

Федеральное государственное бюджетное учреждение науки
ИНСТИТУТ ПРОБЛЕМ ИНФОРМАТИКИ
РОССИЙСКОЙ АКАДЕМИИ НАУК

В.Д. ИЛЬИН

**СИСТЕМА ПОРОЖДЕНИЯ
ПРОГРАММ.
Версия 2013 г.**

М.: ИПИ РАН, 2013

© В.Д. Ильин, 2013
ISBN 978-5-91993-030-3

[Об издании](#)

УДК 004 + 336

ББК 32.973.26-018

Ильин, Владимир Дмитриевич. Система порождения программ. Версия 2013 г.
[Электронный ресурс] = The System of program generating. Version 2013 :
[монография] : для специалистов в автоматизации программирования, разработчиков
программных средств, преподавателей, аспирантов и студентов вузов / В. Д. Ильин.
— Электрон. текстовые дан. (1 файл). — М.: ИПИ РАН, 2013. — 1 электрон. опт. диск
(CD-ROM); 12 см. — Систем. требования: компьютер типа десктоп, ноутбук,
планшетный; процессор 800 Мгц; ОЗУ 512 Мб; операц. система Windows, OS X,
Android, iOS; программа для просмотра pdf-файлов; видеосистема с диагональю
экрана не менее 5 дюймов; стандартная акустическая система. — [Загл. с экрана.](#)

Монография посвящена проблеме автоматизированного конструирования
программных систем с заданными характеристиками.

Изложены теоретические основы методологии, названной И-порождением целевых
программных систем.

Приведено детальное описание её применения для конструирования пакетов
программ в системе ГЕНПАК.

В версии 2013 г. представлены новые s-модели задачных конструктивных объектов и
задачных графов.

Для описания s-моделей применена новая версия языка TSM.

Книга может быть полезна преподавателям, аспирантам и студентам вузов.

Электронное научное издание

© Ильин Владимир Дмитриевич, 2013

Федеральное государственное бюджетное учреждение науки
ИНСТИТУТ ПРОБЛЕМ ИНФОРМАТИКИ
РОССИЙСКОЙ АКАДЕМИИ НАУК

В.Д. ИЛЬИН

**СИСТЕМА ПОРОЖДЕНИЯ
ПРОГРАММ.
Версия 2013 г.**



Москва
ИПИ РАН
2013

© В.Д. Ильин, 2013
ISBN 978-5-91993-030-3

ОБ АВТОРЕ

Ильин Владимир Дмитриевич

Доктор технических наук, профессор.

Зав. лабораторией «Методологических основ информатизации» в Институте проблем информатики РАН.

Автор книг и статей по S-моделированию, автоматизации программирования, методологии информатизации экономической деятельности и государственного управления, системам знаний.



Книги, связанные **методологией S-моделирования** с этой монографией

1. Ильин А.В., Ильин В.Д. *Основы теории S-моделирования*. М.: ИПИ РАН, 2009, 143 с. – ISBN 978-5-902030-78-2
2. Ильин А.В., Ильин В.Д. *S-моделирование объектов информатизации*. М.: ИПИ РАН, 2010, 412 с. – ISBN 978-5-902030-86-7
3. Ильин А.В., Ильин В.Д. *Символьное моделирование в информатике*. М.: ИПИ РАН, 2011, 204 с. - ISBN 978-5-91993-005-1
4. Ильин А.В., Ильин В.Д. *S-моделирование задач и конструирование программ*. М.: ИПИ РАН, 2012, 146 с. – ISBN 978-5-91993-013-6
5. Ильин В.Д. *S-моделирование имущественного обмена*. М.: ИПИ РАН, 2008, 80 с. – ISBN 978-5-902030-62-1
6. Ильин В.Д. *Модель нормализованной экономики*. М.: ИПИ РАН, 2009, 125 с. – ISBN 978-5-902030-77-5
7. Ильин В.Д. *S-модель нормализованной экономической системы*. М.: ИПИ РАН, 2010, 103 с. – ISBN 978-5-902030-79-9
8. Ильин А.В., Ильин В.Д. *S-экономика: механизм хозяйствования в эпоху Интернета*. М.: ИПИ РАН, 2011, 105 с. – ISBN 978-5-902030-94-2
9. Vladimir D. Ilyin. *S-economics*. M.: IPI RAN, 2012, 54 p. – ISBN 978-5-91993-017-4
10. Ильин А.В. *Экспертное планирование ресурсов*. М.: ИПИ РАН, 2013, 58 с. – ISBN 978-5-91993-022-8
11. Ильин А.В., Ильин В.Д. *Научно-образовательные веб-ресурсы. S-моделирование*. М.: ИПИ РАН, 2013, 110 с. – ISBN 978-5-91993-023-5

[Содержание <](#)

УДК 004 + 336

В.Д. Ильин

Система порождения программ. Версия 2013 г.

– М.: ИПИ РАН, 2013. – с. 142 – ISBN 978-5-91993-030-3

Монография посвящена проблеме автоматизированного конструирования программных систем с заданными характеристиками.

Изложены теоретические основы методологии, названной И-порождением целевых программных систем.

Приведено детальное описание её применения для конструирования пакетов программ в системе ГЕНПАК.

В версии 2013 г. представлены новые s-модели задачных конструктивных объектов и задачных графов.

Для описания s-моделей применена новая версия языка ТСМ.

Для специалистов в автоматизации программирования и разработчиков программных средств.

Книга может быть полезна преподавателям и студентам вузов.

Издано по решению Учёного совета Института проблем информатики Российской академии наук (ИПИ РАН)

© В.Д. Ильин, 2013

ISBN 978-5-91993-030-3

Vladimir D. Ilyin

The System of program generating. Version 2013

– М.: IPI RAN, 2013. – р. 142 – ISBN 978-5-91993-030-3

The monograph is devoted to the problem of computer-aided design of software systems with the specified characteristics.

The book describes the theoretical foundations of the methodology called I-generating the target software systems.

An example of the application of this methodology is the system GENPAK which is described in detail.

New s-models of the task constructive objects and task graphs are presented in the version of 2013.

New version of the TSM-language is used to describe s-models.

For professionals in the automation of programming and software developers.

The book can be useful to university teachers and students.

Issued by decision of the Academic Council of Institute for Informatics Problems, Russian Academy of Sciences (IPI RAN)

© Vladimir D. Ilyin, 2013

ISBN 978-5-91993-030-3

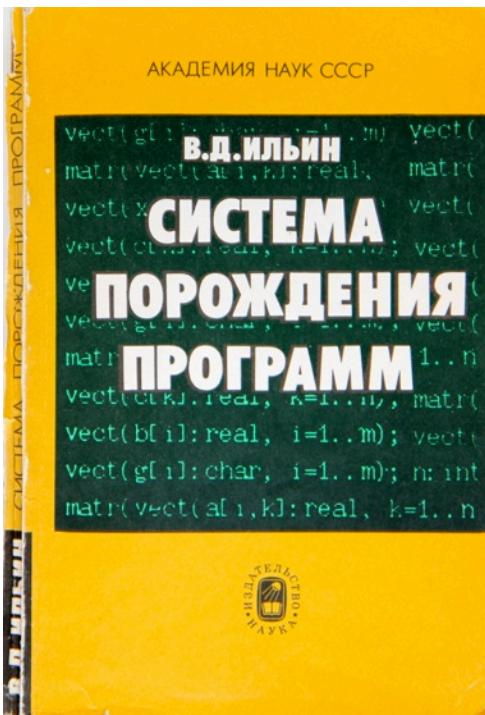
Содержание

Предисловие	8
О содержании	8
О судьбе книги «Система порождения программ»	8
Введение	12
О реализации	12
О представлении материала	13
TSM-комплекс средств описания s-моделей	14
Уровни фрагментов описания	14
Выделения	14
Сокращения	15
Умолчания	15
Формулы	15
Типы: специализация и обобщение	16
Графическая форма записи определений	17
Применимость	18
Программирование поведения символьных автоматов: проблема продуктивности	19
Автоматизация разработки программ: обсуждение	24
Программное обеспечение автоматизированной разработки программ	25
Языки: средства повышения производительности	30
Понятия ЧТО- и КАК-языков	31
Объектно-ориентированный подход к программированию	32
Тенденции развития	34
Повторное использование программ	36
САР-методология	36
Генерация частичных систем	41
Генераторы программ	42
Генераторы приложений	42
Генераторы приложений и сервис-ориентированные архитектуры	46
И-порождение: конструирование целевых программных систем	47
Концептуальные основы И-порождения	48
S-моделирование задач	55
Задачный конструктивный объект (s-задача)	55
Связи по памяти между s-задачами	56
Конструирование s-задачи	57
Конкретизация s-задачи	57
Атомарная s-задача	57
Система знаний об s-задачах	58
Модель задачной области	58
Интерпретация и разрешающая структура	58
Задачные графы	58
Исчисление s-задач	60

Содержание <

Конструирование разрешающих структур на задачных графах	62
Инф: модель унифицированной s-машины	65
Вычислительная s-машина	65
Исчислительная s-машина	65
И-порождение целевых систем: табс-представление	66
Табс	66
Табс-структура как форма для представления задач	66
Табс-представление р-объектов	70
Прикладные задачи	72
Пользовательские задачи	73
Табс-представление целевых систем	73
Табс-представленный инф	79
Табс-представление целевых знаний	81
Правила табс-навигации	81
Конструирование табс-представленных целевых систем	82
Прототипы целевых систем: средство накопления задачных знаний	82
И-порождение: характеристика и применимость	85
Порождённая многоцелевая система: пример	86
Одноцелевая система: пример	87
Табс-ориентированные языки	90
T-спецификация	91
T-программирование	91
Язык спецификации табс-форм	92
Язык T-кон	95
ГЕНПАК: система И-порождения пакетов программ	102
Назначение и общая характеристика	104
Состав	105
Система управления табс-базами данных	106
Взаимодействие с пользователем	108
Спецификация табс-форм и формирование отчётов	110
Программирование табс-вычислений	115
Диалоговый монитор	122
Функциональное наполнение	125
Р-задачи и их связь с R-задачами	127
И-порождение: среди методологий информатики	133
И-порождение и традиционное программирование	133
Интерактивный преобразователь ресурсов: пример реализации И-порождения	134
S-модель интерактивного преобразователя ресурсов	134
РЕСУРС-комплекс: архитектура, характеристика реализации, оценка функциональной эффективности	135
Литература	143

Предисловие



Учитывая запросы на электронный вариант книги «Система порождения программ», в 2012 г. была опубликована монография [[Ильин А.В., Ильин В.Д. 2012](#)], в которую (в сокращённом виде) был включён материал книги «Система порождения программ».

Но запросы продолжались: интересовавшиеся по-прежнему давали поисковым машинам задание найти именно книгу «Система порождения программ».

В ответ поисковики находили сайты, где предлагалось скачать неавторский вариант книги «Система порождения программ» (полученный сканированием бумажного оригинала книги с последующим программным распознаванием, как правило, добавлявшим немалое число ошибок). Потому автором и было решено сделать электронный вариант книги, получивший название «Система порождения программ. Версия 2013 г.»

О содержании

В книге представлен (с рядом исправлений, сокращений и дополнений) материал монографии В.Д. Ильин. Система порождения программ. М.: Наука, 1989, с.264 - ISBN 5-02-006578-1.

Он содержит изложение первой версии теоретических основ и метода порождения целевых систем, описание языков специфирования и программирования, ориентированных на порождение.

Представлено описание системы порождения ГЕНПАК, ориентированной на продуцирование пакетов программ для решения задач учёта и планирования (ГЕНПАК относится к семейству ИГЕН-генераторов).

Дан сравнительный анализ различных подходов к автоматизации разработки программного обеспечения.

О судьбе книги «Система порождения программ»

«Система порождения программ» - первая монография В.Д. Ильина, опубликованная в 1989 в издательстве «Наука».

В эти дни (ноября 2013 г.) её можно найти не только в основных библиотеках России и бывших республик СССР, но и в крупнейших библиотеках США, Великобритании и Германии.

С 1990 ссылки на эту книгу можно встретить в научных работах разного назначения, связанных с автоматизацией программирования.

URSS — российская издательская группа научной и учебной литературы (монографий, журналов, сборников трудов институтов РАН и др.) - через свой [интернет-магазин принимает заказ на продажу этой книги](#).

[Крупнейшая в мире \(по обороту\) американская компания, занимающаяся интернет-торговлей - amazon.com - тоже пытается продавать эту книгу](#).

Размещают неавторский файл книги (полученный путём сканирования бумажного оригинала) на сайтах-библиотеках, с которых предлагают его скачать (как правило, небесплатно).

[Содержание <](#)

Примеры таких сайтов:

- http://www.bookarchive.ru/computer/programming/other_programm/234819-sistema-porozhdjenija-programm.html
- <http://www.twirpx.com/file/1044788/>
- <http://2programmers.com/news/read/Sistema-porozhdjenija-programm.html>
- <http://mirknig.com/knigi/programming/1181593667-sistema-porozhdeniya-programm.html>
- <http://www.kodges.ru/komp/program/188890-sistema-porozhdeniya-programm.html>
- http://www.takelink.ru/knigi_uchebniki/knigi_komputernie/179697-ilin-vd-sistema-porozhdeniya-programm.html

ГЕНПАК - система порождения пакетов программ / В. Н. Барышников, В. А. Борисов, В. Д. Ильин и др.; Под общ. ред. В. Д. Ильина; АН СССР, Ин-т пробл. информатики 2 89-12/467	М. : ИПИАН, 1988
Система порождения программ / В. Д. Ильин; Отв. ред. Г. Д. Фролов; АН СССР, Ин-т пробл. информатики 1 89-29/314 1 89-29/315	М. : Наука, 1989
Представление знаний о задачах в системе порождения программ / В. Д. Ильин; АН СССР, Ин-т пробл. информатики 2 90-8/1027	М. : ИПИАН, 1989
Архитектура вычислительного ядра комплекса программных средств ресурсного обоснования решений / А. В. Ильин, В. Д. Ильин; Рос. акад. наук, Ин-т проблем информатики 3 96-5/592	М. : Б. и., 1995
Концепция ситуационной информатизации государственного управления России / В. Д. Ильин; Рос. акад. наук, Ин-т проблем информатики 3 96-5/610	М. : Б. и., 1995
Теория исчислений задачных конструктивных объектов / Ю. В. Гавриленко, В. Д. Ильин; Рос. акад. наук, Ин-т проблем информатики 2 96-8/158 2 96-8/159	М. : Б. и., 1995
Основания ситуационной информатизации / В. Д. Ильин; Рос. акад. наук 1 96-7/488 1 96-7/489	М. : Наука : Изд. фирма "Физ.-мат. лит.", 1996
Информатизация ситуационного управления : учебное пособие / В. Д. Ильин ; Федер. агентство по образованию, Гос. образоват. учреждение высш. проф. образования "Моск. гос. ин-т радиотехники, электроники и автоматики (техн. ун-т)" 3 05-41/565	Москва : Моск. гос. ин-т радиотехники, электроники и автоматики (техн. ун-т), 2005
Модель нормализованной экономики / В. Д. Ильин ; Российская акад. наук, Ин-т проблем информатики 2 10-24/410	Москва : ИПИ РАН, 2009
S-моделирование объектов информатизации [Электронный ресурс] : научное издание / А. В. Ильин, В. Д. Ильин ; Российская акад. наук, Ин-т проблем информатики ИЭР О 14-5/11	Москва : ИПИ РАН, 2010
Символьное моделирование в информатике [Электронный ресурс] = The Symbol modeling in informatics : [монография] : для информатиков, ИТ-проектировщиков, преподавателей вузов и студентов / А. В. Ильин, В. Д. Ильин ; Российская акад. наук, Ин-т проблем информатики ИЭР О 16-4/135	Москва : ИПИ РАН, 2011
S-экономика: механизм хозяйствования в эпоху Интернета [Электронный ресурс] = S-economics: economy in the Internet era : [монография] : для информатиков, ИТ-разработчиков, экономистов, преподавателей вузов и аспирантов / А. В. Ильин, В. Д. Ильин ; Российская акад. наук, Ин-т проблем информатики ИЭР О 16-3/246	Москва : ИПИ РАН, 2011

В Российской государственной библиотеке (Ленинке):
"Система порождения программ" (вторая в подборке, включающей 12 из 15 опубликованных (с 1988 по 2012) монографий (9), препринтов (5) и учебных пособий (1) В.Д. Ильина.

[The Library of Congress Home Page](#)

LIBRARY OF CONGRESS ONLINE CATALOG

The Library of Congress serves as a source for materials not available through local, state, or regional libraries, via interlibrary loan. Consult your local library for details, or view the [LC interlibrary loan policies](#).

Sistema porozhdeniia programm / V.D. Il'in ; otvetstvennyi redaktor G.D. Frolov.

LC Control Number: 90135441

Brief Description: Il'in, V. D.

Sistema porozhdeniia programm / V.D. Il'in ; otvetstvennyi redaktor G.D. Frolov.
Moskva : "Nauka", 1989.
257 p. : ill. ; 22 cm.

ISBN: 5020065781

LC Call Number: MLCS 93/12475 (T)

The Library of Congress is open to researchers and the public for in-house research only. Researchers must register in person at the [Reader Registration Station](#); the Library cannot accept registrations via mail, telephone, or electronically.



The Library of Congress
URL: <http://www.loc.gov/>
Mailing Address:
101 Independence Ave,
S.E.
Washington, DC 20540

Catalog/authority record
errors?
[Use our Error Report Form](#)
Questions about searching?
[Ask a Librarian](#)

Library of Congress Online
Catalog
URL: <http://catalog.loc.gov/>
Library of Congress Authorities
URL: <http://authorities.loc.gov/>

В библиотеке Конгресса США.



EXPLORE THE BRITISH LIBRARY

To view item holdings click the Details or Year link

System number	007787508
Author - personal	Il'in, V D.
Title	Sistema porozhdeniia programm.
Publisher/year	Nauka, 1989.
Added Title	Program generating system.
Holdings (All)	Details
Shelfmark	R213435(P) DSC

В Британской библиотеке.

[Содержание <](#)

Search | Results | Advanced search | Saveset | Help

search [and] all words (ALL) ? sort by year of publication approxim sea

search history ▾ shortlist ▾ title data

Save Analyse Set Additional catalogues Subject search as of 1946 Subject search 1501 - 1955 Reading room Logout

= results search (Pica production number) 128681853

Title: Sistema poroždenija programm / V. D. Iljin. Qty. red. G. D. Frolov
Author: Iljin, Vladimir Dmitrievič
Published: Moskva : Nauka, 1989
Extent: 257 S. : graph. Darst. ; 22 cm
Note: In kyrill. Schr
ISBN: 5-02-006578-1
Further documents: Library of Congress Classification: [MLCS 93/12475 \(T\)](#)

Shelf mark: 44 MA 8018
Location: Außenmagazin

В Берлинской государственной библиотеке.

Даже в библиотеке Эстонии, "решительно порвавшей с советским прошлым", хранят [эту книгу](#) [официальный положительный отзыв на "Систему порождения программ" в 1990 г. написал проф. Энн Тыугу (автор книги "[Концептуальное программирование](#)").

Замечания о части результатов, представленных в [[Ильин В.Д. 1989, 1](#)]

1. Часть аналитического обзора средств автоматизации программирования содержит материал, читая который следует помнить, что написан он был в 1989.
2. Читая описания языков *табс-ориентированного программирования*, также не следует забывать, что 23 года назад не существовало ни нынешних средств графического интерфейса, ни нынешних редакторов.

Введение

Построение целевой системы относится к задачам автоматизации разработки программного обеспечения.

- **Целевой называем программную систему**, предназначенную для решения конечного множества задач, рассматриваемого как задача более высокого уровня (цель). □

Целевая система обладает рядом обязательных признаков.

Её архитектура рассчитана на объединение нескольких систем в целевую систему более высокого уровня.

Существует возможность получить новую систему как сужение исходной.

- **Порождение целевых программных систем** (кратко - порождение) — процесс их построения на основе других целевых систем, называемых порождающими.
Порождение — это конструирование искомых систем из задачных конструктивных объектов. □
- **Среда порождения** — комплекс программных систем, назначением которого является поддержка процесса формирования и реализации замысла разработчиков целевых систем. □

Среду порождения можно рассматривать как совокупность взаимодействующих целевых систем, в которой уже появившиеся на свет системы помогают разработчикам порождать новые.

Функциональный облик среды порождения определяется ее назначением: обучение разработчика, объяснения по его запросу шагов порождения — всё это присуще ей.

В результате последовательных приближений, когда человек формирует и уточняет свои представления о порождаемой системе, он постепенно добивается такого её описания, которому свойственны недвусмысленность и точность спецификации.

Повышение продуктивности разработки целевых систем недостижимо без свободы выбора способа взаимодействия со средой и без снятия нагрузки, связанной с запоминанием правил работы.

Языки взаимодействия с порождающей системой — это интерактивные формо-ориентированные языки описания порождаемой системы.

Они рассчитаны на использование мощных интерактивных средств. Их прагматика ориентирована на эффективную поддержку рассуждений разработчика, обдумывающего функциональный облик порождаемой системы.

О реализации

Только реализация может обнаружить невидимые до неё недостатки и подтвердить существование достоинств.

Поэтому была разработана система ГЕНПАК порождения пакетов программ.

Реализация экспериментального образца доопределила требования к языкам и базовой операционной системе.

Экспериментальная версия была написана на Паскале и работала под управлением ОС РАФОС 2.

Такой выбор языка и операционной системы не означает, что они хороши для этой цели.

Они использованы лишь потому, что к началу разработки существовал задел в виде системы ДИЭКС [Барышников, Ильин В.Д. и др.].

Некоторые компоненты ДИЭКСа предполагалось применить в новой системе.

Ещё менее пригодным для такой разработки был комплекс УВК СМ 1407.01. Конечно, ещё до начала реализации была готовность поменять ОС РАФОС 2 на Unix, Паскаль на Си, а УВК СМ 1407.01, например, на одну из моделей VAX 8000. Однако, тот факт, что экспериментальная версия системы порождения реализована с использованием столь неподходящих для неё средств, подчёркивает довольно высокий уровень независимости методологии порождения целевых программных систем от средств реализации.

О представлении материала

Стремлением к компактному изложению объясняется графическое представление части материала.

Для этого используется форма определений, описание которой дано в разделе о [TSM-комплексе](#).

[Рисунки, полученные сканированием бумажного экземпляра монографии, могли бы выглядеть получше (исполнителю заказа, очевидно, не хватило умения)].

TSM-комплекс средств описания s-моделей

□ *TSM-комплекс средств описания s-моделей* (англ. TSM) – расширяемый набор средств унифицированного описания s-моделей систем понятий и систем знаний. Включает средства одноуровневой записи формул, выделения частей гипермейдийных описаний s-моделей и замены выбранными сокращениями часто повторяющихся фрагментов. □

Первая версия TSM была предложена при работе над теорией порождения программ, где TSM служил средством записи спецификаций задачных конструктивных объектов [[Ильин В.Д. 1989, 1](#)].

Изложенная здесь версия содержит ряд синтаксических улучшений варианта TSM, развитого в [[Ильин А.В. 2007](#)].

Одноуровневые TSM-описания соответствуют стилю, принятому в языках программирования.

Для TSM-описаний достаточно стандартной клавиатуры и набора специальных символов, имеющихся в составе текстовых редакторов Word (пакета MS Office), Pages (пакета iWork), Writer (пакетов LibreOffice или OpenOffice) или др., что существенно для успешного развития TSM.

Универсализации TSM способствовало применение этого комплекса при формировании образовательных ресурсов и разработке системы знаний информатики СИНФ.

Уровни фрагментов описания

□ Фрагмент TSM-описания – часть описания, включающая не менее одного полного абзаца (без заголовка или с заголовком). □

Выделяется косыми (slashes), размещаемыми в начале фрагмента: /k/ (k – номер уровня вложенности).

Для первого и второго уровней значения k не указываются (/ – первый уровень вложенности; // – второй); для третьего и последующих уровней можно указывать (начало фрагмента третьего уровня можно обозначить как /// или как /3/).

Выделения

Для выделения определений, замечаний, примеров, имен понятий и отдельных частей описания используются следующие средства:

△ <фрагмент описания> △ ≈ часть описания с фиксированными в ее пределах обозначениями (здесь и далее символ ≈ заменяет слово означает);

□ <фрагмент описания> □ ≈ утверждение (определение, аксиома и др.);

◊ <фрагмент описания> ◊ ≈ замечание;

⊗ <фрагмент описания> ⊗ ≈ пример;

[Содержание <](#)

Δ <фрагмент описания> $\Delta \approx$ рекомендация или комментарий составителя описания;

{S<фрагмент описания><список>S} \approx здесь <фрагмент описания> \approx набранный курсивом текст (может быть пустым), который следует интерпретировать как расширенный префикс s-<фрагмент описания> для выделенных курсивом элементов списка;

⊗ {S-модель<список>S} – здесь расширенным префиксом служит s-модель; {S<список>S} – здесь префикс s-.⊗.

Курсивом могут быть выделены:

- первые вхождения названий понятий [определяемых или определённых (последние могут быть гиперссылками)];
- фрагменты описания, к которым автор хочет привлечь внимание;
- формулы.

Сокращения

Для часто повторяющихся названий понятий:

СМ \approx символьное моделирование;

S-моделирование \approx СМ произвольных объектов в человеко-машинной среде;

s-машина \approx машина, помогающая создавать и применять s-модели;

s-среда \approx совокупность взаимодействующих людей и управляемых ими s-машин, предназначенная для решения задач S-моделирования.

Умолчания

Так как в s-среде имеем дело только с s-моделями, вместо s-модель символа, s-модель кода, s-модель сообщения, s-модель информации и т.д., пишем s-символ, s-код, s-сообщение, s-информация и т.д. Слово s-модель не опускаем лишь там, где может возникнуть контекстная неясность.

Формулы

Для теоретико-множественных и других формул применяется одноуровневая форма записи.

/ Индексы, пометы

Не накладывается никаких ограничений на максимальное число индексов для переменных и помечающих символов (помет).

Все индексы и пометы записываются в строчку внутри квадратных скобок, следующих сразу за индексируемой (или/и помеченной) переменной.

Индексы, определяющие элемент массива, отделяются запятыми, индексированные индексы – косой чертой «/».

Верхний индекс от нижнего отделяется точкой с запятой «;». Если в описании индекса точка с запятой не встречается, то индекс считается нижним. Если сразу после точки с запятой стоит закрывающая вертикальная черточка, то — задан только верхний индекс.

⊗ x[out; j=1...n] \approx вектор x из n компонент, имеющий помету out;

a[inp; i=1...m, j=1...n] \approx матрица a размера m*n, имеющая помету inp;

c['; 1] \approx с-один со штрихом (штрих «'» – верхняя помета, 1 – нижний индекс);

[Содержание <](#)

$d[j/i]$ \approx d с верхним индексированным индексом j i-тое (чтобы показать отсутствие нижних индексов, поставлена точка с запятой, за которой сразу следует закрывающая вертикальная черточка);
 $d[j/i]$ \approx d с нижним индексированным индексом j i-тое (отсутствие точки с запятой указывает на отсутствие верхних индексов) ☀.

/ Теоретико – множественные

a: elem A \approx a является элементом множества A;
a, b: elem C \approx a, b — элементы множества C (число элементов, разделённых запятыми, может быть любым);
A: set a \approx A – множество, содержащее элемент a;
A<B (когда оговорено, что A и B рассматриваются как множества) \approx A – подмножество B;
B=D \approx множества D и B совпадают;
C≤B \approx C является подмножеством B или совпадает с ним;
B>A \approx B содержит A;
A≥E \approx A содержит E или совпадает с E;
A^B \approx объединение множеств A и B;
A^B \approx пересечение множеств A и B;
A\B \approx разность множеств A и B;
A*B \approx декартово произведение множеств A и B;
 $R \leq A^* B$ \approx бинарное отношение, заданное на множествах A и B.
Символ 0 обозначает пустое множество или нуль (в зависимости от контекста);
символ # обозначает «не равно».

☀ Если x: elem X, y: elem Y и x = y, то x: elem (X ^ Y);
если X: set x, Y: set y и пара (x, y): elem R, где $R \leq A^* B$, то $(X^*Y)^*(A^*B)\#0$. ☀

/ Функции

Аргументы функции размещаются в круглых скобках, стоящих сразу за идентификатором, обозначающим функцию.

☀ f(x) \approx f от x; f[max;](x[i=1...n]) \approx f с верхней пометой max от x[i = 1...n]. ☀

При записи операций символы «+», «-», «*», «/» обозначают соответственно сложение, вычитание, умножение, деление, а символ «**» – возведение в степень.

Для записи суммы вместо символа « Σ » используется «sum»; при этом индекс суммирования, его начальное и конечное значения записываются в квадратных скобках справа от «sum».

☀ sum[i=1...n]x[i] \approx сумма x[i] по i от 1 до n. ☀

Типы: специализация и обобщение

□ Тип X \approx множество X, элементы которого имеют фиксированные набор атрибутов и семейство допустимых операций.
Может иметь подтипы, называемые специализациями типа X, и надтипы, называемые обобщениями типа X. □

/ Специализация типа

- Специализация типа X – порождение подтипа $X[::rule]$ (здесь сдвоенное двоеточие « $::$ » — символ специализации) с семейством связей, расширенным добавлением связи rule.

Выделяет подмножество $X[::rule]$ множества X .

Специализацией называем и результат $X[::rule]$ этого порождения ($X > X [::rule]$). □

// Специализация типа, заданная последовательностью добавленных связей

$X[::(rule1)::rule2]$ – специализация типа $X[::rule1]$ по связи rule2.

Число специализирующих связей в последовательности не ограничено. При этом имена связей, предшествующие последнему, заключены в круглые скобки, а перед открывающей скобкой каждой пары скобок – сдвоенное двоеточие.

/ Обобщение типа

- Обобщение типа Z – это порождение его надтипа $Z[rule]$ путём ослабления (здесь $\#$ – символ ослабления) связи rule из семейства связей, соответствующей типу Z .

Исключение связи считаем её предельным ослаблением. □

Графическая форма записи определений

Как обычно, определение даётся путём построения дерева атрибутов изучаемого понятия.

Сначала атрибуты перечисляются, а затем, в свою очередь, определяются (если это необходимо).

Листьями так построенного дерева являются терминальные атрибуты (не требующие в данном контексте дальнейшего определения).

Другой способ описания понятия — указание на понятия, связанные с определяемым.

Важной особенностью введённой формы определения понятий являются средства явного указания на ТОЧКУ ЗРЕНИЯ (КОРРЕСПОНДЕНТА, которому адресовано определение), СТАДИЮ (на которой в процессе изучения или работы определение может быть полезно) и ЦЕЛЬ (в процессе достижения которой определение имеет смысл).

Эта триада задаётся явно в тех случаях, когда по контексту её значения могут быть неясны.

Важность явного указания точки зрения не требует пространного комментария: достаточно заметить, что масса недоразумений при определении понятий и их истолковании значительно уменьшилась, если бы авторы точно указывали, КОМУ, на какой СТАДИИ и для какой ЦЕЛИ может пригодиться определение.

Предложенная форма не требует изображения дерева атрибутов определяемого понятия в виде графа.

Определение выглядит как последовательность взятых в рамку утверждений.

Часть площади внутри рамки отведена для значений ТОЧКИ ЗРЕНИЯ (если по контексту эти значения неясны).

Все утверждения, размещённые в рамке, имеют одинаковую структуру:
 $|<\text{определяемое}>| @ |<\text{определяющее}>|$.

[Содержание <](#)

Левая и правая части ограничены вертикальными чёрточками, а связывающий их символ @ принимает одно из трёх следующих значений: @ = (:, ->, ~).

Первое значение ":" (двоеточие) является заменителем слова "это".

Утверждение вида

|<определяемое>| : |<определяющее>|

означает, что текст справа рассматривается как определяющий (основанный на известных понятиях) по отношению к тексту слева, содержащему определяемое понятие.

Второе значение "->" (стрелка) заменяет слова "связано с".

Утверждение вида

|<определяемое>| -> |<определяющее>|

истолковывается, как "определенное понятие (слева) связано с другими (важными в данном контексте) понятиями (справа)".

И, наконец, значение "~" (эквивалентность) используется для переобозначения определяемого понятия.

То есть утверждение вида

|<определяемое>| ~ |<определяющее>|

означает, что справа введено обозначение того, что имеем слева.

Содержимое, заключенное между вертикальными чёрточками, может быть текстом, графическим изображением или их сочетанием.

Никаких синтаксических ограничений, накладываемых на представление такого содержимого, предложенная форма не вводит.

Таким образом, форма позволяет представить определение понятия в виде последовательности утверждений трёх типов, имеющих заданную графику.

Опыт показал, что эта форма определений позволяет не только экономно расходовать ресурс восприятия читателя.

Важно и то, что она дисциплинирует работу того, кто формулирует определения.

Применимость

TSM рассчитан на создание строчных одноуровневых описаний s-моделей посредством QWERTY-клавиатуры.

Применим при s-моделировании объектов любой предметной области.

Программирование поведения символьных автоматов: проблема продуктивности

Программирование поведения символьных автоматов занимает особое место среди всех видов деятельности, связанных с управлением.

Оно делает потенциально короткой по времени и уникальной по эффективности цепочку: замысел — его символьное воплощение — реализация в виде задуманного поведения автомата.

К сожалению, пока только потенциально.

Процесс символьного воплощения замысла в виде сегодняшней разработки программного обеспечения никак не назовёшь коротким по времени.

Из-за этого остаётся лишь потенциальной и уникальной эффективность всей цепочки. Её среднее звено является критическим.

С тех пор как существует программирование, программисты ищут способы повышения производительности своего труда.

Особенно актуальной стала проблема автоматизации теперь, когда недостаточный уровень продуктивности разработки программного обеспечения сдерживает во многих областях процесс их информатизации.

Естественно, что первыми в деле сохранения, накопления и применения знаний с помощью *s*-машин стали разработчики программных средств.

Уже на начальных этапах развития программирования (вместе с языками и трансляторами) они стали создавать различные библиотеки программ.

За вопросами «Как объединить подобные библиотеки?» и «Как добавлять в них апробированные программы?» последовал вопрос «Какой должна стать система знаний о программируемых задачах, чтобы служить основанием автоматизации программирования?»

Наряду с другими ответами на подобные вопросы (на основе опыта успешной реализации и применения генератора программ для создания, обработки и представления табличных структур [[Ильин В.Д. 1987](#)]) автором был предложен подход к построению программ как интерактивному конструированию из задачных конструктивных объектов.

В системе автоматизированного конструирования программ (названной системой порождения программ [[Ильин В.Д. 1989, 1](#)]) система знаний о программируемых задачах представлена в форме конструкций, построенных из задачных конструктивных объектов.

Основания для автоматизации любой деятельности тем значительнее, чем лучше она изучена и чем удачнее формализована её технология [[Ильин А.В., Ильин В.Д. 2010](#)].

С первых шагов автоматизации программирования обозначились два направления, определяемые разными точками зрения на программу как объект разработки и программирование как процесс построения этого объекта.

Первое - представлено теми, кто смотрит на программу, как на формальный объект, а программирование считает процессом доказательства его существования (напр. [[Manna, Waldinger 1971, 1981](#)]).

Второе — теми, кто не считает программу формальным объектом и рассматривает её как сообщение, определяющее поведение автомата с заданными свойствами и

[Содержание <](#)

существующее в символном, кодовом и сигнальном воплощениях, связанных отношениями трансляции.

Не станем приводить здесь аргументацию несостоятельности первого подхода: с ней можно ознакомиться в [[Ильин А.В., Ильин В.Д. 2011](#)].

Ограничимся цитированием аннотации доклада об автоматическом синтезе программ и структурном программировании, сделанного видными представителями первого направления [[Lee, Chang](#)]: «When a computer is used to synthesize a program, there is usually too much information for it to handle. In this paper, we propose the use of stepwise refinement technique, based upon the concept of structured programming, to overcome this difficulty.»

Представитель подходов второго направления (один из авторов системы LabVIEW [[Conway, Watts](#)]) Steve Watts удачно выразил его методологическую особенность: «A lot of techniques and methodologies get bogged down with computer science and forget about the design aspects; our intentions are to always concentrate on design and hopefully some of the computer science.»

Вполне естественно, что продуктивные идеи автоматизации программирования родились в среде программистов (такие, как идея модульного проектирования программ по принципу «сверху вниз», реализованная в технологии структурного программирования [[Dijkstra](#)]).

Учитывая, что к настоящему времени накоплено и продолжает быстро увеличиваться число задач, алгоритмы решения которых представлены программами на различных языках и апробированы в составе системных и прикладных комплексов, целесообразно создать систему их специфицированного описания, пользуясь которой можно было бы найти систематизированные сведения о постановках и алгоритмах, программах и тестовых примерах, чтобы использовать арсенал, накопленный при разработке программных средств и сервисов s-среды.

Системы программирования целесообразно связать с системами знаний о программируемых задачах, а системы знаний — между собой.

Исследователи, работающие в области автоматизации программирования, продолжают поиск способов повышения продуктивности.

Пройден путь от первых Ассемблеров до компилируемых и интерпретируемых языков, от простых редакторов и отладчиков до развитых интерактивных систем поддержки кодирования и отладки.

Работа по расширению традиционных методов программирования продолжается. Создаются новые языки, совершенствуются трансляторы и способы их разработки, редакторы становятся проблемно-ориентированными, интерактивные средства отладки наращивают мощь.

Создание языков все более высокого уровня диктуется стремлением ввести такие информационные объекты и операции на них, из-за отсутствия которых в существующих языках сдерживается рост производительности.

Вслед за АПЛ появились такие языки как SETL, который позволяет программировать, пользуясь математическими понятиями.

[Содержание <](#)

Важным этапом стала разработка пакетной проблематики как технологии решения задач.

Какая часть работы может быть поручена системе?

Что необходимо оставить человеку, определяющему задание на синтез искомой программы?

Велико стремление ряда исследователей [[Darlington](#); [Broy](#)] представить процесс конструирования программ как последовательность формальных преобразований.

Ведётся поиск методов построения генераторов программ, пользующихся растущим спросом [[Horowitz, Kemper, et al](#); [Luker, Burns](#); [Ильин В.Д. 1986, 1987](#)].

Разрабатываются языки и системы спецификации [[Агафонов](#); [Бежанова](#)].

Особое место занимает создание языка Пролог, представляющего собой не только язык программирования, но одновременно и средство, рассчитанное на задачи автоматизации программирования.

Не сдаёт своих позиций Лисп, применение которого для решения задач автоматизации программирования является традиционным.

В частности, Лисп успешно используется в задачах синтеза программ по образцам поведения.

Новый подход к программированию, обозначившийся ещё в языке Симула-67, окреп и утвердился с появлением языка Смолток-80.

[Объектно-ориентированное программирование](#) - идеология, где на первое место поставлен информационный объект, а не процедура его обработки. Создание Смолток-80 подтвердило ещё раз, что мощь языка может быть реализована только в том случае, когда он появляется на свет вместе с развитой средой программирования.

Этот язык показал, что небольшое число понятий, положенных в его основу, позволяет ему быть синтаксически простым, но обладать при этом завидной мощью. Мощь языка обеспечивается богатой динамической средой, в которой интерпретируются его утверждения.

И каждый раз, когда появляется новый подход к разработке программ, обнаруживается, что за этим стоит несколько отличное от предыдущих толкование понятия программа.

Естественно, что вместе с тем иначе толкуется и понятие программирование.

Программы, написанные в кодах машины, имели единственное лицо.

Оно было одним и тем же и для программиста, и для машины (с точностью до двоичного представления в ней).

Изобретение ассемблеров, освободивших программиста от необходимости держать в своей памяти цифровые коды операций и заниматься нудной задачей адресации, ознаменовало первый шаг к неодноликой программе.

Теперь у нее стало два лица.

Их соответствие осталось пока взаимно-однозначным.

Здесь еще не возникли проблемы ошибок перевода и потери эффективности.

[Содержание <](#)

Появление компилируемых, а затем и интерпретируемых языков высокого уровня, упростиивших для программиста не только запись программируемой задачи, но, что гораздо важнее, и обдумывание её, принесло не только облегчение, но и проблемы.

Настало время построения трансляторов.

Время борьбы с ошибками перевода и потерей эффективности.

Число лиц у одной и той же программы возросло.

Возникает вопрос: чем больше лиц у одной и той же программы, тем выше уровень автоматизации процесса её разработки?

Ответить просто — да — вряд ли правильно.

Поэтому скажем, что это связанные вещи.

Предоставляя возможность программисту записывать программируемую задачу в удобном для него виде, освобождая его от необходимости запоминать и выполнять что-то из того, что можно поручить автомату, разработчики средств автоматизации программирования перекладывают эту работу на себя.

При этом они берут на себя и тонкую задачу предвидения того, что будет полезно и удобно программисту.

Чем меньше они оставляют программисту, тем выше, естественно, становится производительность его труда, но тем ограниченнее его возможности повлиять на некоторые детали.

Предельным случаем автоматизации был бы, возможно, такой, когда программисту достаточно было сформировать некоторый начальный, пусть пока нечёткий, замысел задачи, которую он хотел бы запрограммировать.

Далее, взаимодействуя с системой автоматизированной разработки программ, он окончательно оформил бы замысел и с помощью всей той же системы создал описание интересующей его задачи.

Описание, понятное ему и воспринимаемое системой.

Остальное доделала бы система, руководствуясь этим описанием.

Выполняя работу по детализации порученного ей задания на программирование, система предлагала бы программисту сотрудничество в тех пунктах, где выбор из множества альтернативных решений не формализован, а потому требует участия человека.

Приблизиться к такому уровню автоматизации можно за счёт сокращения числа задач, на программную реализацию которых рассчитана система.

Тогда необходимо будет иметь много специализированных систем.

Сделаем еще шаг: представим, что мы объединили бы все такие системы, создав среду, где программирование в его традиционном смысле уступило бы место процессу, больше похожему на просмотр того, чем располагает среда, постановку ей вопросов, изучение её ответов, и только после этого — постановку задачи, которую среда должна представить в виде программы.

Построение решения исходной задачи среда выполняла бы путем представления ее в виде совокупности задач с известными решениями.

При этом программисту отводилась бы роль постановщика исходной задачи, а среде — роль конструктора решения, в соответствие которому она в итоге поставила бы искомую программу, посоветовавшись с программистом о деталях реализации.

Идея создания такой среды явилась отправной для постановки задачи порождения целевых систем.

[Содержание <](#)

На поиск метода её решения наибольшее влияние оказали опыт автора в разработке генераторов прикладных программ [[Ильин 1986](#), [1987](#), [1989](#)] и идеология объектно-ориентированного программирования.

Автоматизация разработки программ: обсуждение

Является ли программирование как деятельность процессом решения задач?

Вряд ли кто-то на такой вопрос ответит отрицательно.

Какие задачи приходится решать, создавая программу?

Этот вопрос вызовет гораздо большие затруднения, чем первый.

Предположим, что и на него удалось получить ответ: что-нибудь в духе Структура данных + алгоритм = программа.

Будем неотступны: как выглядит постановка каждой из задач?

Не обязательно формальная, но недвусмысленная и ясно позволяющая понять, что требуется найти и чем мы располагаем для поиска интересующего нас результата.

