УДК 004
ББК 32

ФОРМАЛИЗАЦИЯ ЗНАНИЙ О ЗАДАЧАХ: ЗАДАЧНЫЕ КОНСТРУКТИВНЫЕ ОБЪЕКТЫ

Ильин А. В.[1], Ильин В. Д.[2]
(*Учреждение Российской академии наук*
Институт проблем информатики РАН, Москва)

Статья посвящена представлению знаний о программируемых задачах. Рассмотрена методология построения символьных моделей задач, рассчитанных на применение в разработках систем знаний о задачах. Методология основана на результатах, полученных при разработке и исследованиях систем автоматизированного конструирования программ. Для формализованного представления задач используется TSM-комплекс описания символьных моделей произвольных объектов в человеко-машинной среде (s-среде). Задачи представлены как конструктивные объекты (s-задачи) с определёнными типами межзадачных связей и правилами конструирования. Описание s-задачи включает постановку, системы обязательных и ориентирующих правил решения, алгоритмы, библиотеки программ и тестовых примеров. В условиях применения программ указаны операционные системы, требования к аппаратным средствам и др.

Ключевые слова: s-моделирование, задачный конструктивный объект, s-задача, s-задачный граф, исчисление s-задач, система знаний о задачах, TSM-комплекс.

1. Введение

Современная *s-среда* [10], ядром которой является Интернет, представляет собой сложный объект с изменяющимися ха-

[1] *Александр Владимирович Ильин*, *кандидат технических наук* (*avilyin@ipiran.ru*).
[2] *Владимир Дмитриевич Ильин*, *доктор технических наук*, *профессор* (*vdilyin@ipiran.ru*).

рактеристиками, в построении которого участвуют тысячи людей, а в пользовании — миллионы. Во времена, когда s-среды не существовало, пути к применению методов решения нетривиальных задач были весьма непростыми. Метод надо было освоить, найдя соответствующую литературу на бумажных носителях. Освоив, можно было (например, с помощью логарифмической линейки, бумаги и карандаша) приступить к решению задачи. В проектировании, исследованиях и других видах деятельности, связанных с решением задач, производительность была в десятки раз ниже, чем теперь. Среди массы попыток создать машину, помогающую решать задачи, одна оказалась существенно успешнее других. В основе успеха – идея построения решателя-автомата со сменяемыми программами, хранимыми в его памяти (в *s-моделировании* такие автоматы названы *s-машинами* [10]).

Шестьдесят лет назад трудно было представить современные масштабы массового применения технологий автоматизированного построения текстов, изображений, аудио- и видео композиций. Вряд ли можно было даже приблизительно оценить масштабы перемен, которые произойдут в массовом применении s-машин [10] и построенной на их основе s-среды[1]. Совершенствование s-машинных решателей задач открыло не существовавшие ранее возможности автоматизации процессов сохранения, накопления, поиска и применения знаний, помогающих решать задачи. Примером одного из наиболее продвинутых воплощений идеи повторного применения профессиональных знаний могут служить современные технологии автоматизированного проектирования в электронике, машиностроении и многих других областях.

2. О двух направлениях в автоматизации программирования

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

[1] Интернет-сервисами (электронной почтой, поиском, IP-телефонией и др.) пользуются миллионы людей.

программных средств. Уже на начальных этапах развития программирования вместе с языками и трансляторами они стали создавать различные библиотеки программ. [3, 30]. 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 [1, 6, 20, 22, 23, 33] в [12] (на основе опыта успешной реализации и применения генератора программ для создания, обработки и представления табличных структур [2]) был предложен подход к построению программ как интерактивному конструированию из *задачных конструктивных объектов.* В системе автоматизированного конструирования программ (названной *системой порождения программ*[1] [13-16]) *система знаний о программируемых задачах* представлена в форме конструкций, построенных из задачных конструктивных объектов. Результаты исследований, посвящённых системе порождения программ, представлены в [4, 5, 7-9, 14-16, 18]. В [31] этот подход был развит применительно к конструированию параллельных программ.

Основания для автоматизации любой деятельности тем значительнее, чем лучше она изучена и чем удачнее формализована её технология [10]. 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 [32, 34]. Второе — теми, кто не считает программу формальным объектом и рассматривает её как *сообщение, определяющее поведение автомата с заданными свойствами и существующее в символьном, кодовом и сигнальном воплощениях, связанных отношениями трансляции* [4, 5, 7-19, 21, 24-36]. В работах [20, 23], где говорится о синтезе программ

[1] В системе порождения программ были предложены и применены трёхмерные многослойные табличные структуры (ТАБСы) (разд. 7.2 книги [13]), которые использовались при реализации.

