Формализация задач в условиях дедуктивного синтеза

А.П. Бельтюков
Институт математики, информационных технологий и физики
Удмуртский государственный университет
Ижевск, Россия
e-mail: belt.udsu@mail.ru

Аннотация[1]

Описывается построение и трансформации формулировок конструктивных задач в парадигме дедуктивного синтеза их решений. Предлагается подход к построению формализации конструктивной задачи, когда уже примерно известны и неформальная постановка задачи, и неформальная идея ее решения. В этой ситуации из неформальной постановки задачи и неформальной идеи ее решения вырабатывается система понятий, положенная в основу онтологии: типы объектов в наиболее общем для данной задачи виде, конструктивные свойства-атрибуты, конструктивные отношения и спецификации исходных и целевых функций и операторов.

1. Введение

Под дедуктивным синтезом в настоящей работе понимается логический вывод решений конструктивных задач из их формальных постановок в определенных системах вывода. Под решениями конструктивных задач здесь понимается построение алгоритмов, структур данных и объектов, содержащих как данные, так и алгоритмы. Более того, задача может быть расширена так, что частями решений могут оказаться как физические объекты, так и компетенции людей. Такие решения можно назвать физико-антропно-техническими. При этом сама система построения таких решений может оказаться физико-антропно-технической, на что мы и рассчитываем в настоящей работе.

Исторически первыми идеями дедуктивного синтеза программ были идеи извлечения алгоритмов из конструктивных доказательств, конструктивно понимаемых логических, логико-математических или иных логико-предметных формул (см. работы С.К. Клини [1], Н.А. Шанина [2], А.А. Маркова[3]). Однако в настоящей работе мы не рассматриваем традиционные логические языки ввиду того, что они оперируют со слишком мелкими действиями, не с теми, к которым привык человек, работающий в области информатики. Это важно, потому что в физико-антропно-технических системах человек является ведущим звеном. Удобство работы человека – ключевой фактор таких систем. Впервые автором такой подход был рассмотрен в работе [4].

Вместо традиционных языков логических формул мы рассматриваем языки спецификации задач, более подходящие для описания предметной области в информатике. Роль языков доказательств у нас здесь играют специальные языки дедуктивного программирования. Наиболее близкое к ним понятие в логике – доказательство как терм – выражение, которое, вообще говоря, можно вычислять (выполнять).

2. Предшествующие исследования

При исследовании сложностных ограничений на синтезируемые алгоритмы в работах [5] и [6] были найдены ограничения на структуру формул, выражающих постановки задач, при которых эффективность получаемых алгоритмов была достаточно высокой. Настоящая работа является продолжением работ автора и С.Г.Маслова [7] и [8].

3. Основная идея

Мы предполагаем, что постановке задачи предшествует изучение предметной области и выявление проблем, связанных с ней. Предполагаем также, что для формулирования задач решения выявленных проблем строится определенная терминологическая система.

Для успешного использования имеющимся традиционным аппаратом конструктивного дедуктивного синтеза терминологическая система должна иметь определенную структуру конструктивной онтологии.

Она состоит в том, что все термины разбиваются на три класса:

- термины, обозначающие исходные типы значений, все исходные значения должны быть конструктивными объектами, то есть однозначно задаваться конечными цепочками символов из заранее выбранного конечного алфавита, с каждым таким термином связано содержательное

Труды Шестой всероссийской научной конференции "Информационные технологии интеллектуальной поддержки принятия решений", 28-31 мая, Уфа-Ставрополь, Россия, 2018

описание на естественном для человека языке и процедуры передачи соответствующих значений между исполнителями строящихся решений задач;

- термины, обозначающие конструктивные предикаты: конструктивные свойства и конструктивные отношения, конструктивность здесь означает, что свойства и отношения должны иметь реализации –специальные значения, конструктивные объекты, подтверждающие соответствующие свойства и отношения, таким образом свойства в сущности являются атрибутами, для передачи этих значений должны быть заданы соответствующие процедуры, а для предикатов – соответствующие неформальные описания, кроме того, для каждого предиката следует зафиксировать количество и типы аргументов; можно считать, что предикат это процедура, вырабатывающая по заданным аргументам определенный тип значений, можно считать что с ложным значением предиката связан пустой тип; иногда при решении задач, связанных с обработкой ошибок, удобно считать, что тип ложного значения не пуст, но состоит исключительно из сообщений об ошибках;
- термины, обозначающие конструктивные функции (отображения, функционалы, операторы), функции могут быть как естественными объектами предметной области, так и искусственными объектами, строящимися при решении задачи, для каждой функции фиксируется число и типы аргументов и структура вырабатываемого значения (см. описание примера языка ниже), функция снабжается и неформальным описанием.

