, разрешима.*

Доказательство. Строим граф обобщенных состояний, а затем граф классов согласно алгоритмам, приведенным выше. Из теоремы 5.1 получаем, что для решения поставленного вопроса достаточно проверить, являются ли построенные графы классов $CG(DN)$ и $CG(DN')$ пребисимулятивными (бисимулятивными). Согласно [7] существует алгоритм распознавания бисимуляции между $CG(DN)$ и $CG(DN')$.

ЗАКЛЮЧЕНИЕ

В рамках данной работы получены следующие результаты.

- Введено понятие временных тестовых эквивалентностей и предпорядков для модели дискретно-временных сетей Петри.
- Получена альтернативная характеризация введенных эквивалентностей и предпорядков (их связь со свойствами языков соответствующих дискретно-временных сетей Петри).
- Определено понятие графа классов для дискретно-временных сетей Петри и установлена взаимосвязь между отношениями бисимуляции графов классов и временными тестовыми эквивалентностями соответствующих сетей Петри.
- На основе этих теоретических результатов получен алгоритм распознавания временных тестовых эквивалентностей для дискретно-временных сетей Петри.

СПИСОК ЛИТЕРАТУРЫ

1. **Ахо А., Хопкрофт Дж., Ульман Дж.** Построение и анализ вычислительных алгоритмов. — М.: Мир, 1979.
2. **Боженкова Е.Н.** Анализ свойств параллельных процессов и процессов реального времени, представленных моделями структур событий. — Новосибирск, 2000.
3. **Aceto L., R. de Nicola, Fantechi A.** Testing Equivalences for Event Structures // Lect. Notes Comput. Sci. — 1987. — № 280**.** — P. 1–20.
4. **Alur R., Courcoubetis C., Dill D.** Model checking in dense real time // Information and Computation. — 1993. — № 104. — P. 2–34.
5. **Alur R., Courcoubetis C., Henzinger T.A.** The observational power of clocks // Lect. Notes Comput. Sci. — 1994. — № 836. — P. 162–177.
6. **Brinksma E., Rensink A., Vogler W.** Fair Testing // Lect. Notes Comput. Sci. — 1995. — № 962. — P. 313–327.
7. **Cleaveland R., Parrow J., Steffen B.** The Concurrency Workbench: A Semantics-Based Tool for the Verification of Concurrent Systems // J. Assoc. Computing Machinery — 1993. — Vol. 15, N 1. — P. 36–72.
8. **Erns K.** Decidability of bisimulation equivalences for parallel timer processes // Lect. Notes Comput. Sci. — 1993. — Vol. 663. — P. 302–315.
9. **Cleaveland R., Hennessy M.** Testing equivalence as a bisimulation equivalence // Lect. Notes Comput. Sci. — 1989. — Vol. 407. — P. 11–23.
10. **Cleaveland R., Zwarico A. E.** A theory of testing for real-time: Proc. / Symp., Logic in Computer Science, 1991. — P. 110–119.
11. **De Nicola R., Hennessy M.** Testing equivalence for processes // Theoretical Computer Sci. — 1984. — N 34. — P. 83–133.
12. **Hennessy M., Milner R.** Algebraic laws for nondeterminism and cocurrency Systems // J. Assoc. Computing Machinery — 1985. — Vol. 32. — P. 137–162.
13. **Park D.** Concurrency and automata on infinite sequences // Lect. Notes Comput. Sci. — 1981. — N 154**.** — P. 561–572.
14. **Stephen B., Weise C.** Deciding testing equivalence for real-time processes with dense time. — Lect. Notes Comput. Sci. — 1993. — N 711**.** — P. 703–713.
15. **Weise C., Lezkes D.** Efficient scaling-invariant checking of timed bisimulation // Lect. Notes Comput. Sci. — 1989. — N 1200. — P. 176–188.
16. **Winskel G.** An introduction to event structures // Lect. Notes Comput. Sci. — 1989. — N 354. — P. 364–397.

Е.А. Сидорова

МЕТОДЫ ИНТЕЛЛЕКТУАЛЬНОЙ ОБРАБОТКИ ДОКУМЕНТОВ, ОСНОВАННЫЕ НА ЭКСПЕРТНЫХ ЗНАНИЯХ

ВВЕДЕНИЕ

Задача разработки систем поддержки документооборота и управления знаниями в крупных компаниях является одной из самых актуальных на сегодняшний день. Часто она рассматривается в контексте создания хранилищ документов и их систематизации с целью облегчения поиска имеющейся информации. Несмотря на важность этих вопросов, возможностей, предоставляемых такими системами, оказывается недостаточно для интеллектуальной организации деятельности, которая представляет собой совокупность средств и методов представления, извлечения и манипулирования знаниями.

Существующие традиционные подходы к автоматизации документооборота [1] можно разделить на два независимых класса — статистические и лингвистические методы обработки документов. Особенностью статистических методов является их универсальность, однако они не учитывают неоднозначность, присущую любым текстам на естественном языке. Этот недостаток начинает проявляться сильнее на ограниченных предметных областях, так как текст изначально имеет контекст, который помогает разрешить неоднозначность с помощью лингвистических методов и игнорируется статистическими методами.

Отметим, что на настоящий момент не существует сколько-либо полного и точного подхода к построению содержательных анализаторов, ориентированных на ограниченную предметную область (ПО).

В данной работе рассматривается подход к автоматическому анализу текста документов и его содержательному индексированию, основанный на использовании предметной онтологии [2].

ПРЕДСТАВЛЕНИЕ ДОКУМЕНТА

Важную роль при автоматизации анализа документа играет его жанровая структура. Она определяет тематические разделы, ограничивая возмож-

ную смысловую нагрузку той или иной части текста документа. В статье мы рассмотрим только один жанр документа — деловое письмо (ДП), как наиболее типичный для задачи интеллектуализации документооборота.

На рис.1. представлена структура делового письма, необязательные подразделы помечены звездочкой.

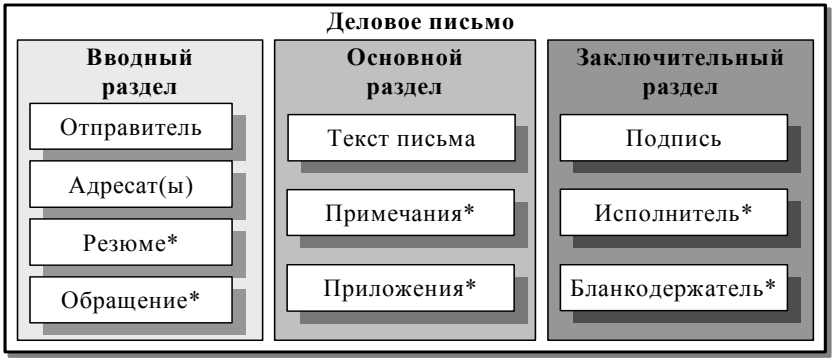


Рис. 2. Структура документа типа «Деловое письмо»

Компоненты структуры документа отражают разбиение текста документа на информационные блоки различной функциональности. Можно говорить об основных и вспомогательных жанровых разделах в структуре ДП. Так, *Обращение* и *Подпись* служат для выделения блока *Основной текст*. К числу основных разделов, необходимых для решения поставленных задач, отнесены *Отправитель, Адресаты, Основной текст.* Процедура анализа использует границы раздела *Отправитель* для определения наименования организации-отправителя ДП, что позволяет применить знания о функциях этой организации. В границах *Основного текста* ищутся ключевые понятия, на основе которых анализируется пропозициональное содержание ДП и выявляется его тема.