Продолжим работу и поинтересуемся тем, как мы решаем эти задачи.

Какую часть решения приходится изобретать каждый раз заново, а какую мы берем с соответствующей полки нашей памяти?

Или, может быть, система, в которой мы ведем разработку, хранит часто повторяющиеся решения в своей памяти и избавляет нас от необходимости запоминать их и, что гораздо труднее, вспоминать, когда нужно?

Есть задачи, решение которых программируем для исполнения машиной, и есть задачи, составляющие сам процесс разработки программ.

Первые могут относиться к разным предметным областям, вторые - к собственно программированию.

Что у этих задач общего и в чем различие между ними, если нас интересует возможность поручить автомату решение не только первых, но и вторых?

Общее то, что и те, и другие являются задачами, а значит относительно каждой из них можно поставить вопрос о спецификации.

Различие - в степени изученности.

Программирование как деятельность исследовано пока не так хорошо, как, скажем, конструирование аппаратной составляющей вычислительных систем.

Его доступность и похожесть на привычное для всех написание текстов создают иллюзии.

Диапазон иллюзий велик: от программирования для всех до программирования — искусства избранных.

И все-таки программа остается всего лишь описанием решения некоторой задачи, рассчитанным на восприятие автоматом.

Независимо от того, может или нет её создатель представить спецификацию задачи, которую он запрограммировал.

Составление спецификации — для многих дело более трудное, чем написание программы, описывающей решение этой задачи.

Наверное, поэтому вошёл в обиход стиль разработки, для которого характерен приоритет действия над осознанием цели, ради которой оно совершается.

Нигде нет такого простора для псевдорешений не полностью осознанных задач, как в программировании.

В результате нередко выясняется (уже на этапе эксплуатации), что программа решает совсем не ту задачу, которую стремился запрограммировать её автор.

[Содержание <](#)

Вспомним ещё раз историю пресловутой программы RANDU из библиотеки SSP, алгоритмическая некорректность которой была выявлена спустя много лет после начала её применения.

А сколько неразвенчанных RANDU-подобных программ продолжает своё существование, вводя в заблуждение тех, кто ими пользуется?

Программное обеспечение автоматизированной разработки программ

Единственный способ избежать неосознанного программирования — сделать его процессом решения задач, от спецификации которых нельзя увернуться.

Итак, программирование как процесс решения задач требует исследования.

В этой главе рассматривается точка зрения на проблему, которая определила предлагаемый подход к разработке программного обеспечения как процессу порождения программных систем.

Обсуждаются и некоторые другие подходы.

Предмет нашего изучения - программное обеспечение автоматизированной деятельности по разработке программных систем.

Каждой предметной области, характеризующейся множеством понятий и отношений между ними, ставится в соответствие множество задач.

Программное обеспечение автоматизированной деятельности в рамках предметной области рассматривается при этом как совокупность всех программных средств, необходимых для решения задач предметной области.

- **Программа (в широком смысле)** - это символьное описание способа взаимодействия с символным окружением, однозначно определяющее изменения его состояния. □

Известно, что причинами ненадежности программ являются пресловутые ошибки перевода.

Но существует ведь и причина причин, объясняющая, почему возникают эти ошибки. Такой причиной является недостаточная приспособленность традиционных языков и сред программирования к поддержке процесса постадийной детализации исходного замысла разработчика, определяющего функциональный и качественный облик программного продукта.

Ошибочность упования на один "самый лучший" язык (в смысле его приспособленности ко всем стадиям разработки) подтверждена не только неуспехом Ада-проекта.

Корни такого подхода глубоки, они уходят в те исторические слои процесса становления программирования, когда обладатель замысла и тот, кто воплощал его в программу, были представлены в одном лице, да и сами программы были гораздо проще.

Что требовалось тогда, чтобы, не расплескав замысел на ухабистом пути кодирования и отладки, добраться до состояния готовности программы, когда она продемонстрирует задуманное автором поведение?

Надо было не только представить постановку задачи, решение которой должна выполнять программа, но и сделать формализованное описание задачи в виде, пригодном для последующего перевода либо на язык блок-схем, либо - сразу на язык реализации.

[Содержание <](#)

Это было не только тяжело и долго.

Для этого нужны были люди, которые в одном лице совмещали и постановщика задачи, и разработчика программ, и, наконец, их пользователя.

В этом тяжком совмещении было одно благо - обладатель исходного замысла на всех стадиях разработки контролировал процесс.

Но это благо было отравлено горечью удручающе низкой продуктивности, свойственной любому "натуральному хозяйству".

Настали времена постадийного разделения труда.

Времена изобретения баз знаний о программируемых задачах, языков спецификации и программирования, редакторов (включая графические — для образного представления программных конструкций), инструментальных систем, поддерживающих символическое воплощение замысла (без искажений при переходе от одной стадии разработки к следующей за ней) [[Ильин В.Д. 1989, 1; Ильин А.В. 2007; Conway, Watts](#)].

Что способствует воплощению замысла при разработке программ

Подходящие языки спецификации и программирования;
инструментальные системы программирования с
удобными редакторами и средствами отладки

Поддержка символического воплощения замысла при разработке программ → Поддержка образного представления символьных моделей программируемых задач и программных конструкций

Расширяемые базы знаний о программируемых задачах

Чтобы можно было поставить вопрос об автоматизации разработки программного обеспечения, разработку необходимо представить как процесс решения задач.

Предположим, что множество задач определено. Рассмотрим эти задачи с точки зрения их возможной формализации. Если задача поддается формализованному решению, то его можно поручить автомату. Те задачи, которые формализовать не удалось, останутся человеку. Тогда вся деятельность в ее автоматизированном виде предстанет как процесс, на одних этапах которого задачи решает человек, пользуясь информационным обслуживанием автомата, а на других - автомат, контролируемый человеком.

Предельным случаем является полная автоматизация, когда человеку в начале остается только определять задание автомату, а в конце изучать полученный результат. Тогда говорят об автоматической работе.

Придерживаясь такого взгляда на автоматизацию разработки программного обеспечения, должны будем выделить работы, которые можем представить в виде задач.

Выбор работ предстоит сделать из известной их совокупности, получившей странное название "жизненного цикла" (видимо, и здесь сказалась из древности идущая привычка "оживлять" слабо познанные процессы, соперничающие по неуправляемости со стихиями).

[Содержание <](#)

Предположим, что множество задач, подлежащих решению с помощью машины, определено. Возьмемся ли мы за разработку, если не располагаем средствами для того, чтобы свести решение исходных задач из множества к решению некоторых базовых задач? То есть, таких задач, для каждой из которых известны: вход, выход, условия, задающие множества допустимых входов и выходов, алгоритм и технология программной реализации.

Если взялись за решение, то неизбежно возникает второй вопрос: какие у нас для этого основания? Собираемся ли мы исследовать каждую задачу, чтобы сначала найти алгоритм, а затем пройти отрезок пути, который связан с программной реализацией?

Предположим, что не видим в таком подходе ничего смущающего нас.

Тогда возникает третий вопрос.

Удастся ли нам обеспечить надлежащее качество и ту производительность труда, которая нас устраивает?

Ведь каждую задачу мы готовы разрабатывать как бы заново.

Представим альтернативный подход.

Вообразим, что в нашем распоряжении среда разработки программного обеспечения, ориентированная на получение архитектурно-подобных программных систем.

Такие системы отличаются одна от другой составом задач, алгоритмами их решения, реализацией задач на том или ином языке программирования, базовой операционной системой, интерфейсом пользователя, входным языком и обучающими пользователя системами, каждая из которых учитывает уровень его подготовленности.

И мы считаем для себя правилом браться за разработку, если установили, что задача, решение которой предстоит запрограммировать, может быть сведена к одной из известных задач или представлена в виде их совокупности.

На всем пути от спецификации задачи до получения готовой системы считаем необходимой поддержку среды, в которой ведем разработку.

Браться только за то, на что рассчитана среда разработки - довольно естественное намерение, если нас интересует надежность получаемых систем и продуктивность нашей работы.

Замысел, задача, программа

Осознание цели в процессе некоторой деятельности ещё не означает ясного представления о том, какие задачи должны быть решены для достижения цели. Это справедливо и для разработки программного обеспечения.

□ В этом смысле **цель** можно истолковывать как задачу более высокого уровня, решение которой требует рассмотрения совокупности других, более мелких задач. □

Естественно предположить, что система автоматизированной разработки программного обеспечения должна поддерживать процесс конкретизации цели. Другими словами, каждая крупная задача с помощью системы должна быть представлена как совокупность более мелких задач.

Чтобы разработать программу, необходимо специфицировать задачу, решение которой должна описывать программа.

Желательно, чтобы система поддерживала формирование замысла и составление спецификации задач, а выбор подходящей структуры данных и алгоритма либо

[Содержание <](#)

полностью брала на себя, оставляя нам совещательную функцию, либо выполняла эту работу в содружестве с нами.

Элементарный шаг в процессе символьного воплощения замысла



Процесс разработки программы можно представить как пошаговое символьное воплощение замысла.

Под реализацией замысла в виде задуманного поведения искусственной среды понимаем продуцирование исходного текста программы системой автоматизированной разработки.

Известные затруднения при формировании замысла вызывает неинформированность о возможностях системы.

Трудно бывает определить её возможности.

Поэтому необходимо предварительное обучение "под руководством" системы и информационное обслуживание в процессе работы.

Изучая её возможности, разработчик формирует представление о пространстве допустимых замыслов.

Действуя таким способом, первый этап работы — формирование замысла с помощью системы — можно будет выполнить более успешно.

Эффективность символьного воплощения замысла зависит и от того, какие языковые средства поддерживает система.

Очень важно, как выглядит реализация языка: текст или графика, или их сочетание. Всё это во многом определяет то, насколько успешным будет процесс символьного воплощения замысла.

Анализ синтаксической правильности берёт на себя система.

Но анализ правильности самого замысла, выполняемый путём изучения его символьного воплощения, — прерогатива человека.

Важно, чтобы система поддерживала определение точек зрения.

Два взгляда на спецификацию

Спецификация: две точки зрения

1	Формализованное описание задания на программирование задач, уменьшающее его неоднозначное толкование	ТОЧКА ЗРЕНИЯ <i>исследователя спецификации</i>
		Стадия: исследование спецификации как средства повышения продуктивности программной реализации
		Цель: определение требований к спецификации
2	Спецификация задач с описанием требований к программной реализации	ТОЧКА ЗРЕНИЯ <i>спецификатора программируемых задач</i>
		Стадия: спецификация задач
		Цель: повышение продуктивности программной реализации

Подчеркиваем необходимость чёткого описания точки зрения, стадии и цели при введении определений: ведь их содержание и форма представления зависят от того, кому они адресованы, с какой целью и на какой стадии работы могут быть полезны.

Программа: точки зрения пользователя и разработчика

Для пользователя программа служит средством решения задач предметной области. Её интересуют вход и выход программы, типы данных, представление результатов, интерфейс, информационное обслуживание в процессе выполнения, документация, требования к среде функционирования (ОС, под управлением которой работает программа, необходимые аппаратные средства).

Другая точка зрения у разработчика.

Ему необходимы чётко и недвусмысленно изложенные требования к функциональности программы, её интерфейсу с пользователем и другими программами, информационному обслуживанию, ОС (под управлением которой должна работать программа), аппаратным средствам, документации и поддержке в процессе эксплуатации.

ЯЗЫКИ: СРЕДСТВА ПОВЫШЕНИЯ ПРОДУКТИВНОСТИ

Языковое направление в автоматизации разработки программ является изначальным и неразрывно связано со всеми остальными.

В каждом языке воплощается взгляд на то, как должно выглядеть символьное описание алгоритма решения задачи (в виде программы, рассчитанной на автомат).

Поскольку используемый язык влияет на интеллектуальную продуктивность, а точки зрения разных людей на один и тот же язык неодинаковы, целесообразно, чтобы среда автоматизированной разработки программ предоставляла возможности программирования на разных языках.

Разработчик должен иметь возможность пользоваться теми языками, которые не кто-то, а именно он считает удобными для решения интересующих его задач.

Разработка языков и сред программирования (как и программирование решения задач) связана с изучением познавательных возможностей человека.

Интеллектуальное поведение связано с манипулированием символами.

Символы служат для представления результатов мышления в форме символьных сообщений, которыми можно обмениваться с другими людьми и которые можно сохранять и накапливать во внешней среде.

Символьные сообщения позволяют в компактном виде определять понятия и связи между ними.

Если структура простых логических операций может быть проанализирована без помощи символьных записей, то структура сложных соотношений, как справедливо заметил Р. Ледли, "не может быть обозрена без употребления символов".

Представление рассуждений в форме символьных сообщений позволяет единым мысленным взором охватывать необходимую для анализа часть рассуждений и, поставив в соответствие высказываниям и операциям над ними символьные конструкции и операции над ними, выполнять обработку высказываний, манипулируя их символьными заменителями.

Язык программирования как формализованная символьная система

Синтаксис Система правил построения предложений языка программирования (ЯП)

Семантика Система понятий ЯП (СПЯ) и система правил интерпретации на СПЯ синтаксически правильных предложений ЯП

Прагматика Характеристика ЯП с точки зрения программиста: восприятие СПЯ и выражений ЯП (как конструктивных составляющих), оценка достаточности совокупности средств записи и синтаксиса ЯП (запоминаемость, компактность)

Точка зрения: [исследователя](#)

Стадия: построение ядра системы понятий Язык программирования

Цель: определить систему понятий Язык программирования

Немало считающих, что идеальным языком программирования решений задач был бы такой язык, пользуясь средствами которого, можно запрограммировать любую задачу.

[Содержание <](#)

Попытки объединить некоторое множество языков в языковую среду более продуктивны, чем попытки создать единый «большой язык».

Стремление к повышению уровня языков существует с тех пор, как существует программирование.

Сегодня в ходу словосочетание "язык очень высокого уровня".

Какие слова понадобятся завтра?

Еще нелепее то, что явно и неявно уровень языка связывается со степенью его непроцедурности: мол, чем более непроцедурным является язык, тем выше его уровень.

При таком взгляде любой декларативный язык по уровню превосходит любой процедурный.

Это заблуждение иногда становится основой для рассуждений о том, что современная тенденция в развитии языков программирования связана с доктриной "ЧТО вместо КАК" (языки, позволяющие описать, ЧТО представляет собой задача, будто бы должны постепенно потеснить языки, предназначенные для описания алгоритма решения задачи, т.е. для ответа на вопрос КАК?).

Понятия ЧТО- и КАК-языков

Языки описания постановок задач (будем называть их ЧТО-языками) так же необходимы, как и языки описания процессов решения задач (КАК-языки). Они не являются конкурентами.

То, что до недавнего времени ЧТО-языки были представлены весьма немногочисленным семейством по сравнению с КАК-языками, говорит лишь о том, что программирование как вид деятельности переживало период младенчества, когда стремление к действию превалирует над стремлением осознать, зачем оно совершается.

Итак, всегда будут необходимы языки разных уровней, как в классе ЧТО-языков, так и в классе КАК-языков.

Признаки, определяющие уровень языка

Выбор языка и его уровня связан с характером задачи, намерением заниматься деталями, категорией пользователя, стремлением к эффективности создаваемой программы.

Во многих случаях было бы удобно иметь в своем распоряжении семейство языков, в которое входили бы и ЧТО-языки (для описания постановок программируемых задач), и КАК-языки (для описания процесса их решения).

В качестве примера приведём следующее сопоставление.

Во второй половине 1980-х среди КАК-языков, ориентированных на непрофессионалов в программировании, неформальным лидером был АПЛ (если оценивать КАК-языки по уровню в следующем смысле).

Уровень языка тем выше, чем более он ориентирован на описание потока данных, а не потока управления; чем более приспособлен к работе с агрегатами данных как с простыми объектами (поскольку размер и форма агрегатов могут меняться, процессор такого языка должен иметь схему автоматического управления памятью). Хотя АПЛ имеет хорошо определенную операционную семантику, он все-таки относится к языкам весьма высокого уровня в классе КАК-языков.

[Содержание <](#)

Ощущение свободы от деталей при работе с АПЛ обеспечено такой важнейшей характеристикой, как УТАИВАНИЕ ПОДРОБНОСТЕЙ.

Можно уверенно говорить о том, что уровень языка тем выше, чем совершенней механизм УТАИВАНИЯ ПОДРОБНОСТЕЙ.

В частности, в АПЛ программист более свободен от деталей, чем в Паскале или Фортране.

Тогдашний триумф АПЛ во многом был, видимо, предопределен ясной позицией, которую имел К. Айверсон, начиная работу над созданием языка.

Поставив целью создать систему записи вычислительных алгоритмов в сжатом виде, он не стал заботиться одновременно и о том, насколько такой язык будет удобен в реализации.

И хотя вначале АПЛ использовался только как предъязык для записи формулировок алгоритмов, которые затем вручную транслировались на выбранный язык программирования, уместно вспомнить, что именно с помощью АПЛ было сделано точное и компактное описание аппаратных средств IBM 360.

И все ныне существующие версии АПЛ так или иначе связаны с АПЛ/360.

Интерактивность и ориентация на обработку структур данных определили так называемый АПЛ-стиль программирования.

Принцип "сначала вычислить, потом проверить", реализация которого обеспечена регулярным характером массивов для вычисления всех значений и возможностью отфильтровать значения, не представляющие интереса, определил парадигму АПЛ-программирования.

Однако многие программисты-профессионалы считают, что такие языки, как Си и Модула-2 более подходят для создания больших систем, предназначенных для алгоритмизируемых задач.

Несложно представить, что их более интересует свобода в определении деталей и эффективность, чем возможность компактной записи алгоритма.

Так что считать язык тем совершенней, чем выше его уровень, - ровно столько же оснований, сколько, например, для объявления о превосходстве языка принципиальных электрических схем над языком монтажных схем (в электронике и электротехнике).

В классе ЧТО-языков влиятельное положение занял Пролог, основанный на исчислении предикатов первого порядка.

Создание этого языка и появление его многочисленных расширений ознаменовали завершение периода младенчества в развитии программирования.

Теперь программисты получили язык для описания замысла при создании программных систем. Важно и то, что программирование сделало пробный шаг навстречу математической логике.

Повышение уровня осознанности и доказательности при разработке программ - вот краткая характеристика Пролог-стиля программирования.

Не сдаёт своих позиций Лисп, основанный на Лямбда-исчислении.

Объектно-ориентированный подход к программированию

Объектно-ориентированное программирование (ООП) [англ. Object-oriented programming (OOP)] — это методология разработки программ для символьного моделирования с помощью компьютеров систем произвольного назначения,

[Содержание <](#)

представимых в виде совокупностей объектов, каждый из которых отнесён к определённому классу и наделён наборами данных (атрибутов объекта) и процедур (методов) их обработки. Классы объектов представлены в виде иерархии наследования атрибутов и методов.

С развитием символьного моделирования и усложнением моделируемых систем всё более явными становились недостатки методологий разработки программ с использованием процедурно-ориентированных языков программирования (Алгола, Фортрана и др.).

Это объясняется отсутствием в них средств, удобных для описания сложных систем, слабой поддержкой повторной применимости ранее разработанных программ и совместных разработок коллективами программистов.

В конце 1960-х был создан язык симула 67, ставший первым языком ООП. В нём объединение данных и процедур их обработки было названо объектом, а совокупность схожих объектов — классом.

Среди языков ООП, созданных вслед за Simula 67, наиболее удачным признан Smalltalk-80 (Смолток-80), разработанный в конце 1970-х.

Его успех способствовал развитию и распространению концепции ООП: в начале 1980-х на основе языка С был создан C++; в середине 1980-х на основе Pascal — Object Pascal; в середине 1990-х был создан язык Java.

Основные понятия ООП

Объект в ООП – это модель экземпляра определённого класса сущностей моделируемой системы.

Класс содержит определение данных и методов, являющихся общими для входящих в него объектов.

Он задаётся атрибутами (свойствами класса), описывающими состояние его объектов, совокупностью процедур (методов класса) и правилами доступа извне (из др. частей программы) к атрибутам и методам, определяющими интерфейс класса.

В иерархии наследования класс-потомок порождается путём добавления одного или нескольких атрибутов и\или методов к атрибутам и\или методам одного или нескольких классов-родителей.

Важнейшей особенностью ООП является возможность утаивания деталей реализации за интерфейсом класса (т.н. инкапсуляция).

Применение и перспективы развития

Преимущества ООП особенно отчётливо проявляются при создании сложных программных систем, выполняемых коллективами разработчиков:

- одни могут проектировать функциональное поведение и структуру системы;
- другие — её составляющие и способы их взаимодействия;
- трети — заниматься программной реализацией.

При этом разработчикам составляющих и занимающимся программной реализацией необязательно знать о системе в целом, а разработчикам системы в целом — о деталях её составляющих, способах их взаимодействия и программной реализации.

ООП, позволяющее повысить безопасность и производительность совместной разработки программного обеспечения коллективами программистов (за счёт

[Содержание <](#)

повторного использования программного кода и утаивания деталей реализации классов) продолжает интенсивно развиваться.

Прирастает семейство языков ООП, совершенствуются системы программирования, увеличивается число программ различного назначения (для [Интернет](#)-сервисов, систем мобильной связи и др.), разработанных на языках ООП (C++, Java и др.).

Парадигма ООП: основание продуктивного программирования

Идеи объектно-ориентированного программирования работают и в предлагаемом здесь подходе к порождению пакетов программ.

Если стиль мышления программистов, не знакомых с ООП, определяется привычкой к языкам, где ведущая роль отведена процедуре, а данные выглядят чем-то зависящим от неё, то рассуждения программиста, работающего на языке объектно-ориентированного программирования, определяются иной схемой.

В ней главная роль отведена созданию «программного объекта», понимаемого как некоторая структура данных с приписанными к ней процедурно-воплощёнными методами её обработки.

Каждый объект имеет память, предназначенную для описания его состояния. Такое описание делается посредством задания значений атрибутов объекта.

Метод, как всякая подпрограмма, может иметь входные и выходные параметры вызова, а также на время своего выполнения назначать локальные переменные, которые уничтожаются, когда метод прекращает свою работу, - в отличие от атрибутов объекта.

Метод может вызывать как другие методы своего класса, так и доступные методы объектов других классов.

Атрибуты, параметры вызова и локальные переменные методов могут принадлежать как простым типам данных, так и другим классам объектов.

Тенденции развития

Наметился серьезный поворот в переоценке роли текстовых языков.

Недостаток многих работ по автоматизации программирования предопределён стремлением к логически завершенному образу системы автоматизации программирования, в которой взаимодействие разработчика с системой осуществляется посредством текстовых сообщений.

Привычка отображать результаты работы на последовательностях бумажных или экранных страниц оказала решающее влияние на архитектуру большинства существующих систем автоматизации программирования.

Успех таких продуктов как dBase III (Ashton-Tate) во многом определяется отказом от упомянутой привычки в пользу табличных и графических форм.

Расширяющиеся исследования по визуальному программированию как основному средству программных систем искусственного интеллекта настоящего и ближайшего будущего подтверждают это.

Язык визуального программирования и языки формо-ориентированного программирования открывают новые возможности в разработке программ. Образное представление структур данных, алгоритмов, процессов обработки - все это шаги в направлении к созданию языковых средств, лучше, чем монотекстовые, приспособленных для отображения замысла разработчика программы.

[Содержание <](#)

Вполне вероятно, что текстовые фрагменты скоро будут играть не превосходящую, как теперь, роль по отношению к нетекстовым фрагментам (например, табличным).

Графические и формо-ориентированные средства хорошо приспособлены для представления сути той или иной программы или системы программирования. Графические методы многомерного отображения в статике и динамике различных точек зрения на программное обеспечение послужат основой для исследования его аспектов, определённых потребностями различных групп разработчиков.

Рост качества терминалов и их удешевление способствует тому, что в ближайшее время доминирующее положение займут интерактивные языки.

Программирование в диалоге с системой станет таким же привычным, как сегодняшнее программирование наедине с собой.

Специфика решения задач автоматизации разработки программного обеспечения такова, что соответствующие языки нуждаются в определенных, характерных для них реализациях.

В противном случае успешное решение задач автоматизации становится затруднительным. Наиболее важной особенностью среды такого языка должна быть способность поддерживать построение сложных структур данных. Такие структуры должны содержать символы без каких бы то ни было ограничений на размер или тип элемента структуры.

Необходимо также освободить разработчика от управления распределением и перераспределением памяти.

Следует позаботиться и об удобной системе настроек, посредством которой среду реализации языка можно привести в соответствие требованиям разработчика.

Повторное использование программ

В этом разделе рассматриваются два подхода к повторному использованию программ в системах автоматизированной разработки: один из них основан на методологии, получившей название САР (Computer Aided Programming - программирование с помощью ЭВМ) [[Basset](#)]; другой - на методологии, изложенной в [[Polster](#)].

САР-методология

К настоящему времени написано и отложено столько программ, что в совокупности они, по-видимому, реализуют большинство задач, встречающихся в различных видах деятельности.

Однако использовать имеющуюся программу для решения конкретной задачи часто не удается из-за того, что она была разработана хотя и для сходной, но отличающейся в деталях задачи.

Из-за этих деталей программу приходится изменять, подгоняя под новую область или условия применения.

Процесс подгонки не только трудоёмок, но и чреват трудно предсказуемыми побочными эффектами.

Зачастую он настолько непрост, что целесообразнее написать новую программу.

В результате мы становимся владельцами двух программ, в значительной степени одинаковых, но существующих как два разных объекта.

Если обе программы входят в одну систему, то заметно увеличиваются затраты на сопровождение, хотя задачная новизна системы возрастает незначительно.

Частично проблема повторного использования готовых программ в новых задачах решается с помощью внешних процедур.

Отдельно компилируемые процедуры, очевидно, полезны, но не удовлетворяют всем требованиям, так как отсутствуют средства для частичного изменения внешних процедур, позволяющего приспособить их к контексту вызывающей программы.

Сомнительна также возможность общего перестроения процедуры так, чтобы она могла удовлетворять вновь возникающие потребности, не нарушая при этом работы прежних пользователей.

Главная проблема — в том, что процедура, обладающая значительной гибкостью на этапе выполнения, во время включения в вызывающую программу не обладает никакой гибкостью (понимаемой как приспособленность к изменениям).

Методология САР основана на идее функционального программирования и реализует подход к решению задач как к описанию области в некотором абстрактном пространстве функций.

Когда поиск решения задачи сводится к работе со списком имеющихся в нашем распоряжении функций, то очевидно, что с такой работой может справиться и непрограммист.

Если задачу можно решить путем объединения некоторых функций под именем новой функции, соответствующей задаче, не принимая при этом во внимание порядок выполнения и взаимосвязи этих функций, то здесь также отсутствует программирование.

Попытка формализации такого подхода привела к понятию пространства функций.

[Содержание <](#)

Каждой конкретной задаче присущи ограничения на область определения входных данных.

При изменении ограничений возникают другие, но сходные задачи, которые могут быть решены с помощью сходных функций.

Назовем изменяющиеся ограничения степенями свободы.

Тогда можно говорить о пространстве функций, предназначенных для решения некоторого множества задач, связанных друг с другом по общим степеням свободы.

Степень свободы в конкретной задаче описывается подфункцией из некоторого множества.

Это множество может быть бесконечным и содержать константы и переменные как простейшие функции.

Тогда любая формализованная запись, которая позволяет определить функцию, ссылаясь на подфункцию для каждой степени свободы, есть, в сущности, язык решения задач, не требующий программирования.

Если бы пользователи всегда имели в своем распоряжении язык, имеющий только те степени свободы, которые необходимы для решения данной задачи, то программирование можно было бы полностью поручить автомату.

Возможно, в будущем будет создан метаязык, на котором можно будет делать описания, по которым специальный комплекс программ сможет порождать подобные языки для новых классов задач.

Но пока при решении задач требуется не только выбирать подфункции из функциональных пространств (этот процесс может быть автоматизирован), но и вручную программировать некоторые уникальные (для конкретной задачи) подфункции.

САР-система обеспечивает процесс удобного и эффективного объединения автоматически создаваемого текста программы с написанными вручную фрагментами. Число языков в программировании возрастает, и создание каждого языка определено желанием уменьшить затраты на описание решения некоторого класса задач.

Чтобы автоматизировать процесс создания программ для заданного класса задач, полезно сгруппировать языки на трёх уровнях (по точности достигаемого описания пространства функций): над-специфицирующие, оптимально-специфицирующие и под-специфицирующие языки.

Язык оптимально определяет функциональное пространство (а, следовательно, и связанный с ним класс задач) тогда и только тогда, когда выполняются следующие требования:

- язык изоморфен пространству функций, т.е. каждая функция описывается одним и только одним выражением языка, и в языке могут быть выражены только функции, принадлежащие данному пространству;
- степени свободы являются независимыми оптимально-специфицированными пространствами констант, переменных или функций;
- выражения языка максимально компактны.

На практике не удается достичь строгого соответствия языка перечисленным требованиям.

Тем не менее, даже квазиоптимально-специфицирующие языки достаточно хорошо приспособлены для функционального описания решения задач (вместо их программирования).

Для оптимального языка характерны спецификации задач.

Три уровня языков

Например, генераторы кода поддерживают именно оптимально-специфицирующие языки.

Под-специфицирующий язык аналогичен оптимально-специфицирующему за исключением того, что между выражениями языка и задаваемыми функциями существует отношение "один ко многим".

Назначение такого языка состоит в получении первого приближения к конечной функции, решающей поставленную задачу, в виде программы на оптимальном или над-специфицирующем языке, в результате чего экономится время на разработку прикладной программы.

В над-специфицирующем языке отношение выражений к реализуемым функциям определяется как "много к одному", а требования 2 и 3 не выполняются.

Над-специфицирующие языки вездесущи.

В частности, универсальные языки являются над-специфицирующими по отношению к функциональным пространствам большинства классов задач.

В то же время, для некоторых ограниченных классов такой язык может быть оптимальным, чем, собственно, и оправдывается его разработка.

Даже Ассемблер в отношении пространства аппаратно-реализованных функций является оптимально-специфицирующим, а для уровня микропрограмм - это под-специфицирующий язык.

Суммируя роль языков различных уровней в решении задач, можно сказать, что в любом случае, когда используемое пространство функций описывается оптимально-специфицирующим языком, программирование можно поручить комплексу компьютерных программ.

База знаний, основанная на фреймах

Фрейм в этом контексте САР-системы - это функционал, множество значений которого представлено пространством функций.

Фрейм реализуется в виде файла, содержащего смесь текста на некотором языке программирования и метакоманд, распознаваемых САР-препроцессором.

Эта смесь называется текстом фрейма.

Фрейм может содержать другие фреймы.

Обращение к фрейму (с целью его копирования или контекстной замены) вызывает работу не только с указанным фреймом, но и со всем деревом иерархии, корнем которого он является.

Главную роль в процессе автоматизации сборки программ из фреймов играют метакоманды. Их число невелико.

Команда COPY позволяет включать в текст заданный фрейм вместе со всеми подчиненными ему фреймами.

Команда SELECT позволяет выбирать один фрейм из заданного набора (аналогично предложению CASE во многих языках программирования).

Команда BREAK определяет "точки разрыва" в тексте фрейма.

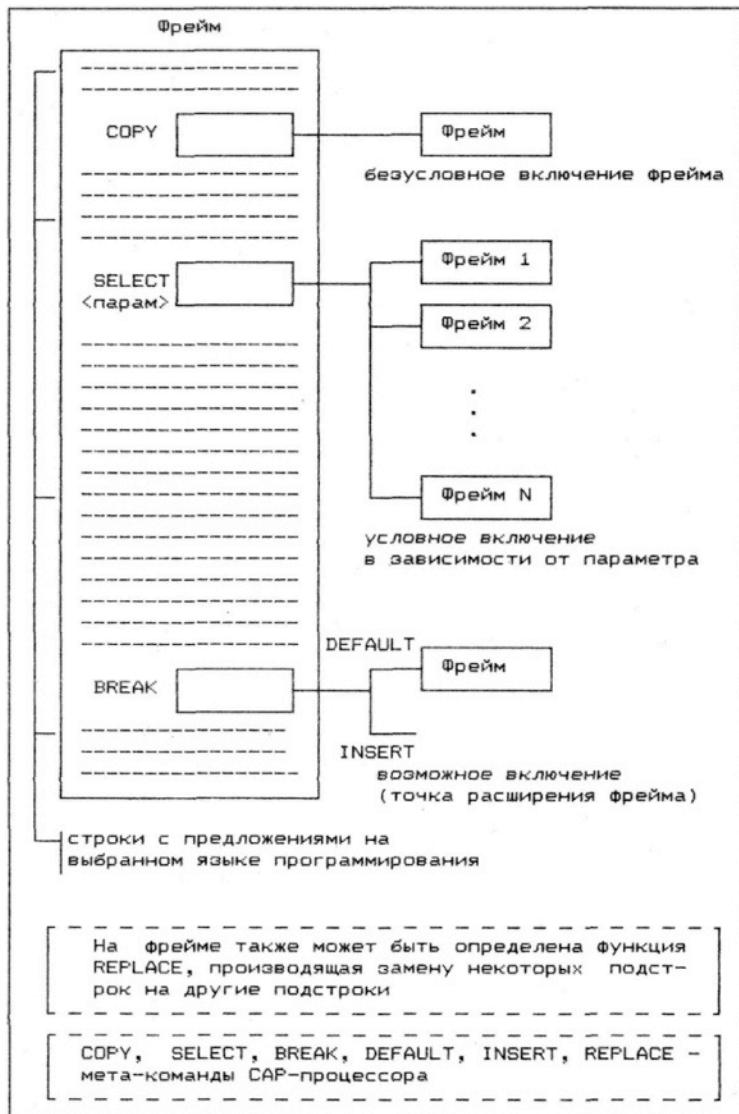
[Содержание <](#)

Эти точки помечают места, куда может быть вставлен (с помощью команды INSERT) другой фрейм, дополняющий и/или замещающий фрейм, присутствующий по умолчанию (определенный заранее посредством команды DEFAULT).

Команда REPLACE позволяет заменять одни строки текста на другие, которые требуются в конкретной программе.

Замена производится на всю глубину иерархии фреймов.

Структура фрейма в САР-системе



Параметры, используемые в метакомандах САР, ассоциируются со степенями свободы функционального пространства.

Все множество программ, тексты которых можно получить из фрейма, задавая различные комбинации параметров метакоманд, можно рассматривать как возможную реализацию множества функций соответствующего пространства.

Спецификацией программы в САР-системе является фрейм, создаваемый пользователем (так называемый SPC-фрейм). Он управляет общим процессом построения текста программы из её фрейм-компонент. Любой SPC-фрейм может быть включён в базу знаний для использования в дальнейших разработках.

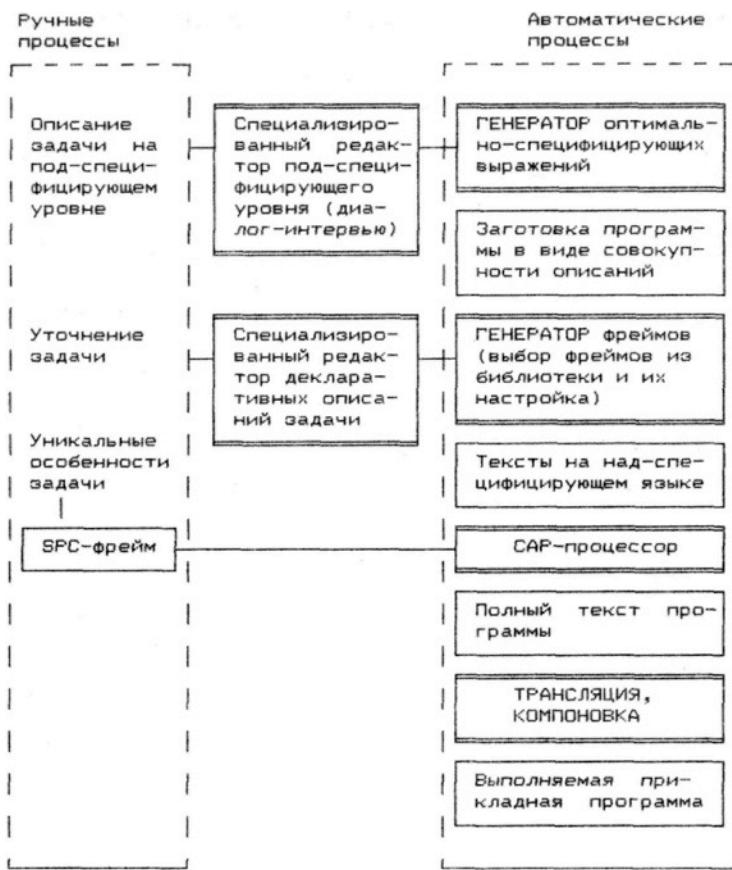
Уровень оптимального специфицирования задач в САР-системе обеспечивается библиотеками фреймов-

шаблонов. Шаблон представляет собой SPC-фрейм, в котором в виде списка собраны имена всех степеней свободы, присущих классу задач.

Фиксируя степени свободы различными способами, можно получать решения разных задач этого класса.

В отличие от внешних процедур, фреймы обладают большей гибкостью благодаря наличию точек BREAK и предложений SELECT, позволяющих иметь "открытые" (зарезервированные) степени свободы, которые при необходимости могут дополняться новыми вариантами решений.

Разработка программ в САР-системе



Существуют классы задач, в которых степени свободы функциональных пространств могут изменяться в таких пределах, что таким классам невозможно поставить в соответствие фреймы-шаблоны.

В качестве примера можно привести задачи поддержки интерфейсов с экраном, клавиатурой, печатающим устройством.

В этой ситуации целесообразно использовать средства под-специфицирующего уровня - фреймы-генераторы, которые порождают тексты новых фреймов, имеющих открытые для программирования степени свободы.

В дальнейшем эти степени свободы могут быть определены в SPC-фрейме на оптимальном или над-специфицирующем уровне (посредством языка программирования).

Развитие САР предполагает возможность создания SPC-фрейма с помощью инструментальных средств под-специфицирующего уровня.

Средства САР могут поддерживать процесс преобразования спецификаций от под-специфицирующего уровня (через оптимально-специфицирующий) до над-специфицирующего с помощью специализированных редакторов.

В числе примечательных достоинств САР - возможность накопления улучшений и модификаций компонент - стандартных фреймов и генераторов фреймов.

Нетрудно сделать так, чтобы область действия новой версии фрейма (функциональное пространство) включала (как подмножество) область действия старой версии. Для этого предусмотрен параметр-идентификатор версии, управляющий предложением SELECT.

Запоминание фрагментов кода программ для частных приложений в отдельном месте, независимо от стандартных фреймов, резко сжимает объем программной документации, снижает затраты на сопровождение программ.

Генерация частичных систем

Повторное использование программ предполагает проектирование и реализацию систем, рассчитанных на часто возникающие задачи.

Типичными примерами служат операционные системы, компиляторы, системы управления базами данных и пакеты статистической обработки данных.

Для конкретного приложения часто бывает достаточно небольшого подмножества функций, обеспечиваемых системой.

При этом использование всей системы в лучшем случае расточительно, а в худшем - невозможно.

Например, из-за её недостаточной эффективности или ограниченности ресурсов, которыми располагает пользователь.

Чтобы избежать таких трудностей или хотя бы сократить их, желательно вместо общей системы использовать её версии, обеспечивающие только те функции, которые требуются конкретному приложению.

Генерация версии полной программы



В подходе, предложенном в [Polster], построение частичной системы предполагает два этапа работ:

- выбор тех частей полной системы, которые реализуют интересующие нас задачи, и объединение их в частичную систему;
- замещение системных параметров в программе, представляющей частичную систему.

В итоге можно получить версию исходной программы.

Сборка версии программы выполняется путём генерации в соответствии со схемой, изображённой на рисунке.

Формальная модель генерации версий программ основана на понятии В-программы, которая состоит из двух типов блоков: подцепочек исходной программы и подстановок.

Генерация версий программ в качестве основного процесса включает обход дерева и соединение цепочек,

соответствующих вершинам маршрута.

Эта модель была, в частности, проверена применительно к генерации частичных систем системы управления базами данных.

Генераторы программ

Широкое распространение получили генераторы приложений и генераторы кодов [[Horowitz, Kemper at al.](#)].

К первым относятся такие продукты, как NOMAD2 (National CSS), FOCUS (Information Builders, Inc.), RAMIS (Mathematica Products Group, Inc.), dBASE II, dBASE III (Ashton-Tate), каждый из которых состоит из СУБД, генератора отчётов, языка запросов базы данных, графического пакета и специального программного обеспечения.

Генераторы кодов предназначены для производства отторгаемого программного кода, который затем может использоваться независимо от генератора.

Генераторы кодов обычно представляют собой диалоговые системы, ориентированные на спецификацию программ.

Диалоговое спецификации выглядит, как процесс уточнения некоторого прототипа, хранящегося в базе данных генератора и представляющего собой начальное приближение к искомой программе.

Растущий спрос на генераторы сделал актуальной проблему создания средств их автоматизированной разработки.

Генераторы приложений представляют собой специализированные программные системы, ориентированные на решение задач данной предметной области.

Общим свойством всех генераторов приложений является то, что они строятся на основе специально выбранной (или разработанной) СУБД, имеют удобные для пользователя средства взаимодействия, графические средства, генераторы отчётов и специальное программное обеспечение, определяющее прикладной облик генератора.

Повышение производительности, обеспечиваемое за счёт применения генераторов приложений, различно (в зависимости от характера приложений и подготовленности разработчиков приложений), но во всех случаях оно оценивается как (5-10)-кратное.

Генераторы приложений

ADF

Особую популярность среди непрофессиональных программистов, занимающихся разработкой приложений, приобрел ADF (IBM).

Он построен и работает на основе СУБД IMS.

Все приложения, разработанные посредством ADF, имеют общую структуру и представляют собой множество программных модулей, так объединённых разработчиком, чтобы они наилучшим образом выполняли задачи конкретного приложения.

Модули содержат средства управления диалогом, доступа к данным, реализации прикладной логики и управления взаимодействием с вышестоящими модулями.

Язык ADF поддерживает также недиалоговый и пакетный режимы работы.

Объединение модулей реализуется с помощью генератора правил, работа с которым выполняется посредством простого англо-подобного языка.

ДИЭКС

К генераторам приложений относится и система ДИЭКС [[Барышников, Ильин В.Д. и др.](#)], предназначенная для решения задач обработки результатов эксперимента и их графической интерпретации.

[Содержание <](#)

Эта система стала отправным результатом для разработки ИГЕН-генераторов [[Ильин В.Д., 1986, 1987](#); [Ильин В.Д., Куров и др. 1987](#); [Ильин В.Д., Мартянов](#)].

Поскольку объектами автоматизированного конструирования, изучаемыми в этой книге, являются генераторы приложений, рассмотрим подробнее их общую характеристику, воспользовавшись материалами обзора [[Horowitz, Kemper at al.](#)].

Генераторы приложений (ГП) - это системы программного обеспечения, созданные первоначально для поддержки разработок приложений, для которых характерны интенсивные потоки данных.

Генераторы обеспечивают язык "программирования" высокого уровня с удобным для пользователя синтаксисом.

Поэтому ГП могут быть легко использованы конечным пользователем, не имеющим программистских навыков.

