УДК 004.832

Автоматическая генерация логического знания

А. Раутиайнен

Кафедра информационных технологий
Российский университет дружбы народов
ул. Миклухо-Маклая, 6, Москва, Россия, 117198

В статье рассмотрены вопросы, которые возникают при попытке генерации автоматическим образом логического знания в системах искусственного интеллекта, в первую очередь, в системах машинного доказательства теорем. Сформулированы три необходимых требования к подобному генератору и рассмотрено, как их можно выполнить.

Ключевые слова: автоматическое доказательство теорем, искусственный интеллект.

1. Введение

Автоматической генерации знаний уже давно отводилась центральная роль в искусственном интеллекте. В первых системах искусственного интеллекта речь шла только об автоматическом использовании знаний, а сами знания вводились в систему «вручную». Однако уже скоро системы стали настолько сложными, что возникла необходимость автоматической генерации знания. В попытках создания машинного разума для игр сталкивались с большим числом неодинаковых, но похожих друг с другом комбинаций — в итоге генерация тактики для разных комбинаций автоматическим образом часто оказывалась самым подходящим решением. Данный подход использовали в самой успешной в истории программе для игры в шашки «Chinook» [1], для компьютерной игры «Го» [2], для кубика Рубика [3] и многих других задач в области игр.

Автоматическая генерация знания также использовалась при создании экспертных систем в области медицины — прикладные правила выводятся из множества основных правил [4]. Пример совершенно другого способа автоматической генерации правил — американская система тактической симуляции для воздушных войск «TacAir–Soar». В нем правила создаются автоматическим образом прямо из наблюдения сенсорами поведения пилотов. Создатели «TacAir–Soar» могут похвастаться, что с 7500 правилами поведения их экспертная система — одна из крупнейших в мире систем, находящихся в практическом применении. Понятно, что создание подобной базы вручную уже не целесообразно — необходима автоматизация [5]. NASA также использует системы, в которых знание генерируется из базы данных CAD автоматическим образом [6].

В большинстве этих систем знание представлено не в логическом виде, а в форме, которая зависит от типа задачи (в случае экспертных систем — продукция, в случае игр — последовательность ходов и т.д.) [7]. Пока вопрос об автоматической генерации именно логического знания мало исследован, несмотря на то, что логическая форма представления знания самая общая, и допускает самое широкое применение. Например, любое предложение вида $\varphi \Rightarrow \psi$ — это своего рода аналог продукции в экспертной системе, оно утверждает, что из любого знания вида φ мы можем вывести знание вида ψ. Но предложение вида $\forall x(\varphi(x) \Rightarrow \psi(x))$ — это уже намного больше, чем просто продукция, это в некоторой степени «суперпродукция», которая описывает бесконечное множество возможных продукций. Таким образом, небходимо говорить также и о совокупном знании, которое нельзя представить в форме продукции, но которое все равно может быть полезным при доказательстве новых теорем.

Прежде всего логическая форма представления знания используется в области автоматического доказательства теорем, но понятно, что она может пригодиться и для других областей искусственного интеллекта.

Статья поступила в редакцию 2 июня 2008 г.

В данной работе решены проблемы создания системы, которая генерирует знания для систем автоматического доказательства теорем. Логическое знание подходит именно для тех систем автоматического доказательства, которые основаны на использовании знания.

Подход к доказательству теорем, основанный на знании, предусматривает методы, в которых в ходе доказательства новых теорем используются знания об уже доказанных теоремах. Такой подход несложно обосновать — новое знание всегда основано на накопленном старом. Однако в области машинного доказательства теорем до сих пор наибольшее внимание уделялось попыткам создания универсальных алгоритмов для доказательства любых теорем без первоначальных знаний, в первую очередь методом резолюции.

Во всех приложениях, как было показано многими авторами, знание играет фундаментальную роль. Но обычно речь идёт о конкретных для данной предметной области знаниях. Нам кажется важным рассмотреть вопрос о построении всех возможных знаний, минимально ограничивая предметную область.

Формализация предметной области — это множества алфавитов, которые означают понятия в конкретной области, и множества аксиом, которые дают характеристику понятий в ней. Возможно, у нас в начале нет полного знания о предметной области, и в процессе работы над предметной областью мы будем добавлять и исправлять множества аксиом. Также возможно развитие множества алфавитов. Но необходимо иметь некоторые аксиомы в самом начале. Знание — это теоремы о предметной области. Генератор знания состоит из генератора предложений (то есть, формул логики без свободных переменных) и системы доказательства теорем, которые пытаются выявить истинность предложений.