Для хранения и поиска документов в архиве необходимо, чтобы при каждом документе хранилась некоторая структурированная информация, характеризующая его содержание и содержащая набор формальных параметров. Такая структурированная информация о документе хранится в виде электронного индекса документа. Под *семантическим индексом* понимается набор атрибутов индекса, в которых отражается смысл документа.

ПРЕДСТАВЛЕНИЕ ЗНАНИЙ

База знаний, необходимая для анализа содержания документов, представляется как совокупность онтологий (высокоуровневое знание) и создаваемых на их основе структур знаний (низкоуровневое знание).

На рис. 2. представлена иерархия знаний, используемых в предлагаемом подходе.

Рис. 3. Система знаний

Рассматриваются пять компонентов системы знаний.

1. **Метаонтология** является ядром системы знаний и отражает сущность решаемых задач данной ПО. Метаонтология фиксирует те базовые содержательные структуры, которые система должна использовать при автоматическом анализе и индексации текстов документов: компоненты семантического индекса, система понятий (базовые классы понятий), семантическая структура высказывания, жанровая структура документа.
2. **Онтология предметной области** отражает общие знания о ПО, такие как иерархия классов понятий, семантические отношения на этих классах.
3. **Предметные знания** — это выделенная часть знаний о ПО, которая содержит только частные понятия и конкретные отношения данной ПО (например, иерархия вложенности объектов, набор существующих организаций и их типы и др.).
4. **Знания о предприятии** представлены списком сотрудников предприятия, структурой его подразделений, фильтрами адресации сотрудников.

5. **Онтология языка документов** — это система языковых средств выражения метаонтологии и онтологии ПО в текстах документов. Лингвистическая информация представлена в словаре с помощью функциональных групп лексических единиц, выделенных классов понятий и набора дополнительных атрибутов, отражающих специфику выражений: синонимы, омонимы, составные понятия и т.п.

Исходными данными для такой системы знаний являются тексты деловых писем и других документов, характеризующих предметную область предприятия-пользователя. Обеспечить автоматическое извлечение знаний из этих данных является главной задачей эксперта при наполнении и настройке системы знаний.

Онтология языка документов

Базовым компонентом для представления онтологии языка документов является Словарь-Тезаурус, который создается с помощью инструментальной системы Alex [3], поддерживающей технологию обработки текстов на основе иерархической системы лексических шаблонов.

В словаре представлена информация о словах и словосочетаниях, необходимая для обработки документов. Эта информация хранится в виде библиотеки именованных шаблонов, привязанных к *понятиям* — именованным сущностям предметной области.

Все понятия разбиваются на классы. Классы понятий выстраиваются в иерархию с учетом их тезаурусных взаимосвязей (отношения синонимии, гипонимии (родовидового), меронимии (часть-целое)).

Шаблон в словаре-тезаурусе — это **единица словаря, которая представляется словарной статьей**.

Словарная статья шаблона включает следующую информацию.

- *Имя шаблона* — соответствует нормализованному виду слова или словосочетания, представляющего некоторое понятие.
- *Определение шаблона* (строковое определение шаблона) — это конструкция на определенном формальном языке, описывающая множество языковых выражений произвольной сложности (в общем случае, разрывных), которые могут представлять понятие в тексте (таким образом, шаблон представляет все возможные синонимичные или "почти синонимичные" языковые выражения данного понятия).
- *Класс шаблона* — наименование конкретной группы шаблонов, отражающее предметную сущность соответствующего класса понятий.

♦ *Атрибут* — представляет собственную параметрическую характеристику понятия, представленного шаблоном, или особенности его лексико-семантической сочетаемости с другими понятиями.

Строковое определение шаблона может содержать:

– *множество словоформ одного слова*, например,

[земля] =
земл**…**
земел**…**

где многоточие представляет флексию (окончание) произвольной длины, в том числе, и нулевую;

– *устойчивое (терминологическое) словосочетание,* представленное цепочкой шаблонов и/или словоформ, например,

[лесные земли] = лесн**…**_**[земля]**

где в определении шаблона используется ссылка на уже имеющийся в словаре шаблон;

– *множество эквивалентных (синонимичных) строковых определений*, например,

[лесные ресурсы] =
[лесные земли]
лесн**…**_**[ресурс]**
([земля])_лесн**…**_фонд**…**

где круглые скобки означают факультативность данного элемента в соответствующей строке текста.

Группы шаблонов. **Словарь содержит следующие группы шаблонов.**

- *Предметно-ориентированные* шаблоны, которые охватывают множество слов и словосочетаний, необходимых для представления ключевых понятий ПО.
- *Служебные* шаблоны, которые не соответствуют ключевым понятиям ПО, но играют роль значимых элементов в процессе анализа документа:
  – *жанровые* шаблоны используются для декомпозиции документа во множество свойственных жанру информационных блоков (*обращение, заключительная реплика*);
  – *разделители* используются для формальной сегментации текста (*точка, номер в списке*);
  – *параметрические* шаблоны представляют числовые значения параметров (*км, номер*);

  - *предлоги* используются для уточнения семантических ролей понятий ПО.
- *Вспомогательные* шаблоны, которые не соответствуют ключевым понятиям ПО, но регулярно используются в качестве подшаблонов при конструировании шаблонов основных типов.

Представленные в словаре шаблоны используются лексическим процессором системы Alex, который извлекает из текста документа необходимые для анализа ключевые понятия предметной области и передает их на дальнейшую обработку.

АНАЛИЗ И ИНДЕКСИРОВАНИЕ ДОКУМЕНТОВ

Обработка документа включает выполнение следующих процедур:

- предварительный анализ — лексический анализ (выделение ключевых понятий), осуществляемый системой Alex, сегментация документа, идентификация сложных объектов;
- семантический анализ — сборка фактов, тематический анализ;
- постобработка — адресация, индексация.

Процесс анализа осуществляется на основе метаонтологии и онтологии ПО, а также процедурных знаний, представленных в системе. Для описания метазнаний и их процедурной интерпретации предлагается использовать Semp-технологию [4].