Традиционный процесс создания программного обеспечения обычно рассматривается как последовательность следующих шагов:

- спецификации системы,
- проектирования архитектуры,
- детального проектирования,
- кодирования,
- тестирования и отладки,
- разработки документации,
- сопровождения.

Применяя ГП, пользователь начинает с того, что специфицирует краткий образец требуемой прикладной системы и затем постепенно расширяет и модифицирует этот образец, пока тот не начинает удовлетворять всем заданным требованиям.

Такой подход имеет целый ряд преимуществ.

Во-первых, отсутствует этап кодирования в обычном смысле.

Спецификация постепенно превращается в программу.

Во-вторых, когда нужно внести изменения, программист возвращается к спецификации, которая имеет более высокий уровень общности (в сравнении с программой на обычном языке программирования).

Упрощаются этапы тестирования и сопровождения.

В-третьих, облегчается процесс составления документации, поскольку спецификация относительно легко читаема.

В-четвертых, языком столь высокого уровня могут легко овладеть пользователи, обладающие небольшим опытом программирования.

Есть основание утверждать, что применение ГП позволяет значительно уменьшить время разработки приложений.

ГП имеют собственные системы управления базами данных и поддерживают доступ к внешним файлам.

Обычно используется иерархическая или реляционная модель данных.

База данных, как правило, состоит из двух файлов: основного - и файла данных.

В основном файле содержится вводимые пользователем данные, описывающие конкретную базу данных (определение типов полей, взаимосвязи между различными сегментами и т.д.).

[Содержание <](#)

Основной файл обычно создается в диалоговом режиме следующим образом: система выводит на экран форму-заготовку для основного файла, а пользователь (в данном случае - администратор базы данных) должен заполнить поля.

Файл данных содержит данные, которые вводятся в базу данных и модифицируются посредством языка запросов.

Чтобы сделать эффективным доступ к данным, пользователь может указать в соответствующем основном файле ключевые слова базы данных и метод доступа, который должен применяться.

Одна из важных особенностей генераторов приложений — наличие генераторов отчетов.

Язык генератора отчетов часто характеризуется как непроцедурный.

Это, однако, не означает полного отсутствия подпрограмм.

Непроцедурный для генератора отчетов означает язык высокого уровня.

Пользователь генератора отчётов должен указать только основные шаги вычисления, но может не вникать в детали нижнего уровня (такие как представление данных или детальная последовательность вычисления).

Ему не обязательно вдаваться в разработку таких деталей нижнего уровня, как определение явной итерации над всеми записями данных и путей поиска для получения полей данных из всех записей.

В генераторе отчетов пользователь может рассчитывать на ассоциативный доступ [доступ к данным, основанный на их характеристиках (таких, как имя поля или набор возможных значений)].

Все действия, связанные с размещением отчета и приведением данных к заданному формату, тоже выполняются генератором отчётов.

В последнее время в ГП стали включать графические пакеты, которые так же, как и генераторы отчетов, взаимодействуют с системами управления базами данных.

Графический пакет в составе ГП - это особый вид генератора отчетов, формирующий отчеты в графической, а не табличной форме.

Для простоты использования синтаксис графического языка — в основном тот же, что и в языке генератора отчетов.

Главное отличие заключается в том, что пользователь должен указать, какой вид графического объекта должен использоваться для представления данных в отчете. Обычно ГП представляют графические объекты в форме гистограмм, диаграмм и графиков.

В идеале язык манипулирования базой данных должен использовать те же синтаксические структуры, что и остальные модули ГП (генератор отчётов и графический пакет).

Язык манипулирования базой данных должен обеспечивать как интерактивное, так и пакетное выполнение модификаций базы данных.

Он должен включать в себя функции для выполнения следующих операций над данными: вставки, удаления, обновления, поиска и вывода (на экран или печатающее устройство), установки ограничений, связанных с доступом.

Если пользователь хочет выполнять редактирование базы данных в интерактивном режиме, то наиболее простым способом является тот, при котором система подсказывает ему необходимые сведения для специфирования соответствующей операции.

Для этого на дисплей выводится таблица-шаблон соответствующей базы данных, а пользователь заполняет свободные поля.

[Содержание <](#)

Манипулирование данными обычно реализуется посредством меню. Это освобождает пользователя от необходимости запоминать синтаксис описания. Кроме редактирования данных, пользователь имеет возможность изменять определение базы данных.

Для этого он должен отредактировать основной файл (содержащий определение соответствующей базы данных).

Если изменение основного файла приводит к противоречию с базой данных, пользователь получает от системы подсказку, как нужно отредактировать данные, чтобы устраниить противоречие.

Многие генераторы приложений включают пакеты специального назначения, с помощью которых легко создавать приложения, имеющие интерфейс с системой управления базой данных.

В таких приложениях, как правило, также используется меню.

Сравним ГП с системами программирования.

Генераторы приложений используются в сочетании с системой управления базами данных.

Генерация приложений, как правило, не связана с применением явных итераций или рекурсий в программах.

ГП не предлагает конструкций с циклами и не обеспечивает возможностей структурирования данных или механизмов типизации.

Они рассчитаны на применение только тех типов данных, которые допустимы в базах данных, использующихся при генерации.

В общем, можно сказать, что программы, созданные с использованием ГП, - это утверждения довольно высокого уровня, описывающие требуемые результаты.

Такие утверждения могут быть сформированы сравнительно быстро, так как они лучше воспринимаются разработчиком.

С другой стороны, языки программирования общего назначения имеют большую, чем языковые средства ГП, гибкость.

Сравнение языковых возможностей ГП и традиционного компилятора

	Генератор приложений	Традиционный компилятор
Входные языки	Диалоговые, формо-ориентированные	Не диалоговые, текстовые, не формо-ориентированные
Выходные языки	Высокого уровня; обычно - более одного языка	Ассемблер или система команд целевой машины; обычно - один

ГП, как правило, обеспечивают интерфейс языка программирования общего назначения с системой управления базами данных и некоторыми командами ГП высокого уровня.

ГП и язык программирования могут рассматриваться как две отдельных компоненты. Если пользователь ГП хочет применить в своей программе команду генератора, он должен создать рабочую область в разрабатываемой программе, а затем вызвать команду генератора (в виде обращения к внешней процедуре).

Доступ к системе управления базами данных имеет только ГП.

ГП направит запрошенную вставку в рабочую область, обеспеченную пользователем. Таким образом, ГП - это система автоматизированной разработки программ, включающая средства управления базами данных с ЧТО-языковыми средствами формирования отчётов (в табличной и графической форме).

Есть два пути повышения гибкости ГП.

Один из них - встроить КАК-язык в уже существующий ГП.

Второй - расширить некоторый КАК-язык таким образом, чтобы он включал в себя возможности ГП.

Генераторы приложений и сервис-ориентированные архитектуры

Mobile Mashup Generator System

Mobile Mashup Generator System for Cooperative Applications of Different Mobile Devices

Prach Chaisatien, Korawit Prutsachainimmit, and Takehiro Tokuda

Department of Computer Science, Tokyo Institute of Technology
Meguro, Tokyo 152-8552, Japan
{prach,korawit,tokuda}@tt.cs.titech.ac.jp

Abstract. This paper presents a development and an evaluation of a mobile mashup generator system to compose mobile mashup applications and Tethered Web services on a mobile device (TeWS). With less programming efforts, our system and description language framework enables a rapid development, a reusability of working components and a delivery of new cooperative mobile mashup applications. Working components in the mashup execution are derived from a combination of existent mobile applications, JavaScript automated Web page data extractions, and RESTful Web service consumptions. The state of art generator system is evaluated with novice and expert composer groups, to validate the usability of the system and the expressibility of our Mobile Application Interface Description Language (MAIDL). Complex mashup examples are provided to demonstrate new cooperative applications of generated Web services, which enable platform-independent functionality exchange across devices via tethered HTTP communications.

Во втором десятилетии 21-го века развитие ГП всё теснее связывается с развитием [сервис-ориентированных](#) архитектур.

В частности, интенсивно развиваются ГП для производства mash-приложений [полученных путём объединения различных приложений (например, чтобы сделать доступными разнородные [информационные ресурсы](#) [\[Ильин В.Д. 2008, 2\]](#), связанные по смыслу, определённому в приложении)].

И-порождение: конструирование целевых программных систем

И-порождением названа методология автоматизированного конструирования программных систем по описанию их характеристик и условий применения.

Предложенный подход к разработке программного обеспечения основан на идее представления разработки программ как интерактивного процесса построения из конструктивных объектов с заданными характеристиками.

В отличие от автоматического синтеза [где по спецификации задачи (представляющей ее формулировку) ищется исходный текст программы на заданном языке реализации] И-порождение - это автоматизированный синтез программной системы с заданной архитектурой, выполняемый по спецификации цели (понимаемой как множество задач, разрешимых с помощью искомой системы) и условий применения (определяющих языки реализации компонент, операционную среду, аппаратные средства).

Как построено изложение

Сначала новые понятия вводятся на содержательном уровне (определяется их прагматика с указанием точек зрения).

Это позволяет ответить на вопросы: зачем вводится понятие, где предполагаем его употреблять и какое множество объектов и отношений между ними ставится в соответствие введенному понятию?

Затем дается формализованное представление понятия (модель понятия).

Об определении понятий

Модель понятия понимается как формализованно представленная составляющая системы знаний.

Модель изобретается как конструктивный объект, рассчитанный на взаимодействие с моделями других понятий из интересующей нас предметной области (понимаемой как система понятий, введенная для изучения некоторой проблемы).

Модели понятий и предметной области представлены так, чтобы уменьшить возможность их неоднозначного толкования и сделать эффективной замену работы с понятиями формализованным манипулированием их моделями в процессе человеко-автоматного взаимодействия.

Формализованное представление понимается здесь шире, чем математическое. Там, где это целесообразно и возможно, формализованная модель строится с применением математических понятий.

О содержании

Раздел, посвящённый теоретическим основам И-порождения, — основной.

В нем введены определения новых понятий (*задачный конструктивный объект, простая и составная задачи, задачная область и задачный граф, разрешающая структура* и др.); дано описание И-порождения как индуктивного построения множества задачных конструктивных объектов.

[Содержание <](#)

В разделе, содержащем описание системы И-порождения, вводятся понятия целевых систем (с определением их типов), целевых знаний (интеллектуального базиса И-порождения целевых систем) и инфа (модели целевой системы, определяющей её поведение в операционной среде).

Раздел, посвящённый воплощению И-порождения, содержит описание подхода к реализации, в основу которого положено табс-представление целевых систем.

В этом разделе ведены понятия табса, табс-представления задачных конструктивных объектов (названных р-объектами), табс-представления целевых систем и понятие Т-инфо (как табс-ориентированного инфа).

Важным для реализации процесса обработки задачных знаний в системе И-порождения является табс-представление целевых знаний.

Правила поведения системы И-порождения при конструировании целевых систем реализуются как правила табс-навигации.

Концептуальные основы И-порождения

И-порождение рассматриваем как индуктивное построение множества задачных конструктивных объектов.

Вводим понятия задачный конструктивный объект (р-объект), И-порождение, простая задача, составная задача, задачный граф, разрешающая структура на задачном графе, поиск разрешающих структур, пространство р-объектов, составляющие понятийное ядро И-порождения.

Объект исследований

Объектом исследований является процесс разработки программных систем с заданными функциональными характеристиками и потребительскими свойствами. Определим состав задач, решить которые необходимо, чтобы получить такую программную систему, как генератор приложений.

Разработка генератора приложений требует решения комплекса задач, завершающихся созданием:

1. языка взаимодействия с системой;
2. модели предметной области, на которой интерпретируются сообщения, представленные на языке взаимодействия с системой;
3. интерпретатора сообщений;
4. интерфейса с пользователем;
5. системы управления базами данных;
6. процессора прикладного ввода-вывода (выполняющего ввод с контролем, формирование отчётов и вывод во внешнюю по отношению к базе данных среду);
7. функционального наполнения системы, определяющего её целевое назначение;
8. монитора системы;
9. специальной системы программирования, ориентированной на задачи заданной предметной области;
10. системы информационного обслуживания пользователя, включающей обучающую подсистему и подсистему документации.

Так как нас интересует проблема автоматизированной разработки генераторов приложений, естественным является вопрос о содержании понятия *автоматизированная разработка программных систем*.

Ответ представим в виде списка задач, из решения которых складывается процесс автоматизированной разработки.

Изобрести способ разрабатывать программные системы с помощью автомата — значит решить следующие задачи поддержки процесса разработки:

1. перевод требований заказчика на язык разработчика;
2. помочь разработчику в переводе заданных требований в описание множества понятий и отношений на этом множестве (необходимого для построения модели предметной области);
3. определение входного языка, предложения которого интерпретируются на модели предметной области;
4. реализация семейства отношений модели предметной области;
5. создание баз данных, воплощающих память программной системы;
6. построение интерпретатора входного языка;
7. построение гибкого взаимодействия с пользователем;
8. выбор языка реализации, операционной системы (под управлением которой будет работать программная система) и конфигурации аппаратных средств;
9. синтез исходного текста программы;
10. построение обучающей системы;
11. документирование системы.

Здесь рассматривается подход к решению задач, указанных в п.п. 2-11.

Проблема создания среды автоматизированной разработки программ

Разработка сложных программных систем - процесс решения так называемых слабоопределённых задач (постановка которых изменяется на основе анализа полученных результатов).

Пока нет той ясности в представлении разработки программного обеспечения (как некоторого процесса решения задач), которая бы позволила алгоритмизировать его. Полагаем, что более продуктивна комбинация исчислительного и алгоритмического подходов в сочетании с неформализованным участием человека (выполняющего роль не только держателя замысла конструируемой программной системы, но и принимающего решения, когда среда разработки не может справиться без его участия).

Анализ, синтез и порождение программ

[Анализ] :	[Найти отношение вход-выход для заданной программы]
[Синтез] :	[Найти программу по заданному отношению вход-выход]
[Порождение] :	[Найти программу по отношению вход-выход, которое может быть представлено структурой отношений из заданного множества, для каждого элемента которого существует программа-заготовка]

Характерные признаки среды порождения программ

Функциональность	Интерактивные средства формирования точки зрения разработчика. Экспериментирование с порождаемыми программами для определения соответствия свойств замыслу разработчика. Накопление знаний, необходимых для поддержки процесса порождения программ. Гибкий интерфейс с пользователем. Информационное обслуживание (сообщения, поясняющие выполняемые действия и состояние среды порождения). Средства обучения на примерах.
Языковая среда	Многоязычная (включая языки объектно-ориентированного, формо-ориентированного программирования). Расширяемость по составу языков и самих языков. Разнообразные типы данных. Поддержка построения сложных символьных структур данных. Программный код - в структурах данных. Структуры данных для представления процедур. Символьная обработка. Обработка списков. Распознавание (по образцу). Представление процедур как данных. Слияние процедура+данные. Обработка процедур как данных. Конструирование новых процедур при выполнении.
Инструментальные средства	Системы управления базами данных (в том числе реляционного типа). Интерактивные средства образного представления данных и процессов. Построители гибкого интерфейса пользователя. Поддержка отладки. Помощь в анализе структуры программы.

Оценивая состояние работ в области автоматизации разработки программного обеспечения в целом, можно заключить, что происходит поворот от разработки одноязыковых систем программирования к созданию многоязыковых сред разработки программ.

Устойчивым стало стремление к интерактивным инструментальным системам разработки программ.

Однако в известных подходах к построению сред разработки программного обеспечения объект разработки определяется недостаточно конструктивно. Отсутствует определение задачи (или программы) как конструктивного объекта, пригодного для построения конструкций в соответствии с некоторыми правилами.

В подходе, предложенном автором, объект разработки определён конструктивно, а сам процесс поэтапного перехода от описания целевого назначения разрабатываемой системы до получения её в готовом виде является процессом автоматизированного конструирования из задачных конструктивных объектов.

Разработка программной системы как задача И-порождения

Рассматривая И-порождение программных систем как задачный подход к их конструированию, будем иметь в виду следующее.

А. Процесс разработки - это последовательность решения задач построения системы: в интерактивном режиме описывая конструируемую систему, решаем цепочки задач конструирования при поддержке системы.

Б. И-порождение как задача специализации: порождение генератора приложений рассматривается как задача специализации исходного генератора (прообраза некоторого множества генераторов приложений).

При специализации на структурах задачных конструктивных объектов (прообраза) отыскиваются подструктуры (соответствующие описанию образа), выполняется специализация языков, баз данных и средств построения интерфейса с пользователем.

С точки зрения исследователя И-порождение - это интерактивный процесс пошагового формализованного описания порождаемой системы с целью построения задачных конструкций-образов на основе конструкций-прообразов.

В процессе И-порождения осуществляется поддержка пошагового переноса замысла разработчика программной системы (определяющего её функциональные характеристики и системные свойства) в описание, на основании которого строится искомая система.

Поддержка реализуется путём предъявления на каждом шаге множества альтернативных шаблонов для ввода специфицирующих описаний (с пояснениями) и пошаговой интерпретации описаний.

Идея редукции исходной задачи к структуре задач с известной программной реализацией

В программировании под порождением обычно понимают построение по заданным правилам некоторого множества объектов на основе заданного множества других объектов.

Говорят, например, о порождении цепочек из элементов алфавита.

Язык при этом рассматривают как множество всех порождённых (правильно построенных) цепочек, каждая из которых является предложением и состоит только из терминальных символов.

Насколько совпадает смысл слова "порождать", когда говорим о порождении программных систем, со смыслом этого слова, когда речь идет о построении множества предложений формализованного языка?

В основу предлагаемого подхода положена идея редукции задачи, подлежащей программной реализации, к некоторой комбинации других задач (разрешающей структуре), для каждой из которых программная реализация известна.

Исходная задача считается разрешимой, если в соответствие ей удалось поставить такую совокупность других задач, что вход исходной задачи содержит в себе вход разрешающей структуры, а выход исходной задачи содержится в выходе разрешающей структуры.

Переход от описания задачи к программе

Мир задач, с которым приходится иметь дело при информатизации различных видов деятельности, так разнообразен (в содержательном отношении), что любые попытки

[Содержание <](#)

определить жёсткий порядок в этом мире можно отнести к заранее обречённым в лучшем случае на исход "с ограниченной полезностью".

Причина - в динамической природе заданного мира, соответствующей эволюционному характеру процесса познания.

Известно, что разные прикладные задачи могут быть решены одним и тем же методом (с точки зрения специалиста по символическому моделированию произвольных объектов в человеко-машинной среде).

Выбор метода решения прикладной задачи (если методов несколько) зависит, в частности, от производительности системы, максимального размера задачи и допустимого времени ожидания результата.

Задачи, которые нас интересуют, назовём программируемыми, выделяя их из множества всех задач.

Идея классификации и накопления знаний о программируемых задачах (задачных знаний) требует для своего воплощения исследования способов представления понятия *задача* и связанных с ним понятий, рассчитанного на реализацию в s-среде.

Концепция порождения программных систем (как методологии их автоматизированного конструирования по описанию заданного облика и условий применения) основана на результатах исследований свойств программируемых задач, не зависящих от содержания этих задач.

Представление задач как конструктивных объектов в сочетании с механизмом обработки таких объектов составляют основу метода И-порождения.

Рассмотрение свойств, общих для всех программируемых задач, имеет при обсуждении И-порождения первостепенный смысл и потому предшествует формальному представлению задач как конструктивных объектов.

Задача И-порождения: подход к решению

Интересующая нас задача порождения целевых программных систем относится к программируемым задачам символического моделирования.

Идея представить разработку программной системы как процесс пошагового изменения описания ее заданного облика достаточно естественна по происхождению. В качестве предельного случая (в смысле полноты поддержки процесса превращения описания задачи в искомую программу) можно рассматривать автоматический синтез программ по спецификации задачи.

Естественно, что за все приходится платить: чем совершеннее (по уровню автоматизации) система автоматического синтеза, тем, как правило, проще класс задач, на который она рассчитана.

Существует вопрос: разве для любой задачи проще на языке спецификации описать формулировку, чем запрограммировать ее решение, не прибегая к помощи системы автоматического синтеза?

Ответ требует постановки встречных вопросов.

Кто предполагаемый пользователь системы синтеза (специалист в той предметной области, на которую ориентирована система, или кто-то иной)?

Каковы его навыки в формулировке задач?

Предполагает ли пользователь расширять систему своими силами?

Если предполагает, то какова его подготовка как разработчика программных систем?

[Содержание <](#)

В зависимости от ответов на встречные вопросы ответ на основной вопрос будет разным.

Однако есть безусловный аргумент в пользу систем автоматического синтеза: они позволяют образовать, сохранить, расширить и применить задачные знания, накопленные специалистами в различных предметных областях.

Применить для получения программ решения задач по описанию их формулировок.

Стимулируя продуктивное мышление, где точный ответ на вопрос ЧТО? предшествует ответу на вопрос КАК?, системы автоматического синтеза являются, кроме всего прочего, альтернативой неосознанному программированию.

То есть, программированию, стиля которого характеризуется преобладанием

"действия" над осознанием "цели", для достижения которой оно совершается.

Это справедливо как для разработчиков таких систем, так и для пользователей.

Другой предельный случай (по отношению к системе автоматического синтеза) — это традиционная система программирования, не рассчитанная на накопление внутри неё знаний о задачах.

При этом предполагается, что держателем таких знаний является разработчик программы.

Система же предоставляет в его распоряжение только инструментальные средства, предназначенные для записи текста решения задачи на некотором КАК-языке, трансляции этого текста, отладки создаваемой программы и т.д.

Итак, между системами автоматического синтеза и традиционными системами программирования существует огромный разрыв в степени ориентации на автоматизированное использование задачных знаний при разработке программ.

Из этого не следует, что один тип систем имеет безусловные преимущества по отношению к другому.

Системы каждого типа имеют свое предназначение.

Основное достоинство систем автоматического синтеза в том, что они позволяют аккумулировать задачные знания, рассчитанные на создание программ.

Представим, что задачные знания, накопленные человечеством в основных предметных областях, воплощены в системах автоматического синтеза программ. Сделаем еще один воображаемый шаг.

Представим, что каждая система автоматического синтеза расширяема и доступна.

Теперь осталось завершить идиллическую картину: профессионалы в области автоматизации программирования разрабатывают новые и расширяют уже существующие системы автоматического синтеза, а пользователи применяют эти системы.

Заметим, что любой проблемно-ориентированный КАК-язык, начиная от незабвенных Фортрана и Алгола, конечно же, содержит средства накопления описаний программно воплощенных методов решения задач (в виде библиотек функций, подпрограмм, процедур).

Но при этом проблема использования библиотек должна быть решена программистом (система программирования не поможет ему даже в выборе элементов из библиотек).

Здесь нет возможности поставить рядом с описанием решения задачи её формулировку и условия применения программы.

Задачные знания здесь отсутствуют.

Присутствует только их процедурная составляющая.

[Содержание <](#)

Какие задачи (по постановке), для каких условий применения целесообразно программировать, используя те или иные элементы библиотек?

Эта часть знаний в автоматизированном виде традиционной системой программирования не поддерживается.

Системы автоматического синтеза программ, не рассчитанные на расширение, предоставляют пользователю возможность специфицировать задачи на языке, относящемся к ЧТО-языкам.

КАК-языков такие системы не предлагают.

Для расширяемых систем характерен следующий набор языков: ЧТО-язык для описания задания на синтез; ЧТО-язык описания модели предметной области, на которой должны интерпретироваться запросы на синтез, и, как минимум, один КАК-язык - для создания заготовок программных элементов, используемых при синтезе.

Некоторые считают удобным существование в системе единого ЧТО/КАК-языка, позволяющего выполнять все перечисленные виды работ.

Например, язык Утопист (система ПРИЗ [[Тыугу](#)]) по нашей классификации относится к ЧТО/КАК языкам.

Позволяя программировать алгоритмы решения вычислительных задач (КАК-составляющая языка), он, кроме того, позволяет создавать описания вычислительных моделей (ЧТО-составляющая языка) для того, чтобы там, где это возможно, вместо программирования алгоритма сделать системе запрос на сборку фрагмента программы из заранее заготовленных программных элементов, используя так называемый "оператор задачи" (ЧТО-составляющая).

Создав запас вычислительных моделей и обеспечивающих их процедур, разработчики системы предоставляют возможность пользователям (для определенного класса вычислительных задач) обходиться запросом на языке предметной области, вместо того, чтобы, например, явно указывать обращения к подпрограммам.

Конечно, если созданная пользователем системы ПРИЗ вычислительная модель не обеспечена необходимыми процедурами ("модулями"), то их ему предстоит создать. Причем, сделать это он должен, как квалифицированный программист (в этой работе система ему не помогает).

S-моделирование задач

Представление связей между понятиями в виде разрешимых задач — необходимое условие построения количественных s-моделей систем понятий. Существует огромное число разрешимых задач, запрограммированных на различных языках и реализованных в составе системных и прикладных комплексов.

Какие это задачи, к каким классам отнесены? Где можно найти сведения об их формулировках, методах, алгоритмах и тестовых примерах? Как использовать этот арсенал при разработке программ?

Идея автоматизации программирования заключается в том, чтобы использовать ранее полученные результаты в последующих разработках. Как должны быть представлены эти ранее полученные результаты, чтобы служить основой автоматизированной разработки?

Уже на начальных этапах автоматизации программирования вместе с языками и трансляторами стали создавать и различные библиотеки программ.

Что требуется для объединения библиотек и тех программ, которые успешно работают, но не входят ни в какие библиотеки? Какой должна быть система знаний о программируемых задачах, чтобы служить основанием для конструирования программных систем?

Один из возможных ответов на этот вопрос был предложен в [[Ильин В.Д. 1989, 1](#), [Ильин В.Д. 1989, 2](#)] применительно к порождению программ. В [[Ilyin V.D.](#)] этот подход был применен в методологии конструирования параллельных программ.

Задачный конструктивный объект (s-задача)

□ **S-задача** – это четвёрка {Formul, Rulsys, Alg, Prog}, где Formul – постановка задачи; Rulsys – множество систем обязательных и ориентирующих правил решения задачи [[Ильин А.В., Ильин В.Д. 2004, 2005](#)] поставленных в соответствие Formul; Alg – объединение множеств алгоритмов, каждое из которых соответствует одному элементу из Rulsys; Prog – объединение множеств программ, каждое из которых поставлено в соответствие одному из элементов Alg.

Постановка задачи Formul – это пара {Mem, Rel}, где Mem – множество понятий задачи, на котором задано разбиение $\text{Mem} = \text{Inp}^* \text{Out}$ ($\text{Inp}^* \text{Out} = 0$) и совокупность Rel связей между понятиями, определяющая бинарное отношение $\text{Rel} \subset \text{Inp}^* \text{Out}$.

Множество Mem назовём также памятью задачи, а Inp и Out – ее входом и выходом, значения которых предполагается соответственно задавать и искать □.

Для каждого элемента из Rulsys, Alg и Prog задано описание применения.

Описания применения элементов Rulsys включают спецификацию типа решателя задачи (автономная s-машина, сетевая коопeração s-машин, коопेrация человек – s-машина и др.); требование к информационной безопасности и др.

Описания применения элементов из Alg включают данные о допустимых режимах работы решателя задачи (автоматический локальный, автоматический распределенный, интерактивный локальный и др.), о требованиях к полученному результату и др.

Описания применения программ включают данные о языках реализации, операционных системах и др.

[Содержание <](#)

◊ Каждая программа сопровождается ссылками на наборы тестовых примеров [[Ильин А.В. 2007](#)]. ◊

В общем случае множества Rulsys, Alg и Prog могут быть пустыми: числа их элементов зависят от степени изученности задачи.

□ **S-алгоритм** — система правил решения задачи (соответствующая одному из элементов Rulsys), позволяющая за конечное число шагов поставить в однозначное соответствие заданному набору данных, принадлежащему Inp, результирующий набор, принадлежащий Out.

Выполнение s-алгоритма состоит из:

распознавания набора входных данных (определения его принадлежности множеству Inp): если набор принадлежит Inp, то – переход к п.2; в противном случае – стоп;
интерпретации набора из Inp (получение результирующего набора данных, принадлежащего Out);

записи результирующего набора данных в заданную область памяти;

стоп. □

[Далее — алгоритм (сравните с [определением из Webopedia](#))].

□ **S-программа** - реализованный (на языке программирования высокого уровня, машинно-ориентированном языке и/или в системе машинных команд) s-алгоритм, представленный в форме сообщения, определяющего поведение s-машинного решателя задачи с заданными свойствами. Существует в символьном, кодовом и сигнальном воплощениях, связанных отношениями трансляции. □

[Далее — программа (сравните с определением [program](#) - в Webopedia)].

□ **Спецификация spec s-задачи** – это пара (Formul, as), где as – описание применения. □

⊗ Пример [TSM](#)-записи задачи линейного программирования.

Вход задачи Inp = {матрица $a[i=1\dots m, j=1\dots n]$ коэффициентов ограничений, вектор $b[i=1\dots m]$ правых частей ограничений, вектор $c[j=1\dots n]$ коэффициентов целевой функции};

Выход Out = {искомый вектор $x[\max; j=1\dots n]$ };

Правило rul максимизации по $x[j=1\dots n]$ целевой функции $c[j=1\dots n]*x[j=1\dots n]$ при ограничениях $a[i=1\dots m, j=1\dots n]*x[j=1\dots n] \leq b[i=1\dots m]$ и $x[j=1\dots n] \geq 0$ имеет следующий вид:

$\max [x[j=1\dots n]]:: (a[i=1\dots m, j=1\dots n]*x[j=1\dots n] \leq b[i=1\dots m]):: x[j=1\dots n] \geq 0] (c[j=1\dots n]*x[j=1\dots n]). \otimes$

Связи по памяти между s-задачами

Связи по памяти между s-задачами определяются тремя типами функций, каждая из которых является функцией двух аргументов и позволяет поставить в соответствие паре s-задач некоторую третью s-задачу, образованную из этой пары.

□ **S-задача a связана с s-задачей b по памяти**, если существует хотя бы одна пара элементов {elem Mem[a], elemMem[b]}, принадлежащих памяти Mem[a] s-задачи a и памяти Mem[b] s-задачи b, относительно которой определено общее означивание (элементы имеют одно и то же множество значений). Пусть S и H – множества s-задач и $D \leq S^*S$. Если каждой паре $(s[i], s[j])$ элементов из D ставится в соответствие

[Содержание <](#)

определенный элемент из H , то будем говорить, что задана функция связи по памяти $h = \text{conn}(s[i], s[j])$.

При этом D будем называть областью определения функции conn и обозначать $D[\text{conn}]$. Множество $R = \{h: \text{elemH}; h = \text{conn}(s[i], s[j]); s[i]: \text{elemD}[h]; s[j]: \text{elemD}[h]\}$ будем называть областью значений функции conn . \square

Тип связи зависит от содержимого пересечения по памяти: составлена ли связь из элементов выхода одной и входа другой задачи; из элементов выходов задач или из элементов их входов; или же связь получена путем комбинации предыдущих способов.

Будем обозначать функцию связи по памяти типа вход-вход через $\text{conn}[x]$, выход-вход – через $\text{conn}[ux]$ и выход-выход – через $\text{conn}[y]$.

Родовые связи между s-задачами

S-задача может быть прообразом некоторого непустого множества s-задач или образом некоторого прообраза; или быть одновременно и образом какой-то одной s-задачи, и прообразом некоторого множества других s-задач.

Предусмотрены следующие типы родовых связей между s-задачами:

- s-(специализация задачи) — указание на s-задачу, частную по отношению к исходной;
- s-(обобщение задачи) — указание на s-задачу, которая служит обобщением исходной.

Конструирование s-задачи

S-(конструирование задачи) реализуется посредством связи по памяти между задачами.

Элементарная задачная конструкция — задачная пара.

Из задачных пар можно построить более сложную задачную конструкцию, если рассматривать их как задачные элементы. Любая задачная конструкция, в свою очередь, может быть использована как составляющая еще более сложной задачной конструкции.

Конкретизация s-задачи

- **S-(конкретизация задачи)** — это переход (формулировка → система обязательных и ориентирующих правил решения задачи → алгоритм → заготовка программы → программа). \square

Для s-задач, имеющих пустое множество программ ($\text{Prog}=0$), конкретизация завершается выбором или разработкой алгоритма. Если и $\text{Alg}=0$, s-задача может быть использована в s-(конструировании задач), но не может быть конкретизирована.

Атомарная s-задача

- **S-задача называется атомарной**, если её формулировка не представлена в виде структуры, заданной на некотором множестве формулировок других s-задач. \square

Будем также говорить об атомарной s-задаче как о простой s-задаче.

Простая задача (с точки зрения построителя задачных конструкций) не наделена внутренней структурой и потому не подлежит декомпозиции.

Каждая задачная конструкция может быть объявлена некоторой новой задачей. В свою очередь, эти новые задачи вместе с атомарными могут быть применены при конструировании задач.

Система знаний об s-задачах

□ **Система рS знаний о задачных конструктивных объектах** (называемых также р-объектами) – это триада $\langle pA, Ing, intr \rangle$, где pA – задачная область, Ing – язык спецификации р-объектов, $intr$ – интерпретатор спецификаций искомых р-объектов на pA . □

Модель задачной области

Пусть P – множество всех р-объектов, а $A \subset P$ – его непустое подмножество. При этом в A (содержащем не менее двух элементов) не существует ни одного элемента, который не был бы связан по памяти хотя бы с одним элементом из A .

□ **S-модель ра задачной области ра** – это р-объект, который задаётся парой \langle память $mem[A]$ множества задач A задачной области $pA\rangle$, \langle семейство $rul(mem[A])$ связей, заданных на $mem[A]$ \rangle .

Непустое множество $mem[A]$ элементов памяти разбито на три подмножества: входов $inp[A]$ задач, выходов $out[A]$ задач и подмножество $or[A]$, каждый из элементов которого является и входом, и выходом некоторых задач.

Любое одно из этих подмножеств может быть пустым; могут быть одновременно пустыми $inp[A]$ и $out[A]$. □

В отличие от памяти задачи, состоящей из входа и выхода, память задачной области содержит подмножество or элементов памяти, каждый из которых может быть или задан (как входной), или вычислен (как выходной).

Будем называть такие элементы памяти обратимыми, а or – подмножеством обратимых элементов.

Подмножество inp будем называть подмножеством задаваемых, а подмножество out подмножеством вычисляемых элементов.

Задачная область pA служит s-моделью [[Ильин А.В. 2007](#)], на которой интерпретируются спецификации искомых задач, составленные на языке Ing .

Интерпретация и разрешающая структура

◊ Интерпретация заключается в постановке в соответствие некоторому подмножеству (или паре подмножеств) памяти $mem[A]$ некоторой подобласти задачной области pA , названной разрешающей структурой.

Интерпретация спецификации искомого р-объекта на pA – конструктивное доказательство существования разрешающей структуры. ◊

Задачные графы

□ Задачный граф служит представлением задачной области, рассчитанным на реализацию процесса р-конструирования и формализацию задачных знаний.

Множество вершин графа составлено из задачных объектов.

[Содержание <](#)

Оно называется задачным базисом графа и обозначается p-basis.

Ребро задачного графа — это пара вершин с непустым пересечением по памяти. Нагрузка ребра определяется множеством всех пар элементов памяти, входящих в это пересечение. Каждая вершина графа имеет память. Память вершины — это память задачи (или задачной области), которую представляет вершина. □

□ **Составная задача comp** – подобласть задачной области pA, которая содержит не менее двух элементов из множества задач A и на памяти которой задано разбиение: $\text{mem}[\text{comp}] = \text{inp}[\text{comp}] \cup \text{out}[\text{comp}]$; $\text{inp}[\text{comp}] \cap \text{out}[\text{comp}] = \emptyset$, определяющее вход $\text{inp}[\text{comp}]$ и выход $\text{out}[\text{comp}]$ составной задачи.

Составной задаче поставлен в соответствие ориентированный граф, вершинами которого являются задачи. Каждая вершина помечена именем задачи.

Ребра графа — это пары задач с непустыми пересечениями по памяти. □

◊ Составная задача может быть построена путём последовательного применения функций связи по памяти. ◊

Типы задачных графов

В зависимости от состава вершин определим следующие типы задачных графов:

- в U-графе имеется множество вершин только из простых задач;
- в C-графе хотя бы одна вершина представлена составной задачей и нет вершин, представляющих собой задачную область;
- в G-графе — не менее одной вершины представлено задачной областью (остальные могут быть простыми и составными задачами).

U-граф

□ Связный граф с непустым множеством рёбер и задачным базисом p-basis, все элементы которого являются простыми задачами, называется U-графом и обозначается U-graph: $\text{U-graph} = (\text{p-basis}, \text{set}[ver])$, где $\text{set}[ver]$ – множество рёбер задачного графа.

Объединение памяти задачных вершин, составляющих базис, называется памятью U-графа и обозначается mem [U-graph].

На памяти U-графа задано разбиение подмножествами:

- $\text{Giv}[\text{U-graph}]$ задаваемых элементов памяти (называется подмножество входных элементов);
- $\text{Comput}[\text{U-graph}]$ вычисляемых элементов памяти (называется подмножество выходных элементов);
- $\text{Or}[\text{U-graph}]$ обратимых элементов памяти (определяется разностью $\text{mem}[\text{U-graph}] \setminus (\text{Giv}[\text{U-graph}] \cup \text{Comput}[\text{U-graph}])$). □

C-граф

□ Связный граф с непустым множеством рёбер и задачным базисом, в составе которого есть хотя бы одна составная задача и нет задачных областей, называется C-графом и обозначается C-graph. □

G-граф

□ Связный граф с непустым множеством рёбер и задачным базисом, в составе которого есть хотя бы одна задачная область, называется G-графом и обозначается G-graph.

Память С-графа и память G-графа обозначаются и определяются аналогично памяти U-графа. □

G-графы — средство формализации знаний о р-объектах

Система знаний об s-задачах обеспечивает процессы р-(специализации, конкретизации и конструирования).

◊ Возможность существования в задачном графе одного или нескольких узлов, являющихся задачными областями, имеет принципиальное значение для формализации задачных знаний. ◊

Конкретным воплощением задачной области может быть граф любого типа (U-, С- или G-граф).

Тот факт, что G-граф может замещать задачный узел, открывает логически неограниченные возможности для усложнения задачной области. Она может быть представлена, в частности, посредством G-графа, базис которого состоит только из вершин, представленных G-графами.

Исчисление s-задач

□ Объединение множеств базовых задачных объектов и сконструированных объектов обозначается через Р и называется множеством р-объектов. □

□ Совокупность пространств, построенных на подмножествах множества Р, называется миром р-объектов (или миром задач). □

Процесс работы с задачными объектами реализуется по принципу «от общего к частному».

В системе задачного конструирования существуют пространства задачных конструктивных объектов, одни из которых содержат представленные в самом общем виде объекты, другие – объекты, полученные путем специализации объектов-прообразов.

Множество пространств р-объектов (мир р-объектов) не имеет логических ограничений на расширение.

□ **Исчисление задачных конструктивных объектов Igen** (которое для краткости называем также исчислением s-задач) — это построение некоторого множества Т конструктивных объектов (t-объектов, обозначаемых через t) на основе множества В базовых конструктивных объектов (b-объектов, обозначаемых через b) по правилам R построения t-объектов из b-объектов и ранее построенных t-объектов.

Igen — это четвёрка (Ing, B, T, R), включающая язык Ing описания объектов и конструкций из объектов, непустое множество B базовых объектов, порождаемое множество T (при этом $B^T = \emptyset$; до начала процесса задачного конструирования $T = \emptyset$) и множество R правил построения объектов. □

[Содержание <](#)

Символы $\text{conn}[x]$, $\text{conn}[yx]$, $\text{conn}[y]$ обозначают функции трёх типов, применение которых позволяет построить t-объект из пары уже существующих объектов или конструкций из объектов.

Символ pre – обозначение прообраза некоторого непустого множества объектов.

Этот символ используется в объявлениях и функциях.

Символ im – обозначение образа некоторого объекта. Этот символ употребляется (так же, как и символ pre) в объявлениях и функциях.

Множество В состоит из базовых объектов (b-объектов), являющихся объектами первого поколения. Иначе говоря, любой b-объект не имеет прообраза.

Любому b-объекту может быть поставлено в соответствие некоторое непустое множество объектов-образов $\text{set}(t[\text{im}; p(b)]) = \text{im}(b)$ (в левой части символ im является верхней погодой, а $p(b)$ — нижней, обозначающей указатель на объект-прообраз b).

Каждый объект-образ $t[\text{im}; p(b)]$ имеет единственный объект-прообраз.

Любой объект-образ может, в свою очередь, иметь образы.

В этом случае он является прообразом своих образов и образом некоторого прообраза.

Объект, множество образов которого пусто, будем называть конечным объектом.

Объекты, имеющие прообраз и непустое множество образов, будем называть промежуточными объектами.

Множество Т строится на базисе В по правилам R.

В Т могут существовать конструкции, построенные из объектов, принадлежащих В, конструкции из b-объектов и ранее построенных t-объектов, а также конструкции из t-объектов.

Определены правила построения t-объектов, в соответствии с которыми работает механизм заданного конструирования при создании конструкций из b-объектов, b- и t-объектов, а также — из t-объектов.

/ Правила построения t-объектов

rul[1]: $\text{tip}(b)=b$

Правило устанавливает, что объект $t|p(b)$ множества Т может быть построен из одного объекта b, принадлежащего множеству В (буква t обозначает некоторое имя объекта из Т; после вертикальной чёрточки — указатель на объект, из которого построена конструкция; p — символ указателя, а в скобках — имя объекта).

rul[2]: $\text{tip}(\text{im}(t))=\text{im}(t)$

Объект $t|p(\text{im}(t))$ множества Т, являющийся образом объекта t, может быть получен p-специализацией t.

rul[3]: $\text{tip}(\text{pre}(t))=\text{pre}(t)$

Объект $t|p(\text{pre}(t))$ множества Т, являющийся прообразом объекта t, может быть получен p-обобщением t.

rul[4]: $\text{tip}(\text{conc}(t))=\text{conc}(t)$

Любой объект, полученный путём p-конкретизации, может быть t-объектом.

[Содержание <](#)

Rul[5]: $\text{tIp}(\text{t}^*; \text{p}(f,s)) = \text{t}^*; \text{p}(f,s)$

Правило устанавливает, что t -объектом может быть любая допустимая конструкция $t^*; p(f,s)$, которую получаем, применяя функцию связи по памяти $\text{conn}^*(f, s)$ (то есть, $t^*; p(f,s) = \text{conn}^*(f, s)$).

Помета $*$ принимает одно из трёх значений (x , ux , y), каждое из которых обозначает определённый тип функции conn^* (имена объектов-аргументов указываются в круглых скобках сразу после символа функции conn^*).

Для пары (f, s) допустимы следующие значения:

- $(f: \text{elem } B, s: \text{elem } B);$
- $(f: \text{elem } B, s: \text{elem } T);$
- $(f: \text{elem } T, s: \text{elem } B);$
- $(f: \text{elem } T, s: \text{elem } T)$.

Конструирование разрешающих структур на задачных графах

Рассмотрим принцип действия механизма конструирования на задачном графе.

Искомая конструкция задаётся спецификацией задачи, содержащей описание её памяти, ограничений на число задачных узлов (и, если необходимо – ограничений, связанных с размером задачи, точностью результата и др.).