Требуемые спецификации исходных типов, предикатов, операторов записываются на специальных языках (см. пример ниже).

Формулировка конструктивной задачи как задачи построения конструктивных функций (функционалов, операторов) также записывается на таком языке и вдобавок снабжается неформальным описанием. Формальная постановка называется спецификацией конструктивной задачи.

Возможны и более сложные задачи, включающие построение не только новых операторов, но и новых типов и предикатов. Как правило, это задачи моделирования.

4. Технология

4.1. Учет предметных терминов

Предметная область, прежде всего, содержит определенные типы предметов.

Это означает, что с каждым выявленным типом предметов требуется связать некоторое имя.

Поскольку мы разрабатываем человеко-машинный подход, то необходимо указать для человека, работающего с этой системой, содержательный смысл этого типа предметов. Например,

Item:Type – детали и сборочные единицы в производственном процессе.

При этом запись x:Item означает, что *x* принадлежит типу Item, т. е. *x* – некоторая конкретная деталь или сборочная единица.

Вообще запись вида *A*:*B*, где сущность *A* относится к классу *B*, будем называть формой спецификации. Для понимания такой записи человеком в человеко-машинной системе желательно комментирование таких форм на языке, естественном для человека:

$$A{:}B, - C.$$

где *C* – комментарий на понятном для человека языке. Такая форма самодокументирования спецификаций используется как для передачи спецификации между людьми для дальнейшей обработки, так и для самодисциплины субъекта-разработчика.

4.2. Учет предикатных терминов

Прежде всего, заметим, что, в соответствии с конструктивно-реализационной парадигмой, конструктивные свойства предметов должны иметь подтверждения – реализации этих свойств. Эти подтверждения являются, в свою очередь, снова предметами. Например, свойство детали быть покупной должно подтверждаться информацией о ее покупке. Таким образом, конструктивное свойство на самом деле – некоторый атрибут. Это же касается и конструктивных отношений: они также должны подтверждаться. Например, утверждение о том, что один продукт может быть переработан в другой продукт, должно подтверждаться некоторым описанием технологии переработки. Такой подход удобен тем, что экономит число сущностей, описываемых при постановке задачи: каждое свойство и отношение скрывают в себе еще одну сущность, их реализацию, которая будет обрабатываться в решении задачи. Поэтому при построении онтологии действует общая рекомендация: как можно больше предметов объявить подтверждениями свойств и отношений. Это упрощает логику решения задач. Свойства и отношения будем называть далее по традиции предикатами.

Далее используем следующие обозначения:

- запись $P(x)$ означает, что предмет x обладает свойством P,
- запись $p{:}P(x)$ означает, что предмет p – подтверждение того, что предмет x обладает

свойством *P* (другими словами, *p* – одно из значений атрибута *P* предмета *x*),

- запись *Q*(*x*, *y*) означает, что предмет *x* находится с предметом *y* в отношении *Q*,
- запись *q*:*Q*(*x*, *y*) означает, что предмет *q* подтверждает то, что предмет *x* находится с предметом *y* в отношении *Q*, то же используем для трехместных отношений и для отношений большей вместимости.

Запись *P*:(*x*:*X*=>*t*:Type) означает, что *P* – одноместный предикат с аргументом типа *X*. Аналогично будем использовать записи

Q:(*x*:*X*, *y*:*Y* => *t*:Type),

где *X* и *Y* – типы аргументов *Q*,

R:(*x*:*X*, *y*:*Y*, *z*:*Z* => *t*:Type)

и т. д.

4.3. Учет конструктивных операторов (функций, функционалов)

При описании предметной области предполагается, что между ее элементами имеются функциональные связи. Это особые применяемые предметы. Ими могут быть, в частности, ранее созданные алгоритмы. Излагаемый здесь подход требует, чтобы свойства этих функциональных предметов могли быть описаны способами, описанными ниже.

Общий вид формы спецификации функционального предмета: *f*:(*A*=>*B*), где

A – цепочка форм спецификаций аргументов функции *f*,

B – цепочка вариантов спецификаций результата применения f к своим аргументам.