Процедура основного анализа документов

На вход модуля основного анализа поступает множество ключевых понятий, выделенных словарным компонентом системы при лексическом анализе текста документа. Дальнейший алгоритм автоматического индексирования документов реализуется с помощью описанной технологии в три этапа.

На этапе *сегментации документа* осуществляется жанровая декомпозиция текста документа, которая определяет тематические разделы, ограничивая возможную смысловую нагрузку той или иной части текста документа.

Последующая обработка документа представляет собой процесс извлечения релевантной информации на основе ключевых понятий, выделенных в границах *Основного текста.*

Идентификация объектов. На этом этапе определяются все возможные атрибуты понятия, позволяющие уточнить объект, описываемый данным понятием (например, для объекта строительства это может быть его номер, начальный и конечный километр участка и т.п.). Кроме того, каждая пара ключевых понятий, отнесенных в тезаурусе к определенному месту иерархии, и удовлетворяющая определенным требованиям порядка слов в тексте письма, проверяется на предмет наличия между ними отношений вложенности (с учетом транзитивности этого отношения). Это позволяет собрать объекты сложной структуры, представленные цепочками наименований, и определить их признаки. Как правило, этот этап относится только к конкретным объектам, представленным в предметных знаниях системы.

Семантический анализ. В зависимости от типа ПО и решаемой задачи возникает необходимость в извлечении из текстов документов различного рода информации (фактов или событий). Общую структуру извлекаемой информации принято представлять в виде некоторой пропозициональной структуры, выражающей связь предиката (процесс, действие, свойство) и множества его аргументов.

В предлагаемом подходе принята следующая базовая структура *Факта*:

$$F = A(S, O, P),$$

где A — действие (представимо понятиями класса *Работа*), S — субъект действия (класс *Организация*), O — объект действия (класс *Объект*), P — место действия (класс *Объект Строительства*).

Схема сборки Факта. Для того чтобы система могла правильно собирать *Факты* по тексту письма, требуется представить в ней знания о сочетаемости понятий. Эти знания представлены в виде отношений, задающих ограничения на допустимые сочетания понятий указанных классов и их значений в *Фактах*.

Исходя из информации о возможных объектных связях между ключевыми понятиями классов *Работа* и *Объект*, строятся *Функции* в границах формальных сегментов текста (предложений). При этом учитываются требование проективности (для исключения пересечений связей) и словарная информация о сочетаемостных свойствах предикатных слов, т.е. о наличии у конкретного предиката класса *Работа* валентности на *Объект* [5].

На рис. 3 отображена общая схема сборки *Факта*.

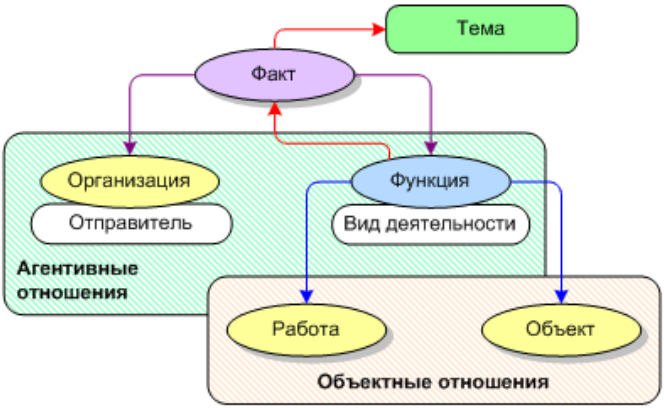


Рис. 4. Схема сборки Факта

Каждой Функции может быть сопоставлен некоторый *Вид деятельности*, который проверяется на семантическую сочетаемость с упомянутыми в тексте *агентами–Организациями*. Вид деятельности представляет некоторый класс функциональных сочетаний *Работы* и *Объекта*, в том числе и таких, которые синкретически представлены в значении лексической единицы (например, лесопользование). Установление соответствующих агентивных связей дает множество *Фактов* документа, совокупность которых определяет тему документа и позволяет определить соответствующие значения полей семантического индекса.

Адресация документов

Для решения задачи адресации документов требуются знания о структуре предприятия и информационных предпочтениях его сотрудников, определяемых их функциональными ролями.

Каждое информационное предпочтение представлено в виде множества семантических фильтров. Простые фильтры представляют собой образцы *Фактов*, которые должны присутствовать или, напротив, отсутствовать в тексте документа, для того чтобы он удовлетворял этому *Фильтру*.

Из простых *Фильтров* могут строиться сложные фильтры, представляющие собой конъюнкцию простых *Фильтров* (*положительная часть Фильтра*) и/или их отрицаний (*отрицательная часть Фильтра*):

$$CF = \bigcap_i SFi \bigcap_j \neg SFj ,$$

где CF — сложный фильтр, SF — простой фильтр.

При адресации совокупность *Фактов* из темы проиндексированного документа сопоставляется с множеством настроенных пользователем фильтров. *Факт* удовлетворяет простому фильтру, если для каждого непустого поля фильтра нашлось хотя бы одно значение из соответствующего поля *Факта* (при этом процедура сравнения учитывает наследование в иерархии классов).

ЗАКЛЮЧЕНИЕ

Данный подход использовался при создании системы документооборота InDoc [6], ориентированной на предметную область и потребности крупной инвестиционной компании, управляющей строительством газопроводов. К программным средствам, разработанным для поддержки предлагаемой в статье методики, относятся: программное ядро, обеспечивающее содержательную обработку документов, модули настройки базы знаний, компонент работы с инструментальной системой ведения Словаря-Тезауруса Alex.

Следующим этапом развития подхода является подключения морфологического словаря для улучшения качества анализа текста, а также разработка и создание конструктора онтологий, который бы позволил объединить все настраиваемые элементы базы знаний в одном пользовательском интерфейсе.

СПИСОК ЛИТЕРАТУРЫ

1. **Хорошевский В.Ф.** Управление знаниями и обработка ЕЯ-текстов // Тр. IX национальной конф. по искусственному интеллекту. — КИИ'2004. — М.: Физматлит, 2004. — Т.2. — С. 565–572.
2. **Нариньяни А.С.** ТЕОН 2: От тезауруса к онтологии и обратно // Тр. междунар. семинара Диалог'2002 «Компьютерная лингвистика и интеллектуальные технологии», Протвино, 2002. — Т.1. — С. 307–313.
3. **Жигалов В.А., Жигалов Д.В., Жуков А.А.и др.** Система Alex как средство для многоцелевой автоматизированной обработки текстов // Тр. междунар. семинара Диалог'2002 «Компьютерная лингвистика и интеллектуальные технологии», Протвино, 2002. — Т.2 — С. 192–208.