для решения задач заданной предметной области, в реализации использованы заготовленные программные модули.

Не станем приводить здесь аргументацию несостоятельности первого подхода: с нею можно ознакомиться в [10]. Ограничимся цитированием аннотации доклада об автоматическом синтезе программ и структурном программировании, сделанного в 1971 году видными представителями первого направления [32]: «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 [24] Steve Watts удачно выразил суть этого направления: «A lot of techniques and methodologies get bogged down with computer science and forget about the design aspects; our intensions are to always concentrate on design and hopefully some of the computer science.»

Разработка программных средств имеет много общего с проектированием человеко-машинных систем управления[1]. Это подтверждает авторский опыт разработки методов и программных средств решения задач диспетчерского управления энергосистемами[2], распределения ресурсов в системах ситуационного управления организационно-техническими комплексами [10, 11], автоматизированного конструирования программ [4, 5, 7-9, 14-16, 18, 31] и построения образовательных ресурсов [19].

Вполне естественно, что продуктивные идеи[3] автоматизации программирования родились в среде программистов. Заметим, что успех систем программирования семейства ОС UNIX во многом определяется возможностями языка С и тем, как в этой операционной системе реализована идея *повторного при-*

[1] Поэтому есть основания полагать, что рассматриваемая в статье методология применима и для построения систем знаний о задачах управления.

[2] Список опубликованных работ можно посмотреть в статье [21] этого же выпуска.

[3] Такие, как идея модульного проектирования программ по принципу «сверху вниз», реализованная в технологии структурного программирования [26].

менения программ[1]. В сообщении ACM[2] о присуждении К. Томпсону[3] и Д. Ритчи[4] премии Тьюринга (1983 г.) за разработку теории операционных систем и ОС UNIX отмечено, что успех этой системы предопределён удачными ключевыми идеями и их элегантной реализацией.

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 (где задачи представлены как конструктивные объекты, содержащие описание постановки, методов, алгоритмов, библиотек программ и тестовых примеров), а системы знаний — между собой. Связанные между собой системы могут служить и образовательными ресурсами. Структурированность, возможности расширения и коррекции таких систем станут эффективными помощниками в борьбе с пресловутой проблемой сложности.

Цель этой статьи — представить методологию формализованного описания программируемых задач, предназначенную для разработки систем знаний о задачах, реализуемых в s-среде[5].

---

[1] В ОС UNIX можно сохранить наработанное для последующего использования (в каталогах /bin, /usr/bin, /etc). UNIX - (каналы, процессы и файловая система) построены с расчётом на повторное применение программ. Принцип повторного применения использовался и при разработке самой ОС.

[2] Association for Computing Machinery http://www.acm.org/

[3] Ken Thompson http://www.linfo.org/thompson.html

[4] Dennis MacAlistair Ritchie http://cm.bell-labs.com/cm/cs/who/dmr/

[5] Автоматизированное конструирование программ [4, 5, 7-9, 14-16, 18, 31] в статье не рассматривается из-за естественных ограничений на её объём.

3. TSM-комплекс: краткое описание

□ *TSM-комплекс,* предназначенный для описания символьных моделей произвольных объектов в человеко-машинной среде (*s-моделей*) [10], включает средства одноуровневой записи формул, выделения частей описаний и замены выбранными сокращениями часто повторяющихся фрагментов. □

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), Writer (пакета OpenOffice.org) или др.

Первая версия TSM была предложена при работе над теорией порождения программ [13]. Факторами последующего продвижения TSM стали отсутствие ограничений на сложность символьных конструкций и расширяемость, а также — минимальные требования [стандартная клавиатура и текстовый редактор (и никаких редакторов формул)]. Универсализации TSM способствовало применение этого комплекса при формировании образовательных ресурсов и разработке системы знаний информатики СИНФ [19].

3.1. ФРАГМЕНТЫ ОПИСАНИЯ

□ *Фрагмент 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/).

3.2. ВЫДЕЛЕНИЯ

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

□ <фрагмент описания> □ ≈ утверждение (определение, аксиома и др.); здесь и далее символ ≈ заменяет слово означает;

◊ <фрагмент описания> ◊ ≈ замечание;

☼ <фрагмент описания> ☼ ≈ пример;

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

☼{S*модель*<список>S} – здесь расширенным префиксом служит s-модель;

{S<список>S} – здесь префикс s- ☼.

