1971
Общность в системах искусственного интеллекта

Джон Маккарти

Станфордский университет

Лекция, прочитанная Джоном Маккарти по случаю получения премии Тьюринга в 1971 г., никогда не издавалась. В публикуемом постскриптуме к лекции, написанном в 1986 г., автор пытается сохранить особенности и дух оригинала, а также прокомментировать его в свете достижений последних пятнадцати лет.

ПОСТСКРИПТУМ

Моя Тьюринговская лекция 1971 г. называлась «Общность в системах искусственного интеллекта». Название темы оказалось слишком амбициозным, так как я обнаружил, что в то время не мог удовлетворительно представить в письменном виде мои мысли по этому поводу. Было бы лучше сделать обзор предшествующих работ, а не пытаться предлагать что-то новое, но тогда это не входило в мои привычки.

Я благодарен Ассоциации вычислительных машин за предоставленную возможность попробовать еще раз. К несчастью для нашей области науки, но, может быть, к счастью для этой статьи, проблема общности в задачах искусственного интеллекта осталась почти столь же нерешенной, как и тогда, хотя сейчас у нас есть много новых идей, которых не было в 1971 г. Статья в большой степени основывается на этих идеях, но она ни в коей мере не является полным обзором существующих к 1986 г. подходов к достижению общности. Место, отведенное каждой из обсуждаемых идей, пропорционально скорее степени моего знакомства с этой идеей и не отвечает какому-либо объективному критерию.

В 1971 г. и даже еще в 1958 г. было очевидно, что системы искусственного интеллекта страдают отсутствием общности. Этот факт продолжает быть очевидным и в настоящее время, и эта очевидность обрастает все более детальными подробностями. Первый явный симптом состоит в том, что небольшое добавление к идее программы часто приводит к переписыванию заново всего начала со структурами данных. Некоторый прогресс был достигнут с появлением модульности структур данных, но по-прежнему невозможно, не переписывая программы, немного изменить стратегию поиска.

Другим симптомом является тот факт, что никто не знает, как создать базу данных, содержащую общеполезные знания об окружающем мире (знания «здравого смысла»), которую могла бы использовать любая программа, нуждающаяся в этих знаниях. Вместе с другой информацией такая база данных содержала бы необходимые роботу знания о взаимодействии движущихся объектов в окружающей среде, знания о том, что человек, как правило, знает о своей семье, и факты, касающиеся купли-продажи. Это не зависит от того, выражены ли знания на логическом языке или с помощью какого-то другого формализма. Когда мы используем логический подход к искусственному интеллекту, недостаток общности проявляется в том, что аксиомы, которые мы создаем для выражения знаний «здравого смысла», оказываются слишком ограниченными, чтобы областью их применимости была вся общая база знаний. По моему мнению, выработка языка выражения общих знаний «здравого смысла» для включения их в универсальную базу данных — это ключ к достижению общности в системах искусственного интеллекта.

Здесь представлены некоторые идеи о том, как достигнуть общности, высказанные как до, так и после 1971 г. Я хотел бы еще раз повторить свое отречение от всеобъемлемости обзора.

ПРЕДСТАВЛЕНИЕ ПОВЕДЕНИЯ С ПОМОЩЬЮ ПРОГРАММ

Фридберг [7, 8] изучал абсолютно общий подход к представлению поведения и предусмотрел способ обучения для его совершенствования. А именно, поведение представляется с помощью программы, а обучение происходит с помощью случайных модификаций программы и тестирования модифицированной программы. Подход Фридберга оправдал себя только при обучении тому, как перенести один бит информации из одной ячейки памяти в другую, а схема вознаграждения инструкций, вошедших в успешные проходы программы, путем уменьшения вероятности их модификаций, как было показано Саймоном [24], уступает способу тщательного тестирования каждой программы и отбрасыванию любой из них, если она оказалась неудовлетворительной. Похоже, никто не пытался следовать идее обучения при помощи изменения всей программы целиком.