В подходе, основанном на знании, доказательство одной теоремы неотделимо от процесса накопления знаний в системе в целом. Система, в которой немного знаний (минимум — множества аксиом в теории, над которой мы работаем), не способна доказывать сложные теоремы. И каждая новая доказанная теорема может пригодиться при доказательстве других теорем. То есть, в каком-то смысле мы уделяем внимание не доказательству отдельных теорем, а развитию формальных теорий в целом. Сама по себе мысль о системах автоматического доказательства теорем не нова, но создаваемая нами система исключительна в том отношении, что у других существующих систем логическое знание не генерируется автоматическим образом.

Системы автоматического доказательства теорем доказывают леммы формальной логики либо в полностью автоматическом режиме, либо во взаимодействии с человеком, который управляет процессом. Автоматическое доказательство теорем — одна из старейших областей искусственного интеллекта, которая возникла ещё в 1960-х годах. В течение последних 20 лет методы автоматического доказательства теорем все больше применяли в производстве, в первую очередь в области верификации микросхем, но также время от времени в области верификации программного обеспечения.

Термин «автоматическое доказательство теорем, основанное на знании» (knowledge-based automatic theorem proving) появляется время от времени в литературе уже в течение 30 лет. Однако не существует определения того, что это означает в точности, и не существует общего согласия на то, в каких контекстах следовало бы использовать термин. Но поскольку значение термина довольно очевидно в интуитивном плане, разные люди, которые в итоге его использовали в независимости друг от друга, его понимали примерно одинаковым образом, то есть как системы автоматического доказательства теорем, которые кроме статистических алгоритмов также используют какое-нибудь динамическое знание.

Первым человеком, который представил подход, относящийся к знанию в области автоматического программирования, был Дэйвид Барстоу [8]. Автоматическое доказательство теорем — область, близкая к автоматическому программированию, и первыми, кто применял эти разработки в области автоматического доказательства теорем, стали Доминик Пастр [9,10] и Доналд Коухен [11]. В 1990-х гг. подход применяли Дж. Дензингер [12,13] и Шие Джу Ли [14]. В 2000-х годах направление появилось в основном как разработка конкретных приложений, самыми важными из которых стали системы автоматического доказательства теорем HOL [15], Isabelle [16], Muscadet [17] и ARGOS [18].

В существующих системах используется база данных из теорем, но эти теоремы создаются «вручную», они основаны на интуиции программиста или пользователя о том, какие результаты могут быть полезными при доказательстве новых теорем. То есть программист или пользователь должен обладать большим запасом знаний о предметной области, а кроме того он должен тратить довольно много времени, если приходится адаптировать систему для новой предметной области, в которой необходимо доказывать теоремы. Этих затрат можно было бы избежать, если система в состоянии самостоятельно создавать базы данных из теорем для каждой предметной области.

Проблема генерации логического знания эквивалентна проблеме нумерации всех синтаксически правильных предложений в заданном формальном языке [19]. Действительно, логическое знание — это некоторая совокупность предложений, и для поиска предложений, которые могут быть полезными, нам нужен способ полного перебора всех предложений. То есть в системе, которая автоматическим образом генерирует логическое знание, должен присутствовать генератор предложений, который должен удовлетворять следующим трём необходимым требованиям:

1) он должен генерировать замкнутые только предложения, т.е. правильные конструкции в заранее указанном синтаксисе;
2) он должен генерировать все возможные предложения, предоставляя возможность обеспечить полный перебор всех предложений;
3) он никогда не должен дважды генерировать одно и то же предложение.

Генератор, который удовлетворяет всем трём требованиям, мы назовём *правильным генератором*.

Естественно, сами предложения пока не являются знанием — знанием могут являться только верные предложения. То есть система должна совершить попытку доказать или опровергнуть каждое предложение, и те предложения, которые системе удалось доказать, сохранить в базе знаний.

2. Полуразрешимость логики предикатов первого порядка

Свойство полуразрешимости означает, что доказательство для некоторой совокупности предложений найдётся (рано или поздно), если оно вообще существует. Однако если доказательства не существует, не факт, что система сможет это обнаружить и прервать процесс доказательства.

