Глава 4. Логический подход к построению систем ИИ. Неформальные процедуры.

Неформальная процедура — это особый способ представления функций.

Неформальные процедуры, выполняемые человеком, обладают рядом специфических особенностей, существенно затрудняющих их представление в ЭВМ с помощью алгоритмических языков программирования.

Чтобы в какой-то степени приблизиться к этому "человеческому" способу представления функций, рассмотрим прежде всего традиционные алгоритмические модели и попытаемся понять, в чем состоит основная трудность их применения для имитации неформальных процедур.

Алгоритмические модели

Алгоритмические модели основаны на понятии алгоритма. Исторически первые точные определения алгоритма, возникшие в 30-х годах, были связаны с понятием вычислимости. С тех пор было предложено множество, как выяснилось, эквивалентных определений алгоритма.

Чтобы оценить возможности использования алгоритмов для представления неформальных процедур, рассмотрим простую задачу.

ЗАДАЧА. Описать процедуру, реализующую преобразование из именительного падежа в родительный для существительных следующих типов: **ДОМ, МАМА, ВИЛКА, КИНО, НОЧЬ, ТОКАРЬ, КИЛЬ**.

Решение 1 указано на Рис. 1 в виде блок схемы соответствующего алгоритма.

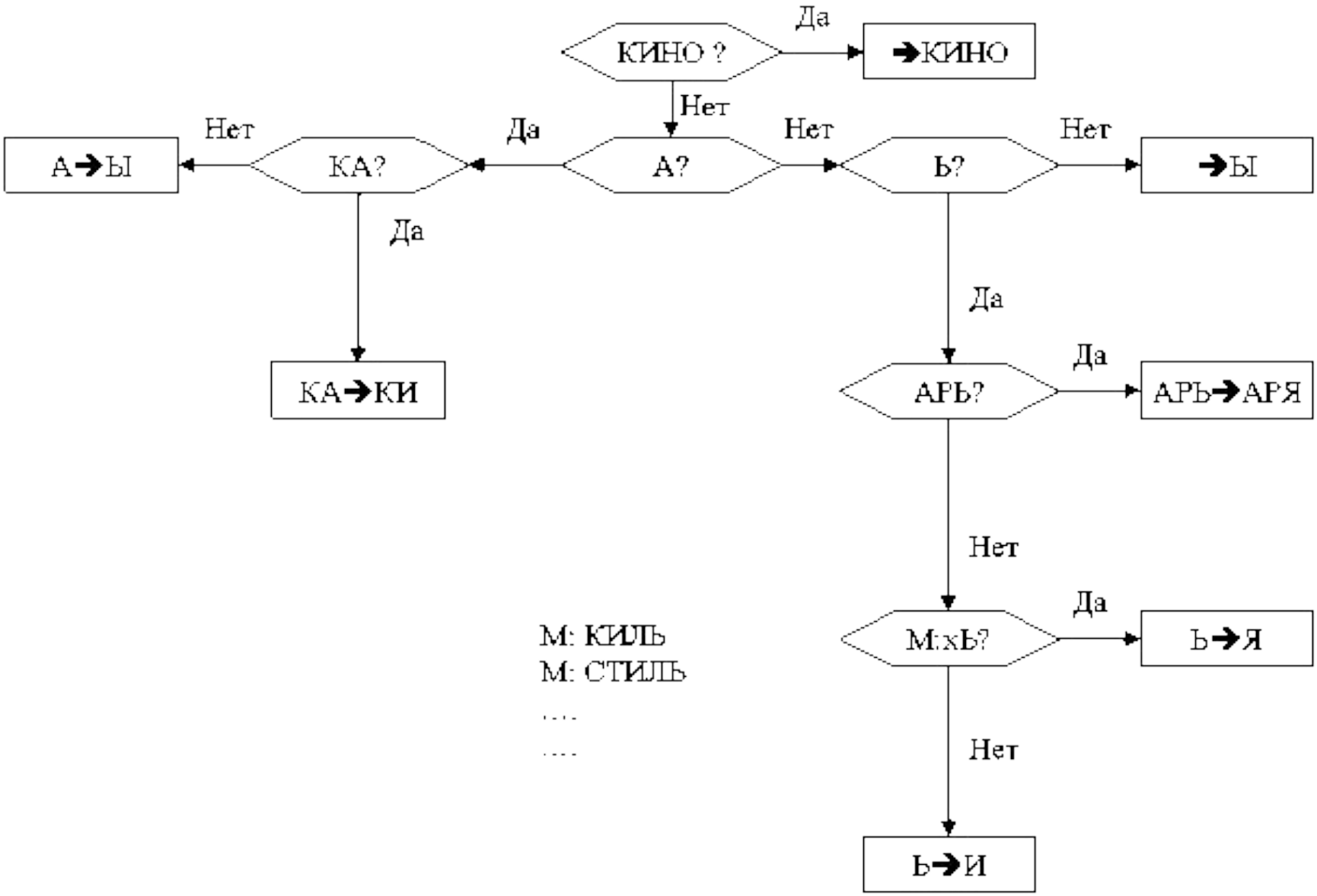

Рис. 1. Решение 1. Алгоритм

С точки зрения программирования на алгоритмических языках достоинства подобного представления очевидны — эта блок-схема без затруднений переводится в текст программы, например, на языке Ассемблера или C++.

Однако само составление подобной блок-схемы при появлении существительных новых типов становится, очевидно, все более и более утомительным занятием. Для иллюстрации этого предположим, что дана

ДОПОЛНИТЕЛЬНАЯ ЗАДАЧА. Расширить алгоритм, представленный на Рис. 1 на слова **ВАСЯ, ВРЕМЯ, АКЦИЯ, ЗАДАЧА**

Разумеется программист без особого труда составит соответствующую блок-схему алгоритма. И все же, если учесть, что подобные изменения и расширения алгоритма при программировании неформальных процедур происходят многократно (реальная сложность неформальной процедуры как раз и проявляется в практической невозможности предусмотреть заранее все случаи), следует признать, что, вполне правильное в статике, решение 1 в динамике неудачно!

Продукционные модели