Курсивом выделяются:

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

3.3. ССЫЛКИ И СОКРАЩЕНИЯ

Перекрёстные ссылки применяются для связи между составляющими описания (в одном файле), а гиперссылки — для связи между описаниями, размещёнными в разных файлах и с внешними ресурсами s-среды (включая её сервисы: поиск, онлайн-перевод и др.)[1].

◊ Применение механизма перекрёстных ссылок позволяет обходиться однократным описанием (при первом вхождении) областей определения аргументов функций, множеств значений индексов и др.◊

◊ Применение механизма гиперссылок позволяет элементы описания связать с файлохранилищами, электронными библиотеками, энциклопедиями и др. информационными ресурсами s-среды.◊

Для часто повторяющихся названий понятий применяются сокращения:

СМ ≈ символьное моделирование;

[1] Ссылки делаются средствами редакторов уровня Word (пакета MS Office), Writer (пакета OpenOffice.org) или др.

S-моделирование ≈ СМ произвольных объектов в человеко-машинной среде;

s-машина ≈ машина, помогающая создавать и применять s-модели;

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

3.4. 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.

3.5. 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 вертикальных черточек «|», 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|* ≈ вектор *x* из *n* компонент, имеющий помету *out*; *a |inp; i = 1…m, j = 1…n|* ≈ матрица *a* размера *m*n*, имеющая помету *inp*; *c |'; 1|* ≈ *c*-один со штрихом (штрих «'» – верхняя помета, 1 – нижний индекс); *d |j/i;|* ≈ *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|* ≈ *d* с нижним индекси-

рованным индексом *j i*-тое (отсутствие точки с запятой указывает на отсутствие верхних индексов)☼.

/ Теоретико – множественные

a: elem A ≈ *a* является элементом множества *A*; *a, b: elem C* ≈ *a, b* — элементы множества *C* (число элементов, разделённых запятыми, может быть любым); *A: set a* ≈ *A* – множество, содержащее элемент *a*;

A < *B* (когда оговорено, что *A* и *B* рассматриваются как множества) ≈ *A* – подмножество *B*; *B* = *D* ≈ множества *D* и *B* совпадают; *C* ≤ *B* ≈ *C* является подмножеством *B* или совпадает с ним; *B* > *A* ≈ *B* содержит *A*; *A* ≥ *E* ≈ *A* содержит *E* или совпадает с *E*;

A ∨ B ≈ объединение множеств *A* и *B*; *A* ∧ *B* ≈ пересечение множеств *A* и *B*; *A* \ *B* ≈ разность множеств *A* и *B*; *A* * *B* ≈ декартово произведение множеств *A* и *B*; *R* ≤ *A* * *B* ≈ 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* ≤ *A* * *B*, то *(X* * *Y)* ∧ *(A* * *B)* # *0*. ☼

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

☼ *f (x)* ≈ *f* от *x*; *f |max;| (x |i = 1…n|)* ≈ *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 заглавной «сигмы» используется *sum*; при этом индекс суммирования, его начальное и конечное значения записываются в вертикальных черточках «|» справа от *sum*.

☼ *sum |i = 1…n| x |i|* ≈ сумма *x | i |* по *i* от *1* до *n*. ☼

3.6. ТИПЫ: СПЕЦИАЛИЗАЦИЯ И ОБОБЩЕНИЕ

□ *Тип X* ≈ 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. □

4. S-задачи: основные понятия

□ *S-задача*[1] – это четверка {*Formul, Rulsys, Alg , Prog*}, где *Formul* – постановка задачи; *Rulsys* – множество систем *обязательных* и *ориентирующих правил решения задачи* [10-11], 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 *Mem = Inp ∨ Out* (*Inp ∧ Out = 0*) и совокупность *Rel* связей между понятиями, определяющая бинарное отношение *Rel < Inp * Out*. Множество *Mem* назовем также *памятью зада-*

[1] Во фрагментах текста, соответствующих определениям (заключены между двумя специальными символами □) — *s-задача,* в других частях текста — *задача.*

чи, 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 кооперация s-машин*, *кооперация 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.

◊ Каждая программа сопровождается ссылками на наборы тестовых примеров (см. разд. 7). ◊

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

□ *S-алгоритм*[1] *— система правил* (соответствующая одному из элементов *Rulsys*) решения задачи, позволяющая за конечное число шагов поставить в однозначное соответствие заданному набору *данных* [10], 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. □

□ *S-программа*[2] *- реализованный* (на языке программирования высокого уровня, машинно-ориентированном языке и/или в системе машинных команд) *s-алгоритм, представленный в фор-*

[1] Далее — алгоритм.

