УДК 004.825

О.А. Мажара, аспирант
Национальный технический университет Украины
«Киевский политехнический ин-т»
(Украина, 03056 Киев, пр. Победы, 37,
тел. (096) 6315931, e-mail: olyamazhara@gmail.com)

Реализация Treat алгоритма на основе сопоставления с образцом в программной оболочке CLIPS

Предложена реализация логического вывода по прикладной базе знаний на основании Rete и Treat алгоритмов сопоставления с образцом для определения оптимального из них по ресурсоемкости и быстродействию. Приведено описание реализации Treat алгоритма для программной оболочки CLIPS, позволяющее сохранить существующие структуры данных и методы оптимизации логического вывода вследствие хеширования, представления сети предкомпиляции и реорганизации базы знаний. Предложенный подход позволяет в дальнейшем расширить программную среду CLIPS дополнительными инкрементными алгоритмами сопоставления.

Запропоновано реалізацію логічного виведення за прикладною базою знань на основі Rete і Treat алгоритмів співставлення зі зразком для визначення оптимального з них за ресурсоємністю та швидкодією. Надано опис реалізації Treat алгоритму для програмної оболонки CLIPS, який дозволяє зберегти існуючі структури даних та методи оптимізації логічного виведення внаслідок хешування, представлення мережі прекомпіляції та реорганізації бази знань. Запропонований підхід дозволяє надалі розширити програмне середовище CLIPS додатковими інкрементними алгоритмами співставлення.

К л ю ч е в ы е с л о в а: сопоставление с образцом, Rete алгоритм, Treat алгоритм, продукционная система, логический вывод, CLIPS.

Продукционные системы широко применяются при решении задач искусственного интеллекта, что обусловило развитие специализированных программных средств их разработки — оболочек. Назначением таких программных средств является предоставление разработчику встроенных механизмов логического вывода. С одной стороны, это упрощает процесс разработки и позволяет быстро и эффективно создавать прикладные продукционные системы, с другой — обусловливает зависимость эффективности работы прикладной системы от механизмов среды ее разработки.

На эффективность продукционных систем относительно затрат ресурсов памяти и времени наибольшее влияние оказывает механизм сопоставления с образцом. Существует прямая зависимость между форматом представления правил (продукций) в базе знаний и быстродействием системы. Однако в современных средах разработки продукционных систем реализуется только один механизм сопоставления с образцом. Наиболее апробированными реализациями этого механизма являются Rete и Treat алгоритмы [1, 2] и их модификации. Результаты исследований свидельствуют о том, что эффективность их использования зависит от особенностей базы знаний, а это делает невозможным выбор оптимального алгоритма заранее. Поэтому задача создания программной среды для разработки продукционных систем, в которой на основе текущей базы знаний можно экспериментально определить ресурсоемкость логического вывода, является актуальной и имеет практическое значение.

Анализ литературных данных и постановка проблемы. Подавляющее большинство современных прикладных продукционных систем создается с помощью специализированных оболочек [3]. Наиболее известными коммерческими оболочками продукционных систем являются Jess, Exsys Corvid, G2. Из свободно распространяемых оболочек продукционных систем наиболее известны CLIPS, Drools, Soar.

Была исследована оболочка CLIPS, так как в настоящее время она является одним из самых распространенных программных средств для разработки прикладных продукционных систем, проведения научных исследований и обучения студентов специальностей, связанных с искусственным интеллектом [4]. Основные преимущества, обусловившие выбор CLIPS, это распространение по свободной лицензии с открытым исходным кодом, который может быть модифицирован и использован без ограничений, а также полная сопроводительная документация и возможность сотрудничества с разработчиками системы. Концептуальными преимуществами является поддержка нескольких парадигм программирования, кроссплатформенность, возможность интеграции с другими языками программирования. Разработчики представили специализированные версии среды для работы в таких операционных системах как Android и iOS, что позволяет создавать экспертные системы для портативных устройств. Кроме того, CLIPS апробирован во многих прикладных продукционных системах. Последняя версия CLIPS 6.3 представлена в марте 2015 года [5].

В среде CLIPS реализована оптимизированная версия Rete алгоритма сопоставления с образцом [1]. Существует ряд подходов к его реализации, которые определяются способами представления Alpha и Beta памяти в сети сопоставления [6]. Текущая версия CLIPS 6.3 имеет ряд усовершенст-

ваний по сравнению с предыдущей. Основные изменения касаются подходов к увеличению быстродействия посредством оптимизации алгоритма сопоставления [7].