В тех случаях, когда либо у теоремы, либо у его отрицания есть доказательство, мы можем обойти проблему возможного бесконечного процесса доказательства путём одновременной генерации предложения и его отрицания. Это не полное решение проблемы полуразрешимости, поскольку в неполных теориях теорема может быть верной, но не доказуемой в заданной аксиоматической системе. Однако данный метод помогает определить неверность леммы в тех случаях, когда доказательство обратного ему утверждения — не слишком затруднительно.

Таким образом генератор должен генерировать одновременно и предложение, и его отрицание. Теперь, чтобы требование 3) правильного генератора выполнялось, мы должны позаботиться о том, чтобы отрицание, которое мы генерировали, не было бы сгенерировано повторно. То есть мы должны разделить множества предложений Π на два непересекающихся множества Π_+ и Π_- с таким свойством, что у каждого предложения $\varphi \in \Pi_+$ есть эквивалентный его отрицанию элемент в множестве Π_-. Естественно, существует бесконечно много способов реализовать подобные разделения, но мы выберем один из них (рис. 1).

3. Инвариантные трансформации в множестве предложений

Предложение φ логики предикатов первого порядка находится в префикснормальной форме, если оно имеет вид

$$Q_0 x_0 Q_1 x_1 \dots Q_r x_r \psi,$$

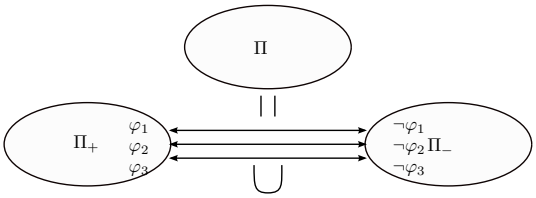

Рис. 1. **Разделение множества предложений**

где все $Q_0x_0Q_1x_1\dots Q_rx_r$ являются либо $\forall x_j$ или $\exists x_j$ для всех $1 \leqslant j \leqslant r$, и формула ψ не содержит кванторов. Можно доказать, что любому предложению логики предикатов первого порядка соответствует эквивалентное предложение в префикснормальной форме. Но префикснормальной формы предложения φ нам пока недостаточно — требуется предложение без отрицания.

Рассмотрим следующие эквивалентности с отрицаниями:

$$\begin{aligned}\neg\neg\varphi &\Leftrightarrow \varphi,\\ \neg(\varphi \vee \psi) &\Leftrightarrow (\neg\varphi \wedge \neg\psi),\\ \neg(\varphi \wedge \psi) &\Leftrightarrow (\neg\varphi \vee \neg\psi).\end{aligned}$$

С помощью этих предложений мы можем перемещать отрицания «вниз» по дереву, добиваясь того, чтобы отрицания появлялись только перед атомной формулой, в виде $\neg R_j^r(t_q,\dots,t_r)$, причём перед атомным выражением может быть не более одного отрицания. Теперь, если расширить множество предикатных символов $\{R_0^1, R_1^1, \dots, R_{p(1)}^1, R_0^2, \dots, R_0^m, \dots, R_{p(m)}^m\}$ символами $\{\tilde{R}_0^1, \tilde{R}_1^1, \dots, \tilde{R}_{p(1)}^1, \tilde{R}_0^2, \dots \tilde{R}_0^m, \dots, \tilde{R}_{p(m)}^m\}$, такими, что выполняется

$$\tilde{R}_j^r(x_1,\dots,x_r) \Leftrightarrow \neg R_j^r(x_1,\dots,x_r),$$

то можно заменить любую формулу $\neg R_j^r(t_q,\dots,t_r)$ атомной формулой $\tilde{R}_j^r(t_q,\dots,t_r)$ и в итоге удалить все отрицания из наших формул.

Таким образом, мы можем ограничить наши исследования в формулах формы

$$Q_0x_0Q_1x_1\dots Q_rx_r\psi,$$

где в ψ нет ни кванторов, ни отрицания. Можно также потребовать, чтобы все переменные $x_0, x_1, \dots, x_r$ также присутствовали в формуле ψ, если бы они не присутствовали, можно просто удалить их из имеющихся кванторов и перенумеровать несвободные переменные без изменения значений предложений.

Теперь рассмотрим, какая атомная формула в крайней слева формуле ψ. Если это формула формы $\tilde{R}_j^r(\dots)$, то определяем $\psi \in \Pi_-$, иначе $\psi \in \Pi_+$. Очевидно, что это разделение выполняет наши требования.