Недостаток подхода Фридберга состоит в том, что, хотя представление поведения с помощью программ является вполне общим понятием, изменение поведения малыми модификациями программы сильно зависит от конкретной ситуации. Небольшие концептуальные изменения поведения обычно нельзя

представить с помощью малых изменений, внесенных в программу, особенно если это программа, написанная на машинном языке, и любая малая модификация текста программы одинаково вероятна.

Может быть, стоило бы попробовать что-нибудь более похожее на генетическую эволюцию, например дублировать подпрограммы, одни копии модифицировать, а другие оставлять неизменными. Обучающаяся система экспериментировала бы: а не лучше ли заменить некоторые вызовы начальных подпрограмм вызовами модифицированных подпрограмм? Скорее всего, этот подход также не будет работать, за исключением тех случаев, когда малые изменения поведения действительно можно получить с помощью небольших модификаций подпрограмм. Может быть, потребовалось бы зарезервировать для модификаций некоторое количество параметров подпрограмм.

Хотя Фридберг занимался задачей обучения на основании опыта, все способы представления знаний с помощью программ наталкиваются на аналогичные трудности, коль скоро приходится комбинировать несопоставимые знания или создавать программы, модифицирующие знания.

УНИВЕРСАЛЬНЫЙ РЕШАТЕЛЬ ЗАДАЧ И ЕГО НАСЛЕДНИК

Один из видов общности в искусственном интеллекте включает в себя методы нахождения решений, не зависящие от проблемной области. Аллен Ньюэлл, Херберт Саймон и их коллеги и студенты были первыми в разработке этого подхода и продолжают им заниматься.

Ньюэлл и его коллеги впервые представили универсальный решатель задач (GPS)[1] на суд ученых в 1957 г. [18]. Исходной идеей являлось представление задач из некоего общего класса как задач преобразования одного выражения в другое при помощи множества допустимых правил. В работе [20] даже утверждалось, что развитие GPS может само мыслиться как задача этого класса. По моему мнению, GPS не мог быть действительно универсальным решателем задач, потому что в общем случае задачи нельзя представить в таком виде и потому что большинство знаний, необходимых при решении задач и достижении целей, не так-то просто представить в виде правил преобразования выражений. Но все же GPS был первой системой, в которой решающие задачу структуры целей и подцелей были отделены от конкретной области задач.

---

[1] General Problem Solver.

Если бы оказалось, что универсальный решатель задач действительно универсален, то, возможно, выполнились бы предсказания Ньюэлла и Саймона о быстром успешном развитии систем искусственного интеллекта. В настоящее время Ньюэлл предлагает для универсального представления задач использовать систему SOAR, которая, насколько я понимаю, работает с преобразованием одного состояния в другое, причем состояния не обязательно должны быть описаны с помощью выражений.

ПРОДУКЦИОННЫЕ СИСТЕМЫ

Первые продукционные системы были созданы Ньюэллом и Саймоном в 1950-х гг., а их идея была впервые описана в работе [21]. Некоторая общность достигается использованием общего механизма целенаправленного поиска для всех типов задач, а изменяются только конкретные продукции. Ранние продукционные системы превратились в оболочки экспертных систем, широко распространенные в настоящее время.

Системы продукций представляют знания в виде фактов и правил, и почти всегда правила сильно синтаксически различаются между собой. Факты обычно соответствуют базисным примерам логических формул, т. е. символам предикатов, примененным к постоянным выражениям. В отличие от систем, основанных на логике, эти факты не содержат переменных или кванторов. Новые факты являются результатом вывода, наблюдения или вводятся пользователем. Переменные зарезервированы для правил, которые обычно имеют вид «образец — действие» (действия по эталону). Правила вводятся в систему программистом в области инженерии знаний и в большинстве систем не могут возникать в результате работы самой системы. В обмен на принятие этих ограничений создатель системы продукций получает относительно быстродействующую программу.