Заданное описание интерпретируется на задачном графе, который служит представлением интересующей конструктора задачной области.

Средством интерпретации спецификаций задач служит механизм конструирования на задачном графе.

Интерпретация на U-графе в процессе задачного конструирования заключается в постановке в соответствие подмножеству (или паре подмножеств) элементов его памяти такого подграфа, память которого находилась бы в данном отношении к введенному подмножеству (или паре подмножеств).

Интерпретации на С-графе и G-графе аналогичны интерпретации на U-графе.

- **Задача t представима на задачном графе $graph$** , если её вход $\text{inp}[t]$ содержится в подмножестве $\text{Giv}[graph] \cup \text{Or}[graph]$, а выход $\text{out}[t]$ – в подмножестве $\text{Comput}[graph] \cup \text{Or}[graph]$ памяти задачного графа; при этом существует не менее одной задачи из базиса графа, вход которой содержит в $\text{inp}[t]$ или совпадает с ним. □
- **Разрешающей структурой $solv [t]$** на графике $graph$, поставленной в соответствие некоторой задаче t , называется подграф с минимальным числом задачных вершин, на котором задача t представима. □

Интерпретация задачного узла U-графа (или С-графа) в процессе поиска разрешающей структуры заключается в соотнесении означенности входа и выхода.

Смысл этого определения поясняют правила интерпретации задачного узла:

- если полностью означен вход, то полностью означен и выход;
- если означенным полагается хотя бы один элемент выхода, то означенным полагается полностью вход.

[Содержание <](#)

Механизм построения разрешающих структур ставит в соответствие спецификации исходной задачи подграф на задачном графе путём реализации трёх типов поведения в соответствии с тремя типами запросов на конструирование.

Существование разрешающей структуры типа solv [xy]

Для заданных подмножеств x и y ($x \cap y = \emptyset$), памяти mem [t -graph], тогда существует разрешающая структура $\text{solv} [xy]$ (здесь xy - помета) с минимальным числом задачных вершин, вход которой определён посредством x , а выход — посредством y , когда найдётся подграф G , множество вершин которого включает хотя бы одну вершину с разрешимой задачей, а объединение выходов вершин подграфа G содержит подмножество y (или совпадает с ним).

Существование разрешающей структуры типа solv [x]

Для подмножества x mem -элементов, заданного на памяти mem [t -graph] задачного графа, тогда найдётся разрешающая структура $\text{solv} [x]$ (x - помета), вход которой определён подмножеством x , а выход является непустым подмножеством памяти графа, включающим максимальное число элементов, которые могут быть определены при заданном x , когда $x \wedge \text{Comput} = \emptyset$ ($\text{Comput} < \text{mem}$ [t -graph]) и найдётся хотя бы одна вершина с разрешимой задачей.

Существование разрешающей структуры типа solv [y]

Для подмножества y mem -элементов, заданного на mem [t -graph], тогда найдётся разрешающая структура $\text{solv} [y]$ с минимальным числом задачных вершин, выход которой содержит y , а вход составлен из элементов, принадлежащих Giv , когда $y \wedge \text{Giv} = \emptyset$.

Для каждого из трёх типов запросов в [Ильин В.Д. 1989, 1] получено конструктивное доказательство существования разрешающей структуры соответствующего типа. Там эти доказательства были получены применительно к задачным сетям. Схемы доказательств существования разрешающих структур на задачных графах аналогичны.

После того как найдена разрешающая структура, становится осуществимым процесс ее конкретизации в соответствии со спецификацией условий применения исходной задачи.

О тестовых примерах

Применительно к конструированию разрешающих структур на задачных графах существование набора тестовых примеров для каждого р-объекта – необходимое условие реализации предложенной технологии.

Заметим, что тестирование полученной разрешающей структуры входит в конструктивное доказательство ее существования.

◊ Нынешние библиотеки программ различного назначения (в том числе – динамически присоединяемые dll) не удовлетворяют требованиям, предъявляемым к s -модели задачного конструктивного объекта.

В частности, они не содержат ссылок на наборы тестовых примеров. Нет возможности протестировать составляющие нынешних операционных систем и работающих под их управлением приложений.

[Содержание <](#)

Правило – продавать программные продукты с заранее оговоренной возможностью доступа к наборам тестовых примеров (выложенных на сайте производителя) – заметно уменьшит число ошибок и уязвимостей на этапе разработки и увеличит вероятность их выявления после начала эксплуатации.

Известно, что составление наборов тестовых примеров требует понимания не только сути задачи, выполняемой тестируемой программой, но и допустимых условий ее применения.

Если правило – предъявлять потребителю наборы тестовых примеров – станет обязательным, изменится отношение разработчиков и к изготовлению таких наборов, и к производству программных продуктов. ♦

Прикладное значение

Конструирование разрешающих структур на задачных графах позволяет путём интерпретации интерактивно формируемой спецификации задачи получить соответствующую ей структуру разрешимых задач.

Построение множества задачных конструктивных объектов в составе системы знаний об s-задачах основано на исчислении р-объектов.

Важными нововведениями по сравнению с [Ильин В.Д. 1989, 1] является то, что задачные сети заменены задачными графиками и в s-модель р-объекта введена гиперссылка на набор тестовых примеров.

Примером применения метода построения разрешающих структур на задачных графах может служить интерактивный преобразователь ресурсов с изменяемыми правилами поведения [Ильин А.В., Ильин В.Д. 2004].

Инф: модель унифицированной s-машины

Инф (как модель унифицированной s-машины) был изобретён в 1989 и впервые представлен в монографии [[Ильин В.Д. 1989, 1](#)].

Там он был назван символным автоматом.

Позднее инф был усовершенствован и стал моделью унифицированной параллельной s-машины [[Ilyin V.D.](#)].

- **Инф** — это s-машина $\langle S, T \rangle$, где S — множество состояний; T — множество пар состояний, принадлежащих декартову произведению S^*S , задающее отношение допустимых (разрешённых) переходов. Пара $(s[i], s[j])$ принадлежит T , если из состояния $s[i]$ разрешён переход в состояние $s[j]$; $i, j=1\dots n[S]$ ($n[S]$ — число состояний). □

Инфы способны самоорганизовываться для кооперативного решения задач и могут иметь реализацию произвольного масштаба в s-среде.

Производительность инф-кооператива зависит от того, сколько и каких ресурсов ему выделено в соответствии с системой правил распределения нагрузки.

Вычислительная s-машина

- **Вычислительной называется s-машина**, если каждому $s[i]$ из множества S состояний правилами переходов поставлено в соответствие не более одного $s[j]$ ($i \# j$). Вычислением называется любая последовательность переходов состояний вычислительной s-машины. □

☼ Примерами вычислительных s-машин могут служить программа s-машины, реализующая некоторый алгоритм, и микропроцессор.

Для программы множество состояний — это множество элементов, каждый из которых представлен экземпляром входа-выхода.

Переход от одного состояния к другому алгоритмически определен последовательностью операторов программы.

Для заданного начального состояния правильная программа обеспечивает переход к искомому конечному состоянию через конечное множество промежуточных.

И сколько бы раз для фиксированного начального состояния не выполнялась программа, вычисление не должно изменяться.

Для микропроцессора множество состояний определяется множеством экземпляров содержимого его регистров.

Переход от состояния к состоянию однозначно определен выполняемой программой (содержимым регистра команд). ☼

Исчислительная s-машина

- **Исчислительной называется s-машина**, если каждому $s[i]$ из множества S состояний правилами переходов может быть поставлено в соответствие более одного $s[j]$ ($i \# j$).

Выводом называется любая последовательность переходов состояний исчислительной s-машины. □

☼ Примером исчислительной машины может служить система автоматического доказательства теорем. ☼

И-порождение целевых систем: табс-представление

Раздел содержит изложение подхода к воплощению И-порождения целевых систем, в основу которого положено табс-представление целевых систем.

Введено понятие о табсе, дано табс-представление р-объектов и целевых систем.

Введено понятие о Т-инфe как инфе с табс-памятью.

Определено табс-представление целевых знаний при реализации обработки заданных знаний.

Правила поведения системы И-порождения при конструировании целевых систем определены как правила табс-навигации.

Табс

□ Табс - это трёхмерная (многослойная) таблица, имеющая два специальных типа клеток для связи с другими трёхмерными таблицами.

Первый тип - указатель на табс (кратко - указатель), второй - вложенный табс.

Значениями типа указатель являются табс-адреса.

Табс-адрес задаётся именем табса и тремя координатами: первая - номер строки, вторая - столбца и третья - слоя.

Значениями типа вложенный являются табсы.

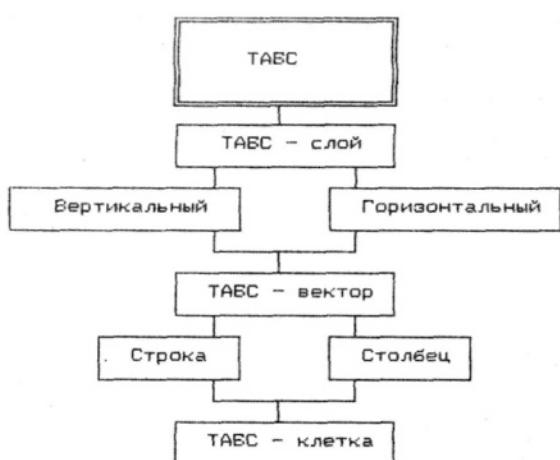
Попадание в клетку типа вложенный делает доступным вложенный в эту клетку табс. □

Об изобретении табса

Идея изобретения табса как формы для описания систем понятий и систем знаний имеет следующее происхождение: автор стремился к созданию удобной для задачного конструирования позиционной системы форм описания множеств понятий и заданных на них семейств отношений.

Искомая позиционная система должна была быть удобной и эффективной для представления на её основе всех структур данных.

Понятие о табсе



Табс-структура как форма для представления задач

□ Табс-структура - множество табсов и семейство заданных на нём отношений. Отношения реализуются посредством типов вложенный табс и указатель на табс. □

Существование типа вложенный табс освобождает от необходимости во всех случаях именовать табсы.

Возможность создавать вложенные табсы полезна и для агрегирования, и для декомпозиции.

Например, клетку с типом вложенный можно рассматривать как некоторый агрегат, представляющий фрагмент памяти целевой системы.

А сам вложенный табс при этом - толковать как декомпозицию упомянутого фрагмента.

Если же в клетке, имеющей тип вложенный, размещено имя задачи, то это означает, что задача относится к составным задачам (её память и семейство связей между

[Содержание <](#)

элементами памяти задаются описаниями, размещёнными в клетках вложенного табса).

Любой вложенный табс может, в свою очередь, иметь клетки типа *вложенный* (в частности, этот тип могут иметь все клетки).

Нетрудно представить богатство возможностей табс-структуроирования, предоставляемое типами *вложенный* и *указатель на табс*.

В системе И-порождения табс-структура используется как форма для размещения описания р-объекта: чтобы сделать удобным для человека и эффективным для выполнения автоматом поэтапный интерактивный процесс специфирования программируемой задачи.

Итак, воплощением позиционного принципа в представлении задачных знаний в системе И-порождения является табс-структура (как форма для размещения в ней описания задачи или задачного графа).

Табс как метаабстрактный тип

Пусть табс А имеет все клетки типа *вложенный табс*.

То есть, множеством значений содержимого каждой клетки служит множество табсов, любой из которых может быть вложен в эту клетку.

Поскольку любой табс является абстрактным типом, то табс А отнесём к мета-абстрактному типу.

Каждый вложенный табс также может иметь все клетки типа *вложенный*.

Легко заметить, что предела усложнению табса таким способом не существует (в данном контексте табс рассматривается как пара: структура данных плюс набор процедур, обеспечивающих связь с внешней средой и выполнение на табсе определённых работ).

Вместо привычного определения абстрактного типа (структуре данных плюс множество допустимых на ней процедур) для табса имеем: табс-структура + множество допустимых процедур.

При этом табс-структура рассматривается как множество упорядоченных трёхмерных таблиц, каждая из клеток которых имеет тип, а множество допустимых процедур - как такая их совокупность, где каждая процедура применима при определенном состоянии памяти табса, принадлежащем некоторому множеству состояний.

Поведение табса во внешней среде задаётся множеством допустимых сообщений.

Данные о нём имеет интерпретатор сообщений данного табса.

Изменение состояния табс-памяти определяется множеством допустимых процедур, применимых на памяти.

A-табс

▫ А-табс, предназначенный для порождения У-табсов (представляющих собой конкретизацию А-табса), будем называть *порождающим табсом*. ▫

▫ Множество А-табсов назовём *порождающим табс-типов*. ▫

А-табс - это метаабстрактный тип, имеющий фиксированное целевое назначение в системе порождения.

Табс-реализация отношений

Поддержка системой управления табс-базами данных (СУТБД) работы с различными структурами данных способствует реализации отношений в базе знаний.

Например, координатное размещение данных в табсах удобно для представления отношения СЛЕДОВАНИЯ.

Отношение СОДЕРЖАТЬ реализуется посредством клеток, имеющих тип *вложенный табс*.

В набор функций перемещения включены функции, разрешающие входить в клетки типа *вложенный табс* и возвращаться обратно.

Допускается, чтобы один и тот же табс существовал в качестве вложенного в нескольких клетках одного и того же или разных табсов.

Это делает возможным построение произвольных табс-реализованных графов. Табс может содержать самого себя в любой из своих клеток типа *вложенный табс*.

Например, табс Q вложен в табс P, табс P вложен в табс S, а табс S вложен в табс Q. Для реализации отношения СОДЕРЖАТЬ может использоваться, кроме вложенных табсов, аппарат ссылок на табсы.

Ссылки могут применяться для выполнения переходов в табс-базе данных (*ТБД*) посредством изменения привязки каналов (реализующих в СУТБД доступ к табс-базам данных).

Ссылки могут быть *прямыми* и *косвенными*.

- *Прямая ссылка* - это табс-адрес, к которому канал должен привязаться, если выполнен переход в клетку со ссылкой. □
- *Косвенная ссылка* - описание маршрута, который каналу следует пройти, чтобы установить требуемую связь. □

Клетка, содержащая вложенный табс или прямую ссылку, интерпретируется независимо от указанного в канале каталога табсов, определяющего доступные табсы.

Результат интерпретации косвенной ссылки зависит от текущего каталога: одна и та же косвенная ссылка может привести к установлению связи с разными табсами (в зависимости от того, какой каталог был определен в канале).

Возможность создания вложенного табса существует от момента определения типа клетки, в которой предполагается его разместить.

Существование вложенного табса начинается в момент заполнения им клетки.

Табс, на который указывает косвенная ссылка, может не содержаться в табс-базе данных в тот момент, когда ссылка заносится в клетку.

Табс, на который сделана ссылка (так же, как и вложенный табс), при копировании не размножается.

Поскольку табс может считаться вложенным в несколько клеток разных табсов, возникает вопрос о принадлежности.

Отношение ПРИНАДЛЕЖАТЬ реализуется также посредством ссылки.

Ссылка при этом не отличается по форме от той, с помощью которой реализуется отношение СОДЕРЖАТЬ.

Особенность такой ссылки - в способе ее реализации: для записи ссылки используется специальная поименованная клетка вне координатного пространства табса.

Задание соответствия: матричное и графовое

СООТВЕТСТВИЕ

Определение

$X \times Y$
 $R \subseteq X \times Y$

$(R, X, Y) - \text{соответствие между множествами } X \text{ и } Y$
 $(x[i], y[i]) \in R; x[i]Ry[i]$

Матричное представление

Пусть $n=\text{card } X$ и $m=\text{card } Y$, тогда

$(R, X, Y) \Rightarrow$

Графовое представление

$(R, X, Y) \Rightarrow$

Каждый фрагмент спецификации, несущий определенную смысловую нагрузку, получает фиксированное место в табс-структуре.

Координатное прикрепление фрагмента (привязка к формоместу) позволяет упростить работу со спецификацией в целом.

Таким образом, представив спецификацию в виде табс-структуры данных, можно обрабатывать её с помощью механизма доступа по заданным координатам.

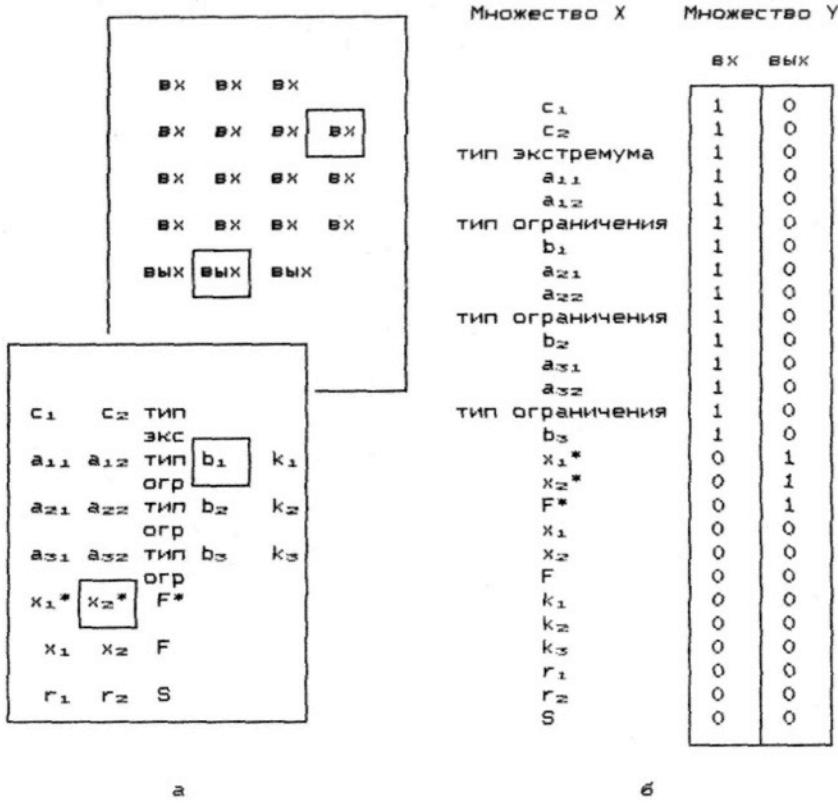
Табсы позволяют экономно представлять соответствия (вместо распространённых матричной и графовой форм). Чтобы оценить это, достаточно рассмотреть пример (см. рисунки: слева и на следующей странице).

Комбинация графовых и реляционных возможностей делает табс-строктуру удобной формоосновой для представления задачных знаний.

В заключение заметим, что такие свойства табсов, как наличие системы координат и типов клеток, позволяет строить сложные структуры табсов, фиксировать синтаксические правила разбора содержимого клеток, и, наконец, определять семантику данных, помещённых в клетки.

Всё это определяет эффективность табс-представления задач.

Задание соответствия двухслойной и однослойной таблицами



Табс-представление р-объектов

- Табс-представление - описание объекта с использованием табс-структур, обеспечивающее работу с ним как с конструктивным объектом.
-

Р-объекты, как участники процессов р- (конкретизации, специализации, замены и конструирования), имеют различные табс-представления. Каждому р-объекту соответствует структура табсов и набор процедур, применение которых допустимо на этой структуре.

Наличие внутренней системы координат даёт возможность использовать позиционный принцип при определении семантики данных, размещаемых в клетках табсов.

Важно, что клетки табсов имеют тип.

В частности, если клетка имеет тип **вложенный**, то она сама является табсом.

Если клетка имеет тип **указатель на табс**, то это означает, что связываются два табса [i-го и (i+1)-го уровней].

Табс первого уровня называется **корневым**.

Число уровней не ограничено: есть возможность построения иерархической структуры табсов.

Табс-представление задач при р-конкретизации

Процесс **р-конкретизации** состоит из четырёх этапов - f, m, t и р.

На этапе f формулируется задача.

На последующих - описываются методы решения (этап m), составляются заготовки программ (этап t) и на этапе р заготовки преобразуются в программы.

Для представления абстрактных задач на этапе f используется специальная иерархическая структура табсов.

Первый слой корневого табса содержит данные **паспорта задачи** [имя р|название р-объекта и описание точки зрения инженера задачного знания (значения атрибутов who, purp и stage); имя паспорта и его атрибуты размещены в разных клетках].

Все клетки (кроме одной) имеют тип **текст** (одна - тип **указатель**: в ней содержится ссылка на табс с формой для описания памяти задачи).

[Содержание <](#)

В клетке второго слоя табса (с теми же двумя первыми координатами) помещается ссылка на табс, хранящий форму для описания отношения formul, заданного на памяти рассматриваемой задачи.

◊ Координаты табс-клеток жёстко связаны с назначением содержимого записанного в клетки: например, в клетке с координатами (1, 1, 1) - всегда имя паспорта, а в клетке (5, 1, 1) - ссылка на табс, содержащий форму для описания памяти задачи. ◊

Итак, корневой табс имеет следующие клетки:

- `pasp[cnc]|name` (паспорт задачи);
- `sp[gen]` (принадлежность к пространству по родовому признаку);
- `sp[us]` (принадлежность к пространству по применению; для абстрактных задач `sp[us] = abstr`);
- `struc` (тип задачной структуры),
- `<form>` (форма описания).

Отличить клетки типа *текст* от типа *указатель* можно по угловым скобкам: если они есть, то клетка имеет тип *указатель*. В клетки типа *текст* записываются символьные строки - имя паспорта и его атрибуты (кроме *form*).

Например, клетка `struc` может содержать `unit` (*простая задача*), а клетка `us` - `abstr` (*абстрактная*).

В клетках `inp`, `out` размещаются данные, задающие *вход* и *выход* задачи. Каждая из этих клеток имеет тип *указатель* и адресует систему к табсу, связанному с корневым.

Отношение `formul` записывается в табс, на который есть ссылка в клетке типа *указатель* второго слоя корневого табса, содержащего имя паспорта задачи и значения его атрибутов.

◊ Зная соответствие между координатами клеток и их содержимым, а также типы клеток (которые определяют и синтаксические правила их заполнения), система порождения работает с табс-представлением задачи как с конструктивным объектом.
◊

◊ Клетки типа *текст* второго слоя табсов первого уровня содержат поясняющие сведения для каждой из клеток первого слоя.

Эти сведения не подлежит обработке интерпретатором: они используется при обучении и поддержке процессов спецификации.

Аналогичную поддержку имеют вложенные табсы и те табсы, на которые сделаны ссылки. ◊

На этапе `m` формируются табс-представления методов решения задачи.

Прообразом задачи с отношением `rel = set(formul|meth)` является задача, для которой `rel = formul` и табс-представление которой предшествует табс-представлению задачи на этапе `m` (поскольку образ наследует значения атрибутов прообраза, то в табс-представлении задачи на этапе `m` появляются указатели на ранее созданные табсы). В клетках первого слоя корневого табса содержится имя паспорта задачи (с пометой `[cnc; m]`), указатель на табс, где описана задача с тем же именем и пометой `[cnc; f]`, и указатель на табс, где содержатся данные о методах решения задачи. Первый указатель делает определёнными атрибуты паспорта задачи и описание её памяти. Второй - табс, где описаны отношения `set(formul | meth)` [здесь же записаны и данные об условиях применения].

На последующих этапах (*t* и *r*) создаются табс-представления заготовок программ, которые затем доопределяются до программ, имеющих вид исходных текстов на выбранных языках реализации (табс-формы заготовок программ привязаны к языкам реализации).

Табс-представление задачи при р-замене

В процессе *r*-замены понятиям, введённым при *r*-конкретизации, ставятся в соответствие заменяющие их (меняется описание памяти: добавляется табс-слой, клетки которого заполняются новыми именами переменных).

Существует возможность записи новых понятий вместо ранее введённых.

Каждая полученная в результате *r*-замены задача имеет свой паспорт.

Табс-представление задачи при р-конструировании

Табс-представление *составной задачи* (как конструкции из некоторого множества простых и ранее определённых составных задач) включает корневой табс, в клетках первого слоя которого записано наименование задачи и указатели на табсы, которые содержат описания задач, входящих в состав конструкции.

Строки и столбцы первого слоя корневого табса используются для задания структуры составной задачи.

Если связь по памяти имеет тип *выход-вход*, то ссылка на первую из задач будет расположена в строке с номером на единицу меньшим, чем номер строки со ссылкой на вторую задачу; если же - тип *вход-вход*, то ссылки помещаются в одной строке.

Аналогичный принцип позиционного размещения используется, когда задача имеет связь по памяти типа *выход-вход* и *вход - вход*.

Связь по элементам памяти для каждой пары задач устанавливается с помощью дополнительного слоя табса (одного на каждую пару).

Эти слои размещаются вслед за слоем с описанием памяти задачи.

В каждой из клеток перечисляются эквивалентируемые элементы памяти связываемых задач (путём указания соответствующих имён табсов и координат клеток).

Табс-представление задач при р-специализации

Процесс *r*-специализации приводит к образованию новой табс-структуре на основе структуры, сформированной на этапе *f*.

Работа выполняется на копии табс-структуры, представляющей специализируемую задачу.

При работе с копией изменяется содержимое табсов, хранящих исходное отношение *formul*.

Если специализируются параметры, то интервалы их изменения записываются в клетки слоя табса, размещаемого вслед за слоем, содержащим описание памяти задачи.

Если выполняется специализация условий, то дополнительные ограничения записываются в слое табса, где находится описание отношения *formul*.

Прикладные задачи

Пространства прикладных задач состоят из задач, соответствующих определённым областям применения.

Исходным материалом для получения табс-представления *прикладных задач* служат табс-представления *абстрактных задач*, полученные на этапе *f* процесса *r*-конкретизации.

[Содержание <](#)

При переходе к табс-представлению прикладных задач следует заменить на `appl` значение `abstr` атрибута `sp[us]`.

Затем - изменить структуру табсов так, как это было сделано в процессе р-замены.

В результате элементы памяти задач, получают новые имена, отражающие прикладную суть задач.

При специализации прикладных задач и создании на их основе задачных конструкций, действуют те же (что и для абстрактных задач) правила табс-представления.

Пользовательские задачи

Основное отличие пользовательских задач от прикладных состоит в том, что пользователь может не только ввести новые понятийные оболочки, но и создать описание подходящего ему ввода-вывода.

Образование новых понятийных оболочек, осуществляющееся в процессе р-замены для пользовательских задач, приводит к табс-представлениям, аналогичным тем, которые создаются при работе с прикладными задачами.

Каждая пользовательская задача требует заполнения очередного слоя корневого табса.

Чтобы создать шаблон ввода/вывода пользовательской задачи, нужно сформировать отдельный табс.

Родовые связи между табс-описаниями задач

Табс, содержащий описание абстрактной задачи, является табсом-прообразом по отношению к табсу, содержащему описание прикладной задачи, полученному на основе описания этой абстрактной задачи.

Один и тот же табс может быть табсом-прообразом конечного множества табсов.

Другими словами, табс-прообраз может иметь конечное множество табсов-образов.

Любой табс может рассматриваться как табс-прообраз, если определены правила получения его образов.

Табс с пустым множеством правил получения образов называется *конечным табсом*.

Табсы-образы, наследуя характеристики своего табса-прообраза, образуют *родовое табс-дерево*, корневым узлом которого является табс, не имеющий прообраза, а листьями - конечные табсы.

Определены правила объединения табсов.

Конечное множество объединённых табсов образуют новый табс, обладающий характеристиками объединённых.

При объединении табсов объединяются и правила получения их образов.

Табс-представление целевых систем

В целевых системах (как конкретизированных инфах) табс-структура является основной структурой данных.

Табс-структуры применяются и как структуры данных задач, и как структуры для представления задач.

Каждый тип целевых систем имеет присущую ему табс-архитектуру.

Целевая ориентация системы определяется *задачной областью*, заданной некоторым *задачным графом*: поэтому рассмотрение табс-представления целевых систем связано с *табс-представлением задачных графов*.

[Содержание <](#)

Табс-представление задачного графа предполагает табс-представление **задачного базиса и условий межзадачной связи** (для этого используется табс, содержащий имена задач, образующих базис, и описание условий межзадачной связи).

Поскольку целевые системы могут взаимодействовать между собой, в их табс-представлении предусмотрены клетки типа *память* (значениями этого типа могут быть ссылки на табс-слой памяти целевой системы или табс-слой памяти р-объекта).

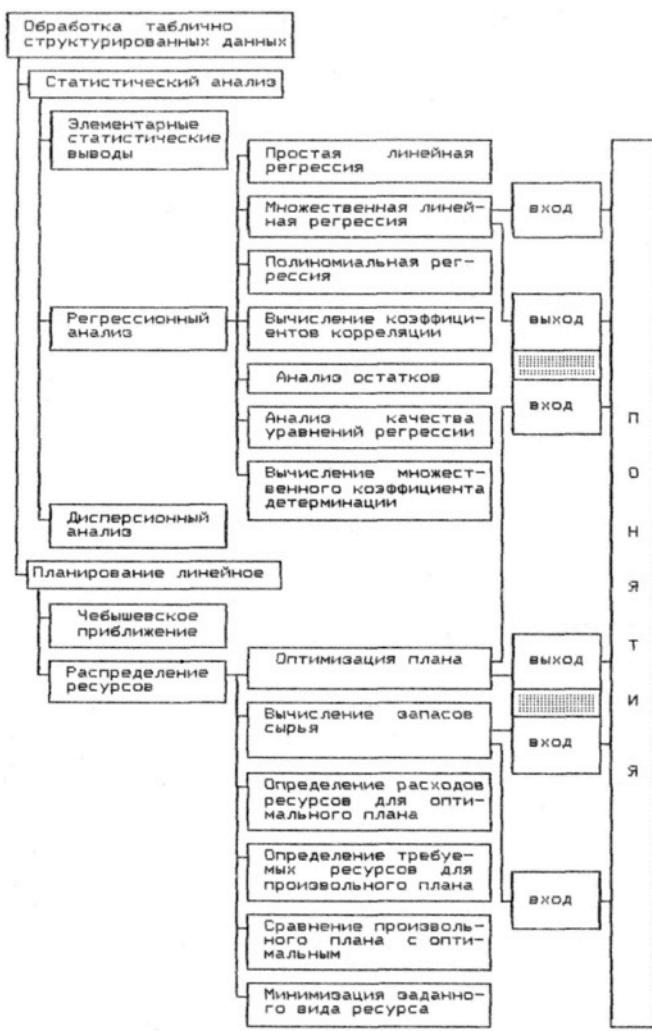
Ссылки имеют вид: *тем <имя табса>*.

Клетка типа *память* применяется в табс-слоях памяти целевой системы и базовых задач.

Переход в клетку типа *память*, даёт возможность работать на памяти целевой системы или - базовой задачи.

❖ Если клетку памяти целевой системы, предназначеннной для решения задач планирования производства, определить как имеющую тип память и занести туда ссылку *тем <РАСПРЕДЕЛЕНИЕ РЕСУРСОВ>* (определив отношение принадлежности системе понятий РАСПРЕДЕЛЕНИЕ РЕСУРСОВ), то становится возможным, например, запрос: найти ОСТАТКИ РЕСУРСОВ по РАСПРЕДЕЛЕНИЮ РЕСУРСОВ. Система при попадании в клетку РАСПРЕДЕЛЕНИЕ РЕСУРСОВ выйдет на табс-слой памяти целевой системы РАСПРЕДЕЛЕНИЕ РЕСУРСОВ. Пользователю будет предложено определить на памяти этой системы запрос на решение задачи распределения ресурсов. Результат решения будет использован в качестве входа для вычисления остатков ресурсов. В итоге пользователь получит интересующие его величины остатков ресурсов. ☀

Взаимодействие целевых систем



Используя определение типа память, связали целевую систему ПЛАНИРОВАНИЕ РАБОТЫ ПРОИЗВОДСТВА с целевой системой РАСПРЕДЕЛЕНИЕ РЕСУРСОВ.

При этом система РАСПРЕДЕЛЕНИЕ РЕСУРСОВ не должна менять своё поведение (из-за этой связи).

Когда, работая на памяти целевой системы ПЛАНИРОВАНИЕ РАБОТЫ ПРОИЗВОДСТВА, попадём в клетку типа память, имеющую значение тем <РАСПРЕДЕЛЕНИЕ РЕСУРСОВ>, система пошлёт сообщение-запрос, адресованное целевой системе РАСПРЕДЕЛЕНИЕ РЕСУРСОВ.

После обработки сообщения будет предоставлена возможность работать на памяти целевой системы РАСПРЕДЕЛЕНИЕ РЕСУРСОВ.

Задав вход, необходимый для решения интересующей нас задачи, в итоге получим вектор оптимального распределения ресурсов.

Полученный результат будет отправлен системе ПЛАНИРОВАНИЕ РАБОТЫ ПРОИЗВОДСТВА (как ответ на её запрос).

Как только клетка РАСПРЕДЕЛЕНИЕ РЕСУРСОВ памяти системы ПЛАНИРОВАНИЕ РАБОТЫ ПРОИЗВОДСТВА станет означенной, возникнет возможность расчёта остатков ресурсов в соответствии с запросом (найти ОСТАТКИ РЕСУРСОВ по РАСПРЕДЕЛЕНИЮ РЕСУРСОВ).

◊ Применение типа память (для клеток табс-слоя памяти) позволяет комплексировать целевые системы, организуя их совместную работу (без коррекции поведения каждой из них).

Ограничений на табс-комплексирование целевых систем не существует. ◊

◊ Ещё пример: в табс-слое памяти целевой системе можно определить клетку, имеющую значение тем <ПЛАНИРОВАНИЕ РАБОТЫ ПРОИЗВОДСТВА>, а в табс-слое памяти системы РАСПРЕДЕЛЕНИЕ РЕСУРСОВ - клетку со значением тем <ВЫЧИСЛЕНИЕ ОГРАНИЧЕНИЙ НА РЕСУРСЫ>. ◊

◊ Все элементы памяти некоторой целевой системы могут иметь тип память.

Кроме того, память целевой системы может содержать клетку типа память, значением которой является ссылка на память системы, все элементы которой имеют тип память. Число таких клеток может быть любым. ◊

[Содержание <](#)

Среда порождения целевых систем может быть представлена как порождающая система высшего уровня, память которой состоит из клеток, каждая из которых имеет тип *память*.

При этом не существует такой целевой системы, память которой содержала бы в качестве одной из своих клеток ссылку на память системы высшего уровня.

Используя понятие *уровень памяти*, можно дать следующее определение среды порождения:

□ Среда порождения - это система порождения с высшим уровнем памяти. □

Возникает вопрос об ограничениях на применение механизма связи по памяти.

Например, допустимо ли системе ссылаться на собственную память?

Такая ссылка допустима, что позволяет реализовать механизм целевой рекурсии (то есть, есть возможность вычисления некоторых элементов входа, которые могут быть получены в результате решения исходной задачи, заданной на памяти той же целевой системы).

Реализация рекурсии требует механизма обработки запросов, учитывающего уровень их вложенности.

Здесь действует правило, аналогичное тому, которое положено в основу машин, управляемых потоком данных: выполняется всё, что обеспечено означенными данными.

Допустимо ли системе с табс-представленной памятью, вложенной в клетку памяти другой системы, иметь, в свою очередь, клетку, в которую была бы вложена память связанной с ней системы?

Иначе: допустима ли взаимная вложенность по памяти?

Исключить такую возможность было бы нецелесообразно, так как и здесь действует то же правило, что и в предыдущем случае.

Если система А ссылается на память системы Б, а та, в свою очередь, - на память системы А, то при обработке запросов достаточно соблюсти очередность, определяемую готовностью по входным данным.

Предложенный механизм комплексирования целевых систем имеет прагматику, опирающуюся на механизм установления связи между понятиями, который применяется при решении комплексов задач.

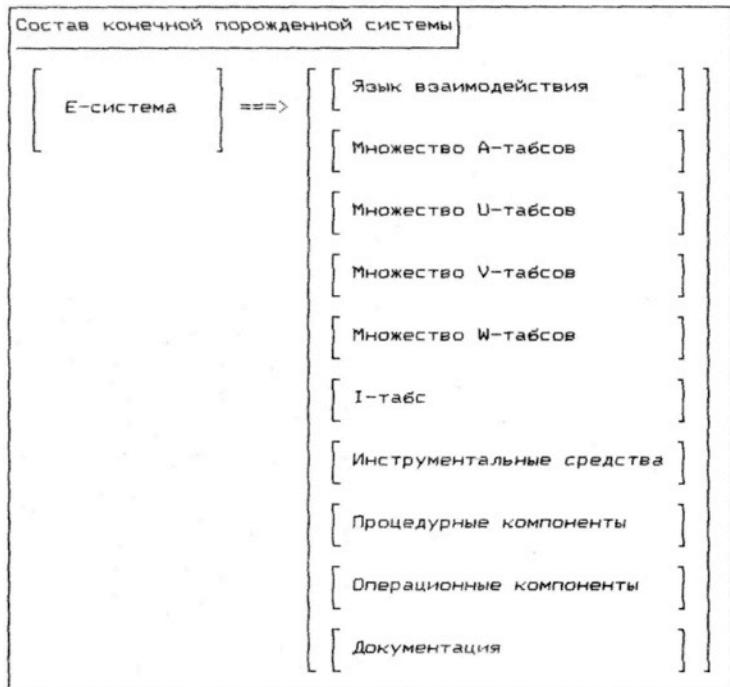
◊ Есть основания полагать, что и в памяти человека существуют "клетки", аналогичные клеткам табс-слоя памяти, значениями которых могут быть ссылки, имеющие вид тем <ИМЯ ЦЕЛЕВОЙ СИСТЕМЫ>.

Это способствует повышению эффективности задачного мышления, так как конструирование целевых систем посредством спецификации механизма связи по памяти реализует довольно естественный принцип связи между понятиями задач. ◊

Основные определения табс-представления целевых систем даны в виде последовательности рисунков (на следующих страницах).

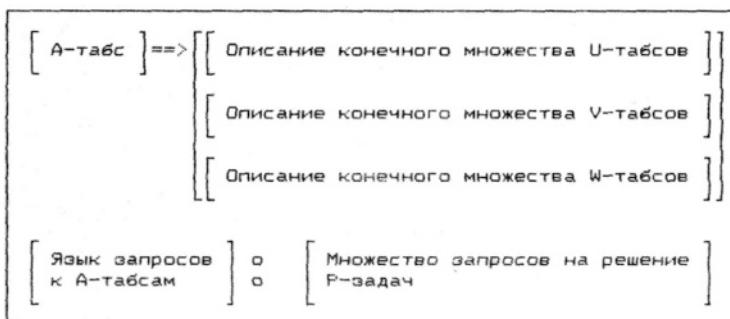
Эти определения положены в основу табс-представления системы [ГЕНПАК](#) и порожденных с её помощью пакетов.

E-система: табс-представление



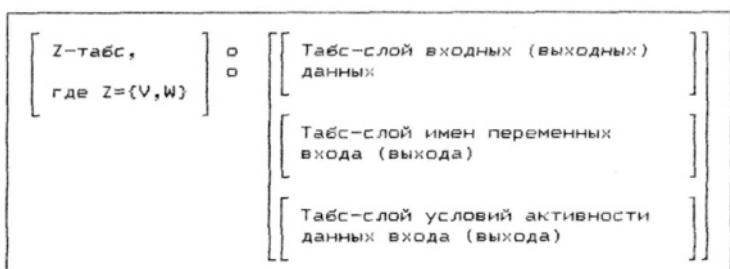
E-система - результат И-порождения.
В её состав входят:
язык взаимодействия
пользователя с системой;
пять множеств табсов: А-необходимых для порождения U-табсов; U- для представления задач и памяти порождённой системы; V- и W- для представления её ввода и вывода; I- для информационного обслуживания пользователя);
инструментальные средства, процедурные и операционные компоненты и документация.

A-табс



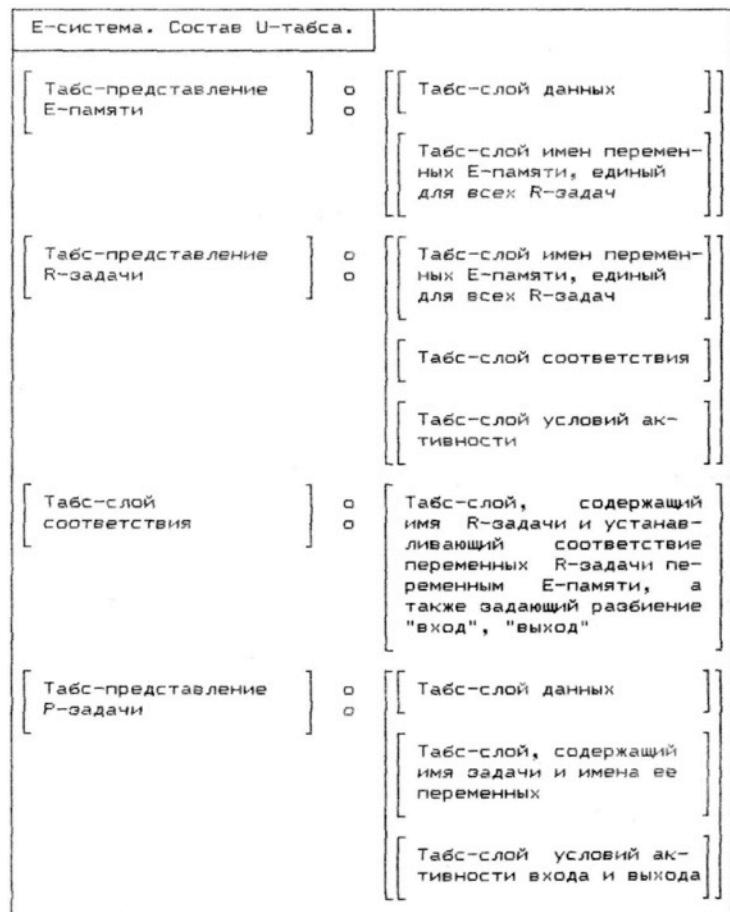
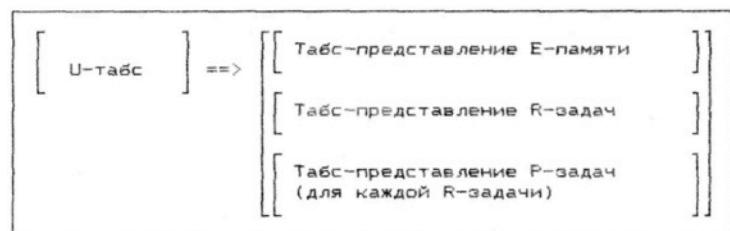
A-табс (названный порождающим) предназначен для порождения U-табсов (представляющих собой конкретизацию A-табса).

Z-табс



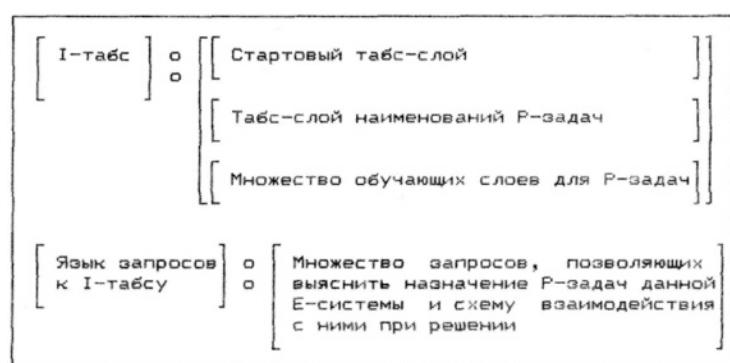
Z-табс предназначен для представления входа и выхода E-системы, а также - условий активности данных.