Усовершенствование Rete алгоритма осуществлялось в соответствии со специфическими требованиями задачи логического вывода [8—10]. Наиболее распространенной модификацией универсального Rete алгоритма является Treat алгоритм, который может быть использован для общего случая прикладной задачи [2]. Его используют в отдельных версиях известных оболочек продукционных систем, таких как Soar и OPS.

В работе [11] приведены схемы сопоставления с образцом по Rete и Treat алгоритмам. Однако пока не установлены формальные критерии, позволяющие выбрать оптимальное относительно вычислительных ресурсов представление продукции в базе знаний (БЗ) на этапе проектирования продукционной системы. Поэтому необходимо создать среду разработки для реализации двух подходов к сопоставлению с образцом с целью экспериментального выбора оптимального алгоритма в зависимости от текущей БЗ. Предлагается создать такую среду на базе оболочки CLIPS со встроенным алгоритмом Rete и реализацией Treat алгоритма.

Для достижения указанной цели поставлены следующие задачи:

1. Определить программные модули CLIPS, обеспечивающие предкомпиляцию Rete-сети.

2. Определить модули CLIPS, обеспечивающие сопоставление с образцом в режиме логического вывода.

3. Разработать подход к реализации Treat алгоритма с сохранением существующей структуры представления данных и оптимизации сопоставления с образцом.

Инкрементное сопоставления с образцом с использованием Rete алгоритма или его модификаций является наиболее распространенным в прикладных производительных системах. Инкрементный подход предусматривает сохранение состояния задачи в процессе логического вывода. Он основан на том, что в прикладных задачах на каждом шаге вывода изменения рабочей памяти влияют лишь на сравнительно небольшое количество продукций. Сопоставление осуществляется не по условной части правила, а по заранее созданной на этапе предкомпиляции сети анализа антецедентов продукций. Таким образом, сопоставление происходит в двух режимах: на этапе предкомпиляции базы знаний продукций формируется Rete-сеть потока данных; на этапе логического вывода проводится непосредственно сопоставление по сети.

Rete-сеть состоит из двух частей, α- и β-сетей, которые соответствуют этапам сопоставления. Согласования в рамках одного шаблона продукции

происходят в узлах α-сети и называются внутренними тестами. Внутренние тесты предусматривают проверку типов, константных значений, согласования переменных, встречающихся в условной части правила только один раз. На этапе выполнения в узлах α-сети хранятся сведения о фактах, которые привели к их активации, т.е. α-память.

В узлах β-сети происходит согласование шаблонов в рамках отдельной продукции — внешние тесты. В Rete-сети β-узлы имеют два входа: левый и правый. Левый вход всегда получает информацию о согласовании от родительского α-узла, правый — от родительского β-узла по результатам предварительных согласований. В каждом β-узле содержится информация о фактах, которые привели к выполнению внешних тестов и его активации. Листовые (терминальные) узлы сети потока данных отражают активацию продукции и содержат информацию о фактах рабочей памяти, которые с ней согласуются. Таким образом, терминальные узлы сети сохраняют текущий конфликтный набор из активированных продукций.

В случае добавления факта к рабочей памяти информация об изменениях поступает к вершине сети сопоставления. Далее происходит распространение изменений по сети посредством выполнения тестов в узлах и их активации. Продукция активируется, когда эти изменения достигают листового узла. В классической версии Rete удаление фактов предусматривает те же действия, что и добавление. Факт поступает в сеть с отметкой об удалении и приводит к деактивации соответствующих вершин. Такой подход к удалению называют симметричным [12]. Для обработки негативных условных элементов в сети добавляют специализированные β-узлы, содержащие счетчик фактов, которые согласуются с отрицательным условным элементом и препятствуют активации.

Реализация сопоставления с образцом в оболочке CLIPS. В текущей версии CLIPS используется реализация сопоставления с образцом с хешированием α- и β-памяти в сети потока данных при асимметричном удалении. Сопоставление с образцом связано с функционированием системы в двух режимах работы: предкомпиляции и логического вывода. Архитектура системы отражает оба этапа сопоставления: построение сети потока данных и проверку изменения фактов рабочей памяти.

Синтаксический и семантический анализ продукций происходит на этапе загрузки правил в систему. Его целью является формирование сети Rete алгоритма. Добавление продукций во время логического вывода в прикладной задаче в CLIPS не предусмотрено.

