OSTIS-2014
(Open Semantic Technologies for Intelligent Systems)

УДК 004.89

МАТРИЧНОЕ ПРЕДСТАВЛЕНИЕ КОНЕЧНЫХ ПРЕДИКАТОВ ДЛЯ АВТОМАТИЗАЦИИ ЛОГИКО-СЕМАНТИЧЕСКОГО АНАЛИЗА

Зуенко А.А.

Институт информатики и математического моделирования КНЦ РАН, Апатиты, Россия
zuenko@iimm.kolasc.net.ru

В статье рассматривается программная система, основанная на матричном представлении конечных предикатов и теоретико-множественной интерпретации логических операций, принятой в алгебре кортежей. Система реализует оригинальные алгебраические методы решения таких задач логико-семантического анализа, как проверка правильности следования; вывод абдуктивных заключений; анализ семантических некорректностей. Также система может быть полезна при решении задач удовлетворения ограничений, которые сводятся к поиску выполняющих подстановок конъюнктивной нормальной формы конечного предиката.
Ключевые слова: алгебра кортежей, логико-семантический анализ, матричное представление конечных предикатов.

Введение

Данная работа продолжает цикл публикаций, посвященный методам алгебры кортежей (АК).

Ранее в [Кулик и др., 2010], [Зуенко и др., 2011], [Зуенко и др., 2013] рассматривались методы АК, предназначенные для решения различных задач логико-семантического анализа таких, как: проверка правильности логического следования; получение возможных следствий из системы посылок при заданных ограничениях; вывод абдуктивных заключений; анализ семантических некорректностей. Особенностью задач, решаемых средствами АК, является то, что они описываются на языке конечных предикатов.

В АК используется матричное представление конечных предикатов и теоретико-множественная интерпретация логических операций с ними. Близкий подход применяется также в [Zakrevskij, 2012] для решения задач распознавания образов.

В АК конечные предикаты можно сжато представить с помощью двух типов структур: *C*-систем и *D*-систем. Они формируются в виде матриц из подмножеств доменов атрибутов, называемых компонентами. В их число входят две фиктивные компоненты: полная компонента (обозначается «*») – это множество, равное домену соответствующего (по месту ее расположения в кортеже) атрибута; пустое множество – ∅. В виде *D*-систем удобно представлять конъюнктивные нормальные формы (КНФ) конечных предикатов, а в виде *C*-систем – их дизъюнктивные нормальные формы (ДНФ). Так, например, в АК процесс поиска всех выполняющих подстановок некоторой КНФ сводится к преобразованию *D*-системы в *C*-систему.

В [Зуенко, 2013] показаны методы АК решения задач удовлетворения ограничений (constraint satisfaction problem, CSP). CSP характеризуются тем, что в них используются переменные, которые имеют конечные области определения. К числу таких задач относится задача раскраски карты (раскраски графа). Также к категории задач с конечными областями относятся булевы задачи CSP. Булевы задачи включают в качестве частных случаев некоторые *NP*-полные задачи, такие как 3*SAT* [Рассел и др., 2006]. Решение CSP в этом случае сводится к определению выполняющих подстановок (единственной подстановки) конъюнктивной нормальной формы конечного предиката.

Методы ускорения вычислительных процедур, основанные на матричном представлении конечных предикатов, позволяют на практике получать требуемые решения упомянутых задач за приемлемое время. Дополнительные возможности ускорения связаны с анализом семантических ограничений предметной области [Кулик и др., 2010].

Рассмотрим некоторые особенности программной реализации разработанной среды для автоматизации вычислений с АК-объектами.

1. Представление логических матриц и операций над ними с помощью булевых векторов

В основе всех операций с АК-объектами лежат операции с компонентами, из которых АК-объекты состоят. На уровне кода программы, на языке С++, компоненты представляются с помощью булевых векторов: каждая компонента моделируется последовательностью бит, чья размерность равна размерности соответствующего домена. Биту соответствует значение, принадлежащее определенному домену. Если некоторый бит установлен в “true”, считается, что соответствующий ему элемент домена присутствует в компоненте (см. рисунок 1). Здесь и далее в рисунках, иллюстрирующих операции, будем слева показывать булевы векторы, а справа – соответствующие им компоненты АК-объектов или АК-объекты целиком.