Здесь предполагается, что число аргументов функционального предмета фиксировано, а результат будет иметь один из заранее заданного конечного списка видов. Следовательно, необходимо привести описание предметной области к этому виду. Например, при наличии переменного числа аргументов нужно ввести понятие списка аргументов и т. п.

Формы спецификации в цепочке *A* будем разделять запятыми.

Например, запись *A* может выглядеть так:

n:*I*, *c*:*C*(*i*).

Это означает, что специфицируемый функциональный предмет применяется к упорядоченной паре предметов, первый из которых имеет тип *I*, а второй - подтверждение свойства *C* для первого предмета.

Варианты в *B*, если их несколько, будем разделять вертикальной чертой, каждый из этих вариантов, так же как и *A*, - цепочка форм спецификаций. Например, *B* здесь может иметь вид:

d:*S*(*i*)|*j*:*I*, *r*:*R*(*i*, *j*).

Это означает, что в первом варианте получается подтверждение свойства *S* для предмета *i*, а во втором варианте - пара, состоящая из еще одного предмета типа *I* и подтверждения того, что эти предметы находятся в отношении *R*. В итоге всю форму спецификации будем записывать как

f:(*n*:*I*, *c*:*C*(*i*) => *d*:*S*(*i*)|*j*:*I*, *r*:*R*(*i*,*j*)).

Будем также использовать и сокращенную запись, получающуюся по очевидным правилам:

f:(*n*:*I*, *C*(*i*) => *S*(*i*)|*j*:*I*, *R*(*i*,*j*)).

Заметим, что здесь все используемые имена должны быть введены локально, как имена в формах спецификаций или известны из контекста (конструктивной онтологии). Предопределенным считается имя Type. Все остальные имена требуется определять.

Возможны и вложенные спецификации функциональных объектов, например,

f:(*x*:*X*, *g*:(*y*:*X*, *r*:*P*(*y*) => *s*:*Q*(*y*)), *p*:*P*(*x*) => *q*:*Q*(*x*)).

Последнюю запись можно представить, как запись задачи построения программы для некоторого исполнителя, который по предметам *x*, *g* и *p* указанных типов строит предмет *q* указанного типа.

Вообще говоря, онтологию для решения задачи можно представить как посылку одной большой спецификации функционального предмета. В итоге постановка задачи в целом представляется как форма спецификации одного функционального предмета - решения поставленной задачи.

Приведем простой пример задачи.

Требуется построить функциональный предмет t1 со следующей спецификацией:

t1:(

Node:Type, – имеется тип "узлы связи".

Base:(n:Node=>t:Type), – имеется свойство узла связи "быть базовой станцией".

Link:(m:Node,n:Node=>t:Type)

– Имеется отношение между узлами связи "можно установить связь".

– Имеются четыре функциональных предмета для осуществления следующих действий.

bs:(x:Node=>y:Node,b:Base(y),l:Link(x,y)) – связать узел с базовой станцией.

bc:(x:Node, y:Node, Base(x), Base(y) => l:Link(x,y)) – связать базовые станции.

rc:(x:Node, y:Node, Link(x,y) => Link(y,x)) – обратить связь,

tr:(x:Node, y:Node, z:Node, Link(x,y), Link(y,z) => Link(x,z)) - соединение каналов,

m:N – дан узел.

n:N – дан еще один узел.

=>

Link(m,n) – требуется связать эти узлы.

)

4.4.Запись решений задач

Мы предлагаем для записи решений задач чисто аппликативный подход. Это значит, что запись любого алгоритма состоит только из записей применений одних предметов к другим и записей построений новых искусственных предметов (алгоритмов) для решения частных задач. При этом не будем придерживаться широко распространенного минималистского лямбда-языка, а из соображений удобства работы человека в человеко-машинной системе предлагаем использовать немного более богатые средства. Тем не менее, такие традиционные средства, как организация циклов и рекурсий, работа со сложными структурными объектами, будем считать опциональными. Это значит, что если нам нужны такие средства (например – организация цикла), то считаем соответствующие инструменты частью предметной области, описываемой онтологией. Вопросы логической корректности этих средств в настоящей работе не рассматриваются и являются предметом отдельных исследований.