Сопоставление с образцом является одним из базовых механизмов логического вывода, поэтому его реализация отражается в нескольких модулях CLIPS. В данном случае цель состояла в реализации Treat алго-

ритма с сохранением предложенного для Rete формата представления данных. Для ее достижения в первую очередь были определены различия между классической версией Rete алгоритма (Forgy), реализованной в среде CLIPS (Riley), и Treat (Miranker) алгоритмом. Результаты сравнения представлены в табл. 1, из которой видно, что в последней версии среды CLIPS реализован способ удаления фактов, предложенный для Treat алгоритма. Для реализации Treat алгоритма на основе существующего функционала проведен анализ текущей внутренней архитектуры CLIPS и особенностей представления данных. Фрагменты архитектуры CLIPS, необходимые для реализации Treat алгоритма, были распределены по категориям: неизменяемые и требующие модификации.

Предкомпиляция Rete-сети сопоставления. Формирование сети потока данных в среде CLIPS происходит в несколько этапов. На первом этапе осуществляется обработка БЗ с целью преобразования в формат, пригодный для топологии Rete-сети. Затем проводится оптимизация представления правил для обработки сети. Промежуточное представление БЗ используется для определения позиций переменных, на основании чего происходит формирование выражений, которые в дальнейшем используются для оценки согласования шаблона и факта. Последним этапом является интеграция правила в сети сопоставления и связывания.

Таблица 1

Способ реализации	Rete (Forgy)	CLIPS Rete (Riley)	Treat (Miranker)
Компиляция сети потока данных	α-сеть внутренних тестов	α-сеть внутренних тестов	α-сеть внутренних тестов
	β-сеть внешних тестов	β-сеть внешних тестов	Динамическая сеть согласования
Сохранение данных	α-память согласования фактов и β-память частичного согласования с шаблонами	α-память согласования фактов, β-память частичного согласования с шаблонами, конфликтное множество	α-память согласования фактов и конфликтное множество
Обработка добавления фактов	Распространение от корневого узла в статической сети потока данных	Распространение от корневого узла в статической сети потока данных	Распространение по статической α-сети и построение динамической сети согласования переменных в случае необходимости
Обработка удаления фактов	Симметричное удаление	Асимметричное удаление	Асимметричное удаление

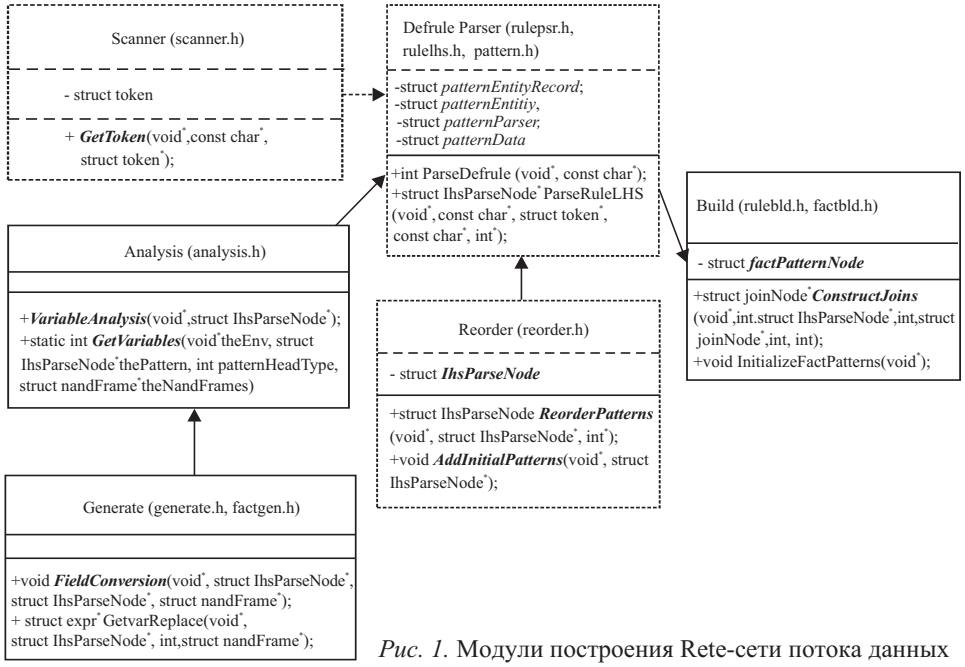

Рис. 1. Модули построения Rete-сети потока данных

По результатам данного исследования выделены фрагменты архитектуры системы, отвечающие за построение Rete-сети (рис. 1). Штриховыми линиями на рис. 1 обозначены модули, не требующие изменений. Структура каждого модуля содержит название, соответствующее функциональному назначению, основные данные и функции в том виде, в котором они объявлены в системе.

В табл. 2 приведено описание логических модулей и указаны необходимые модификации для обеспечения Treat-сети потока данных.

Модули Scanner и Defrule Parser обеспечивают считывание продукций из входного потока данных и их представление во внутреннем формате системы.