Рисунок.1. Пример представления компоненты на основе булева вектора

Конечный пользователь формирует компоненты логических матриц привычным для АК способом – в виде множеств, а уже благодаря внутрипрограммному представлению, операции с компонентами сводятся к побитовым операциям с булевыми векторами.

Проиллюстрируем операцию пересечения компонент (см. рисунок 2):

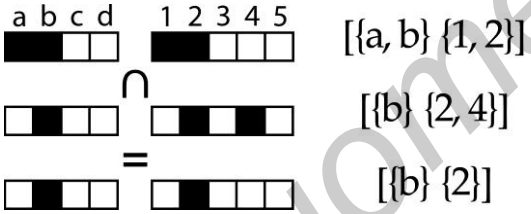

Рисунок 2. Пример операции пересечения компонент

Также можно продемонстрировать операцию объединения компонент (см. рисунок 3):

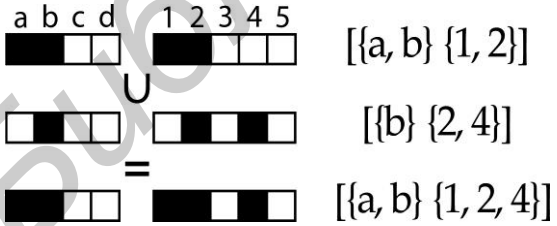

Рисунок 3. Пример операции объединения компонент

Рассмотрим операцию дополнения компоненты (см. рисунок 4):

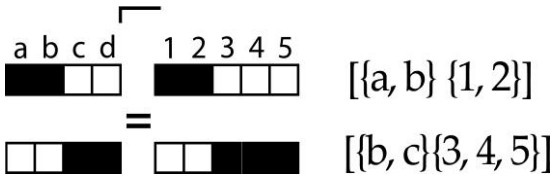

Рисунок 4. Пример операции дополнения компонент

Далее перейдем к описанию системы команд, с помощью которой конечный пользователь формирует алгоритмы (АК-программы) в рассматриваемой среде автоматизации преобразований логических матриц.

2. Описание системы команд программной среды

Система команд построена на присваиваниях, в результате которых создается новая переменная, и ей приписывается некое значение. Команда в общем виде выглядит следующим образом:

<NEWNAME> = <DATA>

Здесь <NEWNAME> – любое алфавитно-цифровое имя новой переменной, не совпадающее ни с одним из зарезервированных или уже использованных слов в документе, а <DATA> – любое из выражений <DOM>, <ATT>, <SCH>, <EX>.

Объявление домена с именем (<DOMNAME>) и типом (<DOMTYPE>), а также с указанием его содержимого производится в следующем виде:

<DOMNAME>=domain(<DOMTYPE>;[<EL>{,<EL>}])

Здесь тип домена <DOMTYPE> может принимать одно из следующих значений: «string», «int», «float», при этом домен будет иметь соответственно строковый, целочисленный или десятичный с плавающей запятой тип. Элементы домена <EL> указываются через запятую и должны соответствовать указанному типу. В числах с плавающей запятой в качестве разделителя используется точка. Приведем следующие примеры объявления домена:

Dom_0 = domain (string; *mon*, *tue*, *wed*, *thu*, *fri*, *sat*, *sun*);

Dom_1 = domain (int; 1, 2, 3, 4, 5, 6, 7, 8, 9);

Dom_2 = domain (float; 3.1415, -897.6,0.14, -15.1345).

Объявление атрибута с именем <ATTNAME>, и указанием его домена выглядит следующим образом:

<ATTNAME>=attribute(<DOM>).

Примером объявления атрибута может служить следующая запись:

A_1 = attribute (Dom_1).

Объявление схемы <SCH> может быть как указанием на уже существующую схему по ее имени (<SCHNAME>), так и объявлением новой схемы с указанием входящих в нее атрибутов:

<SCHNAME>=scheme([<ATT>{,<ATT>}]).