U-табс



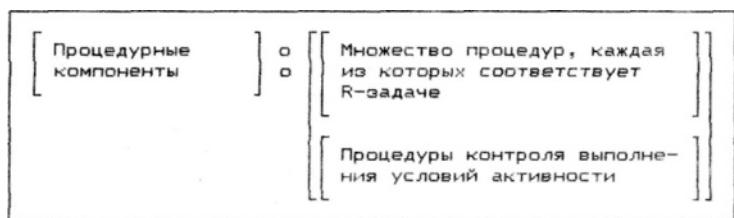
U-табс служит для представления памяти Е-системы, множества задач, которые можно решить (R-задач), и для каждой R-задачи - множества задачных конструктивных объектов (P-задач), используемых при её решении.

I-табс



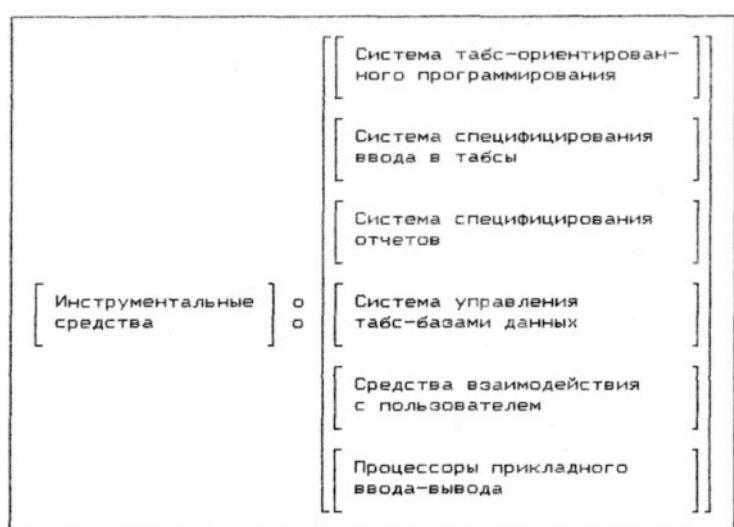
I-табс предназначен для информационного обслуживания пользователя.

Процедурные компоненты



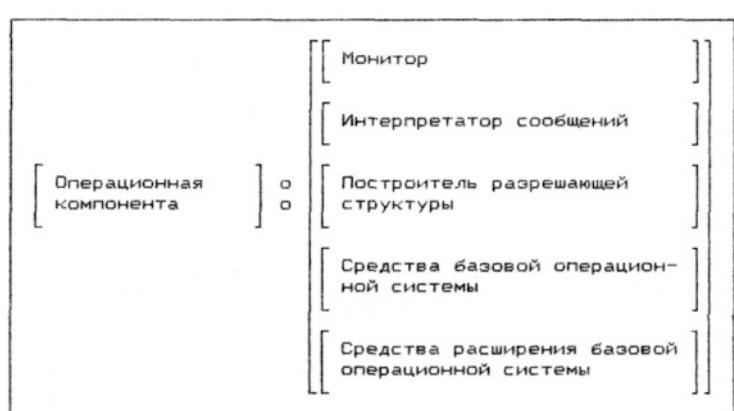
Для каждой R-задачи определено множество процедур, включая процедуры контроля выполнения условий активности данных.

Инструментальные средства



Инструментальные средства Е-системы - её инфраструктура, необходимая для решения Е-задач.

Операционная компонента



Основное назначение операционной компоненты: управление Е-системой, интерпретация сообщений пользователя и построение разрешающих структур, которые служат основанием для построения R-задач.

Табс-представленный инф

Если нас интересует системная сторона табс-представленной целевой системы, удобно иметь специальное название для инфа, являющегося моделью такой системы.
□ Будем называть *T-инфом* инфа, имеющий табс-представленную память.

Т-инфи образуют классы, где каждому наследнику передаются свойства его предка. Свойства Т-инфо, являющегося главой рода, наследуют все табсы класса.

[Содержание <](#)

Принцип наследования снижает трудоёмкость описания создаваемых Т-инфов, что весьма важно при реализации процесса порождения.

Типы Т-инфов в классе могут быть связаны отношением частичного порядка, которое соответствует *включению, заданному на свойствах*.

Для Т-инфа, вводимого в класс, достаточно указать его предка, как всё, чем обладает предок (например, средства интерпретации спецификаций и средства обработки табсов), становится достоянием наследника.

Предок может иметь несколько наследников.

Наследник имеет единственного предка.

Т-инф - многограничный объект: это и формо-ориентированное описание табс-автомата, (когда просматриваем его); это и сам табс-автомат, когда он работает как целевая система; это также и то, с помощью чего можно создать другие Т-инфы.

Его нельзя уподобить традиционно понимаемой программе или абстрактному типу данных.

Т-инф - это табс-представленная модель символьной конвейерной машины, созданной для работы в кооперации с себе подобными машинами □

Корреспонденты Т-инфа, обменивающиеся с ним сообщениями, могут установить разные точки зрения на Т-инф.

Существует также механизм создания понятийных оболочек: каждый элемент памяти задачного графа Т-инфа, а также заданные вершины графа могут получить имена, выбранные корреспондентом.

После этого можно посыпать сообщения, представляющие собой запросы на языке, лексикон которого определён созданной понятийной оболочкой.

Синтаксическая форма и реализация одинаковы для сообщений всех корреспондентов фиксированного Т-инфа.

Число понятийных оболочек не ограничено. При создании каждой из них допускается синонимия: один и тот же элемент памяти или задача могут получить более одного имени.

Другим важным средством является *маскирование*.

Можно замаскировать часть памяти Т-инфа (она становится невидимой для создавшего маску корреспондента).

Маска может быть абсолютной или относительной.

При абсолютной маске элемент "не существует" и при записи, и при чтении.

Относительная маска может устанавливаться только для одной из этих операций.

Каждая задача в составе задачного графа Т-инфа имеет свою память.

Наличие памяти у задач позволяет ввести следующий механизм связи между Т-инфами, основанный на принципе кооперативного решения задач.

Например, если составная задача попала в разрешающую структуру, то работа по её решению будет передана Т-инфу, специализирующемуся на решении этой составной задачи.

Предусмотрен еще один вариант взаимодействия Т-инфов при кооперативном решении задач.

[Содержание <](#)

Он осуществляется путём укрупнённого представления памяти составной задачи. Это означает, что в Т-инфо, где некоторая задача находится в составе его задачного графа, не каждой переменной входа и выхода этой задачи поставлена в соответствие отдельная клетка в табс-памяти задачи.

Переменные могут быть сгруппированы.

Предельным случаем группировки является тот, когда существует всего две группы: вход и выход.

❖ Приведем пример. Если известно, что среди входных есть группа переменных, принадлежащих Giv, то в её пределах можно сделать укрупнение, не противоречащее содержательной роли переменных.

Когда встречается составная задача с укрупненным представлением памяти, то работа с ней поручается специализирующемуся на ней Т-инфу. ❖

Есть две причины, по которым работа с составной задачей передаётся другому Т-инфу: если задача включает другие составные задачи и если она имеет укрупненное представление памяти. Как правило, эти причины неразлучны.

Заметим, что механизмы укрупнения памяти и укрупнения задач неразрывно связаны с механизмом агрегирования Т-инфов.

Первые два механизма являются обеспечивающими по отношению к третьему: агрегирование основано на укрупнении задач и памяти.

Табс-представление целевых знаний

□ Под табс-представлением целевых знаний будем понимать реализацию модели задачной области на основе Т-инфов.

При этом язык, интерпретируемый на модели, - это язык взаимодействия с объединением Т-инфов. □

Правила табс-навигации

Как реализуются правила поэтапного описания конструируемой системы при порождении?

Ответ на этот вопрос весьма важен: ведь эффективность системы порождения во многом определяется реализацией работы с правилами на каждом из этапов И-порождения.

Учитывая это, правила поведения системы порождения воплощены как правила навигации по табс-структурям, содержащим различные фрагменты описания порождаемой системы.

Элементы описания создаются разработчиком порождаемой системы в пошаговом режиме и содержат зависимые один от другого фрагменты.

Правила навигации содержатся в базе знаний порождающей системы.

В каждом фиксированном пункте маршрута разработчик имеет следующие возможности:

- сделать описание;
- предложить системе выполнить описание;
- сделать остановку (система запоминает точку остановки);
- пройти в пошаговом режиме весь пройденный к этому моменту маршрут в обратном порядке;
- перейти в предыдущую точку остановки;
- переименовать любую точку остановки;
- перейти в указанную переименованную точку;

[Содержание <](#)

- предложить системе отложить выполнение описания в данной точке маршрута (система даёт ответ относительно возможности реализации предложения: не во всякой точке можно отложить описание);
- предложить системе в данной точке принять некоторое временное описание, носящее экспериментальный характер (система помечает такое описание, как допускающее изменение по её инициативе в случае возникновения у разработчика трудностей при выборе вариантов описания в дальнейшем).

Семантика описания в каждой точке маршрута задана.

Она (по желанию пользователя) может быть напомнена ему системой.

Синтаксис также не должен обременять память разработчика: правила построения

правильного описания можно увидеть, не смещаясь из данной точки маршрута.

Для этого достаточно сообщить о своем намерении системе.

Конструирование табс-представленных целевых систем

Метод порождения целевых систем является методом их инф-конструирования.

Независимо от содержания задач, на которые рассчитана порождаемая система, правила её построения определены системным обликом Т-инфов.

Задачный потенциал Т-инфра определяется его задачным графом.

Порождение как интерактивное конструирование Т-инфра

Процесс создания искомой системы [от табс-представления в пространстве прикладных задач до табс-представления в выбранном пользовательском пространстве (с учётом конкретных условий применения)] реализуется как процесс конструирования Т-инфра, обладающего требуемыми свойствами.

Сначала определяется задачная область путём описания задачного графа с базисом из задач этой области.

Затем поэтапно выполняется спецификация условий применения: максимальные размеры задач, допустимое время обработки сообщения, режим работы пользователя (или группы), операционная система и характеристика аппаратных средств машины; язык реализации; формы и режимы ввода-вывода; подготовленность пользователя, построение интерфейса; требования к информационному обслуживанию (объяснения, обучение), средства поддержки во время работы; профессиональный лексикон пользователя, понятийная оболочка задачного графа и языка, предложения которого интерпретируются на этом графе.

Пошаговый процесс приводит в итоге к построению задачного графа, базис которого состоит из пользовательских задач и для которых определены условия межзадачных связей и условия их реализации при функционировании порождённой системы.

Функциональное наполнение системы определяется созданным задачным графом, а системное - механизмами обработки сообщений, работы с табс-представлением памяти задачного графа и его вершин, табс-ориентированного ввода-вывода, интерфейса с пользователем, информационного обслуживания и обучения.

Прототипы целевых систем: средство накопления задачных знаний

В системе порождения для точек зрения, заданных системными аналитиками, сохраняются образцы целевых систем.

Это относится и к конечным, и к порождающим системам.

Такие образцы принято называть прототипами.

Коррекция прототипа вместо полного описания

Существует возможность создания целевой системы путём коррекции описания прототипа.

Для этого система предоставляет возможность не только подробно ознакомиться с работой прототипа, но и установить зависимость между фрагментами описания и теми функциональными характеристиками, которые интересуют разработчика.

Ему предстоит выяснить, присущи ли прототипу интересующие его характеристики.

Если - да, то необходимо ли их изменить.

Работа выполняется следующим образом.

Пользователь вводит режим КОРРЕКЦИЯ ПРОТОТИПА.

Затем начинает просмотр интересующих его прототипов.

Если ему заранее известны требуемые функциональные характеристики целевой системы, он сообщает их системе, заполнив специальную табс-форму.

Используя введённое описание, система подбирает самый близкий в функциональном смысле прототип и предлагает пользователю посмотреть его, а, если необходимо, то и - демонстрацию функционирования.

Если, поработав с прототипом, пользователь захочет изменить функциональные характеристики, то сможет сделать это существенно успешнее, чем в ином случае.

Если предложенный системой прототип отвергнут, она предложит уточнить спецификацию (путём описания тех компонент характеристики прототипа, которые отвергаются; в пределе - всех, что означает необходимость составить полную спецификацию).

Затем система вновь приступит к подбору наилучшего приближения к искомой системе.

Когда прототип подобран, начинается процесс коррекции его описания. Допустимый объём коррекции и порядок её выполнения контролируются системой порождения. Наиболее целесообразна пошаговая интерактивная коррекция, когда после каждого шага целевая система приводится к виду, допускающему работу с ней в режиме ПРОБА.

Этот путь не самый быстрый, но позволяющий получить систему, наиболее соответствующую замыслу пользователя.

Создание составного прототипа

Заданные пользователем функциональные характеристики могут в общем случае привести к выбору нескольких ранее порождённых целевых систем в качестве подлежащего коррекции прототипа.

Тогда первым шагом, который система предложит выполнить пользователю (после просмотра работы каждой из систем) будет выполнение операции их объединения. Успешное завершение этой операции приведёт к получению целевой системы, с которой пользователю будет предложено ознакомиться.

Дальнейшая работа с ней выглядит так же, как работа с одним прототипом.

Коррекция составного прототипа

Существует возможность сужения ранее созданного составного прототипа посредством указания одной или нескольких целевых систем, которые следует вывести из его состава.

Такой подход является альтернативным по отношению к варианту работы, когда пользователь должен определить требуемые функциональные характеристики. Если он указывает, какие целевые системы следует удалить для получения желаемого приближения к искомой системе, то берёт на себя решение задачи анализа

[Содержание <](#)

функциональных характеристик составного прототипа и сопоставления их с требуемым набором характеристик.

Если искусственный разум системы порождения - база задачных знаний, то совокупность семейств прототипов (по задачным областям) - её богатство. Чтобы приобрести его, требуется не просто время - требуется работа квалифицированных системных аналитиков, производящих целевые системы.

И-порождение: характеристика и применимость

Метод порождения построен на теоретических основах И-порождения, включающих: определения, утверждения и их обоснование [включая доказательства (там, где это необходимо)]; модели задач и задачных областей; языки, интерпретируемые на моделях задачных областей; табс-представление моделей и языков.

Попытки изобретения полностью формального метода решения задачи порождения программных систем имеют не больше смысла, чем формальное описание методологии автоматизированного проектирования сложных станков (и других человеко-машинных комплексов).

Теоретическая платформа И-порождения состоит из связанных между собой концептуальных блоков.

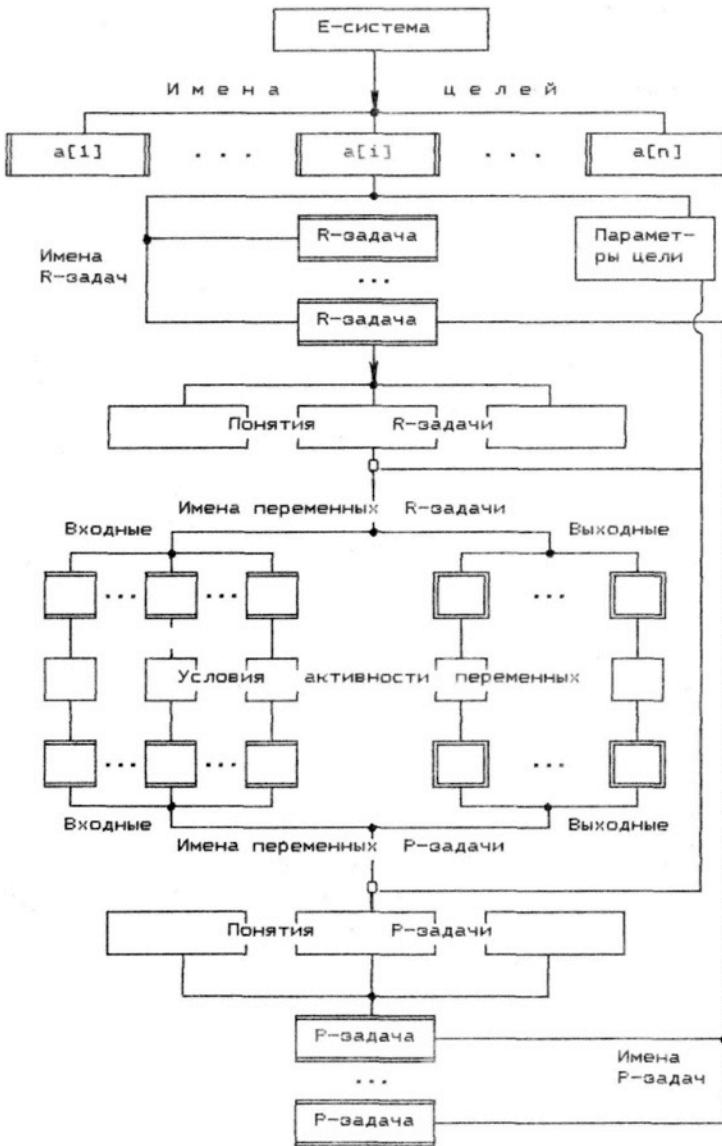
1. Формализм представления задач как конструктивных объектов, имеющих f-, m-, t- и r-представления на различных стадиях порождения. Понятия о *символ-форме* и *код-форме*. Наделение описаний задач свойствами конструктивных объектов имеет принципиальное значение, так как делает возможным создание задачных конструкций, поэтапное специфицирование которых приводит в итоге к получению искомой программной системы. Будучи помещёнными в специальные формы, описания задач приобретают необходимые для процесса порождения свойства конструктивных объектов.
2. Три типа пространств задачных конструктивных объектов (*абстрактных*, *прикладных* и *пользовательских*) с механизмом взаимодействия между пространствами. Введение *пространств задачных конструктивных объектов* создало основу для накопления и применения задачных знаний трёх видов: независимой от приложений (абстрактной), ориентированной на различные приложения (прикладной) и ориентированной на конкретные условия применения в рамках фиксированного приложения (пользовательской).
3. Типы задачных конструкций и правила их образования. Три типа функций связи по памяти определяют образование задачных конструкций (от составных задач до задачных графов).
4. Конструирование на задачных графах: разрешающие структуры и механизм их построения. Доказательство существования решения задачи порождения как конструктивное доказательство существования разрешающей структуры.
5. Понятие инфа как символьной машины с конечным числом состояний. Инф введён как системная модель задачной конструкции. Понятие инф-конструирования как взаимодействия инфов в процессе порождения.
6. Понятия табса и табс-структуры данных. Табс-структура позволяет моделировать известные структуры данных и играет большую роль в реализации инфа. Понятие Т-инф (инф с табс-памятью).
7. Понятие точки зрения; применение этого понятия в системе порождения. Точка зрения позволяет определить вектор с тремя компонентами (корреспондент, цель, стадия), с помощью которого инженеры задачных знаний, системные аналитики, аналитики-разработчики приложений и конечные пользователи могут, работая с системой порождения или порождёнными продуктами, определить своё отношение к объекту, для которого установлена точка зрения.

Ограничения применимости

Применение И-порождения целесообразно, если порождаемая программная система предназначена для решения достаточно сложного комплекса задач, где существует необходимость межзадачного взаимодействия на основе баз данных (как в задачах

автоматизированной разработки программного обеспечения и автоматизированного проектирования других сложных человеко-машинных систем).

Многоцелевая система



Порождённая многоцелевая система: пример

В общем случае целевая система может служить для достижения нескольких целей, каждая из которых понимается как множество разрешимых задач. Задачная область каждой цели представлена пользовательским задачным графом.

Элемент памяти задачного графа (в общем случае) определяется некоторым множеством переменных.

Множество может состоять либо из фиксированного числа элементов (в частности, одного), либо число элементов может определяться значениями параметров цели.

Параметры цели имеют целочисленные значения.

Диапазон их изменения задаётся в процессе порождения, а конкретные значения - при решении задач с помощью порождённой системы.

Число параметров цели постоянно для фиксированной цели.
Итак, пусть целевая система рассчитана на множество целей А
 $\{a_1, a_2\}$

где $a[i]$ - фиксированная цель ($i = 1..n$), а $M[i;] = \{m[i;1]..m[i; l/i]\}$ - множество параметров цели $a[i]$.

Каждому элементу $a[i]$ ставится в соответствие задачный граф с базисом $R[i] = \{r[i; 1]..r[i; n/i]\}$:

каждой задаче $r[i; j]$ - множество понятий $T[i; j] = \{t[i; j/1]..t[i; j/n]\}$.

Причём, для любого множества $T[i; j]$ найдётся хотя бы одно отличное от него множество, пересечение с которым не пусто. Это условие выражает связь по памяти между задачами.

Объединение всех множеств $T[i]: i]$ образует множество понятий

$$T[i:] = \{t[i; 1], t[i; p/i]\}$$

цели а[i].

Каждому понятию $t[i; s] = \text{elem}(T(i))$ соответствует множество переменных $K[i; s] = \{k[i; s/1], k[i; s/2]\}$.

[Содержание <](#)

Число переменных $s[i]$, определяющих понятие $t[i; s]$, является в общем случае функцией параметров цели $s[i] = s[i](m[i; 1]..m[i; l/i])$.

При фиксированных значениях параметров $m[i; 1]..m[i; l/i]$ объединение множеств переменных $K[i; s]$ образует табс-слой имён переменных памяти системы - множество $K[i;] = \{k[i; 1..k[i; v/i]]\}$, где $v[i]$ - число переменных памяти.

Каждой задаче $r[i; j]$ ставится в соответствие пара непересекающихся подмножеств множества $K[i;]$ - подмножество $v(r[i; j])$ входных переменных (вход) и подмножество $w(r[i; j])$ выходных переменных (выход).

Одноцелевая система: пример

Задачи распределения ресурсов

P	Найти экстремум целевой функции $c[1]*x[1] + \dots + c[n]*x[n]$ при ограничениях $a[1,1]*x[1] + \dots + a[1,n]*x[n] \leq b[1]$ \vdots $a[m,1]*x[1] + \dots + a[m,n]*x[n] \leq b[m], m > n,$ $b[i] > 0, i = 1..m; x[j] \geq 0, j = 1..n$
A	Определить остатки ресурсов $k[i] = b[i] - \sum_{j=1..n} a[i,j]*x[^;j], i = 1..m$ После реализации оптимального плана $x[^;] = (x[^;1]..x[^;n]),$ обеспечивающего экстремум целевой функции $c[1]*x[1] + \dots + c[n]*x[n]$ при ограничениях $a[1,1]*x[1] + \dots + a[1,n]*x[n] \leq b[1]$ \vdots $a[m,1]*x[1] + \dots + a[m,n]*x[n] \leq b[m], m > n,$ $b[i] > 0, i = 1..m; x[j] \geq 0, j = 1..n$
C	
P	
E	
D	
E	
L	Вычислить отклонения $r[j] (j = 1..n)$ компонент произвольного плана $x = (x[1], \dots, x[n]),$ удовлетворяющего ограничениям $a[1,1]*x[1] + \dots + a[1,n]*x[n] \leq b[1]$ \vdots $a[m,1]*x[1] + \dots + a[m,n]*x[n] \leq b[m], m > n,$ от соответствующих компонент оптимального плана $x[^;] = (x[^;1]..x[^;n]),$ обеспечивающего минимум целевой функции $c[1]*x[1] + \dots + c[n]*x[n]$ при тех же ограничениях
N	
I	
E	
R	
E	Найти разность значений целевой функции $F = c[1]*x[1] + \dots + c[n]*x[n],$ для произвольного плана $x = (x[1]..x[n]),$ удовлетворяющего ограничениям $a[1,1]*x[1] + \dots + a[1,n]*x[n] \leq b[1]$ \vdots $a[m,1]*x[1] + \dots + a[m,n]*x[n] \leq b[m], m > n,$ $b[i] > 0, i = 1..m; x[j] \geq 0, j = 1..n$ и оптимального плана $x[^;] = (x[^;1]..x[^;n]),$ обеспечивающего минимум целевой функции F при тех же ограничениях
S	
Y	
R	
C	
D	Найти значение целевой функции $F,$ если $x[1]..x[n]$ удовлетворяют ограничениям $a[1,1]*x[1] + \dots + a[1,n]*x[n] \leq b[1]$ \vdots $a[m,1]*x[1] + \dots + a[m,n]*x[n] \leq b[m], m > n,$ $b[i] > 0, i = 1..m; x[j] \geq 0, j = 1..n$
B	

Множество A в этом случае состоит из одного элемента - цели $a[1]$, объединяющей, например, оптимизационные задачи класса «Распределение ресурсов» (PP). Математические формулировки пяти задач PP, составляющих задачный базис, приведены на рисунке.

Множество $R[1;]$ состоит из пяти элементов $r[1; 1]..r[1; 5]$ (в дальнейшем при рассмотрении этого примера индекс i , везде равный единице, опускается).

Каждый из элементов $r[j]$ отождествляется с именем соответствующей задачи.

Так, задача $r[1]$ - основная задача линейного программирования: задана целевая функция

$\sum_{j=1..n} c[j]*x[j]$ и система $m > n$ линейных неравенств (ограничений) $\sum_{j=1..n} a[i, j]*x[j] \leq b[i], m > n; i = 1..m.$

Найти неотрицательные значения $x[0; j] (j = 1..n)$ переменных $x[j],$ обеспечивающие достижение максимума (минимума) целевой функции при выполнении ограничений.

Задача $r[5]$ - это задача отыскания

значений целевой функции для произвольных значений переменных, удовлетворяющих ограничениям.

Задаче $r[1]$ ставится в соответствие множество понятий

$T[1] = \{t[1]..t[7]\},$ где

$t[1]$ - коэффициенты целевой функции, $t[2]$ - тип экстремума,

$t[3]$ - коэффициенты ограничений,

[Содержание <](#)

$t[4]$ - типы ограничений, $t[5]$ - свободные члены, $t[6]$ - оптимальный вектор переменных,

$t[7]$ - оптимальная величина целевой функции;

Задаче $r[5]$ соответствует множество понятий $T[5] = \{t[1], t[3], t[4], t[5], t[8], t[9]\}$, где $t[8]$ - допустимые значения переменных; $t[9]$ - величина целевой функции, соответствующая допустимому вектору переменных.

Пересечение множеств $T[1]$ и $T[5]$ - непусто: оно включает элементы $t[1], t[3], t[4]$ и $t[5]$.

№№	Понятия цели «Распределение ресурсов» (PP)	Множества понятий задач PP				
		T1	T2	T3	T4	T5
1	Коэффициенты целевой функции	+	+	+	+	+
2	Тип экстремума	+	+	+	+	
3	Коэффициенты ограничений	+	+	+	+	+
4	Типы ограничений	+	+	+	+	+
5	Свободные члены (запасы ресурсов)	+	+	+	+	+
6	Оптимальный вектор переменных	*	+		+	
7	Оптимальная величина целевой функции	*			+	
8	Допустимый вектор переменных			+	+	*
9	Величина целевой функции, соответствующая допустимому вектору переменных				+	*
10	Вектор остатков ресурсов			*		
11	Разность компонент допустимого и оптимального векторов переменных				*	
12	Разность величин целевой функции для допустимого и оптимального векторов переменных				*	

Обозначения: + вход, * выход

Параметры цели РР - число переменных n и количество ограничений m , поэтому множество М параметров состоит из двух элементов $m[1] = m$ и $m[2] = n$. Каждое из понятий 2, 7, 9 и 12 определяется единственной переменной, тогда как остальным понятиям соответствует множество переменных, число элементов которых зависит либо от параметра $m[1]$, либо от $m[2]$, либо от $m[1]$ и $m[2]$:

$s[1] = s[6] = s[8] = s[11] = n$;

$s[4] = s[5] = s[10] = m$;

$s[3] = m^*n$.

[Содержание <](#)

Число переменных, образующих память целевой системы, записывается как функция параметров цели следующим образом:

$$\text{Card mem} = 4*n + 3*m + m^2*n + 4.$$

Каждое множество понятий $T[j]$ разбьём на два непересекающихся подмножества $T[v; j]$ и $T[w; j]$.

Элементами $T[v; j]$ являются те понятия, которые определяются входными переменными задачи $r[j]$, а элементами $T[w; j]$ - понятия, определяемые выходными переменными $r[j]$.

Например, $T[v; 1] = \{t[1]..t[5]\} = \{\text{коэффициенты целевой функции, тип экстремума, коэффициенты ограничений, типы ограничений, свободные члены}\}$, а $T[w; 1] = \{t[6], t[7]\} = \{\text{оптимальный вектор переменных, оптимальная величина целевой функции}\}$.

При фиксированных значениях параметров цели подмножествам $T[v;j]$ и $T[w;j]$ соответствуют множества входных и выходных переменных задачи $r[j]$.

Если, например, $n = 2$ и $m = 3$, то множество входных переменных: $v(r[j]) = \{c[1], c[2]\}$, тип экстремума, $a[1,1], a[1,2], a[2,1], a[2,2], a[3,1], a[3,2]$, тип первого ограничения, тип второго ограничения, тип третьего ограничения, $b[1], b[2], b[3]\}$, где $c[i]$ - коэффициенты целевой функции, $a[i, k]$ - коэффициенты ограничений, $b[j]$ - свободные члены.

Множество выходных переменных $w(r[j])$: $w(r[j]) = \{x[0; 1], x[0; 2], F[0;]\}$, где $(x[0; 1], x[0; 2])$ - оптимальный вектор переменных, а $F[0;]$ - оптимизируемая целевая функция. Формирование множеств переменных $v(r[j])$ и $w(r[j])$ из элементов памяти выполняется посредством задания соответствия между множествами X и Y , где X включает все переменные памяти, а $Y = \{\text{вх, вых}\}$ - множество из двух элементов. Соответствие $(r[j], X, Y)$ определяется как подмножество декартова произведения X^*Y : $r[j] = \text{elem}(X^*Y)$.

Фрагмент табса: задание соответствия

Память	
5	9
max	
1	1
<=	15
5	3
<=	120
5	10
<=	100
x_1^*	x_2^*
F^*	

Слой соответствия переменных

	вх	вх	вх	
	вх	вх	вх	вх
	вх	вх	вх	вх
	вх	вх	вх	вх
	вых	вых	вых	

Для задания соответствия используем двухслойный табс. В первом слое размещаются переменные памяти, а второй - содержит символы "вх" и "вых" в клетках, координаты строк и столбцов которых совпадают с координатами переменных $v(r[j])$ и $w(r[j])$ первого слоя. Преимущества двухслойного задания соответствия связаны с многократным использованием слоя переменных памяти при задании различных соответствий, а

также его наглядностью. Определенные таким способом соответствия $r[j]$ в дальнейшем будем называть R-задачами, снабжая R нижним индексом, когда речь идет о конкретной задаче.

Табс-ориентированные языки

Языки табс-ориентированного программирования (Т-языки) относятся к интерактивным формо-ориентированным языкам.

Т-языки делятся на ЧТО-языки спецификаций и КАК-языки табс-ориентированного программирования.

Языки обоих классов имеют общие черты, что снижает трудоёмкость разработки интерпретаторов.

Основной структурой данных любого из языков табс-ориентированного программирования является табс-структура.

Словарь, правила построения предложений и семантика Т-языка описываются путём интерактивного заполнения табс-форм.

Последовательность заполнения клеток табс-форм и допустимые значения каждой клетки определены.

При затруднениях предусмотрена возможность, прервав на время ввод, получить помогающие сведения (что можно заносить в очередную клетку и/или какая клетка является очередной).

В Т-языке существуют два вида управления: *внешнее* и *внутреннее*.

Внешнее - реализуется посредством механизма взаимодействия между объединением Т-инфов и его окружением.

Сообщение, поступившее из окружения, передаётся Т-инфу, которому оно адресовано.

Т-инф, получивший сообщение (после конвейерной обработки) посыпает своему корреспонденту сообщение о завершении.

Обрабатывая сообщение, Т-инф может, в свою очередь, послать сообщение другому Т-инфу, кооперируясь с ним в выполнении поступившего запроса (например, если Т-инфы представляют задачи, входящие в составную задачу).

Таким образом, сообщение из окружения объединения Т-инфов может вызвать обмен сообщениями между Т-инфами.

Внутреннее управление реализуется посредством механизма взаимодействия между Т-инфами одного и того же объединения.

Принцип вложенности Т-языков

Например, в клетке Т-программы, имеющей тип ИНТЕРПРЕТАТОР, может находиться текст программы на языке Т-кон (одном из языков, входящих в среду Т-языка).

Когда интерпретатор Т-языка попадёт в эту клетку, он пошлет сообщение (с табс-адресом клетки) Т-кон-интерпретатору, который, выполнив свою работу, ответит сообщением (о завершении или об ошибке).

После чего будет продолжена работа по выполнению основной Т-программы.

По такому принципу действует интерпретатор любого Т-языка.

Принцип вложенности Т-языков рассчитан на формирование расширяемой обновляемой Т-языковой среды И-порождения (путём добавления новых языков и расширения существующих).

О реализации

При реализации Т-языка на первой позиции - его прагматика (влияющая на продуктивность реализации системы задачных понятий и её символьной модели, на которой интерпретируются предложения Т-языка).

Важно, какие мысленные образы предлагает Т-язык, насколько способствует процессу формирования и реализации замысла.

При этом скорость работы, безусловно, должна быть такой, чтобы не страдала поддержка процесса воплощения замысла разработчика.

Идеальной для Т-языка является программно-аппаратная реализация: Т-инф создан в расчёте на такую реализацию.

В частности, объединение Т-инфов рассчитано на реализацию в многопроцессорных системах.

Программист, работающий с Т-инф-средой, играет роль её окружения.

В Т-языках, рассчитанных на реализацию в интерактивной среде, предусмотрены средства определения организации процесса разработки программ (чтобы программист имел возможность поэтапного контроля).

Реализация Т-языка предполагает, что средства операционной системы рассчитаны на поддержку механизма обмена сообщениями между Т-инфами и с окружением и - распределения системных ресурсов между активными Т-инфами.

Т-спецификация

По форме спецификация на ЧТО-языке, относящемся к Т-языкам, (Т-спецификация) представляет собой табс.

Составление спецификации на языке Т-спецификации осуществляется в диалоге со структурно-ориентированным табс-редактором и интерпретатором этого языка.

В диалоге существует свобода выбора из заданной совокупности тем, но отсутствует возможность уклоняться от предусмотренной дисциплины разработки в рамках выбранной темы.

Диалоговое обучение (с выполнением упражнений для освоения структурно-ориентированного табс-редактора Т-спецификаций) включает возможности выборочного просмотра табсов систем (порождающих и конечных).

Т-спецификация выглядит как просмотр уже имеющихся табс-спецификаций, создание новых и/или редактирование ранее созданных.

Табс-спецификация, признанная редактором правильной, предлагается интерпретатору.

Обрабатывая в диалоге с системным аналитиком табс-спецификацию, интерпретатор формирует табс-представление порождаемой системы.

Т-программирование

Программирование на таком языке выглядит, как процесс описания решения задач, редуцируемых к решениям задач с известной программной реализацией. Если в результате интерпретации описание признано правильным - значит искомая задача разрешима.

Любой Т-язык - это инф-ориентированный язык (каждый табс принадлежит некоторому Т-инфу).

Т-инфы взаимодействуют путём обмена сообщениями.

Объединение Т-инфов - это Т-инф более высокого уровня.

Табс-структура в составе одного Т-инфра может быть обработана другим Т-инфом. В Т-языке табс-представление данных аналогично представлению программ: те и другие размещены в табс-слоях.

Операции обработки табс-структур рассчитаны на работу с табс-данными и табс-программами.

Т-программа может содержать условные ветвления, циклы и обращения к другим Т-программам.

Обработка табс-структур - основа реализации процессов И-порождения.

Язык специфирования табс-форм

Т-форм - ЧТО-язык интерактивного описания табс-форм спецификаций, названных Z-формами, каждая из которых определяет форму и тип содержимого табс-спецификации (С-формы).

Язык Т-форм - прообраз языка специфирования табс-форм, используемого для описания табличных отчётов.

В языке Т-форм спецификация может быть представлена в форме:

- текстового сообщения, размещённого в одной табс-клетке;
- табс-сообщения (каждая составляющая которого размещена в отдельной табс-клетке Z-формы);
- комбинации текстовых и табс-сообщений.

Используемые при описании С-форм умолчания могут быть изменены.

С-форма представлена в виде иерархии её элементов.

Предусмотрены возможности параметрического определения содержимого спецификации и автоматического определения размеров столбцов и строк таблиц табсов для размещения элементов С-формы [С-форма компонуется из табс-структур, имеющих не более двух измерений (клеток, столбцов, строк, сечений табса по слову, строке, столбцу)].

Чтобы облегчить описание С-форм, в языке введено понятие <таблица> - двумерный массив клеток, получаемый как сечение табса путём фиксации значения по одной из координат, и <ряд> - одномерный массив, получаемый при фиксации значений по двум координатам в табсе.

Свободные координаты в <таблицах> и <рядах> могут ограничиваться диапазоном возможных значений.

Понятия <таблицы>, <ряда> и <клетки> являются определяющими для понятия <фрагмент табса>.

С-форма представляет собой множество фрагментов табсов с фиксированным взаимным расположением.

С-форма характеризуется рядом признаков, названных атрибутами.

Чтобы описать требуемый вид специфицируемой формы, надо определить её атрибуты (размеры, типы клеток).

Все атрибуты С-формы разделены на две группы:

- относящиеся к С-форме в целом (например, размеры);
- имеющие смысл для отдельных полей С-формы - строк, столбцов, клеток.

Общие атрибуты С-формы удобнее описывать текстовыми командами, имеющими формат:

Содержание <

<имя_атрибута> [ДЛЯ <фрагмент_C-формы>] = <значение>

или

<имя_атрибута> [ДЛЯ <фрагмент_C-формы>] В<фрагмент_Z-формы>

Здесь <имя атрибута> - одно или несколько слов, записанных полностью или с сокращениями.

Необязательная запись "ДЛЯ <фрагмент_C-формы>" задаёт множество полей в карте расположения элементов С-формы, для которых действительно данное определение. По умолчанию атрибут определяется для всей С-формы.

Синтаксис записи <значение> различается у разных атрибутов.

Чаще всего это слово-идентификатор, число, числовой интервал.

В отдельных случаях используется специальный формат записи - шаблон числа, набор символов оформления таблицы.

Определения некоторых значений могут включать в себя ссылки на фрагменты табсов, хранящихся в базе знаний (например, содержимое поля С-формы может быть задано ссылкой на клетку с нужным текстом).

Примеры таких ссылок:

- "ТАБЛ1" : 3, 4, 5 - клетка табса ТАБЛ1 с координатами 3, 4, 5;
- "ТАБЛ2" : 1-10, * , 5 - фрагмент табса ТАБЛ2, включающий первые 10 строк 5-го слоя;
- "ТАБЛ3" - все клетки 1-го (по умолчанию) слоя таблицы ТАБЛ3.

Иногда требуется перечислить ряд значений атрибутов.

В этом случае можно использовать перечень в виде одностороннего списка значений, разделённых знаками препинания.

Такие ситуации часто встречаются при определении атрибутов, относящихся к отдельным фрагментам (полям) спецификации.

При этом бывает удобнее разместить их значения в разных клетках, чтобы их расположение было связано с теми фрагментами С-формы, которые они описывают. В этом случае атрибут будет представлен ссылкой на фрагмент Z-формы, где перечислены его значения.

Ссылка на фрагмент определяет координаты клетки или множества клеток, где содержится значение атрибута, и записывается в формате, аналогичном принятому в языке Т-кон способу адресации клеток, столбцов и строк.

Для имен атрибутов, идентификаторов значений, а также для ключевых слов ДЛЯ, В и = можно (на этапе порождения версии языка Т-форм для целевой системы) ввести синонимы, более удобные для пользователя.

Начальная форма. В базе знаний хранится таблица с фиксированным именем - начальная форма.

Она хранит полное описание некоторой спецификации наиболее распространенного типа.

Полное в том смысле, что в нём определены все возможные атрибуты С-формы.

Любая созданная инженером задачного знания Z-форма также содержит определения атрибутов, но здесь уже необязательно представлять полный список - недостающие атрибуты будут заимствованы из начальной формы.

Таким образом, описание С-формы сводится к описанию отличий новой спецификации от определённой в начальной форме.

[Содержание <](#)

Приведённые рассуждения относятся к случаю, когда значения всех атрибутов заданы текстовыми командами.

Если инженер задачного знания хочет вынести значения некоторых атрибутов в отдельные клетки, то в начальной форме соответствующие текстовые описания атрибутов должны содержать ссылки на клетки формы, где записаны значения. Таким образом, реализуется формо-ориентированная запись параметров С-формы.

В общем случае процесс поиска значений атрибутов выглядит следующим образом:

1. определение атрибута ищется в Z-форме; если оно не найдено, то выполняется п. 2; если найдено определение ссылки на значение, то - п.3;
2. это же определение ищется в начальной форме; если найдено определение ссылки на значение, то - п.3;
3. по ссылке производится обращение к соответствующей клетке Z-формы спецификации; если клетка пуста, то - п.4;
4. производится обращение к клетке начальной формы с теми же координатами.

Описание расположения и содержания полей спецификации. Спецификация, как правило, составляется из нескольких элементов.

Для удобства описания их размещения инженеру задачного знания предоставляется следующая возможность: некоторый фрагмент Z-формы (прямоугольное связное множество клеток) определяется как карта расположения элементов.

Столбцы и строки в этом фрагменте вместе с заданными для них в Z-форме размерами будут играть роль координатной сетки.

В клетки карты расположения записываются определения элементов С-формы.

Содержимое поля элемента может быть задано:

- выражением, состоящим из текстовых констант и ссылок на клетки,
- ссылкой на фрагмент таблицы,
- ссылкой на файл ОС,
- вызовом самостоятельной С-формы.

Кроме того, в клетке можно дополнительно определить некоторые атрибуты, которые будут действовать только в её пределах.

Форматирование данных. В С-форме поддерживаются следующие типы данных:

- указатель;
- вложенный табс;
- текст (и его разновидности, задаваемые вслед за символом "/", такие как: параметр, связь, условие, вектор, матрица, программа);
- число.

Для каждого типа данных определяются правила форматирования при выводе в С-форму.

Использование параметров при описании С-форм. Для обеспечения гибкости применения форм в языке реализован механизм передачи в Z-форму параметров на этапе запуска задания на построение С-формы.

В качестве параметров можно передавать имена таблиц и символьные строки, которые будут подставляться в записанные в Z-форме определения.

Для описания числа и типов передаваемых параметров служит предложение ПАРАМ.

[Содержание <](#)

Оно содержит спецификации параметров, состоящие из двух элементов: идентификатора типа параметра (# имя табса, & - строка символов, заключённая в апострофы, % - числовое значение) и номера (целое число от 1 до максимального, определённого при порождении).

Спецификации параметров разделяются пробелами, знаками препинания, а также любыми словами.

Слова улучшают читаемость Z-формы и используются для формирования строки запроса (результатом интерпретации которой являются параметры С-формы).

Перечисленные в ПАРАМ параметры могут использоваться для записи в Z-форме ссылок на объекты, содержимое которых специфицируется, текстовых и числовых значений, подставляемых в текст элементов, значений общих атрибутов С-формы.