Модуль Reorder обеспечивает дополнительную оптимизацию представления правил в системе. Доказано, что такой подход может значительно ускорить логический вывод в прикладной задаче [13]. Дополнительная оптимизация промежуточного представления БЗ предполагает реорганизацию и модификацию.

Реорганизация представляет собой упорядочение условных элементов таким образом, что в антецедентах остается единственный условный элемент «или», расположенный в самом начале. Это позволяет упростить построение сети согласования фактов и уменьшить ее размер.

Таблица 2

Модуль	Назначение	Входные данные	Выходные данные	Предусмотренные изменения
Scanner	Считывание информации из входного потока данных по заданному формату языка. Лексический анализ продукций. Формирование структуры для представления входных маркеров заданного типа	Лексемы из входного потока данных	Маркер в виде структуры данных, содержащей тип и значение лексемы	Без изменений
Defrule Parser	Синтаксический анализ продукций. Формирование промежуточного формата БЗ для обработки	Маркеры лексем	Список правил БЗ в формате внутреннего представления. Промежуточное представление БЗ в виде структур данных	То же
Reorder	Оптимизация представления условных элементов для анализируемой сети	Промежуточное представление БЗ в виде структур данных	Оптимизированное представление БЗ в виде структур данных	" "
Analysis	Сохранение положения переменных в антецеденте и формирования выражений, используемых для оценки согласования в β-сети. Семантический анализ БЗ	Оптимизированное представление БЗ в виде структур данных	Представление БЗ с информацией о тестах, выполняемых с целью согласования переменных для внешних или внутренних тестов	Представление сети для динамического согласования переменных
Build	Интегрирует правило в Rete-сеть	То же	Сеть шаблонов (α-сеть) и сеть согласования (β-сеть)	Представления α-сети для хранения информации, необходимой при согласовании переменных. Представление данных для формирования динамической сети согласования переменных
Generate	Построение узлов β-сети для проверки заданных ограничений на значения условных элементов правила	" "	Выражения для проверки значений в узлах α- и β-сети	Динамическое формирование выражений для проверки согласования переменных в пределах одного правила

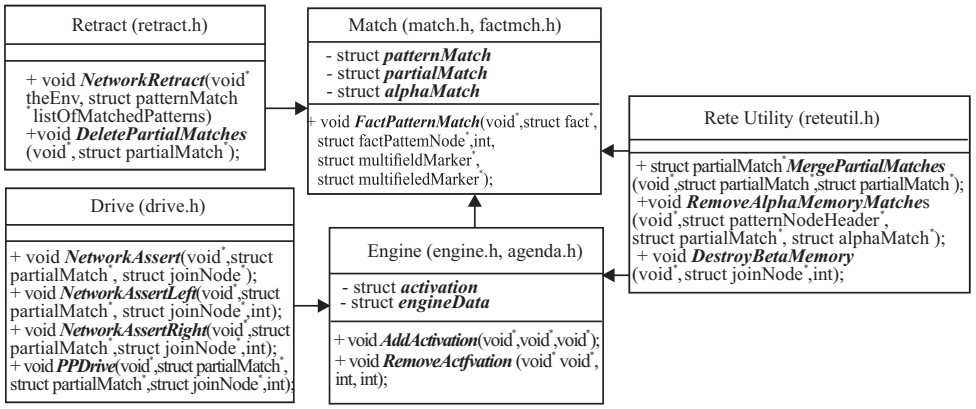

Рис. 2. Модули сопоставления в Rete-сети

Модификация обеспечивает уменьшение избыточности в системе посредством удаления лишних условных элементов и логических связок, не влияющих на результаты согласования.

Созданная общая структура представления продукций может быть одинаково эффективно использована для алгоритмов Rete и Treat. Поэтому для сохранения целостности и однотипности системы эти модули остаются неизменными.

Обработка Rete-сети сопоставления. В режиме логического вывода по сети сопоставления используются различные модули, предназначенные для обработки процесса добавления и удаления фактов. В этом режиме сопоставление происходит в два этапа: согласование фактов с шаблоном и связывание переменных. Для каждого из этапов предусмотрены различные структуры представления данных, определенные при формировании сети Rete. Модули, обеспечивающие непосредственно процесс сопоставления, представлены на рис. 2.

Основным отличием алгоритма Treat от Rete является отсутствие сохранения согласования переменных, а следовательно, и β-сети сопоставления. В Treat алгоритме применяется подход к сохранению конфликтного множества, позволяющий эффективно обрабатывать процесс удаления фактов. В табл. 3 приведены модули обработки сети потока данных в среде CLIPS и определены изменения, необходимые для реализации Treat алгоритма.