Пример объявления схемы:

S_1 = scheme (A_1, B_1).

Выражение <EX> может быть как объявлением АК-объекта (<SYS>), так и объявлением массива АК-объектов (<ARR>). АК-объект <SYS> описывается путем поэлементного указания его компонент (<SYSDEF>) или путем объявлением операции <OP>, результатом вычисления которой этот АК-объект является.

Аналогично, объявление массива АК-объектов <ARR> может быть либо конкретным массивом (<ARRDEF>), либо объявлением операции <ARROP>, результатом вычисления которой служит массив АК-объектов.

Объявление конкретной системы (<SYSDEF>) может быть указанием на уже существующий АК-объект (<SYSNAME>) или объявлением новой системы с указанием ее вида <SYSTYPE>, который принимает значения «c-system» или «d-system». В качестве примера объявления <SYSDEF> приведем следующие команды:

sys_1 = c-system (S_1; [{3, 4, 5}, {a, b}], [{2}, {b}]);

sys_2 = d-system (S_1; [#, {b, c, d}], [{8}, {c, d}], [{7, 8, 9}, {d}]).

На языке АК эти структуры выглядят следующим образом:

$$Sys_1[A_1B_1]=\begin{bmatrix} \{3,4,5\} & \{a,b\} \\ \{2\} & \{b\} \end{bmatrix};$$

$$Sys_2[A_1B_1]=\begin{bmatrix} \varnothing & \{b,c,d\} \\ \{8\} & \{c,d\} \\ \{7,8,9\} & \{d\} \end{bmatrix}.$$

Формат задания массива АК-объектов следующий:

ARRDEF = array([<SYS>{,<SYS>}]).

Все входящие в массив АК-объекты должны принадлежать к одному типу (либо *C*, либо *D*). Приведем пример объявления массива:

arr = array (sys_1, sys_2, sys_3).

Далее опишем основные операции <OP> над *C*- и *D*-структурами, поддерживаемые рассматриваемой системой автоматизации матричных символьных преобразований:

~<SYS> – преобразование АК-объекта в альтернативный класс;

!<SYS> – дополнение;

<SYS> * <SYS> – пересечение;

<SYS> + <SYS> – объединение;

elimination(<SYS>;[<ATT>{,<ATT>}]) – элиминация атрибутов, где через запятую в любом порядке указываются все атрибуты, подлежащие элиминации;

projection(<SYS>;[<ATT>{,<ATT>}]) – проекция, где через запятую в любом порядке указываются все атрибуты, которые должны остаться в результирующей системе.

Аналогичным образом определены операции преобразования в альтернативный класс, дополнения, пересечения, объединения, элиминации атрибутов, проекции для массивов АК-объектов:

<ARR>‘[„<INDEX>‘]‘‘– индексация массива;

~<ARR> – преобразование в альтернативный класс массива АК-объектов;

!<ARR> – вычисление дополнения для массива АК-объектов;

<ARR>*<SYS> – пересечение каждого элемента массива АК-объектов <ARR> с АК-объектом <SYS>;

<ARR>+<SYS> – объединение каждого элемента массива АК-объектов <ARR> с АК-объектом <SYS>;

elimination‘(„<ARR>;[[<ATT>{,<ATT>}]‘)‘‘,

projection‘(„<ARR>;[[<ATT>{,<ATT>}]‘)‘‘ – соответственно операции элиминации и взятия проекции для каждого элемента массива в отдельности. Помимо этого, определена операция нахождения всех возможных проекций:

all_projections‘(„full|not_full|all;<SYS>‘)‘‘.

В зависимости от заданного параметра (full, not_full, all) результатом операции будут соответственно полные, неполные, или все возможные проекции системы (массива).

Далее показаны примеры использования представленных ранее команд.

3. Примеры использования командного процессора

Пример 1. Проверить правильность логического вывода для следующего рассуждения: "Если Джонс не встречал этой ночью Смита, то либо Смит был убийцей, либо Джонс лжет. Если Смит не был убийцей, то Джонс не встречал Смита этой ночью, и убийство имело место после полуночи. Если убийство имело место после полуночи, то либо Смит был убийцей, либо Джонс лжет. Следовательно, Смит был убийцей".