Системы продукций редко используют фундаментальные знания из тех областей, для которых они созданы. Например, в медицинской системе MYCIN [2] имеется много правил, как, исходя из симптомов и результатов лабораторных анализов определить, какая именно бактерия является причиной заболевания. Однако в ее формализме нет способа описать тот факт, что бактерии — это организмы, которые живут внутри тела человека. Кроме того, в системе MYCIN нет способа представления процессов, протекающих во времени, хотя в других системах продукций имеется возможность представлять процессы приблизительно на уровне ситуационного исчисления, которое будет описано в следующем разделе.

В системе продукций результатом сопоставления с эталоном является замена аргументов в эталонной части правила константами. Следовательно, система продукций не выводит общих суждений. Например, рассмотрим определение: контейнер стерилен, если он запечатан и бактерии не могут в него попасть, а все бактерии, находящиеся внутри, мертвы. Система продукций (или логическая программа) может воспользоваться этим утверждением, только подставляя различные виды бактерий вместо переменных. Следовательно, она не может сделать вывод, что нагревание запечатанного контейнера приведет к его стерилизации, если известно, что бактерии погибают при нагревании, потому что она не может рассуждать о неопределенном множестве бактерий в контейнере. Дальнейшее обсуждение этих идей приведено в работе [14].

ПРЕДСТАВЛЕНИЕ ЗНАНИЙ НА ЯЗЫКЕ ЛОГИКИ

В 1958 г. мне казалось, что малые модификации поведения в большинстве случаев можно описать как малые изменения представлений о мире, и для этого необходима система, точно отражающая эти представления.

> Если вы хотите, чтобы машина могла выводить абстракции, скорее всего, это значит, что она должна уметь представлять эти абстракции некоторым достаточно простым способом [11, стр. 78].

В 1960 г. возникла идея увеличения общности, которая заключается в том, чтобы воспользоваться логикой для такого описания фактов, которое не зависело бы от того, как эти факты будут использоваться впоследствии. Тогда мне казалось (как, впрочем, и сейчас), что люди по объективным причинам предпочитают общаться с помощью декларативных предложений, а не языков программирования, все равно, является ли субъект общения человеком, существом с Альфа Центавра или компьютерной программой. Более того, и для внутреннего представления проявляются преимущества декларативной информации. Основным преимуществом декларативной информации является общность. Факт, что при столкновении двух объектов они производят шум, может быть использован в различных конкретных ситуациях: произвести шум, избежать шума, объяснить шум или объяснить отсутствие шума. (Я думаю, что те машины не столкнулись, потому что, услышав скрип тормозов, я не услышал звука от удара.)

При построении системы искусственного интеллекта с декларативным представлением информации нужно решить, какой тип декларативного языка использовать. В простейших системах допускается только применение постоянных предика-

тов к постоянным символам, например *on*(*Block1*, *Block2*). Далее можно допустить применение любых термов, построенных из функциональных символов и символов предикатов, например *location*(*Block1*)-*top* (*Block2*). Базы данных, написанные на Прологе, допускают произвольные хорновские выражения, содержащие и несвязанные переменные; например, $P(x, y) \wedge Q(y, z) \supset R(x, z)$ — это прологовское выражение в обозначениях стандартной логики. Более высокий уровень — это логика первого порядка, в которую входят как кванторы существования, так и кванторы общности и произвольные формулы первого порядка. В рамках логики первого порядка выразительна сила языка зависит от того, каким классам объектов принадлежат переменные. Довольно значительную выразительную силу дает использование теории множеств, которая позволяет работать с выражениями для множеств любых объектов.

За любое увеличение выразительной силы приходится платить требуемой сложностью программ, осуществляющих рассуждения и решающих задачи. Другими словами, ограничение выразительности декларативной информации позволяет упростить процедуру поиска. Пролог представляет собой локальный оптимум в этом континууме, потому что хорновские выражения обладают средней выразительностью, но легко интерпретируются логическим решателем задач.