Таким образом, генератор порождает предложения множества Pi_+, то есть множества предложений в префикснормальном форме без отрицания, в котором крайняя слева атомная формула не имеет вида $\tilde{R}_j^r(\dots)$. Ради простоты мы также допускаем, что в наших предложениях нет импликации — их можно удалить с помощью тождества:

$$\varphi \Rightarrow \psi \Leftrightarrow \neg\varphi \vee \psi.$$

Теперь очень просто генерировать отрицание предложения — можно просто заменять все кванторы всеобщности $\forall$ на кванторы существования $\exists$ и наоборот, все дизъюнкции $\vee$ на конъюнкции $\wedge$ и наоборот, и все атомные формулы формы $\tilde{R}_j^r(\dots)$ на атомные формулы формы $R_j^r(\dots)$ и наоборот. Очевидно, что если $\psi \in \Pi_+$, то для формулы φ, которую мы получили путём подобного отрицания, верно, что $\varphi \in \Pi_-$.

4. Рекурсивная нумерация множеств

Рассмотрим сначала теорию нумерации, с помощью которой мы можем формализовать задачи полного перебора и создание генератора предложений в частности.

В аксиоматической теории множеств доказано, что у каждого множества s имеется изоморфный кардинал $|s|$ (рис. 2). Соответственно, если дано конечное или счётное множество s, то каждому натуральному числу $i < |s|$ соответствует единственный элемент $x \in s$

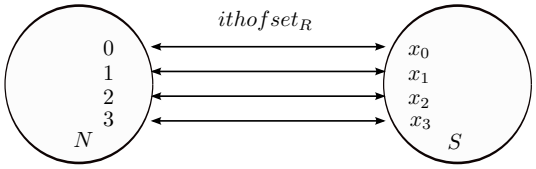

Рис. 2. **Изоморфный кардинал множества**

Обозначим через $ithofset(i, s) = x$, определённый i-й элемент множества s. Если фиксировано s, то это — функция одной переменной $x = ithofset_s(i)$.

Данная функция $ithofset_s$ всегда существует — когда их имеется несколько (как в случае, когда $|s| > 1$), то можно ограничиться одной из них. Но простого знания, что функция существует, нам недостаточно – мы должны уметь вычислять их, то есть найти рекурсивную конструкцию, которая вычислит функцию $ithofset_s$.

Рекурсивную конструкцию, вычисляющую $ithofset_s(i)$, обозначим через ithofset_s и назовём «рекурсивной нумерацией множества s». Проблема нумерации — это проблема нахождения рекурсивной конструкции ithofset_s, если известен способ построения множества s. Мы используем обозначения $\text{ithofset}(i, s) = \text{ithofset}_s(i)$ для рекурсивной нумерации произвольного множества s, в случае, когда она определена.

Если удалось найти нумерацию множества s, то можно осуществить полный перебор множества s, проверяя все элементы

$$\begin{aligned} &\text{ithofset}_s(0),\\ &\text{ithofset}_s(1),\\ &\text{ithofset}_s(2),\\ &\dots \end{aligned}$$

в порядке их нумерации.

Если удалось найти нумерацию множества всех предложений ithofset_Π, то она также будет генератором предложений, который удовлетворяет всем трём обязательным требованиям, указанным выше, то есть является правильным генератором.

Теория нумерации обеспечивает нас теоремами, по которым мы можем построить нумерацию множества S ithofset_S, исходя из составляющих множества S. Например, если $S = T \times U$, и T и U являются счётными, мы можем определить

$$\text{ithofset}(i, S) = \langle \text{ithofset}(f(i), T), \text{ithofset}(g(i), U) \rangle,$$

где

$$a(x) = \frac{-1 + \sqrt{1 + 8x}}{2},$$
$$f(x) = \lfloor (a(x)(1 + \lfloor a(x) \rfloor - a(x)) \rfloor, \quad g(x) = \lceil a(x)(a(x) - \lfloor a(x) \rfloor) \rceil$$

и так как $h(x, y) = \frac{(x+y)^2+x+3y}{2}$ является биекцией $h : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ и f и g являются проекциями её обратной функции, $h(f(i), g(i)) = i$.

Для нахождения нумерации всех положительных предложений Π_+ нам необходимо представить множество с помощью таких операций теории множества как $\times$ (картезианское произведение), $\sqcup$ (объединения без пересечений) и т.д. Такая конструкция слишком сложна, чтобы представить её в рамках данной статьи, но она подробно изложена в дипломной работе автора.