4. **Zagorulko Yu.A., Popov I.G.** Knowledge representation language based on the integration of production rules, frames and a subdefinite model // Joint Bulletin of NCC&. Ser.: Comput. Sci. — 1998. — № 8. — P.81–100.
5. **Кононенко И.С., Сидорова Е.А.** Обработка делового письма в системе документооборота // Тр. междунар. семинара Диалог'2002 по компьютерной лингвистике и ее приложениям, Протвино, 2002. — Т.2. — С. 299–310.
6. **Загорулько Ю.А., Кононенко И.С., Костов Ю.В., Сидорова Е.А.** Система InDoc: интеллектуальная обработка, распределение и поиск документов в электронном архиве // Тр. V междунар. конф. «Проблемы управления и моделирования в сложных системах», Самара, 2003. — Самара: Самарский Научный Центр РАН, 2003.

А.П. Стасенко

ГРАФИЧЕСКИЙ МЕТАЯЗЫК ДЛЯ ОПИСАНИЯ ТРАНСЛЯТОРА[1]

Одним из препятствий на пути эффективной разработки трансляторов при применении связки утилит LEX [1] и YACC [2] является семантический разрыв [3] между формальным описанием исходных языков и методами реализации трансляторов этих языков. Для его сокращения при создании транслятора языка SISAL [4, 5] во внутреннее представление IR1 [6] было разработано графическое описание алгоритма трансляции, основанное на конечно-автоматной модели [7, 8, 9], которое можно создать в обычном редакторе графов [10]. Для теоретического обоснования разработанного графического метаязыка вводится модель упрощенного магазинного автомата.

LEX И YACC

Обычно для автоматизации построения транслятора используется связка утилит LEX и YACC. YACC применяется для создания синтаксического и семантического анализатора, а для лексического анализа используется утилита LEX.

YACC позволяет работать с грамматиками класса LALR. Этот класс грамматик достаточно широк и описывает конструкции многих языков программирования. По входной грамматике утилита YACC генерирует программу на языке СИ, которая является синтаксическим анализатором для исходной грамматики. Этот анализатор для разрешения неоднозначностей реализует алгоритм перенос-свертка и является достаточно эффективным и быстродействующим.

Однако применение YACC не лишено недостатков, так как получаемый на выходе распознаватель для грамматики без специальных преобразований практически всегда оказывается недетерминированным. Преобразование грамматики к нужному виду, которое в большинстве случаев необходимо производить вручную, часто приводит к потере её наглядности или изменению входного языка.

[1] Работа выполнена при финансовой поддержке Научной программы «Университеты России» (грант УР.04.01.027).

Подобные проблемы возникают потому, что описание синтаксиса осуществляется с использованием формальной грамматики, которая определяет синтаксическую структуру языка, тогда как построение распознавателя основывается на автоматной модели и оперирует соответствующими понятиями из теории автоматов, такими как входной алфавит автомата, множество его состояний, множество магазинных символов и переходов.

Преобразование формального описания в модель автомата 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. Если в процессе разработки будут внесены изменения в автоматную модель, то они уже никаким образом не повлияют на исходное представление, которое часто является также и пользовательским описанием.

Несовпадение объектов и операций, которыми разработчик манипулирует при описании языка, с объектами и операциями, которыми разработчику приходится манипулировать при реализации транслятора, получило название *семантический разрыв*.

МОДЕЛЬ УПРОЩЕННОГО МАГАЗИННОГО АВТОМАТА

Для сокращения семантического разрыва далее предлагается модель упрощенного магазинного автомата (далее УМА). Представление исходного языка с использованием данной модели, с одной стороны, сохраняет описанные выше и свойственные современным метаязыкам иерархичность и высокий уровень представления, а с другой, позволяет описывать любой контекстно-свободный язык и может быть реализовано непосредственно в терминах используемых абстракций.

Модель УМА можно рассматривать как частный случай Машины с Абстрактными Состояниями [11]. Также модель УМА является аналогом модели динамически порождаемых конечных автоматов [3], но в отличие от этой модели позволяет описывать распознаватель в виде одного автомата. Это может быть удобно для более надёжного построения распознавателя, так как отсутствуют неявные зависимости между автоматами. Наглядность графического представления сложного автомата можно поддерживать путём его представления в виде иерархического графа.

Определение

Упрощенный магазинный автомат, аналогично обычному магазинному автомату (далее МА), определяется следующей совокупностью семи объектов: $A = \{P, S, s_0 \in S, F \subseteq S, H, h_0 \in H, f\}$. Где P — это входной алфавит, S — алфавит состояний, s_0 — начальное состояние, F — множество конечных состояний, H — алфавит магазинных символов, записываемых на вспомогательную ленту, h_0 — маркер дна, всегда находящийся на дне магазина.

Функция переходов обозначается как $f: S \times \{P \cup \varepsilon\} \times H \to S \times H^*$, если автомат детерминированный, и $f: S \times \{P \cup \varepsilon\} \times H \to \{S \times H^*\}$, если автомат недетерминированный. Правила функционирования УМА аналогичны правилам функционирования МА, но, в отличие от МА, для УМА $H = S \cup h_0$ и функция перехода может иметь один из следующих трёх видов:

1) $f(s_1, \{p \cup \varepsilon\}, h) = (s_2, h)$ — переход, не зависящий от состояния и не меняющий состояние магазина;
2) $f(s_1, \{p \cup \varepsilon\}, h) = (s_n, hs_2 \ldots s_{n-1})$ — переход, не зависящий от состояния магазина, но записывающий в него цепочку $s_2 \ldots s_{n-1}$;
3) $f(s_1, \{p \cup \varepsilon\}, s_2) = (s_2, \varepsilon)$ — переход в состояние, прочитанное из магазина.

Утверждение

Произвольный контекстно-свободный язык задается с помощью недетерминированного УМА. Убедимся в справедливости этого утверждения. Для этого возьмем контекстно-свободную грамматику $\Gamma = \{V_т, V_a, <I> \in V_a, R\}$, где $V_т$ — терминальный алфавит, V_a — нетерминальный алфавит, <I> — начальный символ грамматики, R — множество правил вывода $\{<A> \to \alpha$, где $<A> \in V_a$, $\alpha \in (V_т \cup V_a)^*\}$. Построим упрощенный магазинный автомат A, такой что $L_A = L_\Gamma$.

Определим компоненты УМА A следующим образом: $P = V_т$, $S = V_a \cup \{r_{i,j}$, где $i = 1 \ldots |R|$ и $j = 1 \ldots |\alpha|$ = длина правой части i-го правила вывода$\}$, $F = \{<I>\}$. В качестве начальной конфигурации автомата возьмем: (<I>, ω = начальная цепочка на входной ленте, h_0<I>). Построим функцию переходов следующим образом.