Одно из основных ограничений, которое обычно принимается, состоит в требовании, чтобы при выведении новых фактов таковыми являлись только формулы без переменных, т. е. в требовании производить рассуждения в высказываниях, подставляя вместо переменных константы. Оказывается, что повседневная жизнь человека по большей части сопровождается именно такими рассуждениями. В принципе Пролог идет дальше этого, потому что выражения, которые прологовские программы находят как значения переменных, могут сами содержать несвязанные переменные. Однако эта возможность редко используется, кроме как для получения промежуточных результатов.

Операция, которая невозможна без привлечения исчисления предикатов в большей степени, чем позволяет Пролог, — это универсальное обобщение. Рассмотрим логическое обоснование консервирования. Мы говорим, что контейнер стерилен, если он запечатан и все бактерии в нем погибли. Это можно написать в виде следующего фрагмента прологовской программы:

sterile(*X*) :-*sealed*(*X*), *not alive-bacterium*(*Y*, *X*).

alive-bacterium(*Y*, *X*) :-*in*(*Y*, *X*), *bacterium*(*Y*), *alive*(*Y*).

Однако прологовская программа, в которую входит этот фрагмент, может простерилизовать контейнер, только убивая персонально каждую бактерию, и при этом она будет требовать, чтобы какая-нибудь другая часть программы последовательно генерировала их имена. Ее нельзя использовать для того, чтобы логически обосновать процесс консервации: запечатывание контейнера и затем его нагревание, чтобы убить все бактерии сразу. Для рассуждений, приводящих к логическому обоснованию консервирования, необходимо использовать кванторы.

Я считаю, что программы, осуществляющие рассуждения и решение задач, должны в конечном счете допустить неограниченное использование кванторов и множеств и иметь достаточно строгие методы контроля, чтобы избежать комбинаторного взрыва.

Хотя идея, выдвинутая в 1958 г., была принята очень хорошо, в те годы было сделано очень мало попыток воплотить ее в программу. Основная из них — это диссертация Блэка на соискание степени доктора философии, защищенная в Гарварде в 1964 г. Сам я большую часть времени потратил на то, что считал предварительными проектами; главным образом это был LISP. Я хотел вначале понять, как знания здравого смысла можно выразить на языке логики, и это основная причина того, что я не стал сразу пытаться реализовать эту идею. Представление знаний здравого смысла на логическом языке и сейчас является моей целью. Мое стремление продолжать работу в этом направлении могло бы и поколебаться, если бы люди, развивающие нелогические подходы, достигли серьезных успехов в обеспечении общности.

Маккарти и Хэйес [12] стали различать эпистемологические и эвристические аспекты задач искусственного интеллекта и предположили, что изучать проблемы общности гораздо проще в рамках эпистемологического подхода. Разница состоит в том, что при эпистемологическом подходе требуется полный набор фактов, гарантирующий, что некоторая стратегия достигает цели, в то время как эвристический подход предполагает поиск приемлемой стратегии исходя из наличных фактов.

Идеей работы [11] было создание базы данных «здравого смысла» общего назначения. Информацию «здравого смысла», имеющуюся у людей, предполагалось записать в логической форме и включить в базу данных. Любая программа целенаправленного поиска могла бы обратиться к ней за фактами, необходимыми для того, чтобы решить, как достигнуть поставленной цели. Наиболее значимыми фактами базы данных должны были быть факты о результатах действий робота, пытающегося перемещать объекты с одного места на другое. Изу-

чение этой проблемы привело к созданию в 1960 г. *исчисления ситуаций* [12], целью которого было найти способ описания результатов действий (операций) вне зависимости от проблемной области.

Основной формализм ситуационного исчисления — это выражение вида

$$s' = result(e, s),$$

где s' — ситуация, возникающая, когда происходит событие e в ситуации s. Ниже приведены некоторые аксиомы ситуационного исчисления для перемещения и раскрашивания блоков.

