УДК 004.89

ВЕРИФИКАЦИЯ ДАННЫХ В СИСТЕМАХ ОТСЛЕЖИВАНИЯ ЗАДАЧ С ПОМОЩЬЮ ПРОДУКЦИОННЫХ ПРАВИЛ

Р.С. Катериненко, И.А. Бессмертный

Предлагается методика верификации данных в системах отслеживания задач с помощью модели продукционных правил. Данная модель позволяет в декларативном стиле формулировать условия, которым должно соответствовать информационное наполнение, и использовать логический вывод. В качестве практических результатов описывается применение разработанной системы верификации для реального проекта разработки программного обеспечения.
Ключевые слова: продукционные правила, логический вывод, системы отслеживания задач, верификация.

Введение

Система отслеживания задач (СОЗ) – прикладная программа, разработанная с целью помочь контролировать ход отдельных задач и проекта в целом. Основные компоненты такой системы – это база данных задач и графический интерфейс с возможностью редактирования задач и построения аналитических отчетов. Информация в подобной системе меняется и используется большим количеством специалистов из разных областей, что является одним из источников ошибок. Какой бы хорошей ни была СОЗ, со временем эксплуатации в содержащейся информации накапливаются неточности, противоречия, ошибки. Этому способствуют человеческий фактор и неточность естественного языка. В свою очередь, от адекватного состояния информационного наполнения зависят продуктивность работы участников проекта и его успех. Другой проблемой является миграция неточностей из базы СОЗ в автоматически генерируемые отчеты. Неправильные данные в таких отчетах могут привести к неверно подсчитанным трудозатратам, финансовым затратам и ошибочному планированию.

В основном СОЗ создаются под каждый отдельно взятый проект на базе общей платформы СОЗ. Платформа для создания СОЗ имеет язык высокого уровня абстракций, подходящий для описания широкого круга проектов в терминах рабочего процесса, артефактов рабочего процесса, связей между артефактами, состояний артефактов и т.д. Из-за высокой универсальности этот язык позволяет реализовать только основные проверки целостности информации на уровне правильности заполнения полей артефактов. Его возможности ограничиваются синтаксическими проверками.

Авторы предлагают использовать для целей верификации информационного наполнения СОЗ логические правила – продукции. Преимущества продукционного языка – декларативность и язык запросов, основанный на логическом выводе. Продукционный язык обладает большими выразительными возможностями, позволяющими формулировать сложные ограничения на семантику информации в СОЗ, в том числе рекурсивные правила.

Модель продукционных правил

Модель продукционных правил относится к классу дедуктивных логических систем, в основе которых лежит принцип дедукции – порождение истинных фактов из заданных аксиом и правил. В общем случае дедуктивная система состоит из набора аксиом, правил вывода и изначально заданных формул. В модели продукционных правил с целью повышения эффективности логического вывода введены ограничения:

- правила вывода (продукционные правила) являются хорновскими выражениями;
- факты представляют собой атомарные формулы.

Хорновское выражение (horn clause) – это дизъюнкция литералов с одним единственным положительным литералом. Литералом называется атомарная формула, а атомарная формула с отрицанием называется отрицательным литералом. Если применить эквивалентность $\neg A \vee B \equiv A \to B$, то хорновский дизъюнкт превращается в привычный вид импликации или продукции:

$$\neg A_1 \vee \neg A_2 \vee \ldots \vee \neg A_{n-1} \vee \mathrm{A}_n \equiv A_1 \wedge A_2 \wedge \ldots \wedge A_{n-1} \to A_n .$$

В некоторых продукционных системах поддерживаются рекурсивные правила, отрицание по «принципу закрытого мира» и термы-функции. Согласно «принципу закрытого мира» (closed world assumption) все те факты, которые не заложены в систему и не могут быть получены логическим выводом, объявляются ложными. Другими словами, все, о чем нет информации в системе, объявляется ложным.

Преимуществами указанного типа моделей являются:

- простота и ясность основной единицы – продукционного правила;
- независимость продукций и легкость модификации баз знаний;
- строгость, простота и изученность механизма логического вывода;
- эффективность логического вывода.

К недостаткам можно отнести:

- малую степень структурированности базы знаний;
- неуниверсальность.

Применение модели продукционных правил

Как было показано выше, для построения логической модели необходимы изначально заданные факты и правила. Часть информации СОЗ, подлежащая верификации, должна быть представлена в виде фактов для формирования предикатов. Правилами здесь являются записанные на продукционном языке свойства, которым должны удовлетворять данные. Проверка того, что данные СОЗ удовлетворяют заданным правилам, называется верификацией СОЗ.