Реализация сети Treat алгоритма. На основании абстрактного описания Treat алгоритма, предложенного в работе [2], представим схему его работы (рис. 3). На вход алгоритма поступают изменения рабочей памяти в виде фактов с отметкой (маркером) добавления или удаления. Для

каждого условного элемента правила (condition element (CE)) на пути распространения изменений по сети выполняются тесты в зависимости от типа узла и значения маркера.

В Treat алгоритме α-сеть потока данных используется так же, как в Rete. Для обеспечения функционирования Treat-сети в каждом узле α-памяти хранится список продукций, с которыми она согласовывается. В предложенной в работе [2] версии для параллельных вычислений не применяется оптимизация вследствие совместного владения несколькими продукциями вершин сети. Однако для последовательной реализации CLIPS сохранение подхода к совместному владению вершинами в α-сети позволяет избежать избыточных затрат памяти.

Таблица 3

Модуль	Назначение	Входные данные	Выходные данные	Необходимые изменения
Retract	Обработка сети потока данных после удаления факта	Факт для удаления и список согласуемых с ним шаблонов	Сеть согласования после обработки удаления факта	Удаление обработки β-сети. Обеспечение удаления факта с использованием сохраненного конфликтного множества
Drive	Обработка сети потока данных в случае добавления фактов к рабочей памяти	Добавленный факт	Сеть согласования с изменениями в соответствии с добавленным фактом	Создание функционала динамического формирования и обработки сети согласования. Удаление обработки β-сети
Match	Определение основных структур данных для представления α- и β-сети	Узлы α-сети	Значение α-памяти в узлах сети	Модификация структур данных для обеспечения сохранения конфликтного множества и формирования сети согласования в режиме логического вывода
Engine	Выполнение правила из конфликтного множества, избранного по заданной стратегии разрешения конфликта с сохранением целостности сети потока данных	Текущее конфликтное множество продукций	Изменения в рабочей памяти в результате выполнения консеквента продукции	Модификация функционала обеспечения целостности сети потока данных с учетом подхода к сохранению конфликтного множества
Rete Utility	Инициализация и доступ к α- и β-сети с внешних модулей	Сеть потока данных или запрос на ее инициализацию	α / β-сеть	Модификация инициализации и доступа к сети потока данных

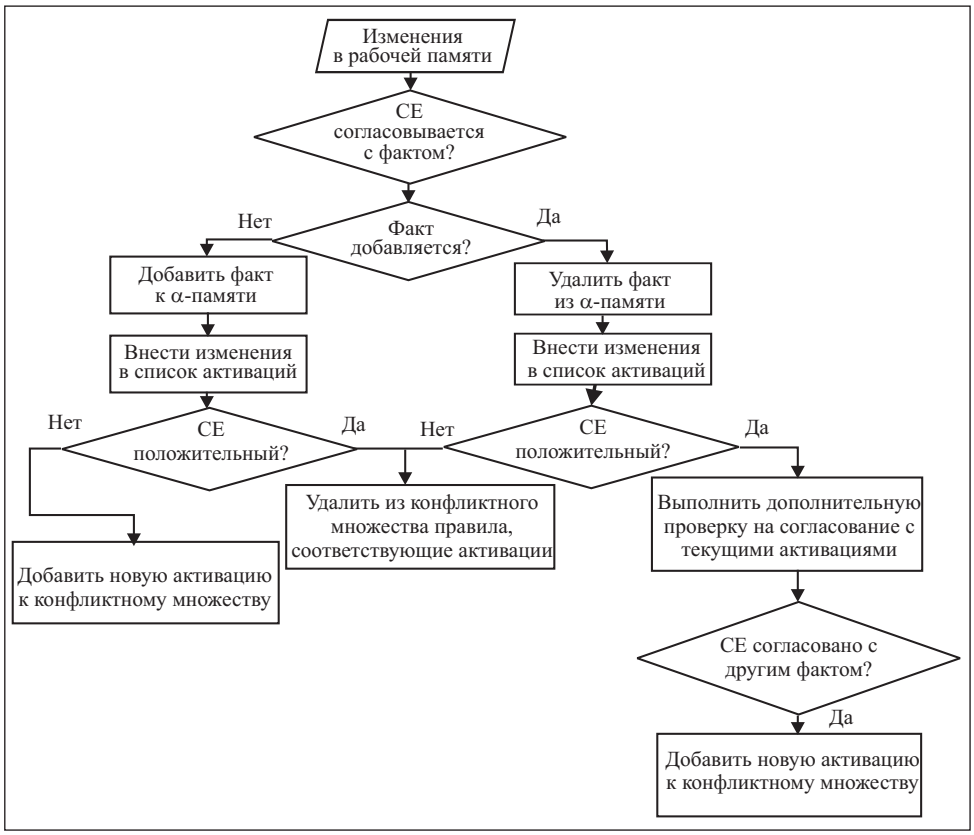

Рис. 3. Схема работы Treat алгоритма

β-сеть отсутствует в Treat алгоритме. Согласование переменных осуществляется в режиме логического вывода по динамически созданной сети. Сеть согласования формируется только для продукций, в которых были активированы все α-вершины.