Аксиомы результатов действий

$$\forall\, x\ l\ s.\ clear(top(x), s) \wedge clear(l, s) \wedge \neg\, tooheavy(x) \supset loc(x, result(move(x, l), s)) = l.$$

$$\forall\, x\ c\ s.\ color(x, result(paint(x, c), s)) = c.$$

Аксиомы фреймов

$$\forall\, x\ y\ l\ s.\ color(y, result(move(x, l), s)) = color(y, s).$$

$$\forall\, x\ y\ l\ s.\ y \neq x \supset loc(y, result(move(x, l), s)) = loc(y, s).$$

$$\forall\, x\ y\ c\ s.\ loc(x, result(paint(y, c), s)) = loc(x, c).$$

$$\forall\, x\ y\ c\ s.\ y \neq x \supset color(x, result(paint(y, c), s)) = color(x, s).$$

Заметим, что во все описания исполнения действий посылки входят в явном виде и что утверждения (называемые аксиомами фреймов) о том, что не изменяется при выполнении того или иного действия, также выписаны в явном виде. Без этих утверждений было бы невозможно что-либо сказать о выражении *result(e2, result(e1, s))*, так как мы не знали бы, выполнены ли в выражении *result(e1, s)* посылки для того, чтобы событие *e2* имело ожидаемый результат.

Заметим также, что ситуационное вычисление применимо только в том случае, когда рассуждения о дискретных событиях, результатом каждого из которых является новая общая ситуация, имеют смысл. Непрерывные события и события, происходящие одновременно, теорией не охватываются.

Оказалось, что, к сожалению, практически невозможно использовать ситуационное исчисление предложенным выше способом даже для довольно ограниченных задач. Использование универсальных программ для доказательства теорем приводило к слишком медленной работе программы, потому что в 1969 г. программы для доказательства теорем [9] не имели средств управления поиском. Все это привело к созданию системы STRIPS [6], в которой используются только логические рас-

суждения в рамках конкретной ситуации. Формализм системы STRIPS был более ограниченным, чем исчисление ситуаций в полном объеме. Чтобы не ввести противоречия, необходимо было аккуратно выбирать факты, входившие в число аксиом. Эти противоречия могли возникать при невозможности удалить высказывание, которое не являлось бы истинным в результате происшедшей в результате действия ситуации.

НЕМОНОТОННОСТЬ

Вторая проблема, связанная с аксиомами ситуационного исчисления, состоит в том, что они также не обладают достаточной общностью. Это *проблема уточнения,* и до конца 1970-х гг. не было найдено способа ее решения. Предположим, что мы хотим ввести в *базу данных «здравого смысла»* аксиому о том, что птицы могут летать. Очевидно, что аксиому нужно каким-то образом уточнить, потому что пингвины, мертвые птицы, а также птицы, ноги которых замурованы в бетон, летать не могут. Аккуратно сформулировать аксиому удастся, если ввести в нее исключения — пингвинов и мертвых птиц; однако ясно, что можно придумать сколько угодно дополнительных исключений, таких, как птицы, у которых ноги замурованы в бетон. Формализованные немонотонные рассуждения (см. [4], [15]—[17] и [23]) дают возможность сказать на формальном языке, что птицы умеют летать, если только ситуация не является ненормальной, а также сделать заключение, что будут рассматриваться только те ненормальные ситуации, появление которых является следствием принятых во внимание фактов.

Немонотонность значительно расширила возможности выражения универсальных знаний о результатах событий в ситуационном исчислении. Она также дала метод решения *проблемы фрейма,* которая была еще одним препятствием для достижения общности, как отмечено в [12]. Проблема фрейма (этот термин используется по-разному, но я первый его ввел) возникает в случаях, когда допустимыми являются несколько действий, каждое из которых изменяет определенные признаки ситуации. Необходимо каким-то образом сказать, что действие изменяет только те признаки ситуации, к которым оно непосредственно относится. Когда имеется фиксированное множество действий и признаков, можно прямо указать, какие признаки не изменяются в результате действия, даже если для этого потребуется много аксиом. Однако если предположить, что в базу данных можно вводить дополнительные признаки ситуаций и дополнительные действия, то возникнет проблема, суть которой в том, что аксиоматизация действий никогда не будет

завершена. Маккарти [16] придумал способ, как можно с этим справиться, используя методику ограничений (*circumscription*), но Лифшиц [10] показал, что этот метод надо улучшить, и сделал свои предложения по этому поводу.