Возможно найти рекурсивную нумерацию множества формул, множества предложений и множества предложений в префикснормальной форме без отрицания, но невозможно найти рекурсивную нумерацию множества теорем (то есть, множества верных предложений). Означает ли это, что наше знание носит только синтаксический, не семантический смысл? Не совсем. Естественно, часть предложений, которые генерируются, являются тривиальными, простыми следствиями от других предложений. В будущем нам ещё предстоит работа по поиску эвристик, которые позволят отбросить «тривиальные» формулы. Но сами условия, которые мы сформулировали выше для генератора, уже позволяют избежать большей части тавтологий — по крайней мере, можно быть уверенными, что ни одно предложение не повторяется, поскольку все предложения в префикснормальном виде без отрицаний и тривиально эквивалентные им формулы, которые получаются из них путём инвариантных трансформаций с кванторами и отрицаниями, в ходе генерации не появятся.

Рассмотрим в качестве примера аксиоматическую теорию множества ZFC и арифметики Пеано. Вначале необходимо выбрать для них однозначный алфавит. На самом деле, поскольку процесс генерации не учитывает аксиоматику, единственное что нас волнует, это выбор логических связок, предикатных символов и функциональных символов. Константные символы мы считаем функциональными символами нулевого порядка. Предикатные символы и связки нулевого порядка означают логические константы типа $\bot$, то есть их количество должна совпадать.

Таким образом алфавит полностью определяется тремя массивами произвольной размерности со значениями из натуральных чисел — значение $c[i]$ первого массива c показывает сколько в алфавите i-местных логических коннективов, значение $p[i]$ второго массива p показывает сколько в алфавите i-местных предикатных символов, и значение $f[i]$ третьего массива f показывает сколько в алфавите i-местных функциональных символов.

Для удачного разделения множества предложений нам требуется множество логических связок $\{\vee, \wedge\}$, то есть $c = [0, 0, 2]$. В случае аксиоматической теории множества ZFC у нас нет функциональных символов, а множество предикатов равно $\{=, \in\}$, которое мы расширяем в $\{=, \in, \neq, \notin\}$, то есть $f = [0]$, $p = [0,\ 0, 4]$. В случае арифметики Пеано, у нас единственный предикатный символ $=$, но множество предикатов мы расширяем в $\{=, \neq\}$, то есть $p = [0,\ 0, 2]$. У нас есть константы 0,1 и функции $+$, $\times$, то есть множества функции $f = [2,\ 0, 2]$.

Первые 10 положительных предложений, которые генерируются в аксиоматической теории множества ZFC:

$\forall x_0(x_0 = x_0)$	неверно,	$\exists x_0 \forall x_1(x_0 = x_1)$	неверно,
$\exists x_0(x_0 = x_0)$	верно,	$\forall x_0 \exists x_1(x_0 = x_1)$	верно,
$\forall x_0(x_0 \in x_0)$	неверно,	$\exists x_0 \exists x_1(x_0 = x_1)$	верно,
$\exists x_0(x_0 \in x_0)$	неверно,	$\forall x_0 \forall x_1(x_0 \in x_1)$	неверно,
$\forall x_0 \forall x_1(x_0 = x_1)$	неверно,	$\exists x_0 \forall x_1(x_0 \in x_1)$	неверно.

Первые 10 положительных предложений, которые генерируются в арифметике Пеано:

$0 = 0$	верно,	$\exists x_0(x_0 = x_0)$	верно,
$0 = 1$	неверно,	$\forall x_0(x_0 = 0)$	неверно,
$1 = 0$	неверно,	$\exists x_0(x_0 = 0)$	верно,
$1 = 1$	верно,	$\forall x_0(0 = x_0)$	неверно,
$\forall x_0(x_0 = x_0)$	неверно,	$\exists x_0(0 = x_0)$	верно.

5. Система доказательства теорем с автоматической генерацией знаний

Необходимо, чтобы компонент автоматического доказательства теорем не был полностью автоматическим, а был бы полуавтоматическим, поскольку практика показывает, что в полностью автоматических системах доказательства пока не удаётся получить хорошие результаты для достаточно широкого класса задач. Все системы, которые широко используются в приложениях, являются именно полуавтоматическими, т.е. пользователь направляет процесс в «правильном направлении» в самых общих чертах, а компьютер разбивает задачи на подзадачи и «работает конвейером» во множестве наиболее простых подзадач.