Для обеспечения модификации Rete алгоритма в Treat необходимо выполнить следующие шаги:

1. Модифицировать представление сети сопоставления для обеспечения сохранения данных, необходимых для работы Treat; удалить избыточную информацию.

2. Удалить функционал формирования и обработки β-сети.

3. Реализовать динамическое формирование сети согласования и ее обработки с применением текущих подходов к представлению данных в системе.

Изменения в системе, необходимые для реализации Treat алгоритма сопоставления с образцом, приведены в табл. 4. Для каждого из модулей

Таблица 4

Модуль	Функция	Назначение	Статус	Уровень обработки Treat-сети
Analysis	Variable Analysis	Образует взаимные ссылки на переменные в пределах одного правила. Проверяет семантические ошибки, связанные с согласованием переменных	Модифицированная	Построение α-сети
	Create External Network Test	Генерирует выражения для согласования переменных в динамической сети согласования	Добавлена	Построение динамической сети согласования
	Get Variables	Распространяет по сети потока данных информацию о расположении переменных в антецеденте в одном семантическом пространстве	Без изменений	Построение α-сети
	Process Variable	Обеспечивает обработку переменной, встречающейся в правиле один раз	То же	То же
Build	Construct Joins	Интегрирует внутренние и внешние тесты в соответствующие вершины α- β-сетей	Модифицированная	Построение сети потока данных
	Initialize FactPatterns	Добавляет шаблоны фактов к сети потока данных	Без изменений	Построение α-сети
	PlaceFact Pattern	Интегрирует шаблоны фактов в α-сеть	То же	То же
	Construct Dynamic Joins	Интегрирует внешние тесты в динамическую сеть согласования	Добавлена	Построение динамической сети согласования
Generate	FieldConversion	Генерирует выражения для тестов, связанных с ограничениями на значение в шаблонах, и интегрирует их в сеть потока данных	Модифицированная	Построение сети потока данных
	Getvar Replace	Заменяет переменные в узлах сети на вызовы функций, получающие текущее значение переменной из частичного согласования или активации	То же	Построение и обработка сети потока данных
Match	FactPattern Match	Обеспечивает выполнение сопоставления в α-сети	Без изменений	Обработка α-сети

Окончание табл. 4

Модуль	Функция	Назначение	Статус	Уровень обработки Treat-сети
Retract	Network Retract	Удаляет из сети потока данных информацию о предварительном согласовании с заданным фактом	Модифицированная	Обработка процесса удаления факта в сети потока данных
	Delete Partial Matches	Удаляет все частичные согласования из сети	То же	Удаление фактов из сети в процессе перезагрузки системы
	Delete Partial Join Matches	Удаляет узлы сети согласования, не приведшие к активации, из динамической сети потока данных	Добавлена	Обработка динамической сети согласования
Drive	Network Assert	Обрабатывает согласования переменных и формирование активаций для добавления в конфликтное множество	Модифицированная	Обработка согласования переменных в динамической сети
	Network AssertLeft	Проверяет согласования первого активированного условного элемента с другими шаблонами антецедента	То же	То же
	Network Assert Right	Обеспечивает проверку согласования переменных в пределах антецедента	" "	" "
	PPDrive	Интегрирует результаты согласования в α-сети и результаты согласования переменных	" "	Обработка сети потока данных
Engine	AddActivation	Добавляет активации в конфликтное множество	" "	То же
	Remove Activation	Удаляет активации из конфликтного множества	" "	" "
Rete Utility	Merge Partial Matches	Объединяет частичные согласования для уменьшения затрат памяти	" "	" "
	Remove Alpha Memory Matches	Удаляет α-память из узлов сети	Без изменений	Обработка α-сети
	Destroy Beta Memory	Удаляет результаты согласования переменных в узлах сети согласования	Модифицированная	Обработка согласования переменных

выделены основные функции и структуры данных, которые обеспечивают процесс построения и обработки сети сопоставления алгоритма. Статус функции определяется объемом необходимых изменений. Считается, что функция осталась без изменений логики, если модификация касалась только взаимодействий между модулями. Уровень обработки Treat-сети определяется этапом сопоставления и подмножествами узлов сети потока данных, которыми оперирует функция. Построению α-сети соответствует этап предкомпиляции. Формирование динамической сети согласования и последующая обработка сети потока данных происходят на этапе логического вывода.