Сначала выразим данное рассуждение на языке исчисления высказываний [Kulik et al., 2010]. Введем обозначения:

A – Джонс встречал этой ночью Смита;

B – Смит был убийцей;

C – Джонс лжет;

D – убийство имело место после полуночи.

Тогда заданное рассуждение можно сформулировать так:

первое предложение: $\neg A \supset (B \oplus C)$;

второе предложение: $\neg B \supset (\neg A \wedge D)$;

третье предложение: $D \supset (B \oplus C)$;

следствие: B.

Преобразуем первые три предложения для того, чтобы избавиться в них от импликации и строгой дизъюнкции. Получаем:

1) $\neg A \supset (B \oplus C) = A \vee (B \wedge \neg C) \vee (\neg B \wedge C)$;

2) $\neg B \supset (\neg A \wedge D) = B \vee (\neg A \wedge D)$;

3) $D \supset (B \oplus C) = \neg D \vee (B \wedge \neg C) \vee (\neg B \wedge C)$.

Для представления этих формул АК-объектами используем универсум $X_1 \times X_2 \times X_3 \times X_4 = \{0, 1\}$, где $A = B = C = D = 1$ и $\neg A = \neg B = \neg C = \neg D = 0$. Тогда посылки, которые являются ДНФ, выражаются C-системами:

$$P_1 = \begin{bmatrix} \{1\} & * & * & * \\ * & \{1\} & \{0\} & * \\ * & \{0\} & \{1\} & * \end{bmatrix};$$

$$P_2 = \begin{bmatrix} * & \{1\} & * & * \\ \{0\} & * & * & \{1\} \end{bmatrix};$$

$$P_3 = \begin{bmatrix} * & * & * & \{0\} \\ * & \{1\} & \{0\} & * \\ * & \{0\} & \{1\} & * \end{bmatrix},$$

а следствие – C-кортежем $B[X_2] = [\{1\}]$.

Чтобы создать домен, пользователь вводит команду:

dom = domain (int; 0, 1).

Также необходимо определить атрибуты:

x_1 = attribute (*dom*);

x_2 = attribute (*dom*);

x_3 = attribute (*dom*);

x_4 = attribute (*dom*).

Далее задаются необходимые схемы:

sch_1 = scheme (x_1, x_2, x_3, x_4);

sch_2 = scheme (x_2).

Для создания АК-объектов, моделирующих посылки и заключение требуются команды:

p_1 = c-system (sch_1; [{1}, *, *, *], [*, {1}, {0}, *], [*, {0}, {1}, *]);

p_2 = c-system (sch_1; [*, {1}, *, *], [{0}, *, *, {1}]);

p_3 = c-system (sch_1; [*, *, *, {0}], [*, {1}, {0}, *], [*, {0}, {1}, *]);

b = c-system (sch_2; [{1}]).

Если вычислить АК-объект, соответствующий выражению $P_1 \cap_G P_2 \cap_G P_3$, получим:

$P[X_1X_2X_3X_4] = P_1 \cap_G P_2 \cap_G P_3 =$

$$\begin{bmatrix} \{1\} & \{1\} & * & \{0\} \\ * & \{1\} & \{0\} & * \\ \{0\} & \{0\} & \{1\} & \{1\} \end{bmatrix}.$$

Для этого введем соответствующую команду:

p = compression(1; p_1 * p_2 * p_3).

Здесь операция compression используется для устранения повторяющихся кортежей.

Решается задача путем определения пустоты АК-объекта $P_1 \cap_G P_2 \cap_G P_3 \cap_G \overline{B[X_2]}$.

Можно использовать уже полученный ранее промежуточный результат $P[X_1X_2X_3X_4]$, а можно ввести команду целиком, поскольку рассматриваемая система осуществляет разбор сложных выражений:

check = p_1 * p_2 * p_3 * ~!b.