Рассмотрим каноническую запись наших алгоритмов. Все другие варианты записи, вводимые для удобства, будем считать сокращениями этой канонической записи. В свою очередь, конечно, и саму каноническую запись можно перевести на лямбда-язык, но это сделает ее менее удобной для человека.

Любой алгоритм будем записывать состоящим из 4 частей:

1) цепочка имен локальных предметов, обозначающих следующее:

1.1) сначала идут соответственно аргументы алгоритма (входы),

1.2) затем идут соответственно варианты завершения алгоритма (выходы);

2) имя действия, которое необходимо выполнить первым;

3) в скобках через запятую – аргументы этого действия, каждый аргумент – либо имя предмета, либо в скобках снова запись алгоритма (для наглядности будем использовать здесь фигурные скобки), аргументом также может быть и спецификация в случае, если нужно передать тип значения;

4) если выполненное действие – не выход, то алгоритмы, продолжающие выполненное действие в каждом из вариантов его завершения, каждый из алгоритмов заключается в (фигурные) скобки, последний алгоритм в скобки можно не заключать; заметим, что имена выходов в этих алгоритмах не нужны, так как используются выходы объемлющего алгоритма; если выполненное действие – выход, то эта часть пустая: у выхода нет своих выходов и продолжения ему не требуются. Заметим, что обрабатываемые предметы могут быть как обычными предметами области, так и типами или предикатами. В последнем случае исполнитель (будь то автомат или человек) получает описание этого типа, которое, в частности, должно обязательно содержать правила работы с предметами этого типа (как их хранить, передавать, утилизировать) на языке, обрабатываемом этим исполнителем.

В соответствии с парадигмой дедуктивного синтеза любое решение задачи, согласованное с ней по типам данных, является в сущности доказательством постановки задачи, если ее понимать как конструктивное утверждение, т. е. логическую формулу.

В качестве примера проиллюстрируем часть описанных средств для записи алгоритма t1, задача построения которого была описана выше. Текст снабдим двумя видами комментариев. Одни комментарии начинаются с двоеточия, это формальные комментарии, указывающие тип предмета для только что определенного имени. Такой комментарий заканчивается запятой или точкой с запятой. Другие комментарии - неформальные они начинаются со знака "-" и заканчиваются точкой.

Итак, решение задачи *t*1 можно записать следующим образом:

t1={

Node:Type, - исполнитель получает правила работы с узлами.

Base:(n:Node=>Type), - исполнитель получает правила работы с информацией о базовых станциях.

Link:(m:Node,n:Node=>Type), - исполнитель получает правила работы с информацией о соединениях.

bs:(x:Node=>y:Node,b:Base(y),l:Link(x,y)), - искатель базы.

– Здесь предполагается, что приведенной информации достаточно, чтобы исполнителю корректно работать с функциональным предметом.

bc:(x:Node,y:Node,Base(x),Base(y)=>l:Link(x,y)) – связь баз.

rc:(x:Node,y:Node,Link(x,y)=>Link(y,x)) – обращение связи.

tr:(x:Node,y:Node,z:Node,Link(x,y),Link(y,z)=>Link(x,z)) – транзитивность связи.

m:N – первый узел.

n:N –второй узел.

return:(lmn:Link(m,n)) – выход для соединения этих узлов.

– Выходы специфицируются в скобках, чтобы отличаться от входов.

bs(m)p:Node,bp:Base(p),lmp:Link(m,p) – поиск первой базы.

bs(n)q:Node,bq:Base(q),lnq:Link(n,q) – поиск второй базы.

bc(p,q,bp,bq)lpq:Link(p,q) – соединение баз.

rc(n,q,lnq)lqn:Link(q,n) – обращение одной из связей.

tr(m,p,q,lmp,lpq)lmq:Link(m,q) – соединение каналов.

tr(m,q,n,lmq,lqn)lmn:Link(m,n) – соединение каналов.

return(lmn) – результат готов.

}

Одновременно это решение является доказательством утверждения, которым является спецификация задачи t1.

То, что названо формальными комментариями, хотя и восстанавливается однозначно, опускаться нами не будет. В сложных системах исполнения решений такой комментарий на самом деле описывает вполне конкретное действие - передачу описанного значения. Например, поиск первой базы можно развернуть следующим образом:

bs(m) – выполнение действия bs с передачей аргумента m.

p:Node – импорт первого результата с помощью процедуры Node.

bp:Base(p) – импорт второго результата с помощью процедуры Base с параметром p.

lmp:Link(m,p) – импорт третьего результата с помощью процедуры Link с параметрами *m* и *p*.