[2] Далее — программа.

ме сообщения, определяющего поведение s-машинного решателя задачи с заданными свойствами. Существует в *символьном*, *кодовом* и *сигнальном* воплощениях, связанных отношениями трансляции. □

4.1. СВЯЗИ 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|, elem Mem |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 определенный элемент из *H,* то говорим, что задана функция связи по памяти *h = conn (s |i|, s |j|).* При этом *D* называем *областью определения функции conn* и обозначаем *D |conn|*. Множество *R = {h: elem H; h = conn (s |i|, s |j|); s |i|: elem D |conn|, s |j|: elem D |conn|}* называем *областью значений функции conn.* □

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. Функция связи по памяти типа *вход-вход* обозначена через *conn |x|*, *выход-вход* – через *conn |yx|* и *выход-выход* – через *conn |y|*.

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. Определены два типа *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.

4.2. КОНСТРУИРОВАНИЕ И КОНКРЕТИЗАЦИЯ S-ЗАДАЧ

□*S*-(*конструирование задачи*) реализуется посредством связи по памяти между задачами. *Элементарная задачная конструкция* — это 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.□

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

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

□*S-(конкретизация задачи)* — переход (*Formul → элемент из Rulsys → элемент из Alg → элемент из Prog*), выполненный в соответствии с *описанием применения* [3].□

5. Система знаний об s-задачах: определения

□ *Система pS знаний о задачных конструктивных объектах* (*s-задачах*, называемых также *p-объектами*) – это триада <*pA, lng, intr*>, где *pA* – задачная область, *lng* – язык спецификации р-объектов, *intr* – интерпретатор спецификаций искомых р-объектов на *pA*. □

Пусть P – множество всех s-задач, а $A < P$ – 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 pA* – это *p-*объект, который задаётся парой <память *mem* |*A*| множества задач *A* задачной области *pA* >, <семейство *rel* (*mem* |*A*|) связей, заданных на *mem* | *A*|>. 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, на которой интерпретируются спецификации искомых задач, составленные на языке *lng*.

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

Формально интерпретация спецификации искомого p-объекта на *pA* – это *конструктивное доказательство существования разрешающей структуры* [10, 13].

Задачный граф служит формальным представлением *задачной области*, рассчитанным на реализацию процесса *р-конструирования* и формализацию задачных знаний. Множество вершин графа, составленное из p-объектов, 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 *mem |comp| = inp |comp| ∨ out |comp|; inp |comp| ∧ out |comp| = 0*, определяющее вход *inp |comp|* и выход *out |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.◊

В зависимости от состава вершин определены следующие типы задачных графов:

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

□ Связный граф с непустым множеством ребер и задачным базисом *p-basis*, все элементы которого являются простыми задачами, называется *U-графом* и обозначается *U-graph*: *U-graph* = (*p-basis, set* |*ver*|), где *set* |*ver*| – множество ребер задачного графа. Объединение памяти задачных вершин, составляющих базис, называется *памятью U-графа* и обозначается *mem* |*U-graph*|.

На памяти *U-графа* задано разбиение:

- *Giv* |*U-graph*| *задаваемых элементов памяти* называется подмножеством *входных элементов*;
- *Comput* |*U-graph*| *вычисляемых элементов памяти* называется подмножеством *выходных элементов*;
- *Or обратимых элементов памяти* называется разность (*mem* |*U-graph*|) \ (*Giv* |*U-graph*| ∨ *Comput* |*U-graph*|) □.

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

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

Конкретным воплощением задачной области может быть граф любого типа (*U*-, *C*- или *G*-граф). Тот факт, что *G*-граф мо-

жет 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.

В системе знаний об s-задачах *p*-(*специализация, обобщение, конкретизация и конструирование*) — средства построения задач.

6. Исчисление s-задач: основные понятия

Объединение множеств базовых задачных объектов и сконструированных объектов обозначено через *P* и названо *множеством p-объектов*. Совокупность пространств, построенных на подмножествах множества *P,* названа *миром p-объектов* (или *миром s-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 *p*-объектов (*мир p-объектов*) не имеет логических ограничений на расширение.