Проверка показывает, что вывод неверный. Результат работы системы с программой вычислений приведен на рисунке 5. Действительно:

$$\begin{bmatrix} \{1\} & \{1\} & * & \{0\} \\ * & \{1\} & \{0\} & * \\ \{0\} & \{0\} & \{1\} & \{1\} \end{bmatrix} \cap_G [* \ \{0\} * *] =$$

$[\{0\} \ \{0\} \ \{1\} \ \{1\}] \neq \varnothing$.

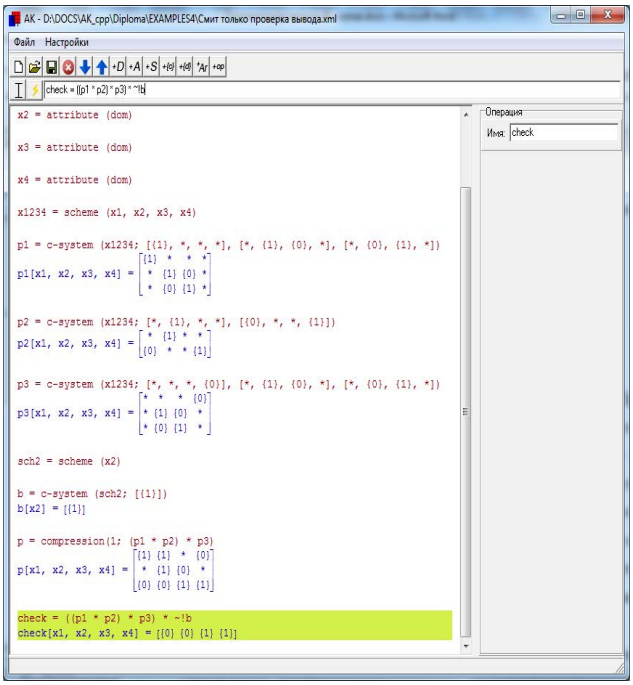

Рисунок.5. Решение задачи проверки правильности логического вывода

Пример 2. В предыдущем примере предполагаемое следствие (Смит был убийцей) не выводимо. Чтобы подтвердить или опровергнуть правильность вывода, требуется уточнить некоторые обстоятельства. Поиск таких обстоятельств можно сформулировать как задачу

поиска абдуктивного заключения [Kulik et al., 2012], [Вагин и др., 2008].

Предположим, что вывод правильный. Тогда необходимо найти подходящие гипотезы, которые можно использовать в качестве посылок. С помощью редактора можно сохранить программу вычислений предыдущего примера в файле с новым именем, удалить ненужные ветки вычислений и добавить новые. Отправной точкой будет следующий промежуточный результат:

$$P[X_1X_2X_3X_4] = \begin{bmatrix} \{1\} & \{1\} & * & \{0\} \\ * & \{1\} & \{0\} & * \\ \{0\} & \{0\} & \{1\} & \{1\} \end{bmatrix};$$

$B[X_1X_2X_3X_4] = [* \ \{1\} \ * \ *]$.

Далее используем алгоритм получения абдуктивных заключений, разработанный в АК [Kulik et al., 2012].

$$R = A \setminus_G B = \begin{bmatrix} \{1\} & \{1\} & * & \{0\} \\ * & \{1\} & \{0\} & * \\ \{0\} & \{0\} & \{1\} & \{1\} \end{bmatrix} \cap_G [* \ \{0\} \ * \ *]$$

$= [\{0\} \ \{0\} \ \{1\} \ \{1\}]$.

Для вычисления этого выражения введем команду:

$R = p * \sim!b$.

Здесь можно выбрать в качестве R_i любую проекцию R. Пусть это будет $R[X_4]$:

$RX_4 = \text{projection}(R; x_4)$.

Для вычисления $H_i = \overline{R_i}$ введем следующую команду:

$HX_4 = \sim!RX_4$.

Поскольку коллизии нам не заданы, проверим, вырождается ли общая предпосылка P при полученной гипотезе:

$$P \cap_G H_i = \begin{bmatrix} \{1\} & \{1\} & * & \{0\} \\ * & \{1\} & \{0\} & * \\ \{0\} & \{0\} & \{1\} & \{1\} \end{bmatrix} \cap_G$$

$$[* \ * \ * \ \{0\}] = \begin{bmatrix} \{1\} & \{1\} & * & \{0\} \\ * & \{1\} & \{0\} & \{0\} \end{bmatrix}.$$

Введем соответствующую команду:

$PH = p * HX_4$.