Возможна и более подробная запись:

bs(m:Node) – выполнение действия bs с передачей аргумента m.

p:Node – импорт первого результата с помощью процедуры Node.

bp:Base(p:Node) – импорт второго результата с помощью процедуры Base с параметром p.

lmp:Link(m:Node,p:Node) – импорт третьего результата с помощью процедуры Link с параметрами m и p.

В принципе можно использовать и еще более подробную запись. Для последней строки это будет

lmp:Link(m:Node:Type,p:Node:Type);

Здесь явно указывается, что сама процедура Node передается универсальной процедурой Type. Считается, что наша система должна изначально владеть процедурой Type и передавать ее не требуется. Заметим, что настолько подробная запись практически не нужна.

Другая строка программы:

bc(p,q,bp,bq)lpq:Link(p,q);

записывается так:

bc(p:Node,q:Node,bp:Base(p:Node),bq:Base(q:Node))lpq:Link(p:Node,q:Node);

Решение без комментариев выглядит следующим образом:

t1={Node,Base,Link,bs,bc,rc,tr,m,n,return;

bs(m)p,bp,lmp;

bs(n)q,bq,lnq;

bc(p,q,bp,bq)lpq;

rc(n,q,lnq)lqn;

tr(m,p,q,lmp,lpq)lmq;

tr(m,q,n,lmq,lqn)lmn;

return(lmn)

}

Возможна и совсем короткая запись, при которой имена, не используемые в качестве аргументов предикатов, опускаются:

t1={Node,Base,Link,bs,bc,rc,tr,m,n,return;

bs(m)p;bs(n)q;bc(p,q);rc(n,q);tr(m,p,q);tr(m,q,n);return()
}.

5. Примеры спецификаций функций

Приведем пример спецификации оператора цикла динамического программирования. С логической точки зрения этот цикл - рекурсия по некоторому строгому частичному порядку R на множестве I. Описание части онтологии со спецификацией этого оператора для заданных I и R с комментариями выглядит следующим образом:

I:Type, - тип переменной цикла.

R:(i:I,j:I=>t:Type), - отношение на значениях этого типа.

C:(B:(i:I=>t:Type)=>u:Type), - ограничение на задачи, которые могут решаться в цикле.

for: - имя оператора цикла как функционального предмета области.

(- Начало спецификации оператора цикла.

B:(i:I=>t:Type) – решаемая циклом задача.

c:C(B) – ограничение, накладываемое на задачу.

n:I – верхняя граница цикла.

do:(– начало спецификации тела цикла.

i:I – параметр цикла.

mem:(– начало спецификации памяти цикла.

j:I, – индекс памяти цикла.

rij:R(i,j) – ограничение памяти цикла.

=> bj:B(j) – содержимое памяти цикла.

) – конец спецификации памяти цикла.

=> bi:B(i) – цель шага цикла.

) – конец спецификации тела цикла.

=> bn:B(n) – цель всего цикла.

) – конец спецификации всего цикла.

Заметим, что приведенный выше оператор for может быть явно выведен из некоторых ограничений, на задачи, решаемые в цикле. Например, ограниченность длины цепочек, связанных отношением *R*, может быть выражена следующим ограничением:

C=

{

B:(i:I=>t:Type), - решаемая задача.

return:(Type);

– возвращается конструктивное логическое значение.

return

((w:I,x:I,y:I,z:I,rwx:R(w,x),rxy:R(x,y),ryz:R(y,z)=>bw:B(w)))

– если найдена слишком длинная цепочка в R, то считаем, что задача решена.

}

Здесь в качестве примера приведено ограничение 3 на длину цепочки предметов, связанных отношением R. Аналогично можно ввести любое другое ограничение.

Пример задачи, решаемой с помощью оператора for:

t2:(

I:Type – пусть I – обозначение для некоторых продуктов.

R:(I,I=>Type) – R(i,j) – технологии получения i из j.

C:((I=>Type)=>Type),

– ограничение на решаемые в цикле задачи.