Версия языка Z-форм, включаемая в состав порождаемой системы. При порождении целевой системы требуется ввести данные в следующие таблицы:

- параметров,
- начальной формы,
- синонимов имен.

Система порождения хранит в базе знаний заполненные образцы таких таблиц. Инженеру задачных знаний не обязательно вводить в них данные заново: достаточно скорректировать имеющиеся.

Содержимое таблицы параметров. Здесь записывается перечень требуемых возможностей, которые должен предоставлять пользователю язык Z-форм, и определяются предельные размеры С-форм.

Исключение ненужных функций, минимизация разности между заданными предельными и практически возникающими размерами С-форм будут способствовать уменьшению объема интерпретатора, повышению его быстродействия.

Содержимое начальной формы. В начальной форме для каждого атрибута должно храниться либо определение его значения, либо ссылка на клетку Z-формы (где это значение записано).

Содержимое таблицы синонимов. Здесь размещены синонимы названий атрибутов и ключевых слов языка.

Синонимы можно употреблять при описании С-форм так же, как и изначальные названия.

Количество используемых синонимов определяется пользователем.

Язык Т-кон

Язык Т-кон - интерактивный табс-ориентированный КАК-язык.

В системе порождения используется для записи условий выполнения межзадачных связей и создания заготовок программ.

Служит прообразом языка ВЫЧ табс-ориентированных вычислений.

Программа, написанная на языке Т-кон, размещается в слое табса.

Именно в этом смысле Т-кон ориентирован на табс-форму.

Основная структура данных в Т-кон - слой табса (далее называемый таблицей).

Допускается также работа с элементами таблиц: строками, столбцами и клетками.

Тот факт, что программы на языке Т-кон представляют собой табс-слои, важен в том смысле, что их можно обрабатывать с помощью других Т-кон-программ.

[Содержание <](#)

Таким способом автоматизируется работа по созданию Т-кон-программ.

Работая с интерпретатором Т-кон, инженер задачного знания имеет возможность видеть на экране (через табличное окно) обрабатываемую таблицу.

Результаты выполнения командных строк отображаются в соседнем табличном окне (по желанию пользователя).

Для хранения промежуточных данных и для передачи данных между табсами используются контейнеры, представляющие собой табличные элементы, которые существуют независимо от обрабатываемой таблицы.

Содержимое контейнеров также может быть показано в специально выделенном для этого окне.

При обработке выбранного слоя текущего табса доступны другие слои этого табса, а также любые другие табсы, существующие в системе.

В командной строке может быть записано либо одно-, либо двухместное выражение, определяющее вычисление некоторого табличного элемента, либо - команда.

Т-кон предоставляет широкий набор базовых команд, в том числе команды манипулирования табсами, просмотра табсов, определения формата представления вычисляемых данных, ограничения области действия, определения текущего слоя, открытия внешних табсов, просмотра контейнеров, ввода/вывода, вычисления условий над табличными элементами, условного выполнения, определения режимов работы подпрограмм (запуска, паузы и продолжения с места, где выполнение было прервано).

В командных строках допускаются комментарии.

Для передачи управления внутри программ в языке Т-кон реализованы команды перехода и вызова подпрограмм.

Переход возможен на метки внутри текущей клетки или на другие клетки таблицы.

Т-кон-подпрограммы могут использовать формальные параметры.

Т-кон содержит в своем составе подсистему расширения языка (которая предоставляет возможность расширения набора команд).

Расширяющие команды реализуются посредством Т-кон-подпрограмм особого вида.

Для добавления новой команды необходимо ввести в специальный табс описания расширений имя команды, описание её синтаксиса (на ориентированном на язык Т-кон метаязыке), а также - имя исполняющей Т-кон-подпрограммы.

Возможность работы в режиме выполнения подпрограмм используется при разработке программ реализации условий межзадачных связей.

Наиболее часто встречающиеся условия реализуются в виде подпрограмм, которые хранятся в базе данных под знаком точки зрения их владельца.

Когда нужно создать некоторую новую программу, реализующую проверку условий межзадачной связи, используются подпрограммы из базы.

Работа с Т-кон максимально приближена к интуитивным представлениям человека о работе с таблицами.

[Содержание <](#)

Описание базового уровня языка Т-кон. Т-кон в процессе работы использует следующие типы экраных окон: табличное, командное, показа контейнеров, сообщений об ошибках, ввода в Т-кон-подпрограммы, вывода из Т-кон-подпрограмм и окно помощи.

Через табличное окно можно просматривать обрабатываемую таблицу и наблюдать результаты выполнения команд.

В командном окне инженер задачного знания вводит команды или исправляет ранее введенные команды.

Здесь же показываются (взятые на выполнение из вызванной Т-кон-подпрограммы) командные строки.

Окно показа контейнеров используется для визуализации контейнеров.

В зоне сообщений об ошибках показываются сообщения о синтаксических ошибках в командах и сообщения об ошибках в процессе исполнения.

Т-кон-подпрограммы могут вести диалог с инженером задачного знания с помощью команд ввода/вывода.

Пользователь может в любой момент прервать работу Т-кон-программы и перейти к редактированию таблицы, показываемой в табличном окне.

Т-кон работает со следующими объектами: табсами, слоями табсов, табличными элементами (строками, столбцами, клетками таблиц; контейнерами-векторами и контейнерами-скалярами), числами, символьными строками, символом "пусто", а также - с переменными Т-кон-подпрограмм.

Контейнеры-скаляры могут хранить числа, имена таблиц, символьные строки, а также - содержимое любых клеток таблиц.

В контейнеры-векторы можно заносить строки/столбцы таблиц.

Переменные Т-кон-подпрограмм могут быть трёх типов: имя таблицы, указатель и часть командной строки.

Переменные типа часть командной строки могут использоваться только в Т-кон-подпрограммах, реализующих расширение языка.

Табсы указываются по имени, заключенному в кавычки.

Табличные элементы идентифицируются спецсимволом элемента (L - строка, G - столбец, C - клетка, V - контейнер-вектор, S - контейнер-скаляр), за которым должны идти указатели элемента.

Указатель может быть номером, заголовком, спецсимволом, обозначающим последний или предпоследний элемент в ряду (символы T и P), переменным указателем, комбинацией заголовка и переменного указателя.

Строки, столбцы таблицы имеют один указатель.

Клетки таблицы определяются двумя указателями (строки и столбца, разделёнными запятой).

Переменный указатель представляет собой контейнер-скаляр или переменную типа указатель, заключённые в круглые скобки.

Указатель контейнера может быть только номером, который, в отличие от остальных видов табличных элементов, может быть пустым (равным нулю).

Перед заголовком в квадратных скобках может находиться указатель вектора, который будет считаться заголовочным для данного элемента.

[Содержание <](#)

Числа могут быть целыми или - с десятичной точкой.

Символьные строки заключаются в апострофы.

В символьных строках, заголовках и именах таблиц допускаются сокращения двух видов:

- до конца текущего слова - символ ".:";
- до конца текущего слова и вместо любого количества слов - символ "**".

Переменные Т-кон-подпрограмм идентифицируются спецсимволом (# - переменная типа имя таблицы, & - переменная типа указатель, \$ - переменная типа часть командной строки), за которым должен следовать номер переменной.

Структура командной строки выглядит следующим образом: [<МЕТКА>] [<ВЫРАЖЕНИЕ ИЛИ КОМАНДА>] [<КОММЕНТАРИЙ>]

В конце командной строки может находиться комментарий.

Символ начала комментария - восклицательный знак "!".

В начале строки может стоять числовая метка - число, за которым следует двоеточие.

В арифметическом выражении над скалярными операндами необязателен адрес результата, в этом случае вычисления идут в режиме калькулятора.

Если операндом в операции является вектор, то результат операции должен быть вектором того же типа.

Не допускаются операции над векторами различных типов.

Одноместная операция копирования - ">" применима и к числовым, и к текстовым данным, находящимся в табличных элементах.

Одноместная операция размножения применяется для занесения содержимого указанной клетки (строки, столбца) в другие клетки (строки, столбцы), если содержимое - числа или "пробел".

В Т-кон-подпрограммах, реализующих расширения языка, может использоваться ограничитель выражения, представляющий собой указатель, стоящий между обратными косыми чертами.

Ограничитель выражения приводит к выполнению векторного выражения только над одним элементом из векторов, участвующих в выражении.

Внешними таблицами (по отношению к текущей обрабатываемой таблице) являются другие слои текущего табса и слои других табсов.

Внешняя таблица определяется именем табса и, если нужно, указателем слоя в нём. Для однажды упомянутой внешней таблицы можно в последующих командах опускать её имя и просто ставить двоеточие.

Данные из внешних таблиц могут быть скопированы в контейнеры или в элементы текущей таблицы, а также использоваться в арифметических операциях.

В такой операции могут участвовать данные только из одной внешней таблицы, и они должны стоять первыми в командной строке.

Функции применяются к элементам таблиц, и их результаты могут заноситься в табличные элементы.

Для просмотра обрабатываемой таблицы в табличном окне служат команды: Ln, Gm, Sp,t, где n и t - номера строк и столбцов текущей таблицы.

[Содержание <](#)

Команда Ln нужна для сдвига табличного окна по обрабатываемой таблице в вертикальном направлении (таким образом, что строка с номером n станет первой строкой, показываемой в табличном окне).

Аналогично команда Gm сдвигнет табличное окно в горизонтальном направлении (так, что столбец с номером m станет первым показываемым).

После выполнения команды Cp, m клетка с указанными координатами окажется в левом верхнем углу табличного окна.

Просмотр контейнера осуществляется после ввода его имени; при этом в контейнере-векторе визуализируется только та его часть, которая соответствует текущему положению табличного окна.

Команды перехода и проверки условий. Команда перехода осуществляет передачу управления при выполнении Т-кон-подпрограммы.

Управление может быть передано либо на числовую метку в этой же клетке, либо на другую клетку Т-кон-программы.

В командах проверки условия и условного выполнения (в зависимости от значения входящего в них условия) либо выполняется одна из двух альтернативных команд (в команде условного перехода - "ЕСЛИ"), либо визуализируется некоторое значение (в команде проверки условия - "?").

Т-кон допускает простейшие двухместные условия, в которых могут использоваться табличные элементы, в том числе элементы внешних таблиц, числа, символьные строки и символ "пусто".

Допустимы следующие виды отношений: ">" - больше, ">=" - больше либо равно, "=" - равно, "<=" - меньше либо равно, "<" - меньше, "<>" - не равно, "<!>" - строго не равно и "B" - входит.

В команде условного выполнения (в зависимости от значения условия) выполняется команда, стоящая либо в части "ТО", либо в части "ИНАЧЕ".

Команды работы с табсами. К командам работы с табсами относятся команды открытия, закрытия, создания, удаления, переименования.

Команда открытия табса приводит к тому, что первый слой указанного табса визуализируется в табличном окне и становится текущей обрабатываемой таблицей, а предыдущий обрабатываемый табс закрывается.

Команда закрытия табса - к тому, что текущий обрабатываемый табс закрывается, его имя заносится в указанную в команде переменную.

Команда создания - к созданию табса с новым именем и заданными размерами, а команда удаления - к исключению табса с указанным именем.

Команда переименования позволяет изменить имя табса.

Подпрограммы. К командам работы с Т-кон-подпрограммами относятся команды запуска Т-кон-подпрограммы, описания формальных параметров, останова, установки и снятия пошагового режима выполнения подпрограммы.

Команда <ВЫЧИСЛИТЬ> запускает выполнение соответствующей Т-кон-подпрограммы.

Выполнение начинается с первой непустой клетки таблицы.

Если нет явных команд передачи управления, то по исчерпании текущей клетки управление передается следующей клетке в столбце.

[Содержание <](#)

Выполнение подпрограммы завершается или при встрече команды останова, или при попадании на пустую клетку, или при достижении последней клетки таблицы, куда записана Т-кон-подпрограмма.

Из одной подпрограммы можно вызывать другие Т-кон-подпрограммы. Определяемые в процессе работы Т-кон-подпрограммы текущая таблица, контейнеры и установленная точность являются локальными (только для вызванной Т-кон-подпрограммы).

Фактическими параметрами могут быть имена таблиц, числа, указатели табличных элементов (перед указателями для наглядности могут стоять спецсимволы этих элементов), а также переменные Т-кон-подпрограмм.

Команда <ПАРАМЕТРЫ> описывает формальные параметры, используемые в Т-кон-подпрограмме.

Формальные параметры перечисляются через запятую.

Обычные подпрограммы могут содержать только два типа переменных в качестве формальных параметров: имя таблицы и указатель.

Т-кон-подпрограммы, реализующие расширения языка, могут применять тип часть командной строки в качестве формальных параметров.

Соответствие фактических и формальных параметров устанавливается по позиционному принципу с учётом типов: формальному параметру некоторого типа ставится в соответствие фактический параметр того же типа.

Команда <ПРОДОЛЖИТЬ> определяет выполнение прерванной Т-кон-подпрограммы.

При попадании управления на команду <СТОП> выполнение текущей Т-кон-подпрограммы заканчивается, управление возвращается в предыдущую Т-кон-подпрограмму, из которой была вызвана текущая, или, если такой нет, то интерпретатор Т-кон переходит в состояние ожидания ввода команды.

Команда <ШАГ> устанавливает режим выполнения Т-кон-подпрограмм в пошаговом режиме.

В этом режиме каждая взятая на выполнение из Т-кон-подпрограммы команда показывается в окне ввода команд и запрашивается ответ (продолжить выполнение; прервать выполнение; выйти из пошагового режима и продолжить выполнение).

Команду можно использовать как в интерактивном режиме, так и в Т-кон-подпрограмме.

При попадании управления на команду <НЕШАГ> отменяется пошаговый режим работы.

Ввод-вывод. Команды ввода-вывода позволяют Т-кон-подпрограммам вести диалог с пользователем.

Команда <ВВЕСТИ> позволяет ввести некоторое значение.

Введённое значение может быть занесено в переменную типа имя таблица или указатель, или в клетку таблицы.

В команде <ВЫВЕСТИ> перечисляются объекты, подлежащие выводу на экран.

Выводиться могут: имена таблиц, символьные строки, содержимое клеток таблиц и переменные Т-кон-подпрограмм.

[Содержание <](#)

Подсистема расширения. Подсистема расширения поддерживает дополнение языка новыми командами (обеспечивая возможность его сравнительно легкой настройки на требования конкретной предметной области).

При этом, как правило, уровень команд языка растёт, что в итоге приводит к значительному повышению продуктивности при составлении Т-кон-подпрограмм.

❖ С помощью этой подсистемы язык Т-кон расширен командами, реализующими основные операции алгебры Кодда и процедуры сортировки. ❖

Средства расширения поддерживают синонимию языковых конструкций, что способствует созданию удобного интерфейса с системой.

Новые команды и конструкции языка реализуются посредством Т-кон-подпрограмм с параметрами, а их синтаксис описывается на специальном метаязыке.

Описания расширений хранятся в базе знаний в специальной таблице, где их находит Т-кон-интерпретатор при встрече расширяющей команды.

ГЕНПАК: система И-порождения пакетов программ

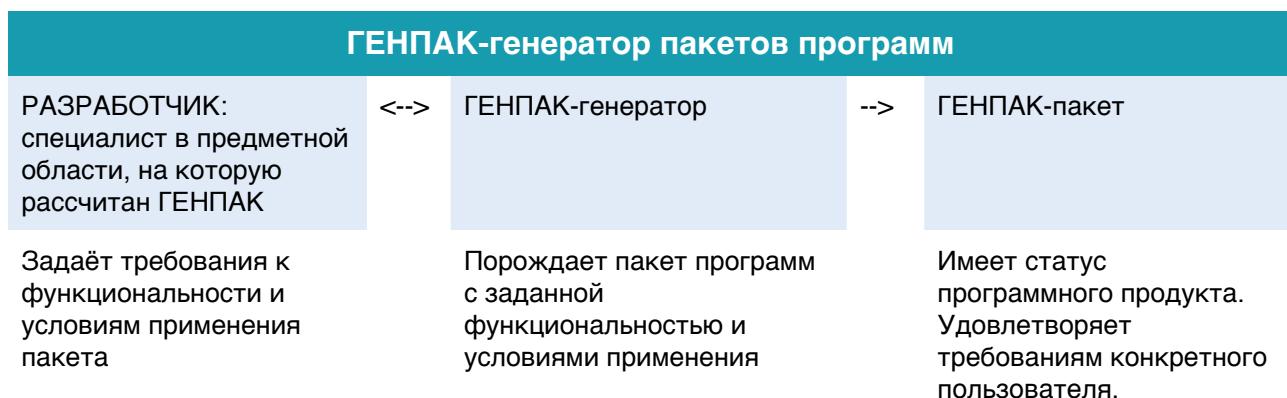
В этом разделе дано описание реализованной системы порождения (системы ГЕНПАК), построенной по принципам, определенным для табс-ориентированной порождающей целевой системы.

Описываемая версия ГЕНПАК - программный продукт.

Основное внимание сосредоточено на функциональной характеристике и потребительских свойствах ГЕНПАК (типичной системы порождения целевых программных систем).

Система порождения ГЕНПАК (ГЕНПАК-генератор) - инструментальная система разработки интегрированных пакетов программ с заданным набором функций для решения задач учёта и планирования.

ГЕНПАК служит средством повышения производительности изготовления надёжных пакетов программ с заданной функциональностью, интерфейсом пользователя и системными условиями применения.



ГЕНПАК является диалоговой программной системой, посредством которой системный аналитик производит архитектурно-подобные пакеты для заданной предметной области.

ГЕНПАК рассчитан на порождение пакетов для решения задач учета в различных областях и задач оптимального планирования.

Около 80% компонент генератора ГЕНПАК не зависят от особенностей операционных систем и аппаратных средств.

Степень сложности порождённых ГЕНПАК-пакетов зависит от задачной области и условий применения, что в итоге определяет системное и функциональное наполнение пакетов.

Каждый ГЕНПАК-пакет представляет собой диалоговую систему, предназначенную для решения задач заданной задачной области при заданных условиях применения.

ГЕНПАК-пакеты и обычные: характерные отличия

	ГЕНПАК-пакеты	Обычные пакеты
Соответствие требованиям конкретного пользователя	Соответствует	Не соответствует: рассчитан на некоторого обобщённого пользователя
Ввод/вывод	Компактный; в выбранной пользователем форме	Обычно: не компактный, в жёсткой форме
Функциональная избыточность	Небольшая	Как правило: большая
Надёжность	Высокая	Не обязательно высокая
Учёт программно-аппаратной специфики машины	Всегда	Не всегда
Обучение, помощь	Учтена подготовка пользователя; помощь удобна и конкретна	Рассчитаны на некоторого обобщённого пользователя
Стоимость разработки	Невысокая	Высокая

В состав каждого ГЕНПАК-пакета входят: диалоговый монитор, система управления табс-базами данных и интерфейсная компонента (обеспечивающие унифицированное взаимодействие между задачами, основанное на механизме передачи сообщений посредством табсов, и гибкий интерфейс пользователя).

ГЕНПАК-пакеты могут иметь функциональность генераторов приложений, если в их состав включены компоненты ВЫЧИСЛИТЕЛЬ (представляющая собой специальную систему программирования на табс-ориентированном языке ВЫЧ) и ФОРМАТ (генератор произвольных отчётов с развитым языком описания выходных документов).

Применение генератора ГЕНПАК обеспечивает получение надёжных и дешёвых пакетов программ для решения массовых задач учёта и планирования силами специалистов, не обязательно имеющих высокий уровень ИТ-подготовки.

Важная особенность генератора ГЕНПАК (отличающая его от других систем автоматизации разработки пакетов) - в том, что порождаемые пакеты могут стать частью прикладного программного обеспечения различных конфигураций целевых машин, принадлежащих к разным семействам ЭВМ.

ГЕНПАК-пакеты могут быть реализованы на различных языках программирования (из заданного множества языков) и могут работать под управлением выбранной операционной системы (из заданного множества ОС).

ГЕНПАК построен как интерактивная расширяющаяся система.

[Содержание <](#)

Расширение генератора возможно как в части системных компонент, так и - компонент функционального наполнения.

Расширение не сопровождается коррекцией существующих компонент, что обеспечивается реализацией в его архитектуре принципов инф-конструирования целевых программных систем.

База знаний генератора ГЕНПАК содержит правила, необходимые для конструирования пакетов методом И-порождения.

При работе с генератором ГЕНПАК в диалоговом режиме системный аналитик описывает порождаемые ГЕНПАК-пакеты.

Эксплуатация генератора ГЕНПАК осуществляется системным аналитиком, являющимся специалистом в той области, для решения задач которой должен быть получен пакет.

Эксплуатация порождённого ГЕНПАК-пакета начинается с обучения пользователя в режиме диалога.

При этом обеспечивается высокий уровень защищённости от ошибок и адекватное текущей ситуации информационное обслуживание.

Удобное для пользователя взаимодействие с генератором ГЕНПАК и ГЕНПАК-пакетами обеспечивается соответствующей настройкой интерфейсных компонент.

Назначение и общая характеристика

ГЕНПАК обеспечивает порождение:

- интерфейсной компоненты пакета,
- системы управления табс-базами данных (СУТБД) пакета,
- формирователя отчетов,
- компоненты ВЫЧИСЛИТЕЛЬ,
- функционального наполнения пакета,
- обучающей системы пакета.

ГЕНПАК рассчитан на:

- обучение работе с генератором,
- настройку средств конструирования пакетов,
- описание требований к порождаемому пакету,
- тестирование пакета (проверку на соответствие требованиям заказчика).

ГЕНПАК-пакеты обеспечивают:

- диалоговое обучение работе с пакетом;
- решение в диалоговом режиме задач учёта и планирования, определённых описанием предметной области [с использованием ТБД, средств редактирования табличных документов, а также - обработки и представления табличных отчетов];
- расчёты по ВЫЧ-программам;
- прикидочные расчёты в режиме калькулятора [без занесения или с занесением результатов в ТБД];
- сохранение результатов промежуточных расчётов в специальных контейнерах с возможностью их оперативного использования;
- использование в качестве исходных данных содержимого табличных документов, хранящихся в ТБД;
- занесение результатов расчётов в любой указанный документ, хранящийся в ТБД;
- создание табсов, размеры которых ограничены только разрядностью ЭВМ и располагаемой внешней памятью;

[Содержание <](#)

- просмотр содержимого табсов через табличное окно, формируемое пользователем, с возможностью редактирования содержимого клеток;
- подсказки и помощь в затруднительных для пользователя ситуациях.

Состав

ГЕНПАК, как система порождения архитектурно-подобных интегрированных пакетов с заданным набором функций, состоит из компонент двух классов: системных и функциональных.

Системные компоненты обеспечивают функционирование генератора (как системы в целом), функциональные - воплощают целевое назначение генератора, реализуя процессы порождения ГЕНПАК-пакетов.

Состав системных компонент:

- диалоговый монитор,
- система управления табс-базами данных,
- интерфейсная компонента,
- формирователь отчетов.

Функциональные компоненты генератора предназначены для порождения:

- компонент ПЛАН, ВЫЧИСЛИТЕЛЬ, ФОРОТ,
- интерфейсной компоненты пакета,
- ТБД.

Состав порождаемых пакетов

Каждый ГЕНПАК-пакет имеет состав, архитектурно-подобный составу генератора ГЕНПАК.

ГЕНПАК-пакет состоит из системного наполнения (СН) и функционального наполнения (ФН).

Минимальный состав СН:

- диалоговый монитор,
- ТБД,
- интерфейсная компонента.

В максимальный состав входит также компонента ФОРОТ.

Минимальный состав ФН может включать только компоненту ВЫЧИСЛИТЕЛЬ (при ориентации только на задачи учета) или только компоненту ПЛАН (при ориентации только на задачи планирования).

Начальный состав задач учёта, их размеры и понятийные оболочки определяются на этапе порождения компоненты ВЫЧИСЛИТЕЛЬ.

Силами пользователя ГЕНПАК-пакета состав задач может быть расширен, так как компонента ВЫЧИСЛИТЕЛЬ представляет собой табс-ориентированную систему программирования табличных вычислений и предоставляет возможности расширения библиотеки ВЫЧ-подпрограмм.

Кроме того, пользователю предоставлена возможность расширения языка ВЫЧ компоненты ВЫЧИСЛИТЕЛЬ.

Состав задач планирования, их размеры и понятийные оболочки определяются на этапе порождения компоненты ПЛАН.

Максимальный состав ФН ГЕНПАК-пакета включает обе компоненты ВЫЧИСЛИТЕЛЬ И ПЛАН.

Система управления табс-базами данных

Для управления табс-базами данных в генераторе ГЕНПАК и ГЕНПАК-пакетах применяется компонента СУТБД.

Она позволяет создавать табс-структуры и манипулировать ими (модифицировать, удалять, копировать).

СУТБД в генераторе ГЕНПАК и продуцируемых им пакетах служит и средством связи между остальными компонентами.

В клетках табсов могут содержаться данные следующих типов:

- строка символов,
- числовые данные,
- указатель (символьная ссылка),
- вложенный табс.

Строка символов: этот тип данных позволяет хранить последовательности символов.

Числовые данные: заносятся в табс и выдаются СУТБД в символьной форме.

Для их хранения используется уплотнённая двоично-десятичная кодировка (2 знака в байте). Набор допустимых символов содержит: ' ', '.', '-' , '+', 'E', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'.

Указатель: этот тип определяет символьную строку, которая является указателем на табс или клетку табса.

Работа с указателями зависит от установленного режима.

Если не установлен режим интерпретации указателей, то с ними можно работать как с обычными символьными данными; в противном случае выполняется работа с данными, на которые указывает ссылка.

Для того, чтобы попасть на указанный табс-адрес, используется операция "переход по указателю".

При попадании в клетку типа указатель выполнение названной операции приведёт к изменению привязки канала.

Координаты сменяемого табса запоминаются в точке трассы.

Возврат в неё возможен с помощью операции "возврат в предыдущую точку".

Указатели могут храниться в ТБД в виде "прямых ссылок".

В этой форме имя табса не может быть визуализировано (указатель хранится в виде двоичного индекса).

Такой способ позволяет обходиться без операций поиска по каталогам при интерпретации указателей, что экономит время доступа к данным.

Символьные ссылки могут быть преобразованы в прямые.

Вложенный табс: вложенным называется табс, размещённый в клетке другого табса и доступный при попадании в неё.

Каталоги в табс-базах данных представляют собой табсы определённой структуры, заполнение клеток которых выполняется посредством СУТБД.

В клетках первого слоя табс-каталога размещаются: в первом столбце - имена вложенных табсов (символьные строки), в последующих столбцах - служебные данные.

Строка табс-каталога содержит описание одного табса.

Изменения в каталог вносятся с помощью специальных функций СУТБД.

[Содержание <](#)

Ядро СУТБД - самостоятельная задача (в мультипрограммной системе). Как самостоятельные задачи реализованы и программы, выполняющие функции защиты и сжатия данных.

Для программ, взаимодействующих с СУТБД, существуют библиотеки процедур, реализующих функции обмена сообщениями с ядром.

При обращении к ним из переданных параметров создаётся запись, которая отправляется модулю-почтальону, который обеспечивает их пересылку ядру СУТБД. Модуль-приёмник (в составе ядра СУТБД) ставит в очередь полученный запрос.

После приёма запросов они попадают к диспетчеру, который определяет содержание запрошенной работы, и передает данные соответствующей подпрограмме ядра.

Подпрограмма может, в свою очередь, вызывать другие подпрограммы ядра.

После обработки результат отправляется задаче, сделавшей запрос, по тому же маршруту, но в обратном направлении.

Кроме работы с клетками табса, имеющими координаты, СУТБД, обеспечивая свои внутренние потребности, работает с клетками для хранения служебных данных (имён табсов, дат последней коррекции и др.).

Важным понятием СУТБД является канал.

Для пользовательских программ канал служит средством связи с табсом.

В процессе работы пользователь может применить для связи с ТБД одновременно несколько каналов.

Можно управлять связью, задавая табс и координаты в нём (чтобы получить доступ к требуемым данным).

Основные функции СУТБД:

- читать (получать от СУТБД тип клетки и/или её содержимое),
- писать (задавать тип клетки и/или заносить её содержимое),
- перемещаться (устанавливать в канале координаты требуемой клетки).

Дополнительные функции:

- перемещать данные (изменять координаты отдельных клеток и их групп);
- контролировать запись в табс или в отдельную клетку.

Функции СУТБД разделены на группы.

Работа с табсами (по их именам):

- создать табс и войти в него,
- удалить табс из каталога,
- переименовать табс,
- войти в табс,
- выйти из табса.

Группа функций выбора данных внутри табса (определение координат клетки):

- определить клетку в табсе или в канале (задать координаты или имя клетки),
- получить координаты или имя текущей клетки.

Работа с типом данных табс-клетки:

- задать тип клетки,
- прочитать тип клетки,
- очистить клетку (задать тип не определен и стереть данные).

[Содержание <](#)

Работа с данными клетки разделяется на группы функций.

Первая - позволяет:

- записать в клетку группу байтов заданной длины (стерев бывшие там данные);
- добавить к записанным данным группу байтов заданной длины.

Вторая группа обеспечивает чтение:

- объёма данных табс-клетки;
- группы байтов, начиная с заданного;
- символа;
- строки.

Группа так называемых комбинированных функций позволяет:

- занести тип и данные в клетку;
- прочитать тип, объём данных и группу байтов из указанной клетки.

Функции перемещения данных в ТБД обеспечивают:

- копирование типа и данных из клетки в клетку,
- удаление/вставку строки/слоя/столбца.

Порождение СУТБД как компоненты производимого пакета производится после порождения остальных компонент.

При порождении функциональных компонент, компоненты ФОРП и интерфейсной компоненты полностью определяются необходимый в пакете набор функций СУТБД. Данные, полученные на предшествующих шагах порождения, сохраняются в ТБД до завершения процесса порождения (поскольку при порождении СУТБД используются данные об уже порождённых компонентах).

Это позволяет включить в СУТБД пакета набор функций, покрывающий необходимые потребности манипулирования данными для всех его компонент.

Взаимодействие с пользователем

Интерфейс с пользователем построен одинаково в генераторе ГЕНПАК и в порождаемых им ГЕНПАК-пакетах.

Различие - в том, что генератор ГЕНПАК содержит средства порождения интерфейсных компонент ГЕНПАК-пакетов.

Взаимодействие пользователя с компонентами генератора ГЕНПАК (или ГЕНПАК-пакета) реализуется посредством ИК (интерфейсной компоненты) и рассматривается как процесс специфирования табсов и редактирования содержимого их клеток.

Пользователь поочередно взаимодействует с редакторами: во время этой работы могут подключаться соответствующие компоненты генератора ГЕНПАК (или ГЕНПАК-пакета), которые будем называть взаимодействующими компонентами (ВК).

В состав ИК входят:

- редактор текста,
- табс-редактор,
- диспетчер окон.

Редактор текста предназначен для типовых работ с текстом в ИК:

- редактирование текста,
- просмотр текста,
- отображение текстовых сообщений.

Пользователь может работать с редактором (используя окно ИК) только с помощью клавишных команд в следующих режимах:

- ввод строки текста,
- просмотр текста,

[Содержание <](#)

- редактирование текста.

В режиме ввода строки текста редактор разрешает ввод и редактирование только одной строки (в этом режиме он вызывается табс-редактором).

В режиме просмотра текста разрешена только его визуализация (любые изменения запрещены).

Редактор текста вызывается табс-редактором при просмотре содержимого клеток табсов.

В режиме редактирования текста разрешена не только его визуализация, но и изменение.

Редактор текста вызывается табс-редактором при редактировании содержимого клеток табсов.

При этом ограничения на вводимые данные зависят от типа редактируемых клеток:

- для клеток типа строка символов разрешены все обычные операции редактора текста;
- для клеток типа числовые данные разрешено вводить только те символы, которые могут появиться в символьном представлении числа в ТБД.

Редактирование текста осуществляется в специальных внутренних текстовых буферах ИК.

Обеспечиваются следующие группы команд:

- перемещение курсора в окне (влево, вправо, вверх, вниз);
- логическое перемещение окна по текстовому буферу (сдвиг окна в начало/конец строки, сдвиг окна влево или вправо на знак, вверх или вниз на строку, в начало/конец буфера);
- удаление/вставка знаков, строк и частей строк;
- поиск по образцу.

Табс-редактор предназначен для операций с табами в ТБД.

Обеспечиваются следующие режимы работы:

- ввод и исполнение текстовых команд;
- просмотр текущего табса;
- редактирование содержимого текущей клетки (в пределах табличного окна).

Табличное окно -разделено на клетки набором вертикальных и горизонтальных прямых (сеткой).

В режиме ввода и исполнения текстовых команд обеспечиваются операции задания имени текущего табса, создания, удаления табсов, задания размеров табсов, распечатки каталога ТБД, определения текущих параметров редактора. При этом для ввода текстовых команд вызывается редактор текста.

В режиме просмотра текущего табса пользователь имеет возможность просматривать (через табличные окна) табсы и изменять типы их клеток.

Обеспечиваются операции поиска текста по образцу в клетках текущего табса, перемещения табличного окна по табсу и модификации типов клеток.

Из этого режима пользователь может перейти в режим редактирования содержимого текущей клетки.

В табличном окне строится визуальное представление прямоугольного участка смежных клеток одного слоя табса.

[Содержание <](#)

При этом в каждой клетке табличного окна отображается видимая часть содержимого соответствующей клетки табса.

В текущей клетке табличного окна организуется редактирование и просмотр ранее невидимой части её содержимого.

Пользователь может перемещать табличное окно как по слоям табса, так и внутри одного слоя.

В этом режиме поддерживается работа с контейнером - специальным табсом для временного хранения содержимого клеток текущего табса.

Обеспечивается выполнение команд записи в контейнер и чтения его содержимого.

В режиме редактирования содержимого текущей клетки табс-редактор вызывает редактор текста, задавая соответствующий режим работы и параметры клетки табличного окна, внутри которой разрешено редактирование.

При редактировании содержимого клетки табса разрешены только действия, определяемые типом редактируемой клетки.

По команде "конец редактирования" осуществляется переход в режим просмотра текущего табса.

Взаимодействие ИК и ВК происходит по схеме, изображённой на следующем рисунке.

Для совместной работы ИК и ВК выделяется специализированный табс связи (ИК-ТАБС).

Этот табс является единственным объектом, используемым для взаимодействия ВК и ИК.

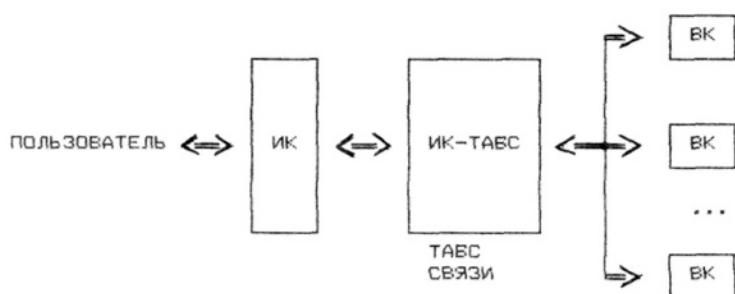
Их взаимодействие осуществляется путём записи команд и данных в ИК-табс, которую выполняет ВК.

Работа ИК полностью определяется данными, содержащимися в ИК-табсе.

Результаты работы ИК записываются в этот же табс для передачи их ВК в процессе порождения.

ИК-табс является результатом порождения ИК и отражает специфику выбранного интерфейса с пользователем.

Схема взаимодействия ИК и ВК



Так как работа ИК определяется содержимым ИК-табса, то при порождении в этот табс записываются тип и параметры физического терминала.

Спецификация табс-форм и формирование отчётов

Назначение форматора отчётов (ФОРТОТ) в генераторе

ГЕНПАК и ГЕНПАК-пакетах: представление данных, хранящихся в ТБД (в виде, удобном для анализа пользователем), и вывод отформатированных данных во внешнюю (относительно ТБД) среду.

Результат вывода называется отчетом.

Вывод данных может сопровождаться их форматированием.

Язык описания выходных документов ФОРТОТа представляет собой реализацию формо-ориентированного языка Т-форм.

[Содержание <](#)

Форма - это таблица ссылок и значений, описывающих содержимое и внешний вид выходных документов (отчётов), создаваемых компонентой.

ФОРПОТ служит интерпретатором содержимого таких форм.

Отчёт компонуется из фрагментов табсов с заданным размещением внутри отчёта.

Фрагментами могут быть клетки, столбцы, строки; сечения табса по слову, по строке, по столбцу (см. рисунок).

Форма отчёта: пример

	G1	G2	G3	G4
L1	ШИРИНА СТРАНИЦЫ РАЗДЕЛИТЕЛЬ В C1,2 ШИРИНА СТОЛБЦА В L2	=I=I-1		
L2	10	40	20	20

Атрибуты отчёта позволяют описать требуемый внешний вид документа (размеры страницы, форматы представления чисел, размеры строк, столбцов и т.д.) Формат записи атрибутов определён при описании языка Т-форм.

Клетка в первом столбце первой строки формы отчёта зарезервирована для хранения описаний атрибутов.

Каждому атрибуту в этой клетке соответствует ссылка (на клетку или множество клеток в форме, содержащих значение атрибута).

Ссылка записывается в формате, принятом в компоненте ВЫЧИСЛИТЕЛЬ для адресации клеток, столбцов и строк (\diamond C1,3; G6; L4; C2-5,3-6; G3-5; L1-10 \diamond)

Для имён атрибутов, идентификаторов значений, а также для ключевых слов ДЛЯ, В и = можно на этапе порождения компоненты ввести синонимы, более удобные для пользователя.

При записи имён и идентификаторов допускаются сокращения (\diamond вместо ФОРМАТИРОВАНИЕ - ФОРМАТИР \diamond).

ФОРПОТ: атрибуты отчётов

Атрибут	Определение атрибута	Определяемость для фрагмента	Форма записи значения
Ширина страницы	Макс. число знакомест строки	нет	число
Длина страницы	Макс. число строк	нет	число
Ширина столбца табл.	Макс. число символов (без учёта разделителей)	да	список пар чисел
Высота строки табл.	Мин. и макс. число строк символов в одной строке табл.	да	список пар чисел
Содержимое поля	Текст	да	описание текста
Разделители	Символы разделителей и границ	да	окаймление
Слияние клеток	Метод объединения клеток	да	идентификатор
Форматирование	Метод форматирования текста в поле отчёта	да	идентификатор
Число	Шаблон для преобразования чисел в таблицах	да	шаблон числа

ФОРМАТ: атрибуты отчётов

Параметр	Колич. и типы параметров отчёта	нет	строка запроса
Структура	Форма, служащая картой размещения элементов отчёта	нет	фрагмент формы

Описание размещения и содержания полей документа

Форма

СТРУКТУРА=L2-B			
V1>1 СОДЕРЖАНИЕ = ТЕКСТ 'РАСЧЕТ ЗАРАБОТНОЙ ПЛАТЫ ЗА '+ "ТАБЛ":1,2,1 ФОРМАТИРОВАНИЕ = ЦЕНТРИРОВАТЬ	>1 СОДЕРЖАНИЕ = ТЕКСТ 'Типовая форма N T-52'		
	СОДЕРЖАНИЕ = ТЕКСТ 'Цех(отдел)'	СОДЕРЖАНИЕ = ТЕКСТ 'Табельный N'	
>3 СОДЕРЖАНИЕ = ТЕКСТ 'Фамилия, И.,О. '+ "ТАБЛ":2,3,1			
>3 СОДЕРЖАНИЕ = ТЕКСТ 'Прфессия(должность) '+ "ТАБЛ":2,4,1			
>1 СОДЕРЖАНИЕ = ТЕКСТ 'Начислено'	>1 СОДЕРЖАНИЕ = ТЕКСТ 'Удержано'		
>1 СОДЕРЖАНИЕ = ТАБЛИЦА "НАЧИСЛ"	>1 СОДЕРЖАНИЕ = ТАБЛИЦА "УДЕРЖ"		
>3 СОДЕРЖАНИЕ = ТЕКСТ 'Сумма к выдаче '+ "ИТОГО":1,1,1			

Документ

РАСЧЕТ ЗАРАБОТНОЙ ПЛАТЫ за г.	Типовая форма N T-52 Цех(отдел) Табельный N															
Фамилия, И.,О. Прфессия (должность) Начислено																
Удержано																
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <th style="width: 30%;">Вид заработной платы</th> <th style="width: 10%;">Сумма</th> <th style="width: 30%;">Вид заработной платы</th> <th style="width: 10%;">Сумма</th> </tr> <tr> <td>1. Сдельно</td> <td>.....</td> <td>1. Аванс</td> <td>.....</td> </tr> <tr> <td>.....</td> <td></td> <td>.....</td> <td></td> </tr> <tr> <td>Всего начислено</td> <td></td> <td>Всего к удержанию</td> <td></td> </tr> </table>	Вид заработной платы	Сумма	Вид заработной платы	Сумма	1. Сдельно	1. Аванс		Всего начислено		Всего к удержанию	
Вид заработной платы	Сумма	Вид заработной платы	Сумма													
1. Сдельно	1. Аванс													
.....															
Всего начислено		Всего к удержанию														
Сумма к выдаче																

Предусмотрена возможность объединения в единое поле клеток, образующих прямоугольное связное множество. Достигается это следующим образом: в клетке, лежащей в верхнем левом углу прямоугольника, охватывающего M+1 клеток по горизонтали и N+1 клеток по вертикали, в начале первой строки должна быть запись: >M VN (>M или VN, если, соответственно, N=0 или M=0). Описания, введённые в левую верхнюю клетку, распространяются на все клетки, принадлежащие объединению клеток.

Есть возможность сформировать общие заголовки столбцов и строк, итоговые строки и др. средства табличного структурирования данных. Атрибут СЛИЯНИЕ КЛЕТОК определяет метод объединения клеток.

Предусмотрены следующие методы:

- СЛИЯНИЕ КЛЕТОК = ВЫКЛ - никакого объединения не производится;

- СЛИЯНИЕ КЛЕТОК = СПРАВА - к заполненной клетке присоединяются все соседние справа пустые клетки в том же ряду;
- СЛИЯНИЕ КЛЕТОК = СНИЗУ - к заполненной клетке присоединяются все соседние снизу пустые клетки того же столбца;
- СЛИЯНИЕ КЛЕТОК = СПРАВА_СНИЗУ - к заполненной клетке присоединяются сначала все соседние справа пустые клетки, затем соседние снизу подстроки пустых клеток;

[Содержание <](#)

- СЛИЯНИЕ КЛЕТОК = СНИЗУ_СПРАВА - к заполненной клетке присоединяются сначала соседние снизу пустые клетки, затем соседние справа пустые подстолбцы.

Формирование общих заголовков

Исходная таблица			Сформированный отчет (СЛИЯНИЕ КЛЕТОК=СНИЗУ_СПРАВА)		
Заголовок 1	Заголовок 2		Заголовок 1	Заголовок 2	
Подзаголовок 21	Подзаголовок 22		Подзаголовок 21	Подзаголовок 22	

В форме задаются размеры столбцов и строк, образующих карту расположения элементов отчёта.

Размеры поля, в котором размещается некоторый элемент, определяются, таким образом, суммарной шириной столбцов и суммарной высотой строк, объединяемых в этом поле.