1. Для i-го правила грамматики $<A> \to \alpha_1 \ldots \alpha_J$ построим команды вида (полагая, что $r_{i,0} = <A>$):
   a) для каждого $j = 1 \ldots J$:

i. если $\alpha_j \in V_т$, то $f(r_{i,j-1}, \alpha_j, h) = (r_{i,j}, h)$,
ii. если $\alpha_j \in V_a$, то $f(r_{i,j-1}, \varepsilon, h) = (\alpha_j, hr_{i,j})$;

b) $f(r_{i,j}, \varepsilon, s) = (s, \varepsilon)$.

2. Для перехода в конечное состояние построим команду $f(<I>, \varepsilon, h_0) = (s_0, \varepsilon)$.

ГРАФИЧЕСКИЙ МЕТАЯЗЫК ДЛЯ ОПИСАНИЯ УПРОЩЕННОГО МАГАЗИННОГО АВТОМАТА

Для визуального представления предлагаемой модели детерминированного УМА был разработан графический метаязык (программы на котором далее называются S-схемами), отображающий модель УМА в виде ориентированного размеченного графа, вершины которого, за исключением специальных “pop-вершин”, соответствуют состояниям автомата, а связи между вершинами — переходам. Метаязыками, наиболее близкими к S-схемам, являются A-схемы [3], синтаксические диаграммы Вирта [12], спецификация Montages [13] и язык спецификаций SDL [14].

В S-схемах существует два типа вершин.

1. Вершины, соответствующие состояниям УМА:
   a) вершины, соответствующие состояниям $S \setminus F$, среди которых выделено начальное состояние;
   b) вершины, соответствующие конечным состояниям, переход в которые означает завершение работы автомата в независимости от состояния входной и магазинной ленты.
2. “Pop-вершины”, служащие для организации переходов третьего типа. Такие вершины не могут иметь исходящих дуг.

Связи также подразделяются на два общих вида.

1. Связи, соответствующие переходам первого типа:
   a) непрерывная дуга с меткой $p \in P$ соответствует функции перехода со вторым аргументом равным p;
   b) непрерывная дуга без метки соответствует функции перехода со вторым аргументом равным $P \setminus$ {метки всех других помеченных дуг, исходящих из той же вершины};
   c) штриховая дуга без метки соответствует функции перехода со вторым аргументом равным ε.

2. Связи, участвующие в формировании переходов второго типа. Обозначаются как непомеченные пунктирные дуги. Из вершины не может исходить более одной такой дуги.

Из вершины не может исходить более одной одинаково помеченной или непомеченной дуги, соответствующей переходу первого типа. Правила определения второго аргумента функции перехода первого типа также распространяются на переходы второго и третьего типа.

Часть S-схемы: **Функция перехода:**

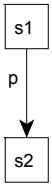


Рис. 1. Переход первого типа

$f(s_1, p, h) = (s_2, h)$, если вершина s_2 не «pop-вершина» и не имеет исходящей пунктирной дуги.

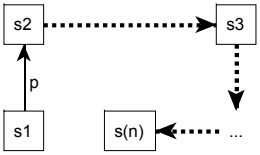


Рис. 2. Переход второго типа

$f(s_1, p, h) = (s_n, hs_2 \ldots s_{n-1})$, где s_n уже не имеет исходящей пунктирной дуги.

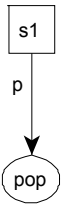


Рис. 3. Переход третьего типа

$f(s_1, p, s_2) = (s_2, \varepsilon)$

Теоретически, в рамках S-схемы, моделирующей детерминированный УМА, возможно избавиться от всех штриховых дуг и, соответственно, от всех “ε-переходов”. Однако на практике это разумно делать только для переходов первого и второго типов, так как в случае переходов третьего типа может возникнуть слишком большое количество дополнительных команд, зависящих от содержимого магазина и сводящих на нет эффект такого преобразования.

Нужно отметить, что описанная модель УМА была введена не случайно. Эта модель автомата может быть наглядно представлена в виде графа, тогда как граф МА был бы перегружен переходами, зависящими от состояния магазина.

Пример грамматики:

Γ_1 = {$V_т$, V_a, <S>, R}:
$V_т$ = { '{', '}', 'a', ',', ' ' },
V_a = { <S>, <L>, <I>, <A> },
R = {
  <S> → {}
  <S> → {<L>}
  <L> → <I>, <L>
  <L> → <I>
  <I> → <S>
  <L> → <A>
  <A> → a<A>
  <A> → <a>
}

Эквивалентная грамматике S-схема:

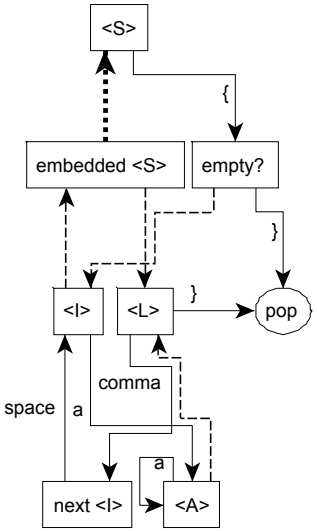


Рис. 4. S-схема для грамматики Γ_1

ПРИМЕНЕНИЕ ГРАФИЧЕСКОГО МЕТАЯЗЫКА ДЛЯ ТРАНСЛЯЦИИ ПРОГРАММ НА ЯЗЫКЕ SISAL ВО ВНУТРЕННЕЕ ГРАФОВОЕ ПРЕДСТАВЛЕНИЕ IR1

К похожим работам можно отнести описание статической семантики для Машины Абстрактных Состояний транслятора функционального языка ML [15].

Процесс трансляции можно разбить на лексический, синтаксический и семантический анализы программы [16]. Для лексического анализа текста программы обычно достаточно детерминированного конечного автомата, который задаётся S-схемой с переходами только первого типа. Для различных действий, связанных с генерацией лексем, к рассмотренной выше S-схеме добавляется дополнительная разметка вершин-состояний и переходов между ними, состоящая из идентификаторов соответствующих действий-процедур. На семантику этих действий не накладывается никаких ограничений, кроме вмешательства в работу УМА, такого как доступ к входной ленте (кроме чтения текущего символа) или магазину.

Ввиду функциональной семантики исходного языка и графовой природы получаемого внутреннего представления IR1, удобно совмещать синтаксический и семантический анализы программы. Для этого к рассматриваемым S-схемам был добавлен специальный тип вершины, переход из которой определяется непосредственно идентификатором действия-функции ей сопоставленной. При этом содержимое входной ленты и магазина не меняется. Такие “переключатели”, не снижая наглядности S-схемы, способны реализовывать произвольные семантически зависимые переходы.

S-схемы строятся с помощью редактора иерархических графов yEd. Для завершенного лексического и синтаксического анализатора они имеют достаточно большой размер, но благодаря иерархии графов, используемой для структурирования транслируемых понятий, сохраняют высокую читаемость. Высокая читаемость также обеспечивается средствами иерархического размещения графов, встроенными в редактор графов yEd.