B:(I=>Type) – B(i) – стоимость продукта *i*.

c:C(B) – ограничение на решаемую задачу

for:

(B:(I=>Type),c:C(B),n:I,(i:I,(j:I,R(i,j)=>B(j))=>B(i))=>B(n))

– спецификация оператора цикла.

db:(i:I=>B(i)|j:I,R(i,j)) – база данных, в которой записано, что каждый продукт – покупной с известной стоимостью или получается из другого продукта.

cn:(i,j:I,R(i,j),B(j)=>B(i)),

– процедура вычисления стоимости производимого продукта по стоимости предшественника и способу производства.

n:I – данный продукт.

=>B(n) – требуется найти стоимость данного продукта.

)

Пример решения этой задачи:

t2=

{I,R,C,B,c,for,db,cn,n,

– комментарии к этим именам приведены выше.

return:(B(n)) – спецификация оператора возврата из программы.

for – запускаем цикл.

(B – в цикле рашаем задачу B.

c – задача B удовлетворяет условию C.

n – цикл выполняем до достижения продукта n.

{ – начало тела цикла

i:I – параметр цикла

mem: – память цикла.

(j:I, – индекс памяти цикла.

R(i,j) – ограничение памяти цикла

=>B(j) – содержимое памяти цикла

),

continue:(B(j)) – оператор завершения шага цикла.

db(i) – рассматриваемый в цикле продукт ищем в базе данных.

{ – первый случай: продукт - покупной.

bi:B(i) – его известная стоимость.

continue(bi) – эту стоимость подаем на завершение шага цикла.

} – второй случай: продукт производится из другого продукта

j:I – продукт предшественник.

rij:R(i,j) – способ получения текущего из предшественника.

mem(j,rij) – ищем продукт-предшественник в памяти цикла.

bj:B(j) – найденная стоимость предшественника.

cn(i,j,rij,bj) – вычисляем стоимость текущего продукта.

bi:B(i) – вычисленная стоимость текущего продукта.

continue(bi) – эту стоимость подаем на завершение шага цикла.

} – конец тела цикла.

) – конец цикла.

bn:B(n) – результат работы цикла.

return(bn) – подаем результат на возврат из программы.

}

Решаемая в цикле задача может быть специфицирована сложной формулой. При этом эта сложная формула рассматривается как конструктивная логическая функция, которая подставляется на место предиката, определяющего задачу, решаемую в цикле.

6. Заключение

- В работе представлен понятийный аппарат, предназначенный для использования в человеко-машинных и физико-антропно-технических системах.
- Предложенные понятия позволяют формулировать и решать конструктивные задачи с помощью таких систем и для таких систем.
- Понятия образуют иерархию типов и опираются на небольшие универсалии, включающий понятие передачи типа значений и базовые языки постановок и решений.
- Предложенный аппарат предназначается для описания конструктивных физико-антропно-технических систем.

Список используемых источников

1. Kleene S.C. Introduction to metamathematics – North-Holland, 1951, 500 p.
2. Шанин Н. А. Об иерархии способов понимания суждений в конструктивной математике // «Тр. Матем. ин-та АН СССР», 1973, т. 129, с. 203-266.
3. Марков А. А. Попытка построения логики конструктивной математики // В сб.: Исследования по теории алгорифмов и математической логике, М., 1976, т. 2, С.3-31.
4. Бельтюков А.П. Малые сложностные классы и автоматический дедуктивный синтез алгоритмов // Известия института математики и информатики УдГУ, Ижевск, 1995, вып. 2, С.3-89.
5. Beltiukov A.P. Intuitionistic formal theories with realizability in subrecursive classes // Annals of Pure and Applied Logic, 89, 1997, P. 3-15.
6. Beltiukov A.P. A strong induction scheme that leads to polynomially computable realizations // Theoretical Computer Science, 322 (2004), P. 17-39.
7. Beltiukov A.P., Maslov S.G. Organizing understanding in the framework of ergatic system // CSIT’2015: Proceeding of the 17th International Workshop on Computer Science and Information Technologies, Rome, Italy, September 22-26, 2015, V.1, – Ufa.: Ufa State Aviation Technical Univercity, 2015, P. 60-64.
8. Бельтюков А.П., Маслов С.Г. Логико-дедуктивный синтез программно-информационных объектов в системах принятия решений // ITIDS'2016: Proceedings of the 4th International Conference on Information Technologies for Intelligent Decision Making Support - Ufa, Russia, May 17-19, 2016, V.1 - Ufa: Ufa State Aviation Technical University, 2016, – P. 25-30.