В случае, когда отчёт состоит из одной таблицы, картой расположения элементов (по умолчанию) служит сама выводимая в отчёт таблица, а каждая её клетка может считаться отдельным полем.

В одном предложении для атрибута ШИРИНА СТОЛБЦА может быть задан список значений.

Первое значение в списке задает ширину крайнего левого столбца в определяемом фрагменте формы; второе (если указано) - ширину второго столбца и т.д. Последнее значение из списка задает ширину всех оставшихся столбцов в определяемом фрагменте формы.

Если ширина столбца не определена в форме, то она выбирается равной ширине соседнего слева столбца, либо (для крайнего слева) берется из начальной формы.

Высота строки таблицы обычно определяется объемом выводимых данных.

Часто бывает, что точные размеры столбцов и, особенно, строк указать нельзя, да и нежелательно.

В подобном случае пользователю даётся возможность определить в форме ограничения на величины столбцов и строк.

Размеры столбцов могут быть заданы диапазоном возможных значений (например, 5-10 задает размер столбца в пределах от 5 до 10 символьных позиций, 40 - минимальная ширина столбца 40 символов).

В этом случае ФОРМАТ сам выберет такие значения, чтобы документ уместился на странице и занимал её (по ширине) полностью.

Пример: запись ШИРИНА СТОЛБЦА ДЛЯ G1-6 = 5, 10-20, 40,5 - определит ширину столбцов отчета следующим образом: 1-й столбец - 5 символов; 2-й - от 10 до 20 символов (в зависимости от заданной ширины страницы); 3-й столбец - 40 символов; 4-й, 5-й и последующие - не менее 5 символов.

Длинный отчёт автоматически будет размещён на нескольких страницах.

Окаймление таблицы и разделители клеток. Разделители клеток задаются шестёрками символов, из которых формируются, соответственно, верхнее, правое, нижнее и левое окаймления, горизонтальный и вертикальный разделители клеток.

Разделители могут включать в себя пробелы.

Символы, образующие окаймление таблицы, выводятся:

- по краям отчёта;
- на границе фрагмента отчёта.

Форматирование данных. ФОРПОТ может выводить в отчёт следующие типы данных:

- текст,
- число.

Для каждого типа данных определяются правила форматирования при выводе в отчет.

Форматирование текста. Текст (как содержимое клетки) при выводе в отчёт может быть преобразован следующим образом:

- центрирован (ФОРМАТИРОВАНИЕ = ЦЕНТРИРОВАТЬ),
- сдвинут к левому краю поля (ФОРМАТИРОВАНИЕ = ВЛЕВО),
- сдвинут к правому краю поля (ФОРМАТИРОВАНИЕ = ВПРАВО),
- выровнен по обоим краям за счет добавления лишних пробелов между словами (ФОРМАТИРОВАНИЕ = ВЫРАВНИВАТЬ).

Во всех случаях ФОРПОТ заново разбивает текст на строки максимальной длины таким образом, чтобы число строк было наименьшим.

Слова переносятся по слогам согласно правилам переноса.

Текст, разбитый на абзацы, сохраняет свою структуру.

Специальным случаем форматирования можно считать вывод текста без преобразования (ФОРМАТИРОВАНИЕ = ВЫКЛ).

Форматирование текста

Определение	Пример форматирования
ФОРМАТИРОВАНИЕ = ЦЕНТРИРОВАТЬ	Отметка кладовой о приеме деталей
ФОРМАТИРОВАНИЕ = ВПРАВО	Форма 2-нк
ФОРМАТИРОВАНИЕ = ВЫРАВНИВАТЬ	Для печати используется рулонная или фальцованная бумажная лента с краевой перфорацией

Форматирование чисел. Числа размещаются в клетках с учётом двух требований:

- выравнивания всех чисел в столбце по десятичной точке,
- центрирования.

Числа могут выводиться в трёх представлениях: целые, с фиксированной точкой, с плавающей точкой. Представление задаётся строкой описания

формата, имеющей в общем случае вид:

+9999.9999E+99,

где:

'+' - указывает на обязательность вывода знака числа; если '+' в строке описания формата не указан, то знак выводится только для отрицательных чисел;
знаки '9' перед десятичной точкой - число символьных позиций, отводимых под поле для размещения целой части числа;

знаки '9' после десятичной точки задают количество выводимых цифр в дробной части числа.

Если число имеет больше разрядов после точки (чем задано для вывода), то лишние разряды отбрасываются с округлением; если меньше - то выводятся дополняющие нули.

'E' или 'e' - признак представления числа в формате с плавающей точкой.

Показатель экспонента может выводиться со знаком (или без) в зависимости от присутствия знака '+' после символа Е (или е)

[Содержание <](#)

Числа размещаются в столбце так, что одинаковые разряды оказываются друг под другом.

Форматирование чисел

ЧИСЛО = +999.999	ЧИСЛО = 9.999e+99	ЧИСЛО = 999
+34.069 -4.287 +637.500	1.000E+03 4.123E-15	365 -12 8

Программирование табс-вычислений

Для программирования табс-вычислений используется компонента ВЫЧИСЛИТЕЛЬ, которая предлагает командный

язык ВЫЧ как версию языка Т-кон.

ВЫЧИСЛИТЕЛЬ предназначен для автоматизации табс-вычислений в табс-базах данных и ориентирован на пользователя, являющегося специалистом в своей предметной области, но не имеющего (возможно) опыта компьютерной обработки данных.

Вычисления могут вестись либо в интерактивном режиме, либо в режиме выполнения заранее созданных ВЫЧ-программ, хранящихся в специальных табсах.

ВЫЧИСЛИТЕЛЬ обрабатывает простую таблицу (слой табса) и её элементы (строку, столбец, клетку).

Используются контейнеры, представляющие собой табличные элементы, существующие независимо от обрабатываемой таблицы.

При обработке таблицы доступны не только данные всех слоёв текущего табса, но и данные других табсов открытой для доступа табс-базы.

Имена данных в ВЫЧ. Табсы указываются по имени, заключённому в кавычки (⊗ "ТАБЛ1"; "ПЛАН 2" ⊗).

Для обозначения табличных элементов используются следующие символы: L - строка, G - столбец, C - клетка, V - контейнер-вектор, С - контейнер скаляр.

За символом следует указатель элемента: номер (для клетки - пара номеров), заголовок, символ последнего (T) (или предпоследнего (P)) элемента в ряду, переменный указатель, либо комбинация заголовка и переменного указателя.

Переменный указатель - это контейнер-скаляр (или переменная типа указатель), заключённый в круглые скобки.

Указатель контейнера может быть только номером (в частности, равным нулю).

Перед заголовком в квадратных скобках может находиться указатель вектора.

Числа могут быть целыми или - с десятичной точкой.

Символьные строки заключаются в апострофы.

В символьных строках, заголовках и именах таблиц допустимы сокращения:



"ПЛАН ПРОИЗВОД* НА IV КВАРТАЛ"

G 'МАТЕР. НА СКЛ.'



Переменные ВЫЧ-подпрограмм идентифицируются следующими символами: # - переменная типа имя таблицы, & - переменная типа указатель, \$ - переменная типа часть командной строки.

За символом должен следовать номер переменной (⊗ #1 &0 \$1 ⊗).

Выражения языка. К числам, вводимым пользователем, и к числовым элементам таблиц применимы следующие операции:

[Содержание <](#)

- сложения - '+'
- вычитания - '-'
- умножения - '>*
- деления - '/'
- вычисления процента - '%'
- группового сложения - '++'
- занесения - '>'
- размножения - '>>'.

При этом из табличных элементов в операциях принимают участие только числовые и пустые клетки.

Если в операнде встречается нечисловая клетка (и это может отразиться на результате), выполнение операции приостанавливается и запрашивается продолжение.

Операция размножения является одноместной; она применима только к скалярному числовому операнду и к символу "пусто".

Если операндом является вектор, то результат операции должен быть вектором того же типа.

Не допускаются операции над векторами различных типов.

В арифметическом выражении над скалярными operandами необязателен адрес результата.

В этом случае вычисления идут в режиме калькулятора:

☼
12/234 Cl , l>S
5%G4>G5 C2 , 1++C2 , 8>C2 , 9
Gl++GT>V10 >>L5
☼

Одноместная операция копирования " $=>$ " применима к числовым и текстовым данным, находящимся в табс-элементах:

☼
L5=>L6
C1 , 1=>#1
S=>\$2
☼

В ВЫЧ-подпрограммах, реализующих расширения языка, может использоваться ограничитель выражения, представляющий собой указатель, стоящий между обратными косыми чертами: он задаёт требование выполнить векторное выражение как скалярное.

☼ Выражение $|S1| G1*C4 , 5 > G2$, где S1 равен 10, выполнится как $C10,1 * C4 , 5 > C10 , 2$ ☼

Данные из слоёв табсов, не являющихся текущими, могут быть скопированы в контейнеры или в элементы текущей таблицы.

Для однажды указанной внешней таблицы (слоя табса, не являющегося текущим) можно опускать её имя (просто ставить двоеточие):

[Содержание <](#)

☼

"ТАБ2", 2 : G5 => G 'ЦЕНА'

: G6 => V1

☼

В арифметических операциях могут участвовать данные только из одной внешней таблицы, и они должны стоять первыми в командной строке:

☼

"ТАБ1" : C1 , 1 + S > S1

☼

Функции. Функции применяются к элементам таблиц, и их результаты могут заноситься в табличные элементы.

Следующие функции применяются к векторам, а результатом их выполнения является скаляр:

- среднее - SR
- сумма - SUM
- максимум - MAX
- минимум - MIN
- номер максимального элемента - NMAX
- номер минимального элемента - NMNIN

☼ SR G1 > S

NMAX L1 > S ☼

Функции округления (до целых - ОКР1, до пяти - ОКР5, до десяти - ОКР10, до ста - ОКР100, до тысячи - ОКР1000) могут применяться к любым табличным элементам с числовым содержимым.

Во всех случаях тип результата - такой же, как у аргумента :

OKR5 V > L1

Команды. Команды просмотра таблицы и контейнеров: LN, GM, CN,M , где N и M - номера строк/столбцов текущей таблицы.

Просмотр контейнера осуществляется путём ввода его имени.

При этом в контейнере-векторе визуализируется только та его часть, которая соответствует текущему расположению табличного окна в обрабатываемой таблице:
C25 , 30 V2

Команда определения текущего слоя задаёт слой текущего табса как текущую обрабатываемую таблицу:

СЛОЙ 3

Команда ограничения области действия задаёт область действия векторных операций над текущей таблицей:

☼

ГРАНИЦЫ L5, L20

ГРАНИЦЫ L3, L15, G 'НАЧИСЛЕНО', G 'ИТОГО'

☼

[Содержание <](#)

Команда задания точности ТОЧНОСТЬ определяет требуемую точность представления вычисляемых данных.

Допустимо сокращение - ТОЧН.

Задаётся либо округление дробной части, либо округление до целых, пятерок, десятков, сотен или тысяч.

При задании округления дробной части задаётся ещё и вид округления (с сохранением всех знаков дробной части или только значащих):

ТОЧН 5

ТОЧН .000 - округление до трёх знаков после точки

ТОЧН .99 - округление до двух значащих знаков (не нулей) после точки

Команда перехода НА осуществляет передачу управления при выполнении ВЫЧ-подпрограммы.

Управление может быть передано либо на числовую метку в этой же клетке, либо на другую клетку ВЫЧ-программы:

☼

НА 5

НА С4, 4

☼

Недопустимо сравнивать векторы различных типов.

При сравнении двух векторов рассматриваются только числовые и пустые клетки.

В векторном условии после запятой или обратной косой черты может быть указан побочный параметр, принимающий значение номера первого по порядку элемента, для которого отношение не выполняется (или выполняется соответственно).

Побочный параметр может быть контейнером-скаляром или переменной типа указатель.

Если перед началом проверки условия побочный параметр не определён, либо меньше, либо равен нулю, то проверка условия ведётся от начала векторов, участвующих в условии (в противном случае проверка начинается с индекса, равного значению побочного параметра плюс единица).

☼ В условии

GI > G2 , S1,

где S1 равен 5, проверка начнётся с шестого элемента в каждом векторе ☼

Команда проверки условия выводит в зону сообщений ВЫЧИСЛИТЕЛЯ значение "+" или "-" (в зависимости от значения условия, если нет побочного параметра) или значение побочного параметра, если он указан:

☼

? GI > G2

? L2 > 5 , S1

☼

В команде условного выполнения в обеих частях можно использовать побочный параметр, если он указан в условии:

☼

ЕСЛИ GI > G2 ТО НА С1 , 1

ЕСЛИ GI > 5\S ТО С(S) , 1 > С(S), 8 ИНАЧЕ НА 1

☼

[Содержание <](#)

Команды работы с табсами: команды открытия, закрытия, создания, удаления и переименования табсов:

```
☼  
ОТКР "ТАБ1"  
ЗАКР #1  
СОЗД "ПЛАН1",10, 20, 2  
УДАЛ #3  
ПЕРЕИМ "ТАБ1" В "ТАБ2"  
☼
```

Команды работы с ВЫЧ-подпрограммами: команды запуска ВЫЧ-подпрограмм, описания формальных параметров, останова, установки и снятия пошагового режима выполнения ВЫЧ-подпрограмм:

```
☼  
ВЫЧ "Т1"  
ВЫЧ #1 НАД #2  
ПАРАМ &1, &2, #1  
ПРОД ШАГ  
☼
```

Команды ВВОДА/ВЫВОДА позволяют ВЫЧ-подпрограммам вести диалог с пользователем:

```
☼  
ВЫВЕСТИ 'Неверно начислено L=', &1, 'Ф.И.О.=', С(&1)  
ВВЕСТИ #1  
☼
```

Расширения языка. Расширения реализуются через ВЫЧ-подпрограммы с параметрами, а их синтаксис описывается на специальном метаязыке. Описания расширений хранятся в ТБД в специальной таблице, откуда их берёт ВЫЧ-интерпретатор при встрече расширяющей команды.

Рассмотрим пример расширения ВЫЧ следующей командой:
ДЛЯ <условие> ВЫП <операторы>.

```
☼  
ДЛЯ G1='ОРЕЛ' ВЫП G2++G4 > G5; C1,1 >> G6  
Для расширения ВЫЧ такой командой необходимо:  
• завести в спецтаблице хранения описаний расширений новую строку и указать в её первой клетке идентификатор новой команды - <ДЛЯ>;  
• описать допустимый синтаксис этой команды-расширения:  
• написать ВЫЧ-подпрограмму, принимающую в качестве параметров условие (G = 'ОРЕЛ') и операторы; при этом обеспечивается выполнение операторов только в тех строках обрабатываемой таблицы, где в первой колонке стоит 'ОРЕЛ';  
• указать в строке описания команды <ДЛЯ> имя исполняющей её ВЫЧ-подпрограммы.  
☼
```

[Содержание <](#)

Метаязык описания синтаксиса расширений. Синтаксис новых команд и конструкций, расширяющих язык ВЫЧ, описывается на специальном метаязыке. Основные объекты метаязыка:

имя таблицы - Т

условие - У

выражение языка ВЫЧ - В

элемент (табличный) - Э

указатель адреса - А (число, символьный указатель или клетка).

Структура таблицы описания расширений

1	2	3	4	5	6	7
ДЛЯ	3	G, L	У(G,L), В(1), [В(1)], [В(1)], [В(1)], [В(1)]	ДЛЯG	ДЛЯL	
ВЫБРАТЬ	4	G, L	У(G,L), Т	ВЫБG	ВЫБL	
OKP 10	5		ВО	OKP 10		

Виды основных объектов:

выражение одноместное - ВО

векторный над столбцами - G

векторный над строками - L

клеточный - С



GI > G2; C1, 2 >> LI; G1 => V - это ВО;

У(G) - обозначение условия над столбцами.



Типы табличных элементов, указателей, условий и выражений указываются за объектом в круглых скобках.

Допустимо указание сразу нескольких типов через запятую.

Тип объекта может указываться и номером.

Это означает, что объекту присваивается такой же тип, как у объекта с указанным номером (тип переходит от объекта с указанным номером к определяемому объекту).

Описание синтаксиса новой команды состоит из списка описаний объектов, входящих в неё.

В предыдущей таблице приведены примеры описания команд <ДЛЯ>, <ВЫБРАТЬ> и <OKP10>.

Приведём пример задания команды <ВЫБРАТЬ>:



ВЫБРАТЬ G 'МЕСТО РАЗМЕЩЕНИЯ' = 'СКЛАД4' В "ТАБЛИЦА 3"



Применение специального языка для описания синтаксиса расширений позволяет организовать вывод диагностических сообщений для неверных команд (в терминах этих команд), что повышает степень комфортности для пользователя.

[Содержание <](#)

Реализация расширений языка ВЫЧ-подпрограммами. Команды, расширяющие язык ВЫЧ реализуются посредством ВЫЧ-подпрограмм. Этим подпрограммам разрешается использовать формальные параметры типа часть командной строки. Для примера приведём ВЫЧ-подпрограмму, реализующую команду <ДЛЯ>.

```
! Подпрограмма, реализующая команду <ДЛЯ>
ПАРАМ $1, $2, $3, $4, $5, $6
! В качестве параметров в программу передаются:
! одно условие - $1 и до пяти выражений ($2-$6);
! все они передаются как часть командной строки
! (все эти данные получены из описания
! синтаксиса расширения).
0 > S0
5:
ЕСЛИ $1 \SO ТО НА 7 ИНАЧЕ НА 9
! Конструкция $1 \SO означает, что нужно найти
! номер строки, в которой выполняется условие $1
! (в нашем примере это условие G1 = 'ОРЕЛ');
! поиск номера идет от предыдущего значения SO.
! Если такой строки нет, то переход на СТОП
! (конец подпрограммы).
7:
! Теперь организуем выполнение переданных
! выражений над элементами строки SO
\SO\$2
\SO\$3
\SO\$4
\SO\$5
\SO\$6
! Конструкция \SO ограничивает область выполнения
! следующего за ней векторного выражения так, что
! выражение G2++G4>G5 будет выполнено как
! C(S0), 2++C(S0), 4>C(S0), 5;
! а выражение C2, 2>>G6 будет выполнено как
! C2, 2>C(S0), 6
НА 5
! Переход на определение следующего номера S0
9: СТОП
! Конец подпрограммы
```

Таблица описаний расширений содержит (по столбцам):

- в первом - полное имя команды;
- во втором - минимальное число символов, до которого можно сократить имя команды;
- в третьем - допустимые символы окончания имени команды; естественно, что для реализации команды "ДЛЯ" нужны две ВЫЧ-подпрограммы (одна - для выполнения выражений над строками, другая - над столбцами; поэтому символ G (или L), следующий сразу за именем команды, считается частью имени, определяющей, какую из двух возможных ВЫЧ-подпрограмм надо вызывать);
- в четвертом - описания синтаксиса расширений;

[Содержание <](#)

- в пятом, шестом и седьмом - имена таблиц с ВЫЧ-подпрограммами, реализующими расширения.

Команды <ДЛЯ>, <ВЫБРАТЬ> реализуются парой ВЫЧ-подпрограмм, а команда <ОКР10> - одной.

В ВЫЧ-подпрограммах, реализующих команды-расширения, могут использоваться не только команды ядра языка ВЫЧ, но и другие команды-расширения, имеющиеся в компоненте ВЫЧИСЛИТЕЛЬ.

Такой подход позволяет легко осуществить синонимию команд.

Для введения команды-синонима достаточно повторить имеющееся описание расширения для другого имени команды.

Для разных областей учёта подсистема расширения позволяет создавать различные наборы команд, отличающихся не только составом, но и семантикой (при внешней похожести команд).

Вызов ВЫЧИСЛИТЕЛЯ осуществляется выбором клетки стартового табса, в которой размещено имя компонента, а прекращение работы - вводом команды <ВЫХОД>.

ВЫЧИСЛИТЕЛЬ в процессе работы использует несколько специальных табсов, скрытых от пользователя:

- табс-контейнер (используется для хранения контейнеров, фактических и формальных параметров; на каждую виртуальную линию, работающую с ВЫЧИСЛИТЕЛЕМ, создается табс-контейнер с новым именем);
- табс возобновления (хранит данные, необходимые для возобновления работы прерванной ВЫЧ-подпрограммы; создаётся для каждой виртуальной линии);
- табс описания расширений (хранит данные о реализованных расширяющих командах; создаётся один табс на всю систему).

Порождение. Порождение компоненты ВЫЧИСЛИТЕЛЬ сопровождается:

- выбором из имеющегося набора команд языка ВЫЧ тех из них, которые полезны в данном приложении;
- построением новых команд, отвечающих специфике приложения и представлениям пользователя об удобной работе.

Диалоговый монитор

Управление работой пакета осуществляется с помощью диалогового монитора, который запускается в момент начала работы пакета.

Монитор имеет свои табсы, которые заполняются при порождении пакета.

Стартовый табс монитора предъявляется пользователю при запуске пакета. Он содержит таблицу наименований разделов работ (РАБОТА С ТАБСАМИ, ВЫЧ, ПЛАН, ФОРТО).

Пользователь формирует М-запрос (запрос монитору).

Интерпретируя запросы пользователя, монитор передает управление соответствующей компоненте пакета.

О завершении работы компоненты монитор узнаёт по факту записи данных (компонентой СУТБД) в фиксированную клетку определённого табса.

После этого монитор вновь включается в работу.

[Содержание <](#)

Диалоговый монитор пакета (МП) реализует заданное управление взаимодействующими компонентами.

Пользователь выбирает из табсов МП нужную ВК.

Если указаны ВК ВЫЧИСЛИТЕЛЬ или ФОРТОТ, управление передаётся соответствующей компоненте, а после завершения работы - возвращается монитору. В процессе работы ВЫЧИСЛИТЕЛЯ или ФОРТОТА управление выполнением полностью реализуется средствами этих компонент.

Если же выбрана компонента ПЛАН, то сохраняется возможность (с использованием средств МП) выбирать (в режиме меню) интересующую задачу. Кроме того, можно задавать условную или безусловную последовательности выполнения задач.

Можно также определить задание, выбрав множество понятий (при этом ищется разрешающая структура).

Интерпретация запроса завершается планированием последовательности выполняемых задач и запуском.

После завершения выполнения заказанной работы управление возвращается МП.

Запуск МП осуществляется командой

<@GENPAK>

После запуска в табличном окне появляется 1-й слой стартового табса МП, который содержит меню ВК, включённых в пакет.

Монитор использует два окна (режимов и помощи).

В окне режимов монитор предъявляет пользователю табсы четырёх типов, которые определяют соответствующие режимы работы:

- меню задач,
- безусловная последовательность задач,
- условная последовательность задач,
- меню понятий.

В окне помощи представлены сведения, характеризующие ВК или задачу.

В режиме меню задач пользователю показывается одностолбцовая таблица, в клетках которой содержатся наименования либо ВК, либо задач.

Пользователь должен установить курсор на выбранную клетку и ввести команду <ВЫБОР>; управление будет передано выбранной компоненте.

Режим безусловной последовательности задач. Пользователю показывается таблица из двух столбцов: в первом - наименования задач, во втором - пусто. Редактирование второго столбца заключается в присвоении задачам порядковых номеров их запуска (целых чисел без знака; значения могут быть любыми, важно лишь отношение порядка).

Запуск выбранной последовательности задач осуществляется по команде <КОНЕЦ РЕДАКТИРОВАНИЯ>.

Режим условной последовательности задач. Пользователю показываются три столбца: первые два - такие же, как в режиме "Безусловная последовательность задач"; клетки третьего столбца могут быть заполнены символом "Д" (наличие этого символа вызовет остановку выполнения последовательности задач перед задачей с символом "Д").

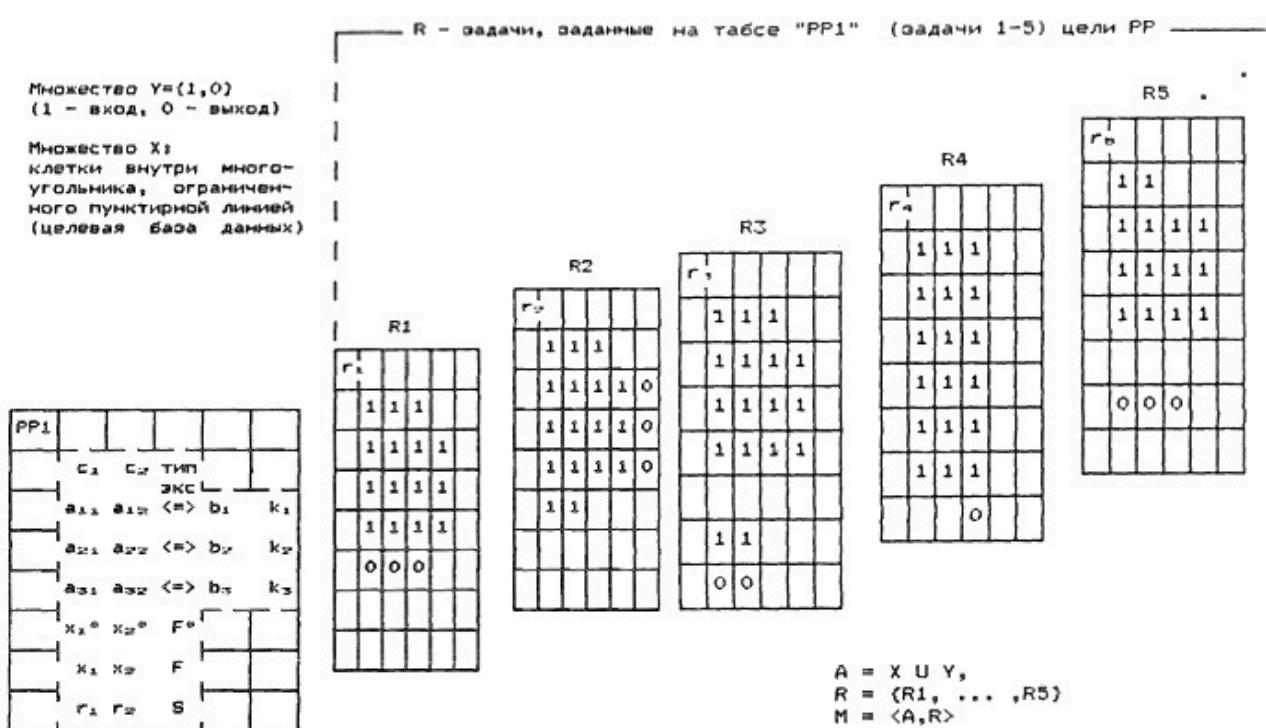
Пользователю будут показаны результаты выполнения предыдущей задачи и поставлен вопрос: удовлетворяют ли они его?

Содержание <

Положительный ответ приведёт к возобновлению выполнения прерванной последовательности задач, а отрицательный - к возврату в tabs "Условная последовательность задач" (для редактирования).

Как и в предыдущем режиме, запуск выбранной последовательности задач осуществляется по команде <КОНЕЦ РЕДАКТИРОВАНИЯ>.

Табс-представление модели М целевой системы «Распределение ресурсов»



Режим меню понятий. Табс этого режима - двухстолбцовый: первый - содержит имена понятий, а второй - должен быть заполнен символами "+" (напротив имён понятий, интересующих пользователя при решении задач выбранной предметной области).

Для заданного пользователем множества понятий выполняется интерпретация запроса на пользовательском задачном графе (модели задачной области). Результат интерпретации - разрешающая структура задач, решение которых обеспечит нахождение значений выбранных понятий.

Интерпретация может выполняться в автоматическом или в диалоговом режиме.

Пример интерпретации в диалоговом режиме показан на следующем рисунке.

В диалоговом режиме пользователь получает возможность доопределять постановку задачи в процессе поиска разрешающей структуры (решение слабоопределенных задач).

Помощь в работе. Помощь может быть оказана во всех режимах работы МП.

Выдаваемые сообщения привязаны к текущей строке табса.

Для получения помощи нужно ввести команду <ПОМОЩЬ ПО ВЗАИМОДЕЙСТВУЮЩЕЙ КОМПОНЕНТЕ>.

По этой команде курсор перемещается в окно помощи.

[Содержание <](#)

Здесь (в режиме просмотра) можно ознакомиться с поясняющими сообщениями. Чтобы вернуться в окно режимов, нужно ввести команду <КОНЕЦ РЕДАКТИРОВАНИЯ>.

Выход из монитора. Прекращение работы монитора приведёт к завершению работы с пакетом.

Выход из МП можно осуществить:

- выбрав из меню пункт "конец работы пакета" и дав команду <ВЫБОР>,
- введя команду <НОРМАЛЬНЫЙ КОНЕЦ РАБОТЫ ПАКЕТА>,
- воспользовавшись командой <АВАРИЙНЫЙ КОНЕЦ РАБОТЫ ПАКЕТА>.

Функциональное наполнение

Решение задач планирования обеспечивается компонентой ПЛАН.

Прикладной смысл задач может быть различным.

С математической точки зрения эти задачи относятся либо к линейным минимаксным, либо - к задачам линейного программирования.

Решение задач осуществляется в диалоге с пользователем, который может выбрать требующиеся задачи, определить последовательность их решения и условия перехода от выполнения одной задачи к другой.

Кроме того, запрос на решение задач может быть сделан указанием понятий, интересующих пользователя.

Для ввода данных предлагается заполнить табсы в соответствии с имеющимися в пакете шаблонами.

Предусматривается помочь в затруднительных ситуациях.

Полученные результаты могут быть выведены на экран или распечатаны (с помощью компоненты ФОРМ).

Окна, через которые пользователь просматривает результаты, определяются при порождении пакета.

Постановки R-задач. R-задача - это прикладная задача, включённая в пакет.

Здесь рассматриваются задачи из класса "Линейное планирование и чебышевские приближения к плану".

Приведём примеры формулировок R-задач.

Отыскание оптимального плана: найти максимум прибыли $\text{sum}[j=1..n] c[j]*x[j]$ при ограничениях на ресурсы $\text{sum}[j=1..n] a[i, j]*x[j] \leq b[i], m > n; x[j] \geq 0, j = 1..n; b[i] > 0, i = 1..m$.

Расчёт остатков ресурсов: определить остатки ресурсов

$$k[i] = b[i] - \text{sum}[j=1..n] a[i, j]*x[j], i = 1..m$$

после реализации оптимального плана $x['] = (x[1]..x[n])$,

обеспечивающего экстремум целевой функции $\text{sum}[j=1..n] c[j]*x[j]$

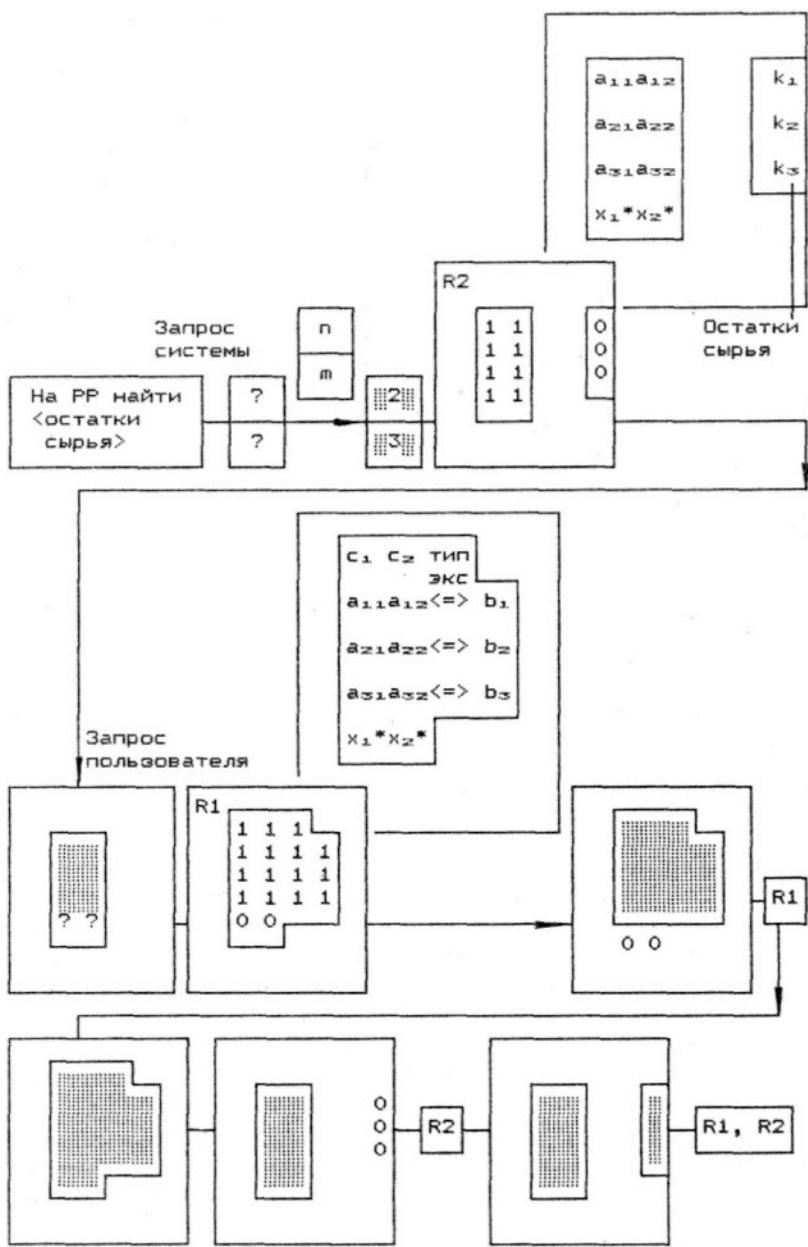
при ограничениях:

$$\text{sum}[j=1..n] a[i, j]*x[j] \leq b[i], m > n,$$

$$x[j] \geq 0, j = 1..n,$$

$$b[i] > 0, i = 1..m.$$

Схема доказательства существования решения Р-задачи



Расчёт таблицы оптимальных расходов ресурсов. Определить элементы $R[i, j] = a[i, j]*x['; j]$, $i = 1..m$, $j = 1..n$, соответствующие j -м компонентам $x['; j]$ оптимального плана $(x['; 1]..x['; n])$, обеспечивающего максимум прибыли $\sum[j=1..n] c[j]*x[j]$ при ограничениях $\sum[j=1..n] a[i, j]*x[j] \leq b[i]$, $m > n$.

Расчёт отклонений вариантурного значения плана от оптимального. Найти величину $S = \sum[j=1..n] c[j]*(x['; j] - x[j])$ как разность значений прибыли $F = \sum[j=1..n] c[j]*x[j]$ для оптимального плана $x[';] = (x['; 1]..x['; n])$, обеспечивающего максимум функции F и удовлетворяющего ограничениям $\sum[j=1..n] a[i, j]*x[j] \leq b[i]$, $m > n$, $x[j] \geq 0$, $j = 1..n$.

и вариантурного плана $x = (x[1]..x[n])$ при тех же ограничениях.

Расчёт значения целевой функции. Найти значение прибыли

$$\sum[j=1..n] c[j]*x[j],$$

если $x[1]..x[n]$ удовлетворяют ограничениям

$$\sum[j=1..n] a[i, j]*x[j] \leq b[i], m > n,$$

$$x[j] \geq 0, i = 1..m, j = 1..n.$$

Каждой R-задаче соответствует множество понятий.

Каждое понятие определяется множеством переменных, число элементов которого - функция параметров R-задачи.

Максимальные значения параметров определяются при порождении пакета, а конкретные - при эксплуатации пакета.

[Содержание <](#)

В сформулированных R-задачах используются следующие понятия:

- максимальная прибыль,
- удельные расходы ресурсов,
- запасы ресурсов,
- коэффициенты целевой функции,
- вариантный план,
- оптимальный план,
- значение прибыли, соответствующее вариантному плану.

Множество понятий каждой j-й задачи разбивается на два непересекающихся подмножества $T[v; j]$ и $T[w; j]$.

Элементами $T[v; j]$ являются понятия, которые определяются входными переменными, а элементами $T[w; j]$ - понятия, определяемые выходными переменными.

R-задачи и их связь с R-задачами

Множество R-задач - основа для решения различных пользовательских задач.

Для применения R-задач в конкретных приложениях существует механизм создания так называемых *R-понятийных оболочек*, позволяющий установить соответствие между R-понятиями и понятиями пользовательских задач конкретной области применения (R-понятиями).

В итоге на основе R-задач и созданной R-понятийной оболочки автоматически формируется множество R-задач.

Приведем пример R-задачи, решение которой сводится к решению R-задачи.

Пусть для изготовления каждого из n видов продукции употребляется m видов сырья, причем расход i-го вида сырья на единицу j-го вида продукции составляет $a[i, j]$ единиц.

Пусть далее на каждой единице продукции j-го вида предприятие получает прибыль $c[j]$ рублей.

Требуется определить, сколько единиц продукции каждого вида $x[1]..x[n]$ должно изготовить предприятие, чтобы обеспечить максимальную рентабельность производства, если учесть, что сырья i-го вида имеется $b[i]$ единиц ($i=1..m$).

R-понятиям поставлены в соответствие следующие R-понятия (см. следующий рисунок):

- коэффициенты целевой функции - прибыль на единицу каждого вида продукции,
- тип экстремума - максимум (прибыли),
- коэффициенты ограничений - расход сырья на единицу продукции,
- свободные члены - запасы сырья,
- оптимальные значения переменных - число единиц каждого вида продукции.

Поиск чебышевской точки (R-задача) может иметь следующий прикладной смысл.

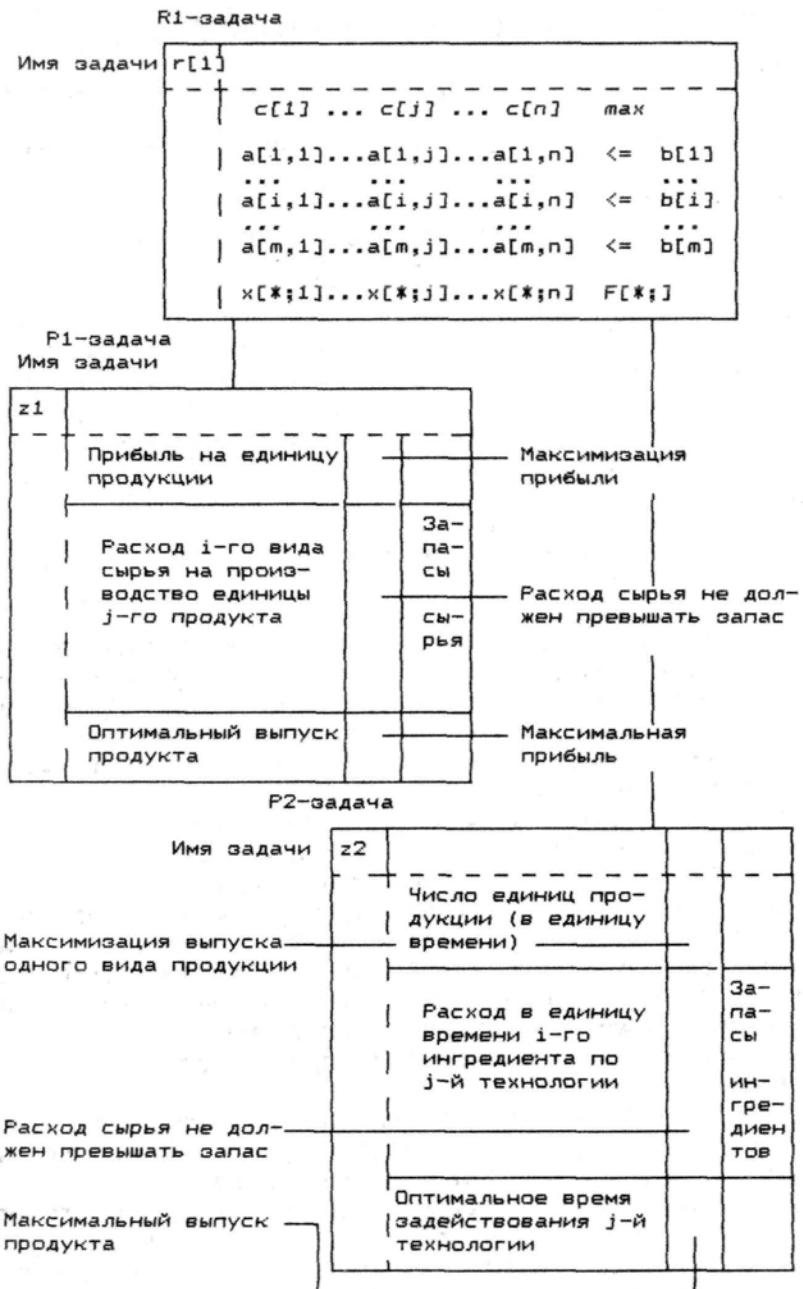
R-задача. Предположим, что запасов сырья не хватает для отыскания плана, обеспеченного ресурсами.

В этих условиях можно найти наилучшее приближение к плану (чебышевскую точку).

Соответствие понятий R- и R-задач будет следующим:

- тип экстремума - минимакс,
- коэффициенты ограничений - расходы сырья на единицу продукции,
- свободные члены - запасы сырья,
- чебышевская точка - число единиц каждого вида продукции, обеспечивающее наилучшее приближение к плану.

Понятийные связи R1-задачи с Р1- и Р2-задачами



Установление соответствия между Р-понятиями и наименованиями переменных может быть выполнено и при работе с готовым пакетом.

Табс-представления задач.

Представления задачных знаний в генераторе ГЕНПАК и ГЕНПАК-пакетах воплощены семейством специальных табсов, связанных между собой.

Механизм вывода реализован как механизм управления табс-навигацией и образования требуемых табс-структур.

Описания R- и Р-задач хранятся в табсах.

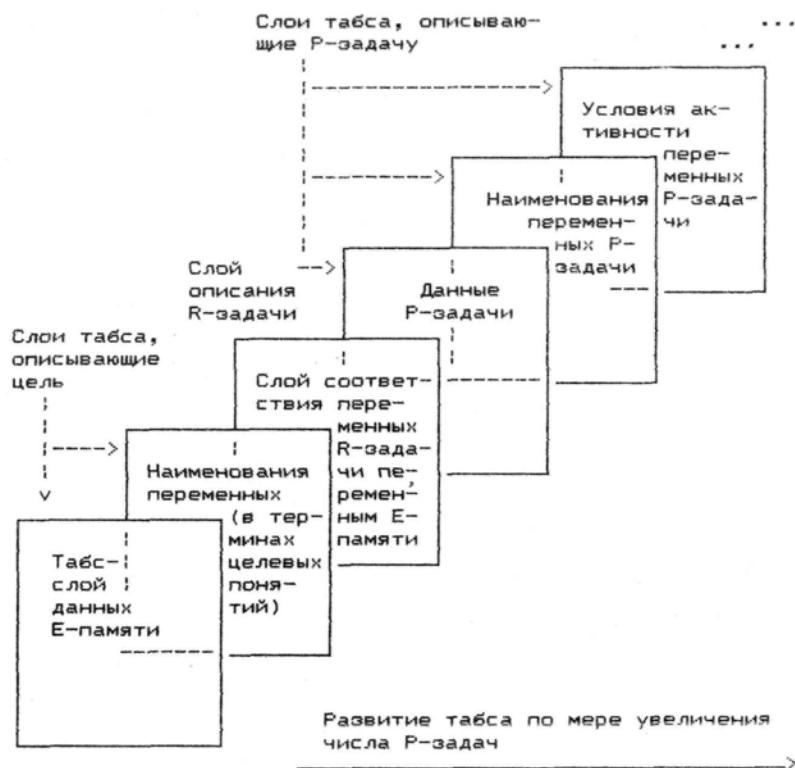
Существуют табсы двух типов: табс генератора и табс пакета.