Далее граф S-схемы транслируется специально разработанной утилитой из языка GraphML [17] (Graph Markup Language, основанный на XML) во внутреннее представление, пригодное для непосредственного исполнения простым интерпретатором, написанным на произвольном языке программирования (например, C++). Указанная утилита дополнительно минимизирует транслируемый автомат и разворачивает все “ε-переходы” для правил первого типа. Для правил второго типа “ε-переходы” не разворачиваются

ввиду внутренних особенностей получаемого внутреннего представления, отказ от которых был сочтен нецелесообразным.

Уже во время исполнения происходит сопоставление идентификаторов действий с непосредственно самими действиями, которые также могут иметь произвольный язык реализации. Тем самым, рабочий транслятор представляет собой совокупность действий размеченной идентификаторами S-схемы, её интерпретатора и реализации этих действий.

То, что автомат хранится в виде отдельного файла, а не представлен в неявном виде текста программы, генерируемого с помощью YACC, позволяет использовать оптимизацию исполнения УМА. Например, высчитываются вероятности срабатываний переходов из отдельно взятого состояния, и далее условия переходов из этого состояния проверяются уже в порядке убывания вероятности их истинности. Также существенно упрощается автоматическое построение системы тестов транслятора, покрывающей все его состояния.

ЗАКЛЮЧЕНИЕ

Для описания предложенной модели упрощенного магазинного автомата разработан графический метаязык (S-схема), который используется при реализации транслятора функционального языка программирования SISAL. Создана утилита для трансляции из удобного для построения S-схемы графового формата GraphML во внутреннее представление, подходящего для непосредственного исполнения. Написан модуль интерпретации S-схемы. Закончена разработка S-схемы и реализация идентификаторов действий для лексического анализатора языка SISAL. Продолжается работа по созданию S-схемы для синтаксического и семантического анализов языка SISAL.

СПИСОК ЛИТЕРАТУРЫ

1. **Lesk M. E.** Lex: a lexical analyzer generator. — Murray Hill, N.J., 1975. — (Comput. Sci. Tech. Rep. / AT&T Bell Laboratories; N 39).
2. **Johnson S. C.** YACC: yet another compiler compiler. — Murray Hill, N.J., 1975. — (Comput. Sci. Tech. Rep. / AT&T Bell Laboratories; N 32).
3. **Легалов А.И.** Трансляторы. Методы разработки. — `http://www.softcraft.ru/translat.shtml`
4. **Бирюкова Ю. В.** SISAL-90: руководство пользователя. — Новосибирск, 2000. — 84 с. — (Препр. / РАН. Сиб. Отд-ние. ИСИ; № 72).

5. **Касьянов В. Н., Бирюкова Ю. В., Евстигнеев В. А.** Функциональный язык Sisal 3.0 // Поддержка супервычислений и интернет-ориентированные технологии. — Новосибирск, 2001. — С. 54–67.
6. **Стасенко А. П.** Внутреннее представление системы функционального программирования SISAL 3.0. — Новосибирск, 2004. — 54 с. — (Препр. / РАН. Сиб. Отд-е. ИСИ; № 110).
7. **Шалыто А.А.** Switch-технология. Алгоритмизация и программирование задач логического управления. — СПб.: Наука, 1998. — 628 с. — http://www.softcraft.ru/auto/switch/lsabook/
8. **Кузнецов Б.П.** Психология автоматного программирования // BYTE. — 2000. — №11. — С. 22–29.
9. **Любченко В.С.** Конечно-автоматная технология программирования. — 2001. — http://www.softcraft.ru/design/katech.shtml
10. **yEd** — Свободно-распространяемый редактор иерархических графов — http://www.yworks.com/en/products_yed_about.htm
11. **Gurevich Y.** Evolving Algebras 1993: Lipari Guide // Specification and Validation Methods. — Oxford University Press, 1995. — P. 231–243.
12. **Вирт Н., Йенсен К.** Паскаль. Руководство для пользователя и описание языка / Пер. с англ. — М.: Финансы и статистика, 1982. — 151 с.
13. **Kutter P., Pierantonio A.** Montages: Specifications of Realistic Programming Languages. // J. of Universal Computer Science. — 1997. — №3(5). — P. 416–442.
14. **Ellsberger J., Hogrefe D., Sarma A.** SDL: Formal Object-Oriented Language for Communicating Systems. — Prentice Hall, 1997.
15. **Cater S.C., Huggins J.K.** An ASM Dynamic Semantics for Standard ML // Abstract State Machines, 2000. — P. 203–222.
16. **Касьянов В.Н., Поттосин И.В.** Методы построения трансляторов. — Новосибирск: Наука, 1986.
17. **Brandes U., Eiglsperger M., Herman I., Himsolt M., Marshall M.S.** GraphML Progress Report // Structural Layer Proposal: Proc. / 9th Intl. Symp. Graph Drawing (GD '01). — Springer-Verlag, 2002. — P. 501–512. — (Lect. Notes Comput. Sci.; № 2265).

Д.В. Шкурко

СИСТЕМЫ ПЕРЕПИСЫВАНИЯ ГРАФОВ: ВЫБОР ЛИДЕРА И РАСПОЗНАВАНИЕ ТОПОЛОГИИ В АНОНИМНЫХ СЕТЯХ

1. ВВЕДЕНИЕ

Существующие универсальные алгоритмы выбора лидера [8] используют различные способы ослабления анонимности, один из которых наличие глобальной информации, доступной в вершине, и имеют в каждом процессоре сети сложность по памяти, зависящую от размера всей сети. Фактически, вершины обмениваются хранящимися в них описаниями частей всей сети, увеличивая размер частей, описание которых имеется в вершинах. Это требует значительных затрат на обмен информацией и выполнения сложных вычислений в каждой вершине. В связи с этим рассматриваются узкие классы графов сетей, для которых существуют алгоритмы, требующие меньше ресурсов на хранение, передачу и обработку информации.

В данной работе представлены результаты о построении алгоритмов выбора лидера для двух классов графов. Построенные алгоритмы одновременно являются алгоритмами распознавания топологии для соответствующих классов. Алгоритмы используют формализм систем переписывания графов. Построенные системы переписывания используют конечное множество меток и используют только локальную информацию.

2. МОДЕЛЬ РАСПРЕДЕЛЕННОЙ СИСТЕМЫ

Распределенная система (РС) представляется сети в виде связного неориентированного размеченного графа $G = (V,E,\lambda)$, $\lambda: V \cup E \to \Lambda$. Вершины графа $v \in V$ обозначают процессоры сети, дуги $e \in E$ — двусторонние линии связи, а метки $\lambda(x), x \in V \cup E$, приписанные элементам графа, — состояния элементов сети. Мощность множества Λ напрямую связана с памятью, необходимой для представления каждого элемента этого множества, поэтому представляет интерес простейший случай, когда множество состояний каждого элемента графа конечно и не зависит от размера графа.