Таким образом, детализированы все модификации, обеспечивающие расширение оболочки продукционных систем CLIPS дополнительным алгоритмом сопоставления с сохранением основных подходов к представлению данных. Это позволяет реализовать два механизма сопоставления в единой среде с возможностью выбора одного из них пользователем — разработчиком прикладной продукционной системы.

Для примера тестирования расширенной оболочки CLIPS приведем одну из общепринятых в продукционных системах тестовых задач, Manners, решающую комбинаторную проблему рассадки гостей по интересам (задано 64 человека). Для описания этой задачи используется восемь правил (продукций), в антецеденте которых представлены условия, определяющие соседей за столом. Эти условия формализованы в виде фактов, представляющих отношение, связывающее имя гостя, его пол и хобби.

Факты являются образцами для механизма сопоставления. Для выполнения программы на входных данных по размещению 64-х гостей требуется запуск 2271 правила. При этом среднее число фактов рабочей памяти — 1307, среднее число активаций — 63. Затраты памяти для Rete алгоритма — 1828 Kb, для Treat — 1327 Kb. Результаты тестирования подтверждают преимущества Treat алгоритма по ресурсоемкости.

Выводы

Предложенный способ расширения продукционной оболочки CLIPS дает возможность в дальнейшем дополнять систему добавочными инкрементными алгоритмами сопоставления с сохранением единого формата представления данных.

СПИСОК ЛИТЕРАТУРЫ

1. *Forgy C.L.* On the Efficient Implementation of Production System. PhD thesis. — Pittsburg, 1979. — 356 p. — [Электронный ресурс]. — Режим доступа: http://reports-archive.adm.cs.cmu.edu/anon/scan/CMU-CS-79-forgy.pdf.
2. *Miranker D.P.* TREAT: A New and Efficient Match Algorithm for AI Production Systems. — London: Pitman/Morgan Kaufmann, 1987. — 144 p.
3. *Sanjay K.* Importance of Expert System Shell in Development of Expert System// Intern. J. of Innovative Research and Development. — 2015. — Vol. 4, Issue 3. — P. 128 — 133. — [Электронный ресурс]. — Режим доступа: http://www.ijird.com/index.php/ijird/article/view/62073.
4. *Stefano A., Gangemi F., Santoro C.* ERESYE: artificial intelligence in Erlang programs // ERLANG'05: Proc. of the 2005 ACM SIGPLAN workshop on Erlang. —Tallinn, Estonia, 2005. — P. 62—71. — [Электронный ресурс]. — Режим доступа: http://dl.acm.org/citation. cfm?id=1088373&dl=ACM&coll=DL&CFID=542447529&CFTOKEN=27514249.
5. *CLIPS* Rule Based Programming Language.— [Электронный ресурс]. — Режим доступа: http://sourceforge.net/projects/clipsrules/files/CLIPS/6.30/.
6. *Doorenbos R.* Production Matching for Large Learning Systems. — Pittsburg: Computer Science Department, Carnegie Mellon University, 1995. — 208 p.
7. *Riley G.* CLIPS Implementation of Rete // Proc. of October Rules Conference. — Chantilly, 2009. — [Электронный ресурс]. — Режим доступа: http://sourceforge.net/p/ clipsrules/ discussion/776945/thread/8460f0d6/e8cb/attachment/CLIPS_Implementation_of_ Rete.pdf .
8. *Sellis T., Lin C.C., Raschid L.* Implementing large production systems in a DBMS environment: concepts and algorithms // Management of data : ACM SIGMOD international conference. — N Y: ACM Press, 1988. — P. 404—423.
9. *Berstel B.* Extending the RETE algorithm for event management // Temporal Representation and Reasoning: Ninth Intern. Symposium. IEEE Computer Society Press. — Washington, 2002. — P. 49—51.
10. *Kang J.A.* Shortening matching time in OPS5 production systems// IEEE Transactions on Software Engineering. — 2004. — № 30 (7). — P. 448—457.
11. *Мажара О.О.* Порівняння Rete та Treat алгоритмів співставлення зі зразком // Адаптивні Системи Автоматичного Управління: Міжвідомчий науково-технічний збірник. Вип. 1 (24) — Київ: НТУУ «КПІ», 2014. — С. 53—61.
12. *Proctor M.* Symmetrical and Asymmetrical Rete. — [Электронный ресурс]. — Режим доступа: — http://blog.athico.com/2008/10/symmetrical-and-asymmetrical-rete.html.
13. *Джаррантано Дж., Райли Г.* Экспертные системы: принципы разработки и программирование. 4-е издание; пер. с англ. — М. : ООО «И.Д. Вильямс», 2007. — 1152 с.