Рассмотрим предлагаемую методику на примере СОЗ, эксплуатируемой в проекте разработки программного обеспечения длительностью 10 лет. Основными артефактами являются спецификации, тикеты, проекты. Тикетом будем называть представление задачи в СОЗ. Тикет является минимальной атомарной единицей информации о задаче. В данном примере одной задаче соответствует один тикет. В проекте может быть произвольное число тикетов и спецификаций, связанных между собой различными связями. Тикет бывает следующих видов:

- описание дефекта;
- описание задачи для разработки;
- описание исследовательской задачи;
- описание дизайнерской задачи;

и т.д.

У тикета имеется около 30 обязательных и необязательных для заполнения полей. Помимо полей, у тикета есть именованные связи с другими тикетами. Связи представляют собой ссылки от одного тикета к другому и могут иметь следующие типы: «has clone», «duplicates», «introduces», «incorporates», «relates». Симметричными им являются соответственно: «clone of», «is duplicated by», «introduced by», «is incorporated by», «is related to». Семантика связей следует из англоязычных названий связей и не является существенной для раскрытия содержания темы, поэтому выходит за рамки настоящей работы.

Существенная для верификации информация экспортируется из системы отслеживания задач (СОЗ) в предикаты следующим образом.

- Каждому полю тикета сопоставляется двухместный предикат тикет–значение. Например, если тикет № 8520 имеет поле «Номер спецификации» («requirement»), равное 328, то соответствующий предикат будет иметь вид *hasRequirement(T1,T2): hasRequirement(8520, 328) = true*.
- Каждому типу связи ставится в соответствие двухместный предикат тикет–тикет. Например, двум тикетам, связанным связью вида «introduces», будет соответствовать предикат вида *introduces(T1, T2)*. Свойства, которым должно удовлетворять информационное наполнение системы, формулируются в виде продукционных правил. В нашей системе одним из правил было наличие у каждого тикета ссылки (связи) на спецификацию, под которой по нему велись работы. Если тикет был причиной других тикетов, то спецификация должна была наследоваться. При составлении биллинговых отчетов часы, затраченные на выполнение спецификации, брались как сумма часов по связанным с ней тикетам. Нарушение указанного правила приводило к большим расходам. Сформулируем это ограничение в виде правил – продукций:

  introducesClosure(T1, T2) :– introduces(T1, T2) (1)

  introducesClosure(T1, T3) :– introduces(T1, T2), introducesClosure(T2, T3) (2)

  requirementForClosure(R, T2) :– requirementFor(R, T1), introducesClosure(T1, T2) (3)

Правила (1) и (2) определяют предикат *introducesClosure* как транзитивное замыкание по свойству «introduces». После выполнения логического вывода он должен содержать все пары тикетов, первый из которых является причиной возникновения второго. Правило (2) задает предикат *requirementClosure*, который должен содержать все пары вида спецификация–тикет, учитывая наследование спецификаций. Эти три правила ((1)–(3)) задают условие, которому должно соответствовать правильное информационное наполнение. Но в реальной системе всегда содержатся ошибки. В связи с этим следующим шагом является написание диагностического правила:

failRequirement(R, T) :– requirementForClosure(R, T), not requirementFor(R, T) (4)

Предикат *failRequirement* должен содержать те пары вида спецификация-тикет, которых не было в системе, но которые должны в ней быть, следуя описанным ограничениям. Исходя из этого, срабатывание данного правила соответствует ошибочной ситуации, когда в СОЗ у тикета *T* не проставлена спецификация *R*. После верификации информационное наполнение может быть скорректировано администратором СОЗ таким образом, чтобы диагностические предикаты стали «пустыми» после следующей итерации.

Общий алгоритм верификации

На рис. 1 в схематическом виде приведены основные этапы алгоритма верификации и данные, передаваемые между ними.

1. Выгрузка данных. На этом этапе происходит выгрузка данных из СОЗ в виде отчетов, результатов выполнения SQL-запросов или html-страниц.

2. Преобразование. Из выгруженных данных формируется логическая модель: предикаты, домены (области определения аргументов предиката).
3. Логический вывод. К логической модели добавляются продукционные правила для верификации (1), (2), (3) и вспомогательные диагностические продукционные правила (4) для локализации ошибочной информации.
4. Коррекция данных. На данном шаге аналитик или администратор системы должен исправить ошибочную информацию, указанную в диагностических предикатах.