□ *Исчисление задачных конструктивных объектов Igen* (для краткости названное *исчислением s-задач*) — это построение некоторого множества *T* конструктивных объектов (*t-объектов*, обозначаемых через *t*) на основе множества *B* 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 (*lng, B, T, R*), включающая язык *lng* описания задачных объектов и конструкций из объектов, непустое множество *B* базовых объектов, порождаемое множество *T* (*B ^ T = 0*; до начала процесса задачного конструирования *T = 0*) и множество *R* правил построения объектов. □

В *Igen* символы *conn* |*x*|, *conn* |*yx*|, *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 объяв-

лениях и в функциях. Символ *im* – обозначение образа некоторого объекта (рассматриваемого как прообраз). Этот символ употребляется (так же, как и символ *pre*) в объявлениях и функциях. Множество *B* состоит из базовых объектов (*b-объектов*), являющихся объектами первого поколения. Иначе говоря, любой *b*-объект не имеет прообраза, но может иметь образы. Любому *b*-объекту может быть поставлено в соответствие некоторое непустое множество объектов-образов *set* (*t* |*im; p* (*b*)|) = *im* (*b*) (в левой части символ *im* является верхней пометой, а *p* (*b*) — нижней, обозначающей указатель на объект-прообраз *b*). Каждый объект-образ *t* |*im; p (b)*| имеет единственный объект-прообраз. Любой объект-образ может, в свою очередь, иметь образы. Объект, множество образов которого пусто, будем называть *конечным объектом*. Объекты, имеющие прообраз и непустое множество образов, будем называть *промежуточными объектами*.

Множество *T* строится на базисе *B* по правилам *R*. В *T* могут существовать конструкции, построенные из объектов, принадлежащих *B*, 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 задачного конструирования при создании конструкций из *b*-объектов, *b*- и *t*-объектов, а также из *t*-объектов.

Rul |*1*| : *t* || *p* (*b*) = *b*. Правило устанавливает, что объект *t* || *p* (*b*) множества *T* может быть построен из одного объекта *b*, принадлежащего множеству *B*. Перед двойной вертикальной чертой «||» размещена буква *t*, обозначающая некоторое имя объекта из *T*; после двойной вертикальной черты размещён указатель на объект, из которого построена конструкция; буква *p* – символ указателя, а в скобках – имя объекта.

Rul |*2*| : *t* || *p* (*im* (*t*)) = *im* (*t*). Объект *t* || *p* (*im* (*t*)) множества *T,* являющийся образом объекта *t*, может быть получен *p-специализацией t*.

Rul |*3*| : *t* || *p* (*pre* (*t*)) = *pre* (*t*). Объект *t* || *p* (*pre* (*t*)) множества *T*, являющийся прообразом объекта *t*, может быть получен *p-обобщением t*.

Rul |*4*| : *t* || *p* (*conc* (*t*)) = *conc* (*t*). Любой объект, полученный путем *p-конкретизации*, может быть *t*-объектом.

Rul |5| : $t \,||\, p\,(t|*|\,||\,p\,(f, s)) = t\,|*|\,||\,p\,(f, s)$. Правило устанавливает, что *t*-объектом может быть любая допустимая конструкция $t\,|*|\,||\,p\,(f, s)$, полученная применением функции связи по памяти *conn* | * | (*f, s*), то есть $t\,|*|\,||\,p\,(f, s) = conn\,|*|\,(f, s)$. Помета * принимает одно из трёх значений (*x, yx, y*), каждое из которых обозначает определённый тип функции *conn* | * |. Конструкция $t\,|*|\,||\,p\,(f, s)$ получена путём применения функции *conn* |*|, аргументами которой служит пара (*f, s*), для которой допустимы следующие значения:

- (*f*: *elem* (*B*), *s*: *elem* (*B*));
- (*f*: *elem* (*B*), *s*: *elem* (*T*));
- (*f*: *elem* (*T*), *s*: *elem* (*B*));
- (*f*: *elem* (*T*), *s*: *elem* (*T*)).

7. 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 заданном отношении к введенному подмножеству (или паре подмножеств). Интерпретации на *C*-графе и *G*-графе аналогичны интерпретации на *U*-графе.

□ *Задача t представима на задачном графе graph*, если вход *inp* |*t*| задачи содержится в подмножестве *Giv* |*graph*| ∨ *Or* | *graph*|, а выход *out* |*t*| – в подмножестве *Comput* |*graph*| ∨ *Or* | *graph*| памяти задачного графа; при этом существует не менее одной задачи из базиса этого графа, вход которой содержится в *inp* |*t*| или совпадает с ним. □

□ *Разрешающей структурой solv |t| на графе graph*, поставленной в соответствие некоторой задаче *t*, называем подграф с минимальным числом задачных вершин, на котором задача *t представима*. □

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

Смысл этого определения поясняют *правила интерпретации задачного узла*:

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

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.