В системе должен обязательно присутствовать компонент «разделитель», который будет отделять отрицательные формулы от эквивалентных положительных и отправлять их на разные процессы доказательства. Разделитель должен также уметь собирать результаты доказательств и прерывать второй процесс при успешном завершении первого. То есть часть системы должна выглядеть следующим образом (рис. 3):

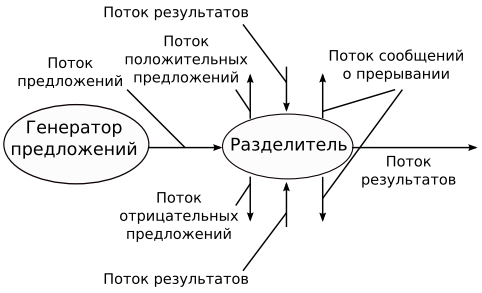

Рис. 3. **Часть системы: Разделитель**

Что означает «неопределённое предложение»? Как правило, все алгоритмы доказательства теорем имеют экспоненциальную сложность. Если невозможно распределить любое количество системных ресурсов в процессе доказательства, которое может длиться часами (или годами), возможно определить максимальное время, после которого процесс прерывается и предложения загружаются в базы данных предложений (вместе с предложением, эквивалентным его отрицанию) с пометкой «неопределённое предложение».

Также необходимо, чтобы наша система выполняла параллельно не только одно предложение и его отрицание, а также разные предложения одновременно. То есть человек может заниматься одной проблемой, которую компьютеру не удалось решить, и компьютер может одновременно стараться доказать множество других предложений (вместе с отрицаниями). Параллелизм задаёт синергию, в которой развитие в одном параллельном процессе может помогать и в остальных процессах, которые находятся в действии. И в настоящей системе, основанной на накоплении знаний, не только компьютер может помогать человеку, но и «человек может помогать компьютеру» — компьютер может исследовать все процессы, которые удалось завершить только человеку, и обобщать новые методы решения, чтобы компьютер справлялся самостоятельно в следующий раз.

Таким образом, в системе должно быть разветвление к разным, параллельным процессам, и компонент «распределитель», который будет управлять распределением задач по разным подпроцессам. Распределитель должен знать, какой из процессов занят и какой доказывает какие предложения. Распределитель должен уметь передать процессу команду о прерывании, которую ему отправил разделитель в случае удачного завершения доказательства предложения, эквивалентного его отрицанию. Также, поскольку система должна состоять из независимых компонентов, и сама система должна быть независима от системы доказательства теорем, над каждым процессом доказательства должен быть ещё отдельный контроллер, который переводит задачи и сообщения о прерывании системы в язык,

понятный компоненту автоматического доказательства теорем. То есть в системе должны присутствовать следующие компоненты (рис. 4):

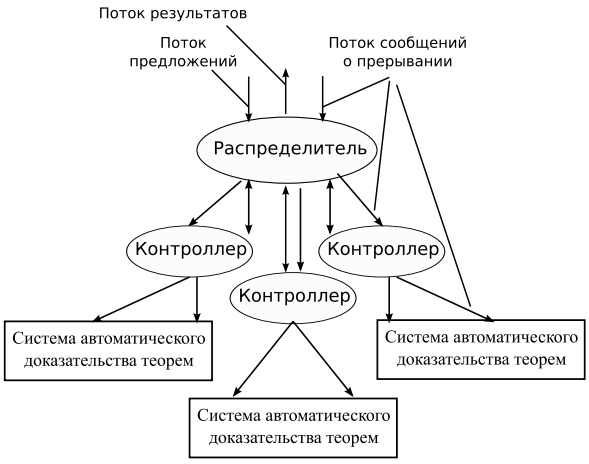

Рис. 4. **Часть системы: Распределитель**

На самом деле, требуется два распределителя: один для отрицательных, другой — для положительных предложений. Оба распределителя могут наблюдать за таким количеством контроллеров, какое допускают системные ресурсы.