Ниже приведены некоторые аксиомы ситуационного исчисления для перемещения и раскрашивания блоков с использованием метода ограничений (*circumscription*) из [16].

Аксиомы о расположении и результатах движения объектов

$$\forall x\ e\ s.\neg ab(aspect1(x, e, s)) \supset loc(x, result(e, s)) = loc(x, s).$$
$$\forall x\ l\ s.ab(aspect1(x, move(x, l), s)).$$
$$\forall x\ l\ s.\neg ab(aspect3(x, l, s)) \supset loc(x, result(move(x, l), s)) = l.$$

Аксиомы о цветах и раскрашивании

$$\forall x\ e\ s.\neg ab(aspect2(x, e, s)) \supset color(x, result(e, s)) = color(x, s).$$
$$\forall x\ c\ s.ab(aspect2(x, paint(x, c), s)).$$
$$\forall x\ c\ s.\neg ab(aspect4(x, c, s)) \supset color(x, result(paint(x, c), s)) = c.$$

Эти выражения показывают, как можно обойти проблему уточнения, потому что любое вообразимое количество условий, препятствующих перемещению или раскрашиванию, может быть добавлено позже, что повлечет за собой введение соответствующего *ab aspect*... . Они также демонстрируют способ решения проблемы фрейма, при применении которого мы не должны декларировать, что перемещения не влияют на цвета и их расположение.

Даже после возникновения формализованных немонотонных рассуждений задача создания универсальной базы данных «здравого смысла» не поддается решению. Проблема заключается в том, чтобы написать аксиомы, которые удовлетворяли бы нашим замечаниям об объединении универсальных фактов о каком-либо явлении. Как только мы приняли предварительное решение о некоторых аксиомах, у нас уже есть средства, чтобы учесть ситуации, в которых они неприменимы, и сделать необходимые обобщения. Более того, предполагаемые трудности часто оказываются похожими на те, которые относились к птицам с замурованными в бетоне ногами.

ВОПЛОЩЕНИЕ

Чтобы рассуждать о знаниях, представлениях или целях, необходимо расширить множество объектов, о которых ведутся рассуждения. Например, программа, которая осуществляет об-

ратный логический вывод о целевых установках, непосредственно использует их как предложения: *on(Block 1, Block 2)*; это значит, что символ *on* используется как предикатный символ языка. Однако программе, которая хочет сказать, что выполнение *on(Block 1, Block 2)* должно быть отложено до того момента, как будет выполнено *on(Block 2, Block 3)*, необходимо предложение типа *precedes (on(Block 2, Block 3), on(Block 1, Block 2)*, и если оно является предложением в рамках логики первого порядка, то символ *on* должен восприниматься как функциональный, а выражение *on(Block 1, Block 2)* должно рассматриваться как объект в логическом языке первого порядка.

Такой процесс изготовления объектов из предложений или других категорий логического языка называется воплощением (*reification*). Этот процесс необходим для увеличения выразительной силы языка, но он опять-таки приводит к усложнению рассуждений. Обсуждение вопроса содержится в работе [13].