Для каждого из трёх типов запросов в [13] приведено конструктивное доказательство существования разрешающей структуры соответствующего типа. Там эти доказательства даны применительно к задачным сетям. Схемы доказательств существования разрешающих структур на задачных графах аналогичны. Поэтому далее приведём только формулировки теорем.

7.1. ТЕОРЕМА СУЩЕСТВОВАНИЯ РАЗРЕШАЮЩЕЙ СТРУКТУРЫ ТИПА T-SOLV|Z|

Для заданных подмножеств x и y (x ∧ y = 0), памяти тет | t-graph|, тогда существует разрешающая структура t-solv|z| 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).

7.2. ТЕОРЕМА СУЩЕСТВОВАНИЯ РАЗРЕШАЮЩЕЙ СТРУКТУРЫ ТИПА T-SOLV|X|

Для подмножества x тет-элементов, заданного на памяти тет |t-graph| задачного графа, тогда найдётся разрешающая

структура t-solv|x|, вход которой определён подмножеством x, а выход является непустым подмножестом тх памяти графа, включающим максимальное число элементов, которые могут быть определены при заданном x, когда x ∧ Comput = 0 (Comput < тет |t-graph|) и найдётся хотя бы одна вершина с разрешимой задачей.

7.3. ТЕОРЕМА СУЩЕСТВОВАНИЯ РАЗРЕШАЮЩЕЙ СТРУКТУРЫ ТИПА T-SOLV|Y|

Для подмножества y тет-элементов, заданного на тет | t-graph|, тогда найдётся разрешающая структура t-solv|y| с минимальным числом задачных вершин, выход которой содержит y, а вход составлен из элементов, принадлежащих Giv, когда y ∧ Giv = 0.

7.4. КОНКРЕТИЗАЦИЯ ПО ОПИСАНИЮ ПРИМЕНЕНИЯ

Спецификация программируемой задачи включает *описание применения*[1], задающее тип *решателя задачи* (автономная s-машина, сетевая кооперация s-машин и др.), требования к *информационной безопасности* и др.

После того как найдена разрешающая структура, становится осуществимым процесс ее конкретизации в соответствии с *описанием применения* исходной задачи.

◊ Применительно к конструированию разрешающих структур на задачных графах существование набора тестовых примеров – необходимое условие реализации. ◊

◊ Нынешние библиотеки программ различного назначения (в том числе – динамически присоединяемые *dll*) не удовлетворяют требованиям, предъявляемым к s-задачам. В частности, они не содержат ссылок на наборы тестовых примеров. Нет возможности протестировать составляющие нынешних операционных систем и работающих под их управлением приложений. Правило — продавать программные продукты с заранее оговоренной возможностью доступа к наборам тестовых примеров

[1] ◊ *Описание применения* как обязательная составляющая спецификации задачи — одна из отличительных особенностей обсуждаемой методологии. ◊

(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.◊

8. Заключение

Рассмотренная в статье методология построения символьных моделей программируемых задач рассчитана на построение систем знаний о задачах.

Методология применима для представления знаний о задачах не только в системах автоматизированного конструирования программ [4-18, 31], при разработке и исследованиях которых она сформировалась, но и — для представления знаний о задачах в САПРах любого назначения.

Накоплен опыт успешного применения этой методологии при разработке приложений [4, 5, 11, 17] (среди которых — диалоговая программная система *РЕСУРС-комплекс*[1] [11]) и формировании образовательных ресурсов. [19].

TSM-комплекс для описания моделей задач, не имеет ограничений на сложность символьных конструкций и предъявляет минимальные реализационные требования [стандартная клавиатура и текстовый редактор уровня Word (пакета MS Office), Writer (пакета OpenOffice.org) или др.][2].

[1] РЕСУРС-комплекс предназначен для проектирования управленческих решений в условиях неполной информированности эксперта о ресурсной обеспеченности. Задачи разделены на типы, для каждого из которых определены базовые конструктивные объекты. Задачи, решаемые в процессе проектирования, представляют собой *конкретизации* конструкций, построенных из базовых задач.

[2] Не нужны никакие редакторы формул.