То есть в целом архитектура нашей системы в упрощённом виде следующая (рис. 5):

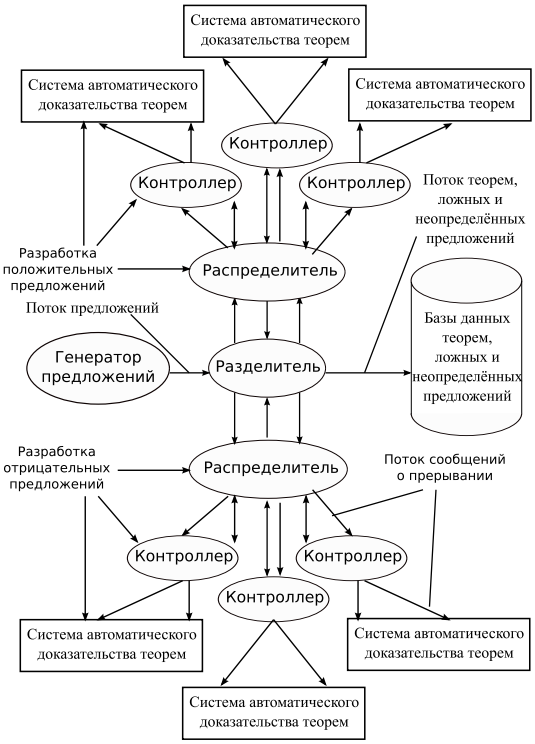

Рис. 5. **Архитектура системы в упрощённом виде**

6. Реализация системы доказательства теорем с автоматической генерацией знаний

Система реализована в операционной системе GNU/Linux, потому что она предоставляет широкий набор возможностей для межпроцессных взаимодействий, и наибольшее число существующих систем машинного доказательства теорем построено в UNIX-подобных системах. Можно менять компонент автоматического доказательства теорем системы, без нарушения системной архитектуры в целом, и UNIX — подобные платформы предоставляют наибольший набор возможных систем доказательства теорем, которых мы можем при желании включить в нашу систему.

Все компоненты системы являются асинхронными процессами в самой операционной системе, чтобы архитектура системы была как можно более независимой от архитектуры компонентов. Из широкого набора возможностей взаимодействий между процессами в UNIX-подобных системах (сигналы, каналы, разделяемая память, терминалы, сокеты,. . .) нами были выбраны сокеты. Преимущества сокетов состоят в том, что они допускают нахождение процессов даже в разных устройствах, и они реализованы в широком наборе языков программировании, в том числе в диалекте языка LISP elisp, которым мы будем пользоваться.

Сообщения содержат информацию о получателе, об отправителе, о номере задачи, о самой задаче и возможные дополнительные сведения.

Для построения генератора нам прежде всего потребуется функциональное программирование. С этой целью используется диалект elisp языка LISP, поскольку текстовый редактор emacs, который является почти стандартным в системах семейства UNIX, всегда содержит интерпретатор elisp, и elisp содержит элементарные функции для создания сокетов в среде UNIX.

В качестве системы доказательства теорем можно выбрать любую, которая действует под Linux, но для данной работы выбрана система Isabelle. Она позволяет использовать множество разных логик. Она использует слабую теорию типов в качестве металогики, и реализует разные системы логики, в том числе аксиоматическую теорию множеств ZFC, в которой предложения генерируемы [16, 20]. Isabelle может действовать интерактивно и в автоматическом режиме. Главный метод доказательства теорем в системе Isabelle это резолюция для логики предикатов высшего порядка, но в ней есть также схемы переписывания термов и система автоматического доказательства методами аналитических таблиц. Преимущества Isabelle с точки зрения подхода, основанного на накоплении знаний, в том, что в ней есть вложенная концепция «теории», которая поддерживает идею о постепенно расширяющихся знаниях. Предшественник системы Isabelle, HOL является одной из наиболее популярных систем в области верификации в различных приложениях. В самой системе Isabelle доказано немало нетривиальных математических теорем (теорема неполноты Геделя, теорема Геделя о непротиворечивости аксиомы выбора), и также ряд прикладных задач (о корректности протокола безопасности и свойств семантики языков программирования).

7. Заключение

В данном исследовании был решён ряд вопросов, связанных с созданием системы искусственного интеллекта с генерацией логического знания. Был продемонстрирован способ обходить проблему полуразрешаемости логики в ряде случаев, и показан математический принцип создания генератора, который выполняет ряд простых, но нужных условий. Были описаны требуемые компоненты системы и в качестве примера приведена система автоматического доказательства теорем, которая использует принцип автоматической генерации знаний. Решена проблема, связанная с требованием асинхронной работы системы — архитектура каждого компонента не зависит от других компонентов.