Распределенные вычисления в данной модели представляются в виде локальных правил переписывания меток. *Правилом переписывания* называется пара связных размеченных графов (G,G'). Графы G и G', рассматриваемые как неразмеченные, должны быть изоморфными. Применение правила к размеченному графу H состоит в замене меток из образа вложения $G \to H$ на соответствующие метки из G'. Если образы вложений $G_1 \to H$ и $G_2 \to H$ для двух правил не пересекаются, то возможно одновременное применение этих правил. Применения непересекающихся правил соответствуют параллельной работе процессоров сети.

В соответствии с результатами [10], выразительную мощность локальных вычислений можно усилить, введя дополнительные механизмы ограничений применений правил. *Правилом с запрещенными контекстами* называется правило переписывания (G,G') и набор запрещенных контекстов F_1, ..., F_n. Для каждого запрещенного контекста определено вложение $G \to F_i$. Применение правила блокируется, если вложение $G \to H$ может быть продолжено до вложения $F_i \to H$.

Исследования построения алгоритмов в РС, начиная с работы [5], используют следующее фундаментальное понятие накрывающего отображения графов. *Окрестность* вершины v — это множество $N_G(v) = \{u \in V(G) \mid \{u, v\} \in E(G)\}$. *0-накрывающее отображение* — это эпиморфизм графов $f : G \to H$, который инъективен в любой окрестности. Граф G называется *накрытием*. *Шар* $B_G(v,k)$ радиуса k — это подграф, индуцированный вершинами графа G, находящимися на расстоянии не более k от вершины v. *k-накрывающее отображение* — это эпиморфизм графов $f : G \to H$, который инъективен на любом шаре $B_G(v,k)$. Граф G называется *k-накрытием*.

3. ПРОБЛЕМА ВЫБОРА ЛИДЕРА

Одной из базовых проблем координации работы РС, позволяющей решать другие проблемы, например, построение накрывающего дерева, рассылку сообщений и т.п., является проблема построения алгоритма выбора лидера (ВЛ). Алгоритм ВЛ должен гарантировать, что в конце его работы единственный процессор находится в выделенном состоянии и переход процессора в это состояние означает невозможность никакого другого про-

цессора перейти в такое же состояние. Проблема ВЛ рассматривалась во многих работах при разных предположениях о количестве информации о всей сети, доступной в каждой вершине, например, [5, 8, 9, 13].

С самого начала исследований РС изучалось построение алгоритма ВЛ в анонимных сетях. *Анонимной сетью* называется сеть, начальное состояние которой представляется размеченным графом (V,E,λ), где $\lambda(x) = c$. Полностью анонимной сетью называется такая анонимная сеть, что c не зависит от (V,E). В соответствии с результатом из [5], существует максимальный класс графов $\mathcal{G}$, удовлетворяющий необходимому условию существования алгоритма выбора лидера в полностью анонимной сети. Этот класс у́же класса всех графов. Наиболее общий известный алгоритм для класса $\mathcal{G}$, представленный в [8], использует предположение, что $c = c(|V|+|E|)$. Этот алгоритм использует объем памяти в каждой вершине, зависящий от размера сети и требующий сложных вычислений в каждой вершине.

4. ПРОБЛЕМА РАСПОЗНАВАНИЯ ТОПОЛОГИИ

Решение проблемы определения топологии сети состоит в построении системы переписывания графов, которая удовлетворяет следующим условиям:

- любая последовательность переписывания конечна;
- по полученной в конце применения максимальной последовательности разметке можно определить, принадлежит или нет граф сети требуемому классу графов.

Проверка финальной разметки осуществляется с помощью проверки справедливости логической формулы, полученной из обычных логических связок $\{\neg, \wedge, \vee\}$ и 0-местных предикатов ψ_l, $l \in L$. Справедливость предиката ψ_l для размеченного графа (G,λ) определяется следующим образом:

$$\psi_l \equiv \lambda^{-1}(l) \neq \varnothing \quad .$$

В работах [3] и [2] представлены алгоритмы распознавания соответственно интервальных и хордальных графов, использующие выделенного лидера. Эти алгоритмы используют в качестве меток натуральные числа, не превосходящие размера графа.

5. АЛГОРИТМЫ ВЫБОРА ЛИДЕРА И РАСПОЗНАВАНИЯ ТОПОЛОГИИ

Хордальным называется такой граф, что любой его бесхордовый цикл имеет длину 3.

Для класса хордальных графов справедлива следующая теорема.

Теорема 1. *Можно построить систему переписывания графов с запрещенными контекстами, которая решает проблему распознавания принадлежности классу хордальных графов. Эта система в классе хордальных графов решает проблему выбора лидера. Число правил и меток в правилах в этой системе переписывания конечно. Все графы из правил переписывания и запрещенных контекстов имеют радиус не более 1 и некоторые из них содержат циклы.*

◄ Для описания схемы построения алгоритма потребуется определение симплициальной вершины. *Симплициальной* называется такая вершина v графа G, что $B_G(v,1)$ является кликой.

Алгоритм строится на базе схемы последовательного удаления из графа симплициальных вершин. Корректность построенного алгоритма подтверждается результатом, описанным в [1], в соответствии с которым, хордальные и только хордальные графы имеют последовательность удалений симплициальных вершин, содержащую все вершины графа. В случае хордальности графа последняя оставшаяся вершина становится лидером. Наличие лидера является признаком хордальности графа.

Построение алгоритма проводится в три этапа. На первом этапе строится система с бесконечной системой следующих правил:

- последняя активная вершина становится лидером;
- не являющаяся в данный момент симплициальной вершина переходит в режим ожидания изменений в своей окрестности;
- если активная вершина симплициальна, то она исключается из дальнейшего рассмотрения, и все ожидающие вершины из окрестности переходят в активное состояние.

Первые два пункта кодируются с помощью двух правил переписывания. Последний же пункт требует бесконечного множества правил переписывания, так как имеется бесконечное число вариантов симплициальных вершин (степень вершин не ограничена).

Затем отдельно строится алгоритм проверки симплициальности вершин с конечным числом правил, проверяется его корректность.

На последнем этапе строится комбинация начального алгоритма с алгоритмом проверки симплициальности вершин, проверяется корректность

окончательного алгоритма. Следует отметить, что в случае нехордального графа возможно возникновение блокировок при проверке симплициальности вершин, что не мешает алгоритму распознавания оставаться корректным. ►

Слабо хордальным [7] называется граф, не содержащий бесхордовых циклов длины более 4, а также дополнений циклов длины более 4.

Для класса слабо хордальных графов верна теорема аналогичная теореме 1.