Литература

1. БАБАЕВ И. О., НОВИКОВ Ф. А., ПЕТРУШИНА Т. И. *Язык Декарт — входной язык системы СПОРА* // Прикладная информатика. М.: Финансы и статистика, 1981. Вып. 1. С. 35-73.
2. БАРЫШНИКОВ В. Н., ИЛЬИН В. Д., КУРОВ Б. Н., МАРТЬЯНОВ А. В. *Генератор программного обеспечения процессов создания, обработки и представления табличных структур* // Современные средства информатики. М.: Наука, 1986, С. 125-129.
3. *Библиотека численного анализа*. НИВЦ МГУ. - URL: http://num-anal.srcc.msu.su/lib_na/libnal.htm
4. БОРИСОВ В. А., ИЛЬИН В. Д., КУРОВ Б. Н., МАКАРОВ Е. М. *Программная система поиска разрешающих структур на задачных сетях* // Системы и средства информатики. Вып. 5. 1993. С. 41-50.
5. БОРИСОВ В. А., ИЛЬИН В. Д., КУРОВ Б. Н., МАРТЬЯНОВ А. В. *Редактор задач в среде порождения программ* // Системы и средства информатики. Вып. 5. 1993. С. 30-40.
6. БРЯБРИН В. М. *Ф-язык — формализм для представления знаний в интеллектуальной диалоговой системе* // Прикладная информатика. М.: Финансы и статистика, 1981. Вып. 1. С. 73-103.
7. ГАВРИЛЕНКО Ю. В., ИЛЬИН В. Д. *Пространства локальных объектов для семантических интерпретаций задач. 1* // Известия Российской академии наук. Теория и системы управления. - 1996. - № 5. - С. 5-13.
8. ГАВРИЛЕНКО Ю. В., ИЛЬИН В. Д. *Исчисления задачных конструктивных объектов и их интерпретации. 2* // Известия Российской академии наук. Теория и системы управления. - 1997. - № 5. - С. 56-65.
9. ИЛЬИН А. В. *Конструирование разрешающих структур на задачных графах системы знаний о программируемых задачах* // Информационные технологии и вычислительные системы. – 2007. – № 3. – С. 30–36.

10. ИЛЬИН А. В., ИЛЬИН В. Д. *S-моделирование объектов информатизации*. – М.: ИПИ РАН, 2010. – 412 с. – URL: http://smodeling.files.wordpress.com/2011/01/avvd-ilyin-smoi-2010_dec31_21.pdf
11. ИЛЬИН А. В., ИЛЬИН В. Д. *Интерактивный преобразователь ресурсов с изменяемыми правилами поведения* // Информационные технологии и вычислительные системы. – 2007. – № 3. – С. 67–77.
12. ИЛЬИН В. Д. *Система ИГЕН. Концепция, архитектура, технология программирования* // ЭВМ массового применения. М.: Наука, 1987, С. 28-37.
13. ИЛЬИН В. Д. *Система порождения программ*. М.: Наука, 1989, 264 С.
14. ИЛЬИН В. Д. *Порождение пакетов программ* // Системы и средства информатики. М.: Наука, 1989, С. 39-66.
15. ИЛЬИН В. Д. *Порождение целевых программных систем: элементы теории* // Системы и средства информатики. М.: Наука, 1989, Вып. 2, С. 3-43.
16. ИЛЬИН В. Д., КУРОВ Б. Н. *Специфицирование задач и задачных сетей в системе порождения программ* // Системы и средства информатики. М.: Наука, 1989, Вып. 2, С. 44-58.
17. ИЛЬИН В. Д., КУРОВ Б. Н., ТОЛСТОХЛЕБОВ С. В. *Массовые вычисления в ИГЕН-пакетах прикладных программ* // ЭВМ массового применения. М.: Наука, 1987, С. 45-54.
18. ИЛЬИН В. Д., МАРТЬЯНОВ А. В. *Взаимодействие с пользователем в ИГЕН-среде порождения программных систем* // Программирование, 1988, № 3, С. 48-56.
19. ИЛЬИН В. Д., СОКОЛОВ И. А. *Символьная модель системы знаний информатики в человеко-автоматной среде* // Информатика и её применения, 2007, т. 1, № 1, С. 66-78. URL: http://www.mathnet.ru/php/archive.phtml?wshow=paper&jrnid=ia&paperid=121&option_lang=rus
20. КАХРО М. И., КАЛЬЯ А. П., ТЫУГУ Э. Х. *Инструментальная система программирования ЕС ЭВМ (ПРИЗ)*. М.: Финансы и статистика, 1981, 158 С.