Табс генератора является табсом-прообразом: он содержит описания R-задач и шаблоны для описания Р-задач.

Табс пакета содержит описание структуры пользовательских табсов (U-табсов), т.е. табсов, создаваемых для фиксированных значений

параметров Р-задач (см. следующий рисунок). Порождаемые на основе табса генератора табсы пакета являются табсами-образами.

U-табс: слои



Первый слой U-табса - слой данных (Е-память), второй - отводится для наименований переменных R-задач.

В третьем слое (соответствий), определяющем R-задачу, символами "вх" помечаются клетки, первые две координаты которых равны соответствующим координатам входных переменных в Е-памяти R-задачи.

Аналогично, символами "вых" помечаются клетки, проекция которых на Е-память соответствует выходным переменным R-задачи.

Фрагмент U-табса РР1: слои 1-2 (n=2, m=3)



PP1				
	c ₁	c ₂	тип	экс
	a ₁₁	a ₁₂	тип	b ₁
	a ₂₁	a ₂₂	тип	b ₂
	a ₃₁	a ₃₂	тип	b ₃
	x ₁ *	x ₂ *	F*	
	x ₁	x ₂	F	
	r ₁	r ₂	S	

Далее располагается совокупность слоёв, относящихся к Р-задачам. Для каждой Р-задачи отводятся три слоя: данных, наименований переменных, условий активности.

Вместе со слоем соответствий Р-задачи и слоем Е-памяти они полностью определяют Р-задачу.

Затем идёт слой соответствий Р-задачи.

За ним - слои, относящиеся к Р-задачам, решаемым с помощью R-задачи, и так далее - до исчерпания описаний всех R- и Р-задач.

Запросы пользователя на решение Р-задач могут быть трёх типов:

- выбор задачи из меню (M-запрос);
- задание безусловной или условной последовательностей выполнения задач (Z-запрос);
- указание понятий, являющихся выходными переменными Р-задач (N-запрос).

[Содержание <](#)

М-запрос позволяет выбрать нужную Р-задачу из меню Р-задач, которое формируется при порождении пакета.

Выбор осуществляется установкой курсора в соответствующую клетку меню с последующим нажатием клавиши "ВЫБОР".

Для задания последовательности решаемых задач (Z-запрос) используется формо-ориентированный язык.

Требуется заполнить клетки таблицы, указав номер задачи в последовательности и определив необходимость вмешательства пользователя при переходе к выполнению очередной задачи (если последовательность - условная).

Контролируется правильность задания последовательности решаемых задач.

При N-запросе пользователю предлагается указать интересующие его понятия - выходы Р-задач.

Чтобы найти разрешающую структуру, N-запрос интерпретируется на задачном графе (модели задачной области).

В случае удачного завершения поиска организуется выполнение задач, входящих в разрешающую структуру.

Взаимодействие пользователя с ГЕНПАК-пакетом в процессе выполнения операций ввода-вывода выглядит следующим образом.

Пользователю предлагается заполнить табсы ввода (V-табсы).

Он видит на экране фрагмент таблицы, размер которой определён значениями параметров задачи.

В клетки этой таблицы (в режиме редактирования) он должен ввести значения переменных.

Структура V-табса



В любой момент заполнения V-табса пользователь может сделать прозрачным его первый слой.

При этом на экран выводятся наименования переменных Р-задачи, размещённые во втором слое (это удобно, когда требуется вспомнить содержание и схему расположения переменных в первом слое).

Полученные результаты помещаются в табсы вывода (W-табсы).

На экране эти табсы могут быть просмотрены через табличные окна, конфигурация которых определяется при порождении пакета.

С помощью компоненты ФОРПОТ результаты могут быть напечатаны в табличной форме.

Информационное обслуживание реализуется с помощью специальных таблиц, которые в совокупности образуют информационный табс (I-табс).

[Содержание <](#)

Таблица наименований Р-задач. Таблица содержит столбец с наименованиями задач и столбцы, в которых пользователь указывает последовательность выполнения задач и необходимость вмешательства при переходе к следующей задаче.

V-табс Р1-задачи А-табса РР (n=2, m=3)

Слой условий активности данных			
0, 100 1	0, 100 1	max	
0, 100 1	0, 100 1	\leq	1, 200 1
0, 100 1	0, 100 1	\leq	1, 200 1
0, 100 1	0, 100 1	\leq	1, 200 1

Слой наименований переменных		
Прибыль на единицу продукции	Тип экстремума	
Расход i-го вида сырья на производство единицы j-го продукта	Типы ограничений	Запасы сырья

Слой данных			
5	9	max	—
1	1	\leq	15
5	3	\leq	120
5	10	\leq	100

(окна, обучающие таблицы, таблицы помощи в затруднительных ситуациях);
- тесты.

Решение задач учёта во многом обеспечивается возможностями языка ВЫЧ (компоненты ВЫЧИСЛИТЕЛЬ).

Ограничимся рассмотрением процесса построения новых команд языка ВЫЧ. Пусть, например, в некотором приложении нередко возникает ситуация, когда необходимо провести оценку стоимости материалов или работ в соответствии с нормативными документами.

Обучающая таблица Р-задачи содержит постановку задачи. В ней же даётся характеристика V-табса (т.е. приводится шаблон ввода с указанием переменных, значения которых помещаются в первый слой V-табса). В этой же таблице (для фиксированных значений параметров цели) сформулирована Р-задача с заданными значениями входных переменных.

Показан пример заполнения первого слоя V-табса для этой задачи, состав и заполнение W-табса результатом решения рассматриваемой задачи.

При порождении компоненты ПЛАН определяются:

- состав Р-задач;
- максимальные значения параметров Р-задач и наименования входных и выходных переменных;
- задачный граф;
- допустимые виды запросов на решение задач;
- интерфейс с пользователем

[Содержание <](#)

Удобно представить эту операцию в виде отдельной команды языка - <ТАРИФИЦИРОВАТЬ>, осуществляющей тарификацию данных из графы первой таблицы по данным из графы второй таблице.

Пример обращения к команде:

ТАРИФИЦ G 'ЦЕНА' "СКЛАД 1" ПО G 'ЦЕНА ЗА ЕД*' ИЗ "НОРМАТИВЫ"

В этом примере графы 'ЦЕНА' таблицы "СКЛАД1" тарифицируются по графе 'ЦЕНА ЗА ЕДИНИЦУ ИЗМЕРЕНИЯ' из таблицы "НОРМАТИВЫ".

Для реализации этой команды необходимо составить ВЫЧ-подпрограмму и ввести описание команды в табс описания расширений.

Строка описания команды в табсе описания расширений будет выглядеть так:

ТАРИФИЦИРОВАТЬ ! 7 ! A(G),T, A(G), T ! ТАРИФ ! ! !

Построением множества таких команд (по заказам пользователя), обеспечиваются дополнительные (по отношению к базовому варианту) запросы на решение задач учёта.

И-порождение: среди методологий информатики

Комплекс процессов создания, накопления, поиска, распространения и применения знаний будем называть *int-комплексом*, а (используемый для его воплощения) комплекс символьных, кодовых и сигнальных процессов в человеко-машинной среде - *inf-комплексом*.

Предмет информатики - символное моделирование *int-* и *inf*-комплексов, связанных между собой (как целевые задачи и инструментальные средства их решения).

Символьное моделирование - научная дисциплина, изучающая процессы построения, накопления, поиска, распространения и применения символьных моделей объектов произвольной природы.

Методология И-порождения - теоретическая платформа для создания среды И-порождения, являющейся частью *inf*-комплекса.

Порождать целевую программную систему - значит продуцировать программную систему в диалоге с системой И-порождения.

При этом выбранные задачный и системный облики порождаемой системы воплощаются в соответствии с правилами, хранящимися в базе знаний порождающей системы.

В И-порождении разработчик - решающая сторона в процессе пошаговой (по правилам, хранящимся в базе знаний) детализации описания целевой программной системы.

Процесс автоматизирован: решения принимает человек, а автомат выполняет роль "хранителя нерушимых истин" и "жреца, строго следящего за тем, как эти истины применяются".

И-порождение и традиционное программирование

Как написать программу для решения некоторой задачи (используя систему программирования) - забота программиста.

Он может написать и совершенно бессмысленную, но синтаксически правильную программу (бессмысленную из-за отсутствия задачи, имеющей содержательную интерпретацию, которую можно было бы поставить в соответствие такой программе).

При И-порождении отсутствует свобода составлять описания программных систем, не имеющих определённого целевого назначения.

Языки И-порождения включают интерактивные формо-ориентированные ЧТО- и КАК-языки.

Каждая порождающая система (в отличие от системы программирования) позволяет получить некоторое множество целевых программных систем. Порождённые системы могут отличаться существенно или - только деталями.

Но в одном они похожи (все без исключения): каждая предназначена для решения фиксированного множества связанных задач и создана по правилам, хранящимся в базе знаний порождающей системы.

Интерактивный преобразователь ресурсов: пример реализации И-порождения

Представленные в этом разделе алгоритмы распределения ресурсов ориентированы на интерактивный режим [[Ильин А.В. 2008, 1](#)] вычислительного эксперимента, проводимого экспертом-планировщиком во взаимодействии с программно-аппаратным р-комплексом [[Ильин А.В., Ильин В.Д. 2004](#)].

Программную составляющую р-комплекса будем называть р-преобразователем.

S-модель интерактивного преобразователя ресурсов

С точки зрения пользователя р-преобразователь – это совокупность программно воплощённых вычислительных и интерфейсных средств пошагового решения заданного множества задач преобразования ресурсов во взаимодействии с окружением.

Под р-окружением будем понимать человека-эксперта или автомат (в частности, s-машину), реализующий заданную человеком программу взаимодействия с р-преобразователем.

Для объяснения работы р-преобразователя введём некоторые определения, которые придадут изложению более формализованный характер.

Память р-преобразователя (р-память) – объединение памяти рассматриваемых задач преобразования ресурсов.

Множество располагаемых операторов преобразования р-памяти (располагаемых р-операторов) состоит из фиксированного числа программно воплощённых алгоритмов задач преобразования ресурсов.

Отношения на памяти задаются правилами двух типов:

- обязательные правила определяют отношения, которые не могут быть нарушены;
- ориентирующие правила задают отношения, которые желательно выполнить.

Правила обоих типов могут быть изменены на каждом шаге взаимодействия р-преобразователя с р-окружением.

Выбор р-оператора зависит от состояния р-памяти после выполнения предыдущего шага и системы правил, заданных для текущего шага при взаимодействии с р-окружением.

Не исключено, что при некотором фиксированном наборе обязательных правил ни одно из ориентирующих правил не будет выполнено.

Также не исключено, что может быть задан такой фиксированный набор обязательных правил, при котором не может быть выполнен ни один из располагаемых р-операторов.

На каждом шаге, начиная с первого, р-оператор может быть выбран автоматически или извне (экспертом) из числа тех р-операторов, для которых выполняются обязательные правила.

В противном случае р-преобразователь выдаёт сообщение о невозможности применения располагаемых р-операторов и рекомендует рассмотреть возможность изменения системы обязательных правил.

[Содержание <](#)

Автоматический выбор р-операторов в случае, когда число допустимых р-операторов не менее двух, выполняется следующим образом.

Срабатывает каждый из р-операторов, создавая свою версию состояния р-памяти.

Например, если сработают два р-оператора, то будут созданы две версии состояния р-памяти.

Далее анализатор состояния р-памяти проверяет каждую из версий р-памяти.

Для этого он использует свою систему правил.

Например, такая система может определять выбор той версии состояния р-памяти, которая обеспечивает наилучшее удовлетворение потребителя ресурсов, имеющего высший приоритет.

Р-преобразователь действует до тех пор, пока состояние р-памяти не станет таким, что ни один из расположенных р-операторов не может быть применён, либо из р-окружения не поступит сигнал «стоп».

Поддерживается возможность параллельного выполнения операторов преобразования, если вычислительная среда располагает соответствующими средствами.

Далее приведено краткое описание экспериментальной версии программной реализации р-преобразователя.

Эта версия прошла относительно представительный (по числу и разнообразию тестовых задач) процесс оценки прикладной эффективности и апробации на задачах из различных предметных областей.

Разработанные алгоритмы интерактивного поиска решений линейных задач одноресурсного и многоресурсного распределения, средства редактирования правил, а также необходимые средства взаимодействия с пользователем реализованы на языке С++ и вошли в состав диалоговой программной системы РЕСУРС-комплекс [Ильин А.В. 1999].

РЕСУРС-комплекс: архитектура, характеристика реализации, оценка функциональной эффективности

РЕСУРС-комплекс предназначен для решения различных прикладных задач одноресурсного и многоресурсного распределения в режиме вычислительного эксперимента. РЕСУРС-комплекс предоставляет диалоговую поддержку эксперту, участвующему в анализе ресурсной ситуации и проектировании решений, обладающих требуемыми свойствами реализуемости и эффективности. Область применения РЕСУРС-комплекса – документально представленное сопоставление складывающейся обеспеченности ресурсами с потребностью в них и последующее вариантное планирование ресурсно-обеспеченных управляющих воздействий в условиях изменяющейся информированности.

Архитектура РЕСУРС-комплекса рассчитана на расширение и совершенствование без нарушения привычной для пользователя среды взаимодействия.

Задачи проектирования ресурсно-обоснованных решений разделены на типы.

Для задач каждого типа сформулированы так называемые базовые задачи, для которых разработаны алгоритмы.

Конкретные прикладные задачи, которые необходимо решать в процессе ресурсного обоснования, представляют собой конкретизации задачных конструкций, построенных из базовых задач.

[Содержание <](#)

Основные составляющие РЕСУРС-комплекса — вычислительное ядро [[Ильин А.В., Ильин В.Д. 1995](#)], оболочка графического интерфейса, модуль управления задачами и генератор отчетов.

Вычислительное ядро представляет собой программную реализацию методов решения следующих задач:

- общей задачи многоресурсного распределения (методом целевого перемещения решения),
- задачи бесприоритетного одноресурсного распределения в иерархической системе,
- задачи приоритетного одноресурсного распределения в иерархической системе,
- стандартной задачи ЛП (симплекс-методом),
- задачи поиска компромиссного решения (вычислением чебышевской точки).

При разработке архитектуры программного комплекса была применена методология построения задачных конструктивных объектов, разработанная в [[Ильин В.Д. 1989, 1](#)].

Вычислительное ядро РЕСУРС-комплекса предназначено для решения задач, относящихся к задачной области Распределение ресурсов. Задачи рассматриваются как конструктивные объекты, представленные графами специального вида [[Ильин А.В. 2007](#)]. Рёбра задачных графов представляют связи задач по данным.

Конструирование решения по описанию задачи и условий применения — это процесс построения задачного конструктивного объекта из уже имеющихся (в системе) и последующей конкретизации построенного объекта.

Этот процесс представляет собой конструктивное доказательство существования схемы редукции на задачном графе.

Итогом доказательства служит схема редукции интересующей нас задачи к задачам с программно воплощённой реализацией.

При решении конкретных задач посредством РЕСУРС-комплекса запросы пользователя интерпретируются на модели задачной области Распределение ресурсов.

Архитектура РЕСУРС-комплекса рассчитана на расширение и совершенствование без нарушения привычной для пользователя среды взаимодействия.

Вычислительное ядро РЕСУРС-комплекса написано на «чистом» C++ и не использует внешних библиотек.

Оно является независимым модулем и может использоваться в программных комплексах, работающих в различных операционных средах и на различных аппаратных платформах.

Конкретные оболочки универсального вычислительного ядра, представляющие собой реализации интерфейса БД и пользовательского интерфейса, зависят от целевой платформы и специфики решаемых прикладных задач.

Программы РЕСУРС-комплекса написаны в среде Borland C++ 3.0.

Комплекс работает с базами данных формата DBF.

Экспериментальный характер версии РЕСУРС-комплекса не накладывает ограничений на спектр решаемых задач и их максимальные размеры.

Эта версия была разработана для исследования прикладной эффективности разработанных методов одноресурсного и многоресурсного распределения в режиме вычислительного эксперимента.

Другими словами, версия разработана для опытной проверки исследуемых методов.

[Содержание <](#)

Для первого этапа лабораторных испытаний были разработаны специальные тестовые задачи, прогоны которых проверяли алгоритмическую правильность программно реализованных методов.

На этом этапе программа испытаний была ориентирована на проверку работоспособности вычислительного ядра РЕСУРС-комплекса.

Для проверки эффективности механизма интерпретации запросов на модели задачной области Распределение ресурсов был разработан соответствующий набор тестовых задач.

С их помощью был испытан механизм интерпретации запросов при решении задач в режиме вычислительного эксперимента.

Повышенная пользовательская актуальность задач приоритетного и бесприоритетного одноресурсного распределения в иерархических системах (особенно — задач бюджетного проектирования в иерархических системах) определила необходимость дополнительных испытаний этой функциональной составляющей РЕСУРС-комплекса.

В дополнение к лабораторным испытаниям эффективности реализованных алгоритмов и гибкости диалоговых средств взаимодействия с пользователем был проведен ряд представительных расчетов на реальных данных.

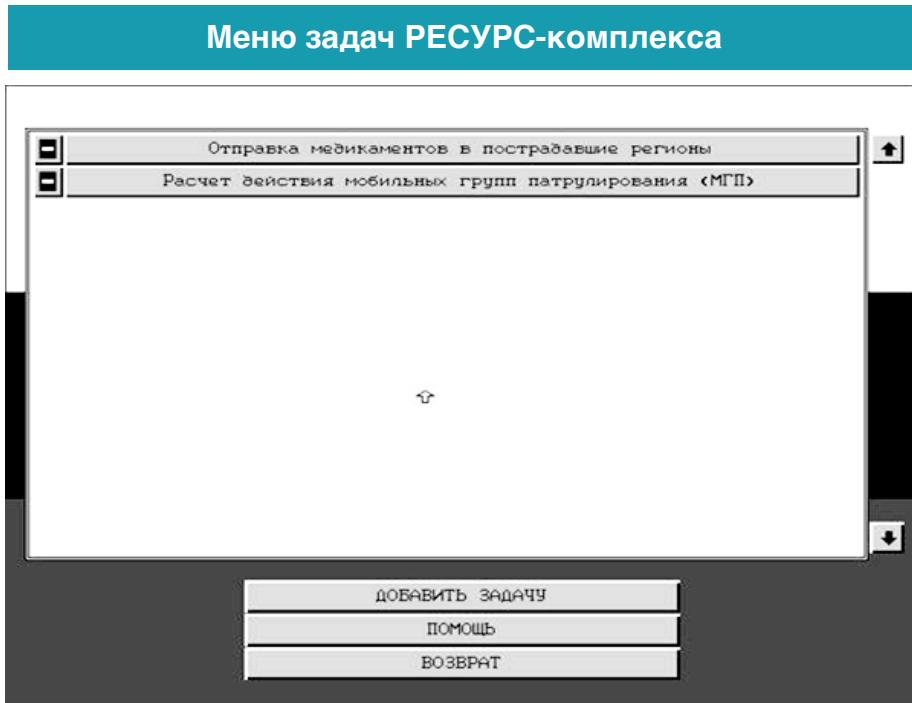
Управление задачами

Модуль управления задачами реализует интерфейс с базами данных.

Он предоставляет возможности описания и сохранения прикладных задач, экспорта и импорта данных, ведения архивов решений.

Каждая задача пользователя представлена в виде набора связанных таблиц.

С помощью меню задач эксперт может создавать и удалять задачи, задавать их параметры.



Для многоресурсного распределения каждой задаче соответствует база данных, содержащая исходные данные задачи, и база различных решений этой задачи с необходимыми пометками и комментариями. РЕСУРС-комплекс запоминает дату и время записи решения на диск и делает пометки (является ли данное решение компромиссом или

результатом оптимизации одной из функций).

Кроме того, пользователю предлагается ввести собственный комментарий к решению.

Графический интерфейс

РЕСУРС-комплекс рассчитан на диалоговую работу по следующей схеме.

[Содержание <](#)

Эксперт описывает иерархию преобразования ресурсов и необходимые на каждом уровне иерархии ресурсы, вводит соответствующие данные и запускает механизм интерпретации запроса на модели задачной области Распределение ресурсов. Встроенный в ядро механизм поиска разрешающих структур на задачных графах находит требуемую задачную структуру и решает входящие в её состав задачи. Результат работы — документ (или пакет документов), содержащий искомые ресурсно-обоснованные управляющие воздействия.

Оболочка графического интерфейса включает специализированные редакторы данных прикладных задач, средства запуска функций вычислительного ядра и визуализации результатов, а также - получения контекстной помощи.

Табличный редактор РЕСУРС-комплекса

РЕСУРС-КОМПЛЕКС			
Действие мобильных формирований защиты особо важных объектов			
Ресурсы	Опт.	1-й козфф-т	2-й
Топливо (т)		0.01	
Средства пожаротушения (т)		1.2	
Средства дезактивации (т)		0.08	
Средства дезинфекции (т)		0.3	
Средства дегазации (т)		0.9	
Площадь обработки (кв.км)		25.0	
Стоимость работы МГП (млн.руб)	min	5.0	
Применение наземных формирований (экипаж/час)		1.0	
Применение воздушных формирований (экипаж/час)		0.0	

Редактирование Буфер Помощь Возврат

Табличный редактор данных РЕСУРС-комплекса позволяет просматривать данные, редактировать и сохранять их, вставлять и удалять строки и столбцы таблиц, содержащих данные задач пользователя. Этот редактор является контекстно-чувствительным: содержит самонастраивающиеся встроенные средства, необходимые для подготовки непротиворечивых данных.

Редактирование данных

ПЛАНИРОВАНИЕ РАСХОДОВ : Эскизный расчет бюджета				
Деньги (млрд.руб.): 118.0.120.61				
Запросы	Приоритеты	мин.запрос	макс.запрос	да
Народное хозяйство	1.8	33.0	35.0	
Соц. обеспечение, наука	1.7	24.0	26.4	
Органы гос. управления	1.9	7.5	9.0	
Оборона	2.0	13.0	15.0	
Гос. внутренний долг	1.7	2.2	3.5	
Внешнеэкономическая деятельность	1.6	6.0	7.0	
Внебюджетные фонды	1.6	40.0	42.0	
Республики СНГ	1.3	4.3	5.5	
Импортные субсидии	1.6	10.0	12.0	

Сохранить таблицу и создать развертку выбранной строки?

Помощь Возврат

Для обмена данными между таблицами и задачами РЕСУРС-комплекс имеет буфер.

Для задач одноресурсного распределения редактор данных позволяет описывать иерархию потребления ресурсов, задавая расходные статьи и их развертки.

В этом режиме строки таблицы, соответствующие расходным статьям, уже имеющим развертки, выделены цветом.

Распределение ресурсов по расходным

статьям (то есть запуск необходимых элементов вычислительного ядра) в задачах одноресурсного распределения производится также в режиме табличного редактора.

Управление вычислениями

ПЛАНИРОВАНИЕ РАСХОДОВ : Эскизный расчет бюджета				
Деньги (млрд.руб.): 118.0.120.61				
Запросы	Приоритеты	мин.запрос	макс.запрос	да
Народное хозяйство	1.8	33.0	35.0	
Соц. обеспечение, наука	1.7	24.0	26.4	
Органы гос. управления	1.9	7.5	9.0	
Оборона	2.0	13.0	15.0	
Гос. внутренний долг	1.7	2.2	3.5	
Внешнеэкономическая деятельность	1.6	6.0	7.0	
Внебюджетные фонды	1.6	40.0	42.0	
Республики СНГ	1.3	4.3	5.5	
Импортные субсидии	1.6	10.0	12.0	
Прочие расходы	1.6	20.0	25.0	

БЕСПРИОРИТЕТНОЕ ПРИОРИТЕТНОЕ

ВСЕГО ВЫДЕЛЕНО ПОМОЩЬ ВОЗВРАТ

В БД Отчет Редактирование Развертка Распределение Буфер Помощь Возврат

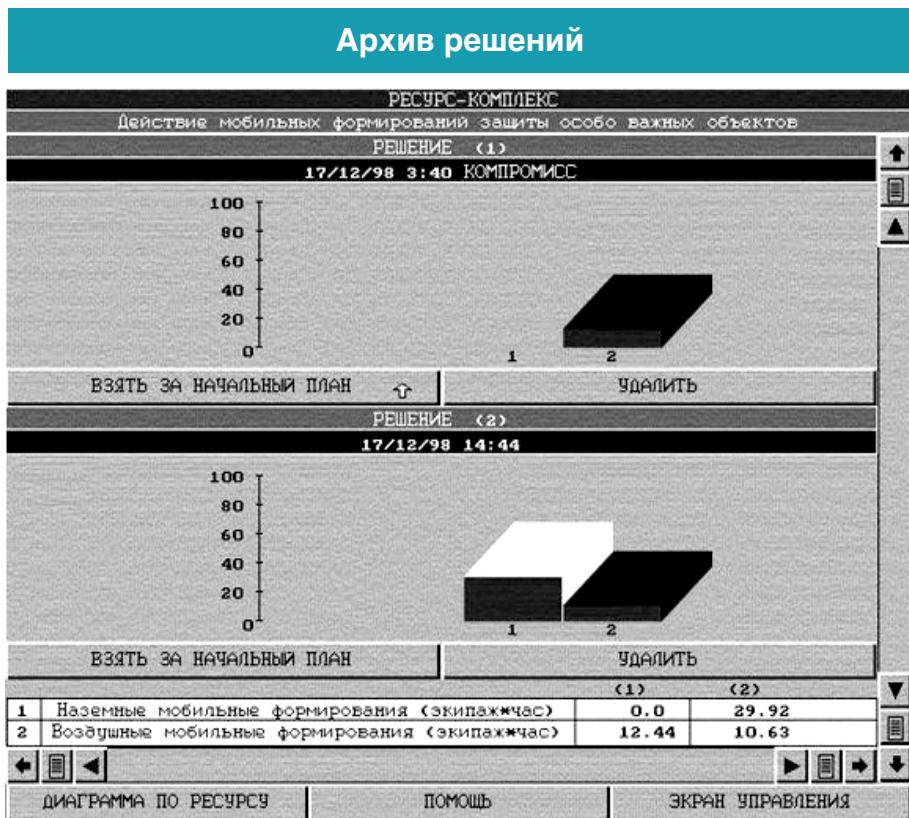
Система управления и настройки РЕСУРС-комплекса включают иерархическую систему меню, визуализируемых в виде наборов кнопок. В многоресурсном распределении имеется система выбора шагов для изменения значений функций и система задания требований к этому изменению.

Система меню позволяет не только переходить на другие уровни иерархии РЕСУРС-комплекса, но и выполнять с помощью

вычислительного ядра расчёты, производить различные действия с таблицами.

[Содержание <](#)

В многоресурсном распределении система меню позволяет выбирать в качестве начального плана для дальнейших расчетов решения из архивов на диске, а также удалять с диска хранящиеся там решения.



В режиме многоресурсного распределения графический интерфейс пользователя позволяет управлять целевым перемещением решения и оценивать его свойства, отслеживая текущие значения ресурсных функций на специальных диаграммах. Пользователь видит изменения интересующих его показателей на каждом шаге перемещения. В случае нарушения ресурсного ограничения

диаграмма соответствующей функции выделяется цветом.

Дружественный интерфейс позволяет оперативно настраивать приложение – эксперт может быстро изменить как направление и дискрет перемещения, так и состав отображаемых диаграммами ресурсных функций.

Траектория перемещения запоминается, что даёт возможность отката изменений.

Предусмотрен механизм 3D-визуализации результатов текущих расчётов и вариантов хранящихся на диске решений, позволяющий просматривать диаграммы решений задач.

Воплощением центра управления вычислительными процессами многоресурсного распределения служит экран управления.

Таблица в верхней части экрана представляет ресурсные функции.

В её строках показаны горизонтальные одиночные диаграммы функций (одиночные в том смысле, что соотношение длин полос под именами функций не отражает соотношения значений этих функций, измеряемых в общем случае разными единицами).

Числовые значения функций показаны в графе <Расход>, а значения запасов для ограниченных функций — в графе <Остаток>.

Графа <Управление> позволяет накладывать требования на любые функции — задавать направление движения в ресурсном пространстве.

Чтобы наложить, изменить или снять с некоторой функции требование, нужно установить курсор мыши в соответствующей клетке графы и последовательными нажатиями левой кнопки мыши установить в кружке нужный значок: стрелка вверх — требование Увеличить; стрелка вниз — Уменьшить; «кирпич» — Фиксировать; пустой кружок — отсутствие требования.

Экран управления

РЕСУРС-КОМПЛЕКС				
Действие мобильных формирований защиты особо важных объектов				
Шаг	Управление	Расход	Остаток	
>>	Топливо (т)	<input type="text" value="0.62"/>	3.38	↑
>>	Средства пожаротушения (т)	<input type="text" value="73.11"/>	-53.11	
>>	Средства дезактивации (т)	<input type="text" value="4.52"/>	0.48	
>>	Средства дезинфекции (т)	<input type="text" value="15.35"/>	-12.35	
>>	Средства дегазации (т)	<input type="text" value="43.4"/>	-31.4	
>>	Площадь обработки (кв.км)	<input type="text" value="5000.0"/>	0.0	↑
>>	Стоимость работы МГП (млн.руб)	<input type="text" value="255.9"/>	144.1	
>>	Применение наземных формирований (экипаж/час)	<input type="text" value="29.92"/>	0.08	
>>	Применение воздушных формирований (экипаж/час)	<input type="text" value="10.63"/>	1.37	
План				
	Наземные мобильные формирования (экипаж/час)	<input type="text" value="29.92"/>		↑
	Воздушные мобильные формирования (экипаж/час)	<input type="text" value="10.63"/>		
				ВПЕРЕД
				Отмена
				ПОМОЩЬ
				ВОЗВРАТ
РЕШЕНИЕ РЕДАКТОР ДАННЫХ ОПТИМИЗАЦИЯ КОМПРОМИСС ВОЗВРАТ				

Графа кнопок <Шаг> слева от таблицы позволяет менять дискрет движение. Чтобы изменить задающий шаг для изменения значения функции, надо щелкнуть левой кнопкой мыши на соответствующей кнопке графы и затем ввести желаемое значение шага.

В общем случае пользователь не обязан вмешиваться в процесс выбора дискрета

движения: РЕСУРС-комплекс способен делать это без его участия.

Значение любой функции можно изменить напрямую, невзирая на общую совокупность требований к функциям.

Для этого нужно щёлкнуть левой кнопкой мыши на соответствующей клетке графы <Расход> или <Остаток> и затем ввести желаемое значение функции или остатка. РЕСУРС-комплекс спроектирует текущее решение на гиперплоскость, заданную введённым значением.

При этом изменение некоторых функций может стать противоречащим наложенным на них требованиям.

В таком случае РЕСУРС-комплекс предлагает два варианта: либо внесение введённых изменений со снятием требований с функций, либо отмену изменений.

Таблица в нижней части экрана представляет значения переменных.

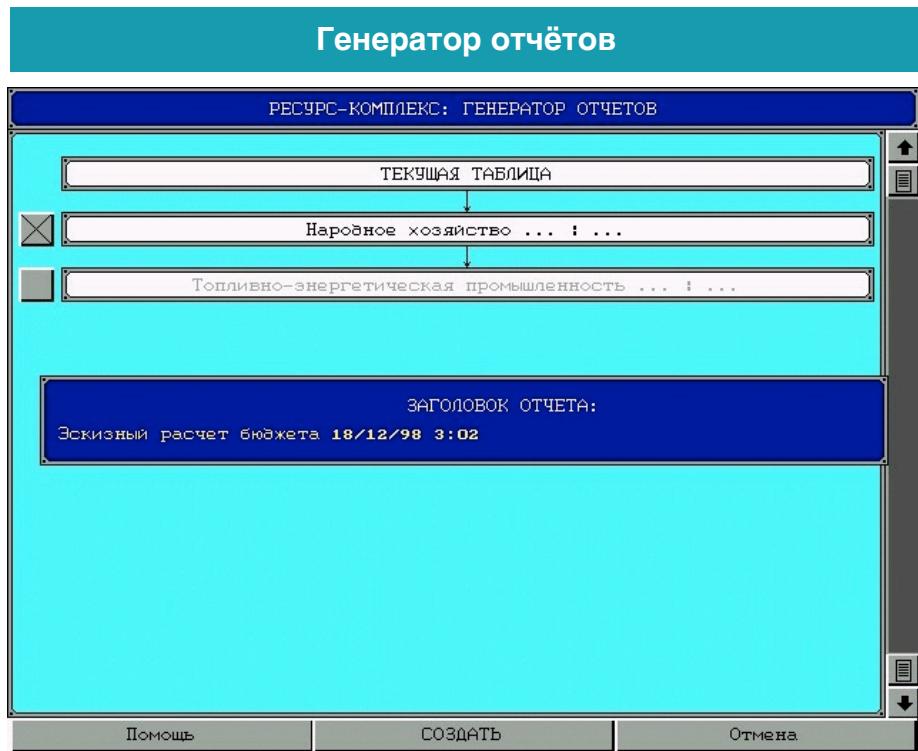
Здесь допустимо изменить значение или имя любой переменной.

Для этого нужно щёлкнуть левой кнопкой мыши на соответствующей клетке и затем ввести нужное число или строку имени.

Генератор отчетов

Генератор отчётов позволяет создавать и распечатывать различные формы выходных документов.

Для одноресурсного распределения генератор позволяет задать степень детализации отчёта, то есть количество уровней иерархии, которые следует включить в отчёт, начиная с текущего.



Можно задать необязательные параметры (алфавитная сортировка, включение приоритетов и величин запросов), а также имя отчёта.

По умолчанию именем отчета становится строка <имя задачи> <дата и время создания отчета>.

Далее показано создание отчёта для решения задачи эскизного расчёта

бюджета. Степень детализации выбирается на схеме уровней разверток расходных статей. Идентификаторы уровней - названия первых расходных статей на каждом уровне, встретившихся РЕСУРС-комплексу при анализе дерева разверток. Включение уровня в отчёт обозначается крестом на кнопке слева от идентификатора.

Литература

Агафонов В. Н. Спецификация программ: понятийные средства и их организация / В. Н. Агафонов. - Новосибирск: Наука, 1987.

Барышников В. Н., Ильин В. Д., Курош Б. Н., Сёмик В. П. Обработка результатов эксперимента в системе ДИЭКС / В. Н. Барышников, В. Д. Ильин, Б. Н. Курош, В. П. Сёмик. - Управляющие системы и машины. - 1984. - № 1. - с. 62-64.

Бежанова М. М. Требования к языку спецификаций пакетов прикладных программ / М. М. Бежанова. - Пакеты прикладных программ. Системное наполнение (Сер. Алгоритмы и алгоритмические языки). - М.: Наука, 1984 . - с. 3-12.

Ильин А. В. Математическое обеспечение процессов преобразования ресурсов / А. В. Ильин. - Системы и средства информатики. - Вып. 9. - М.: Наука, 1999. - с. 159-177.

Ильин А. В. Конструирование разрешающих структур на задачных графах системы знаний о программируемых задачах / А. В. Ильин. - Информационные технологии и вычислительные системы. – 2007. – № 3. – с. 30–36.

Ильин А. В. Интерактивный режим / А. В. Ильин. - Большая Российская Энциклопедия. - т. 11. - М.: Большая Российская Энциклопедия, 2008 - с. 435.

Ильин А. В. Интерфейс в информатике / А. В. Ильин. - Большая Российская Энциклопедия. - т. 11. - М.: Большая Российская Энциклопедия, 2008 - с.457-458.

Ильин А. В., Ильин В. Д. Архитектура вычислительного ядра комплекса программных средств ресурсного обоснования решений / А. В. Ильин, В. Д. Ильин. - М.: ИПИАН, 1995. - 23 с.

Ильин А. В., Ильин В. Д. Интерактивный преобразователь ресурсов с изменяемыми правилами поведения / А. В. Ильин, В. Д. Ильин. - Информационные технологии и вычислительные системы. – 2004. – № 2. – с. 67–77.

Ильин А. В., Ильин В. Д. Распределение ресурсов по обязательным и ориентирующим правилам: сравнительная эффективность алгоритмов / А. В. Ильин, В. Д. Ильин. - Системы и средства информатики. - Вып. 15. - М: Наука, 2005. - с.123-159.

Ильин А. В., Ильин В. Д. Основы теории s-моделирования [Электронный ресурс] = Basics of the theory of s-modeling : [монография] : для информатиков, ИТ-разработчиков, преподавателей вузов и аспирантов / А. В. Ильин, В. Д. Ильин. - Электрон. текстовые дан. – М.: ИПИ РАН, 2009.

Ильин А. В., Ильин В. Д. S-моделирование объектов информатизации [Электронный ресурс] = S-modeling of informatization objects : [монография] : для информатиков, ИТ-разработчиков, преподавателей вузов и аспирантов / А. В. Ильин, В. Д. Ильин. - Электрон. текстовые дан. – М.: ИПИ РАН, 2010.

Ильин А. В., Ильин В. Д. Символьное моделирование в информатике [Электронный ресурс] = The symbol modeling in informatics : [монография] : для информатиков, ИТ-

[Содержание <](#)

проектировщиков, преподавателей вузов и студентов / А. В. Ильин, В. Д. Ильин. - Электрон. текстовые дан. – М.: ИПИ РАН, 2011.

Ильин А. В., Ильин В. Д. S-моделирование задач и конструирование программ [Электронный ресурс] = S-modeling of tasks and construction of programs : [монография] : для специалистов в автоматизации программирования и разработчиков программных средств / А. В. Ильин, В. Д. Ильин. - Электрон. текстовые дан. - М.: ИПИ РАН, 2012.

Ильин В. Д. Система ИГЕН. Концепция, архитектура, технология программирования / В. Д. Ильин. - Современные средства информатики. - М.: Наука, 1986. - с.117-125.

Ильин В. Д. Генератор прикладных задач в системе ИГЕН / В. Д. Ильин. - ЭВМ массового применения. - М.: Наука, 1987. - с. 28-37.

Ильин В. Д. Система порождения программ / В. Д. Ильин. - М.: Наука, 1989. - 264 с.

Ильин В. Д. Порождение пакетов программ / В. Д. Ильин. - Системы и средства информатики. - М.: Наука, 1989. - с. 32-65.

Ильин В. Д. Интернет. Большая Российская Энциклопедия / В. Д. Ильин. - Большая Российская Энциклопедия. - т. 11. - М.: Большая Российская Энциклопедия, 2008 - с. 451-452.

Ильин В. Д. Информационные ресурсы. Большая Российская Энциклопедия / В. Д. Ильин. - Большая Российская Энциклопедия. - т. 11. - М.: Большая Российская Энциклопедия, 2008 - с. 492-493.

Ильин В. Д. Информатизация. Большая Российская Энциклопедия / В. Д. Ильин. - Большая Российская Энциклопедия. - т. 11. - М.: Большая Российская Энциклопедия, 2008 - с. 480-481.

Ильин В. Д. Компьютерная сеть. Большая Российская энциклопедия / В. Д. Ильин. - Большая Российская Энциклопедия. - т. 14. - М.: Большая Российская Энциклопедия, 2009 - с. 711-712.

Ильин В. Д. Компьютерное моделирование. Большая Российская энциклопедия / В. Д. Ильин. - Большая Российская Энциклопедия. - т. 14. - М.: Большая Российская Энциклопедия, 2009 - с. 712.

Ильин В. Д., Куров Б. Н., Толстохлебов С. В. Массовые вычисления в ИГЕН-пакетах прикладных программ / В. Д. Ильин, Б. Н. Куров, С. В. Толстохлебов. - ЭВМ массового применения. - М.: Наука, 1987. - с. 45-54.

Ильин В. Д., Мартынов А. В. Взаимодействие с пользователем в ИГЕН-среде порождения программных систем / В. Д. Ильин, А. В. Мартынов. - Программирование. - 1988. - № 3. - с. 48-56.

Тыугу Э. Х. Концептуальное программирование / Э. Х. Тыугу. - М.: Наука, 1984.

Basset P. Design principles for software manufacturing tools / P. Basset. - Proc. ACM'84 Ann. Conf. Fifth Generation Challenge. October 8-10. 1984. - p. 85-93.

[Содержание <](#)

Broy M. Program construction by transformation: a family tree of sorting programs / M. Broy. - Comput. Synthesis Methodol. - Proc. NATO Adv. Study Inst. Ser.C. - 1981. - Vol. 95. - p.1-49.

Conway J., Watts S. Software Engineering with LabVIEW / J. Conway, S. Watts. - Pearson Education. - 2003

Darlington J. The synthesis of implementations for abstract data types. A program transformation tactic / J. Darlington. - Comput. Synthesis Methodol. - Proc. NATO Adv. Study Inst. Ser.C. - 1981. - Vol.95. - p. 309-334.

Dijkstra E. Structured programming. Software Engineering Techniques / E. Dijkstra. - NATO Science Committee (edited by Burton J. And Randell B.). - 1969. - p. 89-93.

Horowitz E., Kemper A., Narasimhan B. A survey of application generators / E. Horowitz, A. Kemper, B. Narasimhan. - IEEE Software. - 1985. - Jan. - p. 40-54.

Ilyin V. D. A Methodology for Knowledge Based Engineering of Parallel Program Systems / V. D. Ilyin. - Proc. Of the Eighth Int. Conf. Industrial and Engineering Applications of Artificial Intelligence and Expert Systems. - Gordon and Breach Science Publishers, Inc., Newark, NJ. - 1995. - p. 805– 809

Lee R., Chang S. Structured programming and automatic program synthesis / R. Lee, S. Chang. - Proc. of the ACM SIGPLAN symposium on Very high level languages. New York. April 1974. - p. 60-70.

Luker P. A., Burns A. Program generators and generation software / P. A. Luker, A. Burns. - Comp. J. - 1986. - Vol. 29. - No. 4. - p. 315-321.

Manna Z., Waldinger R. Towards automatic program synthesis / Z. Manna, R. Waldinger. - Commun. ACM. - 1971. - Vol. 14. - No. 3. - p. 151-164.

Manna Z., Waldinger R. Deductive synthesis of the unification algorithm / Z. Manna, R. Waldinger. - Comput. Synthesis Methodol. - Proc. NATO Adv. Study Inst. Ser.C. - 1981. - Vol. 95. - p. 251-307.

Polster F. J. Reuse of software through generation of partial systems / F. J. Polster. - IEEE Trans. - Software Eng. - 1986. - Vol. 12. - No. 3, March. - p. 402-416.

[Содержание <](#)

Научное издание

Ильин Владимир Дмитриевич

СИСТЕМА ПОРОЖДЕНИЯ ПРОГРАММ.
Версия 2013 г.

Электронная книга изготовлена автором

Утверждено к изданию

Институтом проблем информатики Российской академии наук

20.09.2013

www.ipiran.ru

119333, Москва, ул. Вавилова, д. 44, корп. 2