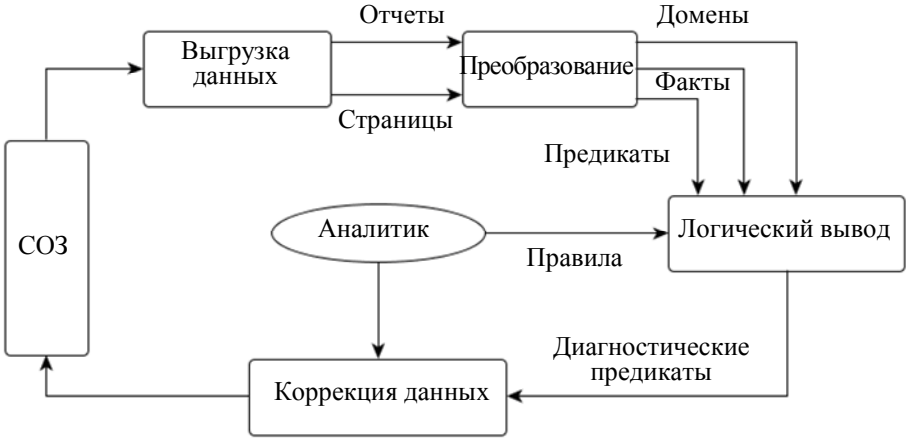

Рис. 1. Основные этапы алгоритма верификации

Описанная итерация может быть проделана несколько раз по мере добавления диагностических правил и коррекции информации. Информация считается верифицированной, когда все диагностические предикаты пусты.

Описание программной реализации

На рис. 2 приведена схема модулей программной реализации.

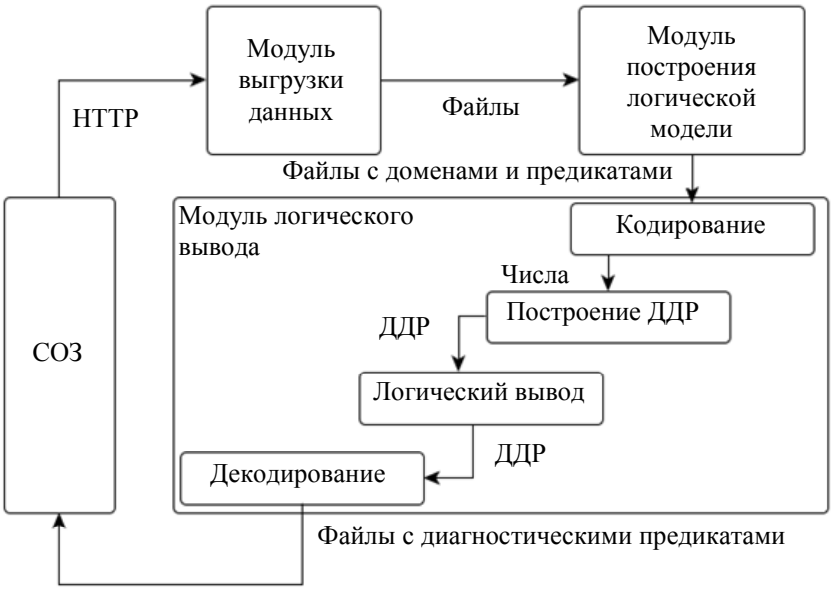

Рис. 2. Схема модулей программной реализации

Основными модулями являются: модуль выгрузки данных, модуль построения логической модели и модуль логического вывода. Функционал модуля выгрузки данных заключается в извлечении верифицируемых данных из СОЗ. В данном случае взаимодействие с СОЗ, построенной на программной платформе JIRA, осуществляется по протоколу HTTP, но может быть реализовано непосредственно через базу данных СОЗ и программное API. Выгрузка данных может быть полной, когда каждый раз выгружаются все необходимые для верификации данные, или инкрементальной, когда добавляются только новые данные.

Полученные данные в виде файлов поступают на вход модуля построения логической модели, где формируются домены и предикаты. В нашем приложении этот модуль позволяет декларативно задать правила для преобразования данных в предикаты. При этом домены создаются автоматически.

Построенная логическая модель и логическая программа, состоящая из продукционных правил, поступают на вход модуля логического вывода. Существуют разные подходы для осуществления логического вывода на продукционных правилах. Авторами был выбран подход на основе двоичных диаграмм решений (ДДР, Binary Decision Diagram), поскольку авторы ведут исследования в области применения ДДР для логического вывода [1–3]. ДДР – это структура данных, хорошо зарекомендовавшая себя на практике в области верификации аппаратных средств, позволяющая выполнять манипуляции над булевыми формулами. Компактность и эффективность операций над ДДР может быть использована для представления отношений и быстрого логического вывода [4–6]. В этом модуле последовательно выполняются следующие шаги:

1. кодирование доменов и предикатов числами;
2. построение ДДР;
3. логический вывод над ДДР;
4. декодирование ДДР.

Заключительным шагом является осуществление коррекции информации в СОЗ.