ФОРМАЛИЗАЦИЯ ПОНЯТИЯ КОНТЕКСТА

По поводу каждой написанной аксиомы критически настроенный человек может сказать, что она верна только в определенном контексте. При наличии некоторой изобретательности критик может придумать более общий контекст, в котором аксиома в ее точном выражении неверна. Это особенно хорошо видно при рассмотрении того, как человеческое мышление отражается в языке. Попробуем аксиоматизировать предлог «на» («on»), для того чтобы потом сделать соответствующие выводы из информации, содержащейся в предложении «Книга лежит на столе» («The book is on the table»). Критик может затеять спор о точном значении слова «на» («on»), придумывая разнообразные трудности, состоящие в том, что что-нибудь может находиться между книгой и столом, или в определении необходимой для употребления слова «на» («on») величины гравитационной постоянной, а также следует ли при этом учитывать центробежные силы. Таким образом, мы сталкиваемся с сократовскими вопросами о том, что означают понятия, взятые в их абсолютной общности, и с примерами, которые никогда не возникают в реальной жизни. На самом деле наиболее общего контекста просто не существует.

Напротив, если аксиоматизация проведена на действительно высоком уровне обобщения, то аксиомы часто становятся длиннее, чем это необходимо для конкретных ситуаций. Поэтому люди предпочитают говорить «Книга лежит на столе» (The book is on the table), пренебрегая указаниями на время

и определениями, на каком именно столе и какая именно книга. Проблема определения необходимого уровня общности обязательно возникает при выражении знаний «здравого смысла» на формальном языке, будь то логика, программа или какой-нибудь другой формализм. (Некоторые ученые считают, что на внутренних уровнях знания представляются только в виде примеров, а строгие механизмы применения аналогий и подобия позволяют их использовать на более общем уровне. Я желаю этим ученым удачи в формулировке точных утверждений о том, что же это за механизмы.)

Возможное решение проблемы состоит в формализации понятия контекста и в комбинировании его с методом ограничений (circumscription) в немонотонных рассуждениях. Добавим в функции и предикаты в наших аксиомах еще один параметр контекста. Каждая аксиома содержит утверждение о некотором контексте, в котором она справедлива. Следующие аксиомы говорят о том, что факты наследуются и в более ограниченном контексте, если не утверждаются исключения. Применимость каждого утверждения немонотонным образом распространяется на определенные более общие контексты, но здесь опять есть исключения. Например, в утверждении о том, что птицы летают, неявным образом подразумевается, что существует атмосфера (чтобы в ней летать). В более общем контексте это может и не предполагаться. Остается определить, чем наследование при переходе к более общему контексту отличается от наследования при переходе к частному контексту.

Предположим, что любое предложение p, находящееся в памяти компьютера, рассматривается в определенном контексте и является сокращенной записью выражения $holds(p, C)$, где C — название контекста. Некоторые контексты могут быть очень частными. Например, Ватсон — это доктор в контексте рассказов о Шерлоке Холмсе и психолог-баритон в трагической опере об истории психологии.

Отношение $c1 \leqslant c2$ означает, что контекст $c2$ является более общим, чем контекст $c1$. Разрешаются предложения вида $holds(c1 \leqslant c2, c0)$, так что предложения, относящиеся к контекстам, сами могут выполняться в определенном контексте. Теория не определяет «самый общий» контекст, так же как и теория множеств Цермело — Френкеля не определяет множество самого общего вида.

Логическая система, использующая контексты, может иметь операции входа в контекст и выхода из контекста, результатом применения которых является то, что можно было бы назвать *ультраестественным выводом,* допускающим последовательные рассуждения вида

$$
\begin{array}{l}
holds(p, C) \\
ENTER\ C \\
p \\
\quad . \\
\quad . \\
\quad . \\
q \\
LEAVE\ C \\
holds(q, C).
\end{array}
$$

Это напоминает обычные логические системы естественного вывода, но по причинам, выходящим за рамки данной статьи, может оказаться некорректным считать контексты эквивалентами множеств предположений и даже бесконечных множеств предположений.

Все это довольно неопределенно, но это гораздо больше, чем я мог сказать в 1971 г.