Теорема 2. *Можно построить систему переписывания графов с запрещенными контекстами, которая решает проблему распознавания принадлежности классу слабо хордальных графов. Эта система в классе слабо хордальных графов решает проблему выбора лидера. Число правил и меток в правилах в этой системе переписывания конечно. Все графы из правил переписывания и запрещенных контекстов имеют радиус не более 2, и некоторые из них содержат циклы.*

◄ Доказательство проводится по схеме доказательства теоремы 1. Принципиальное отличие состоит в замене распознавания с помощью удаления симплициальных вершин распознаванием с помощью удаления симплициальных ребер. Определим понятие симплициального ребра. Ребро графа $e \in G\text{-}S$ называется *S-наполняющим*, если каждый компонент связности подграфа $\overline{G}(S)$ дополнения $\overline{G}$ графа G состоит из вершин, каждая из которых смежна с фиксированным концом ребра e. *Симплициальным* называется ребро $e = \{a,b\}$, являющееся $N(e)$-наполняющим, где $N(e) = \{x \in V(G) \mid \{x, a\} \in E(G)$ или $\{x, b\} \in E(G)\} - \{a,b\}$.

В соответствии с результатом работы [6] граф является слабо симплициальным тогда и только тогда, когда удалением симплициальных ребер можно получить граф без ребер. Лидером хордального графа становится одна из вершин последнего оставшегося ребра. ►

Построенные алгоритмы являются простейшими из возможных в следующем смысле.

Теорема 3. *Не существует системы переписывания графов, которая не содержит графа с циклом среди графов правил переписывания и запрещенных контекстов и решает проблему распознавания хордальности графа.*

Не существует системы переписывания графов, которая не содержит графа радиуса более 1 среди графов правил переписывания и запрещенных контекстов и решает проблему распознавания слабой хордальности графа.

◄ Необходимым условием [11] корректности алгоритма распознавания топологии графа является наличие в правилах переписывания такого графа K, что связные компоненты его прообраза в i-накрывающем отображении $f:H\to G$ (где H — граф, не принадлежащий рассматриваемому классу, $i \geq 0$) не изоморфны K.

Возможные i-накрытия для соответствующих классов графов описаны в следующих технических утверждениях.

Теорема 4.

- *Существуют хордальные графы, имеющие нетривиальные накрытия, не являющиеся хордальными; существуют слабо хордальные графы, имеющие нетривиальные 1-накрытия, не являющиеся слабо хордальными.*
- *Любое 1-накрытие хордального графа является тривиальным; любое 2-накрытие хордального графа является тривиальным.*
- *Если хордальный граф накрывает какой-нибудь граф, то накрытие тривиально; если слабо хордальный граф накрывает какой-нибудь граф, то накрытие тривиально.*

После того как описания i-накрытий графов получены, ограничения на графы в правилах переписывания получаются из необходимого условия. Например, граф без циклов не удовлетворяет необходимому условию для 0-накрытия, граф радиуса 1 не удовлетворяет условию для 1-накрытий. ►

Полные построения алгоритмов и полные доказательства теорем приведены в [4].

6. ЗАКЛЮЧЕНИЕ

Построенные алгоритмы накладывают незначительные ограничения на сложность вычислений и на объем памяти в вершинах. Однако остался открытым вопрос относительно количества передаваемой информации между вершинами во время работы данного алгоритма. Объясняется это отсутствием критерия для оценки этого параметра. В системах с пересылкой сообщений таким критерием является количество отосланных сообщений. Простой перенос этого критерия на системы переписывания графов осложнен тем, что моделирование систем переписывания графов с помощью систем пересылки сообщений требует рандомизации [12].

СПИСОК ЛИТЕРАТУРЫ

1. **Емеличев В.А., Мельников О.И.** Лекции по теории графов. — М.: Наука, 1990. — 384 с.
2. **Евстигнеев В.А., Бояршинов В.А**. Распознавание хордальных графов в рамках распределенной модели вычислений // Проблемы систем информатики и программирования. — Новосибирск, 1999. — С. 107–129.
3. **Бояршинов В.А.** Распознавание интервального порядка на базе локальных вычислений // Проблемы систем информатики и программирования. — Новосибирск, 1999. — С. 130–145.
4. **Шкурко Д.В.** Системы переписывания графов: выбор лидера и распознавание топологии в анонимных сетях. — Магистр. дисс., НГУ, Новосибирск, 2004.
5. **Angluin D.** Local and global properties in network of processors // Proc. of 12-th Symposium on theory of computing. — 1980. — P. 82–93.
6. **Berry A., Bordat J.** Triangulated and weakly triangulated graphs: Simpliciality in vertices and edges. — 2000.
7. **Hayward R.B.** Generating weakly triangulated graphs. — Lethbridge, 1994. — 10 p. — (Tech. Rep. / University of Lethbridge; TR-CS-01-93).
8. **Kameda T., Yamashita M.** Computing on anonymous networks: Part i - characterizing the solvable cases // IEEE Trans. on parallel and distributed systems. — 1996. — Vol. 7. — P. 69–89.
9. **Kameda T., Yamashita M.** Computing on anonymous networks: Part ii - decision and membership problems // IEEE Trans. on parallel and distributed systems. — 1996. — Vol. 7. — P. 90–96.
10. **Litowsky L., Métivier Y., Sopena E.** Different local controls for graph relabelling systems // Math. Syst. Theory. — 1995. — Vol. 28. — P. 41–65.
11. **Lytovsky I., Métivier Y., Sopena E.** Graph relabelling systems and distributed algorithms. — Bordeaux, 1998. — (Tech. Rep./ Universite Bordeaux).
12. **Métivier Y., Saheb N., Zemmari A.** Randomized rendezvous // Mathematics and Computer Science: Algorithms, Trees, Combinatorics and Probabilities. — 2000. — P. 183–194.
13. **Symmetry** breaking in anonymous networks: Characterization / P. Boldi, B. Codenotti, P. Gemmel et al. // Proc. of 4th Israeli Symposium on Theory of Computing and Systems. — IEEE Press, 1996. — P. 16–26.

СОДЕРЖАНИЕ

CONTENTS

МОЛОДАЯ ИНФОРМАТИКА

СБОРНИК ТРУДОВ
АСПИРАНТОВ И МОЛОДЫХ УЧЕНЫХ

Под редакцией
к.ф.-м.н. И. С. Ануреева

Рукопись поступила в редакцию 20.12.04
Редактор З. В. Скок

Подписано в печать 21.07.05
Формат бумаги 60 × 84 1/16 Объем 7.0 уч.-изд.л., 7.7 п.л.
Тираж 75 экз.

ЗАО РИЦ «Прайс-курьер»
630090, г. Новосибирск, пр. Акад. Лаврентьева, 6, тел. (383-2) 330-72-02