О.А. Mazhara

TREAT ALGORITHM IMPLEMENTATION BY THE BASIC MATCH ALGORITHM BASED ON CLIPS PROGRAMMING ENVIRONMENT

Implementation of the logical inference by the applied knowledge base based on Rete and Treat match algorithms for choosing optimal one in terms of resources usage and performance has been proposed. Treat algorithm implementation, which allows saving current approaches for data representation and optimizations by hashing, precompile network representation and knowledge base representation, was formalized for CLIPS. The proposed approach allows further extension of the CLIPS programming environment by additional incremental match algorithms.

K e y w o r d s: match, Rete algorithm, Treat algorithm, production system, inference, CLIPS.

REFERENCES

1. Forgy, C.L. (1979), "On the Efficient Implementation of Production System", Doctor's Degree, Carnegie Mellon University, Pittsburgh, USA.
2. Miranker, D.P. (1987), "TREAT: A New and Efficient Match Algorithm for AI Production System", Doctor's degree, Pitman / Morgan Kaufmann, London, UK.
3. Sanjay, K. (2015), "Importance of Expert System Shell in Development of Expert System", *International Journal of Innovative Research and Development*, Vol. 4 no. 3, pp. 128-133, available at: http://www.ijird.com/index.php/ijird/article/view/62073 (accessed September 1, 2015).
4. Stefano, A. Gangemi, F. and Santoro, C. (2005), "ERESYE: artificial intelligence in Erlang programs", *Proceedings of the ACM SIGPLAN workshop on Erlan*, Tallinn, Estonia, September 26-28, 2005. New York: ACM, available at: http://dl.acm.org/citation.cfm?id=1088373&dl=ACM&coll=DL&CFID=542447529&CFTOKEN=27514249 (accessed August 10, 2015).
5. CLIPS Rule Based Programming Language, available at: http://sourceforge.net/projects/clipsrules/files/CLIPS/6.30/ (accessed August 30, 2015).
6. Doorenbos, R. (1995), "Production Matching for Large Learning Systems", Doctor's Degree, Carnegie Mellon University , Pittsburgh, USA.
7. Riley, G. (2009), "CLIPS Implementation of Rete", *Proceedings of the October Rules Conference*, Chantilly, Virginia, USA, October 25-26, 2009, available at: http:// sourceforge.net/p/clipsrules/discussion/776945/thread/8460f0d6/e8cb/attachment/CLIPS_ Implementation_of_Rete.pdf (accessed September 2, 2015).
8. Sellis, T. Lin, C.C. and Raschid, L. (1988), "Implementing large production systems in a DBMS environment: Concept and algorithms", *Proceedings of the ACM SIGMOD International conference Management of data*, Chicago, Illinois, June 1-3, 1988, pp. 404-423.
9. Berstel, B. (2002), "Extending the RETE algorithm for event management", *Proceedings of the Ninth International Symposium on Temporal Representation and Reasoning*, IEEE Computer Society, Washington, DC, USA , pp. 49-51.
10. Kang, J.A., Mo, A. and Cheng, K. (2004), "Shortening Matching Time in OPS5 Production Systems", *IEEE Transactions on Software Engineering,* Vol. 30 , no.7, pp. 448-457.
11. Mazhara, O.O. (2014), "Comparison of the Treat and Rete match algorithms", *Adaptyvni Systemy Avtomatychnogo Upravlinnya: Mizhvidomchyy naukovo-tehnichny zbirnyk*, Kiev, KPI, Vol. 1, no. 24, pp. 53-61.
12. Proctor, M. (2008), "Symmetrical and Asymmetrical Rete", available at: http://blog.athico.com/2008/10/symmetrical-and-asymmetrical-rete.html (accessed September 3, 2015).
13. Giarratano, J. and Riley, G. (2007), *Ekspertnye sistemy: printsipy razrabotki i programirovanie* [Expert systems: design principles and programming], OOO «I.D. Vil'yams», Moscow, Russia.

Поступила 10.09.15

МАЖАРА Ольга Александровна, аспирант, ассистент кафедры автоматизации проектирования энергетических процессов и систем теплоэнергетического факультета Национального технического университета Украины «Киевский политехнический ин-т», который окончила в 2012 г. Область научных исследований — искусственный интеллект, экспертные и продукционные системы.