21. КУРОВ Б. Н. *Сравнение эффективности алгоритмов управления с учётом точности данных и реализации решений* / Управление большими системами. – 2011. – Выпуск 34. – С. ...- ...
22. ОПАРИН Г. А. *Сатурн — метасистема для построения пакетов прикладных программ* // Разработка пакетов прикладных программ. Новосибирск: Наука, 1982, С. 130-160.
23. ТЫУГУ Э. Х. *Концептуальное программирование*. М.: Наука, 1984, 256 С.
24. BOEHM B. *Software engineering*. In book: Classics in software engineering. Yourdon Press Upper Saddle River, NJ, 1979. http://portal.acm.org/citation.cfm?id=1241515.1241536&coll=DL&dl=GUIDE&CFID=33002831&CFTOKEN=51445937
25. CONWAY J., WATTS S. *Software Engineering with LabVIEW*. Pearson Education, 2003 URL: http://portal.acm.org/citation.cfm?id=861986&coll=DL&dl=GUIDE&CFID=33002831&CFTOKEN=51445937 http://zone.ni.com/devzone/cda/tut/p/id/7117
26. DIJKSTRA E. *Structured programming. Software Engineering Techniques*. NATO Science Commitee (edited by Burton J. And Randell B.). 1969. P. 89-93.
27. *DVM-система*. ИПМ РАН. URL: http://www.kiam.ru/dvm/dvmhtm1107/rus/index.html
28. HINCHEY M., JACKSON M., COUSOT P., COOK B., BOWEN J., MARGARIA T. Software Engineering and Formal Methods // Comm. of the ACM. Vol. 51. № 9. September 2008. P. 54-59
29. HOROVITZ E., KEMPER A., NARASIMHAN B. *A survey of application generators* // IEEE Software. 1985. Jan. P. 40-54.
30. *IBM System/360 Scientific Subroutine Package (360A-CM-03X) Version II Programmer Manual*. IBM Systems Reference Library, H20-0205 2. New York, 1967.
31. ILYIN V. D. *A Methodology for Knowledge Based Engineering of Parallel Program Systems* // 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 URL: http://portal.acm.org/citation.cfm?id=215624.215897&coll=DL&dl=GUIDE&CFID=33002831&CFTOKEN=51445937

32. LEE R., CHANG S. *Structured programming and automatic program synthesis* // Proc. of the ACM SIGPLAN symposium on Very high level languages, New York, April 1974, P. 60-70. URL: http://portal.acm.org/citation.cfm?id=807046&dl=ACM&coll=DL&CFID=33002831&CFTOKEN=51445937
33. LUKER P., BURNS A. *Program generators and generation software* // Comp. J. 1986. Vol. 29. № 4. P. 315-321.
34. MANNA Z., WALDINGER R. *Towards automatic program synthesis* // Commun. ACM. 1971. Vol. 14. № 3. P. 151-164.
35. MALYSHKIN V.E., PEREPELKIN V.A. *Optimization of Parallel Execution of Numerical Programs in LuNA Fragmented Programming System.* MTPP 2010 revised selected papers, Springer, LNCS 6083 (2010), P. 1-10.
36. WIRTH N. Program development by stepwise refinement // Comm. of the ACM, Vol. 14, № 4, 1971, P. 221-227.

THE FORMALIZATION OF KNOWLEDGE ABOUT TASKS: TASK CONSTRUCTIVE OBJECTS

Alexander V. Ilyin, Cand. Sc. (avilyin@ipiran.ru),
Vladimir D. Ilyin, Dr. Sc., professor (vdilyin@ipiran.ru),
Institute of Informatics Problems of RAS, Moscow

Abstract: The article adresses representation of knowledge about programmable tasks. It considers the methodology of constructing symbol models of tasks intended for usage in development of systems of knowledge about tasks. The methodology is based on results achieved in development and research of systems for computer-aided software design. The TSM-complex for description of symbol models of arbitrary objects in human-machine environment (s-environment) is used for formalized representation of tasks. Tasks are repres-

ented as constructive objects (s-tasks) having certain types of relations and rules of design. S-task specification includes formulation, systems of mandatory and adjusting rules for solving, algorithms, sets of programs and test cases. Descriptions of application for programs include data on the operating systems, hardware requirements etc.

Keywords: s-modeling, task constructive object, s-task, s-task graph, s-task calculus, knowledge system about tasks, TSM-complex.