Основные результаты

Мы провели сравнительный эксперимент на реальной СОЗ для нашей задачи, описанной в разделе «Применение модели продукционных правил». К моменту начала логического вывода в базе данных содержалось 4667 факта и 6 продукционных правил. В качестве системы для сравнения была выбрана система DES (Datalog Educational System), не использующая ДДР. Результаты эксперимента приведены в таблице. Время указано в миллисекундах. T_1 – время загрузки фактов в систему DES. T_2 – время логического вывода в системе DES. T_3 – время загрузки фактов в нашу систему. По сравнению с DES, в нашей системе можно получить отдельно время логического вывода и отдельно время записи его результата в файл или выдачи на экран. Этими временами являются T_4 и T_5 соответственно. Для сравнения производительности логического вывода берутся времена получения ответа от систем с уже загруженными фактами – T_2 и T_6. Как видно, T_2 превосходит T_6 более чем в 200 раз. Таким образом, наша система на основе ДДР осуществляет логический вывод в 200 раз быстрее DES. Загрузка фактов осуществляется быстрее в 1,4 раза. Система реализована на языке Java. Верифицируемая СОЗ создана с помощью системы JIRA. Эксперимент проводился на платформе со следующими характеристиками: Intel Core I5 2.3 ГГц, 4 Гб ОЗУ, Windows 7 64-bit, Java 1.6.0_25.

Количество фактов	Время логического вывода, в мс					
	DES T_1	DES T_2	ДДР T_3	ДДР T_4	ДДР T_5	ДДР $T_6 = T_4 + T_5$
4667	1001	130000	721	311	233	544

Таблица. Времена логического вывода

В данной работе получен опыт применения продукционной модели и разрабатываемого механизма логического вывода на реальной прикладной задаче.

Авторы считают, что создание базы знаний не является очень трудоемкой задачей, потому что может быть легко автоматизировано. Как правило, все СОЗ представляют исходные данные, необходимые для верификации, некоторым регулярным для всех проектов образом. Разработанный единожды модуль выгрузки данных под конкретную СОЗ может быть использован на протяжении всей эксплуатации системы без особых изменений. Построение логической модели и дедуктивный вывод осуществляются автоматически разработанной системой. Участие аналитика требуется лишь при коррекции исходных данных по результатам вывода и формулировки правил для верификации.

Выразительность продукционной модели варьируется в зависимости от ее свойств. Решаемая задача дала возможность выделить свойства продукционного языка, которые должны поддерживаться механизмом вывода для широкого применения на практике в подобного рода прикладных задачах: поддержка отрицания, рекурсии, временных предикатов, функций для работы со строками и числами.

Заключение

Предложенный подход может быть реализован в виде отдельного приложения для верификации систем отслеживания задач или в качестве плагина к системе отслеживания задач. В результате применения были обнаружены и скорректированы ошибки в информационном наполнении в эксплуатируемой системе. Выбранный метод логического вывода на основе двоичных диаграмм решений показал хорошую производительность и имеет большой запас по производительности.

На взгляд авторов, методику имеет смысл применять в проектах при выполнении двух условий:

1. над проектом работает достаточно большое количество специалистов разных областей, что способствует быстрому накоплению ошибок в системе отслеживания задач;
2. актуальность и корректность данных в системе отслеживания задач напрямую влияет на ключевые характеристики проекта, такие как трудозатраты, или влияет на качество и работоспособность производимого продукта.

Работа выполнена при финансовой поддержке ФЦП «Научные и научно-педагогические кадры инновационной России» на 2009–2013 годы (соглашение № 14.B37.21.0406).

Литература

1. Bessmertny I.A., Katerinenko R.S. The method of acceleration of logical inference in production knowledge model // Programming and Computer Software. – Springer, 2011. – V. 37. – № 4. – P. 197–199.
2. Бессмертный И.А. Теоретико-множественный подход к логическому выводу в базах знаний // Научно-технический вестник СПбГУ ИТМО. 2010. – № 2 (66). – С. 43–48.
3. Бессмертный И.А. Быстрый логический вывод в среде программирования Visual Prolog // Научно-технический вестник СПбГУ ИТМО. – 2010. – № 3 (67). – С. 50–56.
4. Iwaihara M., Inoue Y. Bottom-up Evaluation of Logic Programs Using Binary Decision Diagrams // Proc. 11th Int. Conf. Data Engineering. – Taipei, Taiwan, Mar. 1995. – P. 467–474.
5. Bryant R. Graph-based Algorithms for Boolean Function Manipulation // IEEE Transactions on Computers. – August 1986. – V. 35. – № 8. – P. 677–691.
6. Andersen H.R. An Introduction to Binary Decision Diagrams // Lecture notes for Advanced Algorithms. – Lyngby, Denmark, 1997. – P. 1–37.

Катериненко Роман Сергеевич – Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики, аспирант, innuWorks@gmail.com

Бессмертный Игорь Александрович – Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики, кандидат технических наук, доцент, igor_bessmertny@hotmail.com