ЛИТЕРАТУРА

1. Black F. A deductive question answering system. Ph. D. dissertation, Harvard Univ., Cambridge, Mass., 1964.
2. Buchanan B. G., Shortliffe E. H., Eds. Rule-based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project. American Elsevier, New York, 1984.
3. Davis R., Buchanan B., Shortliffe E. Production rules as a representation for a knowledge-based consultation program. Artif. Intell. 8, 1 (Feb. 1977).
4. Doyle J. Truth maintanance systems for problem solving. In: Proc. of the 5th International Joint Conference on Artificial Intelligence. 1977, p. 247.
5. Ernst G. W., Newell A. GPS: A Case Study in Generality and Problem Solving. Academic Press, Orlando, Fla, 1969.
6. Fikes R., Nilsson N. STRIPS: A new approach to the application of theorem proving to problem solving. Artif. Intell. 2, 3, 4 (Jan. 1971), 189—208.
7. Friedberg R. M. A learning machine. IBM J. Res. 2, 1 (Jan. 1958), 2—13.
8. Friedberg R. M., Dunham B., North J. H. A learning machine, p. II. IBM J. Res. 3, 3 (July, 1959), 282—287.
9. Green C. Theorem-proving by resolution as a basis for question answering systems. In: Machine Intelligence 4, B. Melter and D. Michie, Eds. University of Edinburgh Press, Edinburgh, 1969, p. 183—205.
10. Lifschitz V. Computing circumscription. In: Proc. of the 9th International Joint Conference on Artificial Intelligence, vol. 1, 1985, pp. 121—127.
11. McCarthy J. Programs with common sense. In: Proceedings of the Teddington Conference on the Mechanization of Thought Processes. Her Majesty's Stationery Office, London. Reprinted in: Semantic Information Processing, M. Minsky, Ed. M. I. T. Press, Cambridge, Mass., 1960.
12. McCarthy J., Hayes P. J. Some philosophical problems from the standpoint of artificial intelligence. In: Machine Intelligence 4, D. Michie, Ed. American Elsevier, New York, 1969.
13. McCarthy J. First order theories of individual concepts and propositions. In: Machine Intelligence 9, D. Michie, Ed. University of Edinburgh Press, Edinburgh, 1979.
14. McCarthy J. Some expert systems need common sense. In: Computer Culture: The Scientific, Intellectual and Social Impact of the Computer, vol. 426. Pagels, Ed. Annals of the New York Academy of Sciences, New York, 1983.

15. McCarthy J. Circumscription — A form of non-monotonic reasoning. Artif. Intell. 13, 1, 2 (Apr. 1980).
16. McCarthy J. Applications of circumscription to formalizing common sense knowledge. Artif. Intell. (Apr. 1986).
17. McDermott D., Doyle J. Non-monotonic logic I. Artif. Intell. 13, 1, 2 (1980), 41—72.
18. Newell A., Shaw J. C., Simon H. A. Preliminary description of general problem solving program-I (GPS-I). CIP Working Paper 7, Carnegie-Mellon Univ., Dec. 1957.
19. Newell A., Shaw J. C., Simon H. A. Report on a general problem-solving program for a computer. In: Information Processing: Proceedings of the International Conference on Information Processing (Paris). UNESCO, 1960, pp. 256—264. (RAND P-1584, and reprinted in Computers and Automation, July 1959.)
20. Newell A., Shaw J. C., Simon H. A. A variety of intelligent learning in a General Problem Solver. In: Self-Organizing Systems, M. C. Yovits and S. Cameron, Eds. Pergammon Press, Elmsford, N. Y., 1960, pp. 153—189.
21. Newell A., Simon H. A. Human Problem Solving. Prentice-Hall, Englewood Cliffs, N. J., 1972.
22. Laird J. E., Newell A., Rosenbloom P. S. Soar: An architecture for general intelligence.
23. Reiter R. A logic for default reasoning. Artif. Intell. 13, 1, 2 (Apr. 1980).
24. Simon H. Still unsubstantiated rumor, 1960. GENERA [W86, JMC] TEXed on May 27, 1986, at 11:50 p.m.