В заключение выражаю глубокую благодарность моему научному руководителю проф. Стефанюку В.Л. за активную поддержку и помощь в развитии представленных в данной работе идей.

Литература

1. Solving Checkers / J. Schaeffer, Y. Bjornsson, N. Burch et al // International Joint Conference on Artificial Intelligence (IJCAI). — 2005. — Pp. 292–297.
2. *Bouzy B., Cazenave T.* Computer Go: An AI Oriented Survey // Artificial Intelligence. — Vol. 132, No 1. — 2001.
3. *Korf R.* Finding Optimal Solutions to Rubik's Cube Using Pattern Databases // Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97). — Providence, RI: July 2005.
4. *Eich H. P., Ohmann C., Lang K.* Decision Support in Acute Abdominal Pain Using an Expert System Fordifferent Knowledge Bases // Tenth IEEE Symposium on Computer-Based Medical Systems, 1997. Proceedings. — 11-13 June 1997.
5. Toward Automating Airspace Management / G. Taylor, B. Stensrud, S. Eitelman et al // Computational Intelligence for Security and Defense Applications (CISDA). — Honolulu, HI: IEEE Press, 2007.
6. Validation of an Automated System Model Generator / A. J. Gonzalez, H. R. Myler, F. D. McKenzie et al // IEEE Transactions on Knowledge and Data Engineering. — Vol. 6, No 4. — August 1994.
7. *Стефанюк В. Л.* Локальная организация систем — модели и приложения. — М.: Мир, 2004.
8. *Barstow D. R.* A Knowledge-Based System for Automatic Program Construction // IJCAI. — 1977. — Pp. 382–388.
9. *Pastre D.* A Humanlike Approach for Automatic Theorem Proving // AISB/GI (ECAI). — 1978. — Pp. 248–252.
10. *Pastre D.* Knowledge-Based Theorem Proving // GI Jahrestagung. — 1980. — P. 429.
11. *Cohen D. N.* Knowledge Based Theorem Proving and Learning. — Ann Arbor: UMI, 1981.
12. *Denzinger J., Schulz S.* Recording and Analyzing Knowledge-Based Distributed Deduction Processes // Journal of Symbolic Computation. — Vol. 21. — 1996. — Pp. 523–541.
13. *Denzinger J., Kronenburg M., Schulz S.* DISCOUNT. A Distributed and Learning Equational Prover // Journal of Automated Reasoning. — Vol. 18, No 2. — August 1994. — Pp. 189–198.
14. *Lee S. J.* An Autonomous Multistrategy Theorem Proving System Using Knowledge-Based Techniques // Journal of Intelligent Information Systems. — Vol. 3, No 1. — August 1994.
15. *Gordon M.* From LCF to HOL: a Short History. — Cambridge University, 1996.
16. *Wentzel M. M.* Isabelle/Isar — a Versatile Environment for Human-Readable Formal Proof Documents. — Munchen: Institut for Informatik der Technischen, Universitet Munchen, 2001.
17. *Pastre D.* Muscadet2.3 : A Knowledge-based Theorem Prover based on Natural Deduction // International Joint Conference on Automated Reasoning IJCAR (Conference on Automated Deduction CADE-JC). — 2001. — Pp. 685–689.
18. *Pagnol J.-P.* Modelisation and Automation of Reasoning in Geometry. The ARGOS System: a Learning Companion for High-School Pupils // Proceedings of the 6th International Conference, ITS 2002. — Springer Verlag, 2002.
19. *Раутиаинен А.* Алгоритм нумерации объединения конечных множеств без пересечении // XLII Всероссийская конференция по проблемам математики, информатики, физики и химии 17-21 апреля 2006 года: Труды конференции. Секция «Программные исследования». — М.: РУДН, 2006. — С. 122–130.
20. *Paulson L. C.* The Isabelle Reference Manual. — Cambridge, United Kingdom: Computer Laboratory, University of Cambridge, 2003.

UDC 004.832

Automatic Generation of Logical Knowledge

A . Rautiainen

Information Technologies Department
Peoples' Friendship University of Russia
6, Miklukho-Maklaya str., Moscow, Russia, 117198

We study problems which arise deriving generating automatically logical knowledge in systems of artificial intellect, first of all in systems of automatic theorem proving. Three necessary conditions for a generator of logical knowledge are proposed and a verification of these ones is presented.