Проверка подтверждает корректность гипотезы. На естественном языке данная гипотеза означает, что убийство произошло до полуночи. Отсюда следует, что вывод будет правильным, если уточнить время убийства.

Редактор поддерживает разбор сложных выражений, то есть все операции можно записать в одну строку и получить тот же результат:

$res = \text{compression}(1; p_1 * p_2 * p_3 * \sim!\text{projection}(p_1 * p_2 * p_3 * \sim!b; x_4))$.

Заключение

Матричное представление ДНФ и КНФ конечных предикатов, основанные на этом представлении методы решения задач логико-семантического анализа и методы ускорения вычислительных процедур позволяют организовывать вычисления с нечисловыми переменными подобно тому, как реализуются численные методы алгебры матриц в специализированных программных средах.

Разработанная система, позволяет решать следующие задачи логико-семантического анализа: проверка правильности логического следования; получение следствий, удовлетворяющих заданным условиям; получение абдуктивных заключений, анализ семантических некорректностей и т.п. Также система может быть полезна при решении задач удовлетворения ограничений, которые сводятся к поиску выполняющих подстановок КНФ конечного предиката.

Система команд описанного программного процессора реализована на основе операций с булевыми векторами с использованием библиотеки boost::dynamic_bitset. Разбор и вычисление сложных регулярных выражений осуществляется с использованием библиотеки boost::regex. Предусмотрено сохранение программы в формате XML.

Работа выполнена при поддержке РФФИ (проекты № 12-07-006689-а, № 12-07-000550-а, № 13-07-00318-а), Президиума РАН (проект 4.3 Программы № 15), ОНИТ РАН (проект 2.3 в рамках текущей Программы фундаментальных научных исследований).

Библиографический список

[Вагин и др., 2008] Вагин, В.Н. Достоверный и правдоподобный вывод в интеллектуальных системах / Под ред. В.Н. Вагина, Д.А. Поспелова. – 2-е изд. испр. и доп. / В.Н. Вагин,Е.Ю. Головина, А.А. Загорянская, М.В. Фомина // – М.: ФИЗМАТЛИТ, 2008. – 712 с.

[Зуенко и др., 2011] Зуенко, А.А. Интеграция баз данных и знаний интеллектуальных систем на основе алгебраического подхода / А.А. Зуенко, Б.А. Кулик, А.Я. Фридман // Открытые семантические технологии проектирования интеллектуальных систем = Open Semantic Technologies for Intelligent Systems (OSTIS-2011): материалы Междунар. научн.-техн. конф. (Минск, 10-12 февраля 2011 г.) – Минск: БГУИР, 2011. – С.59-70.

[Зуенко, 2013] Зуенко, А.А. Матрицеподобные вычисления в задачах удовлетворения ограничений /А.А. Зуенко // Шестая Всероссийская мультиконференция по проблемам управления (30 сентября – 5 октября 2013 г.). Материалы мультиконференции: в 4 т. – Ростов-на-Дону: Издательство Южного федерального университета, 2013. Т.1. – С. 30-34.

[Зуенко и др., 2013] Зуенко, А.А. Интеллектуальные обучающие системы на основе алгебраического представления вопросно-ответных текстов / А.А. Зуенко, Б.А. Кулик, А.Я. Фридман // Открытые семантические технологии проектирования интеллектуальных систем = Open Semantic Technologies for Intelligent Systems (OSTIS-2013): материалы III

Междунар. научн.-техн. конф. (Минск, 21-23 февраля 2013 г.) – Минск: БГУИР, 2013. – С.165-170.

[Кулик и др., 2010] Кулик, Б.А. Алгебраический подход к интеллектуальной обработке данных и знаний / Б.А. Кулик, А.А. Зуенко, А.Я. Фридман // – СПб.: Изд-во Политехнического ун-та, 2010. 235 с.

[Рассел и др., 2006] Рассел, С. Искусственный интеллект: современный подход. 2-е изд. / пер. с англ.; ред. К. А. Птицына./ , С. Рассел, П. Норвиг // – М.: Изд. дом «Вильямс», 2006. 1408 с.

[Kulik et al., 2010] Kulik, B. Logical Analysis of Intelligence Systems by Algebraic Method / Boris Kulik, Alexander Fridman, Alexander Zuenko // Cybernetics and Systems 2010: Proceedings of Twentieth European Meeting on Cybernetics and Systems Research (EMCSR 2010), Vienna, Austria, 2010. – pp. 198-203.

[Kulik et al., 2012] Boris Kulik, Alexander Fridman, Alexander Zuenko. Logical Inference and Defeasible Reasoning in N-tuple Algebra. In: "Diagnostic Test Approaches to Machine Learning and Commonsense Reasoning Systems", IGI Global, P 102-128.

[Zakrevskij, 2012] Arkadij Zakrevskij. Integrated Model of Inductive-Deductive Inference Based on Finite Predicates and Implicative Regularities. In: "Diagnostic Test Approaches to Machine Learning and Commonsense Reasoning Systems", IGI Global, P 1-12.

MATRIX REPRESENTATION OF A FINITE PREDICATES FOR AUTOMATION OF LOGICAL AND SEMANTIC ANALYSIS

Zuenko A.A

Institute for Informatics and Mathematical Modelling, Kola Science Centre of RAS

zuenko@iimm.kolasc.net.ru

The article discusses a program system based on a matrix representation of finite predicates and set-theoretic interpretation of logic operations used in the n-tuple algebra (NTA). The system realizes the original algebraic methods of solving such problems of semantic and logical analysis as: correctness check for a consequence; derivation of consequences from axioms; derivation of abductive conclusions; analysis of semantic inconsistencies. Also, the system can be useful for solving constraint satisfaction problems which are reduced to finding substitutions of conjunctive normal form of the finite predicate.

Introduction

The article continues a series of publications devoted to the NTA methods.

A matrix representation of finite predicates and the set-theoretic interpretation of logical operations are used in NTA.

The NTA methods developed to solve various problems of logical and semantic analysis were presented in our publications earlier. The distinguishing feature of interesting us problems is that they could be described in terms of finite predicates.

Also NTA methods were used to solve constraint satisfaction problems (constraint satisfaction problem, CSP) with finite domains of variables. Such problems include graph coloring problem, Boolean problems CSP and etc.

In this case solving of CSP reduced to the definition of satisfying substitutions (single substitution) conjunctive normal form of the finite predicate.

Methods for accelerating of computational procedures based on the matrix representation of finite predicates allow to practically obtain the desired solutions of the problems in a reasonable time. Additional acceleration capabilities associated with semantic constraints analysis of subject domain.

Main Part

The article presents a tools for development of NTA-algorithms. It allows to automate various procedures of logical-semantic analysis. The features of the software implementation developed environment for automation of calculations with NTA-objects are presented.

The basis of all operations with NTA-objects is the operations with components of NTA-objects. On the program level components are represented by Boolean vectors: each component is a sequence of bit, whose dimension is equal to the dimension of the corresponding domain. If a bit is set on "true" the corresponding element of the domain is present in certain component.

The user determines components of logic matrix in the form of sets as it is taken in the NTA. Through the matrix presentation operations with components could be reduced to bit-wise operations with Boolean vectors on the program level.

The article presents a system of commands of considered command processor, and also examples of using this system for solving some problems of logical analysis.

Conclusion

The matrix representation of DNF and CNF of finite predicates, logical-semantic analysis methods that use this representation, methods for accelerating computational procedures allow to organize computations with non-numeric variables similar to computations based on the methods of matrix algebra in specialized software.

The authors would like to thank the Russian Foundation for Basic Research (grants 12-07-00550, 12-07-00689, 13-07-00318), the Department for Nanotechnologies and Information Technologies of RAS (project 2.11 of the current Programme of Basic Scientific Researches), and the Chair of RAS (project 4.3 of the Program # 16) for their help in partial funding of this research.