В подобных случаях для обеспечения динамичности процессов модификации программ используются те или иные варианты таблиц решений. С учетом этого для исходной задачи более приемлемо решение 2:

Ситуация	Действие	Ситуация	Действие
КИНО	КИНО	-Ь	-И
-ча	-чи	-ие	-ия
-КА	-КИ	-мя	-мени
-А	-Ы	-я	-и
-АРЬ	-АРЯ	-	-А
-Ь & М:хЬ	-Я		

Таблица 1. Решение 2

Соответствующая таблица решений содержит две графы — слева приведены описания ситуаций, справа — соответствующие действия.

Предполагается, что программист разработал интерпретирующую программу для подобных таблиц. Эта программа работает следующим образом.

Для конкретного входного слова, пусть это будет для примера слово **РОЗА**,

- осуществляется последовательный просмотр ситуаций, указанных в таблице, и сравнение их со входным словом.

- Если слово соответствует некоторой ситуации, то выполняется действие, указанное для этой ситуации.

Для слова **РОЗА** будет обнаружено соответствие с ситуацией **"-А"**. В результате выполнения действия **"-Ы"** будет получено выходное слово **РОЗЫ**.

Теперь значительно упрощается расширение на новые классы слов — необходимо лишь обеспечить внесение вставок на нужное место в таблице решений.

Таблицы решений представляют собой частный случай так называемых **продукционных систем**.

В этих системах правила вычислений представляются в виде продукций. **Продукции** представляют собой операторы специального вида и состоят из двух основных частей, для краткости называемых обычно "**ситуация — действие**".

"Ситуация" содержит описание ситуации, в которой применима продукция. Это описание задается в виде условий, называемых **посылками продукции**.

"Действие" — это набор инструкций, подлежащих выполнению в случае применимости продукции.

Режим возвратов

Таблица решений, приведенная в Таблице 1, иллюстрирует так называемую безвозвратную процедуру.
В этом случае на каждом шаге выбирается единственное решение — так, для слова **РОЗА** таким решением будет **РОЗЫ** — проблема выбора решения не возникает.
В общем случае неформальные процедуры являются многозначными, а правильность конкретного выбора, сделанного на некотором шаге, проверяется на следующих шагах. При этом используется так называемый **режим возвратов**.

а). МАТЬ ——————> ЛЮБИТ ——————> ?
что делать? кого?
б). МАТЬ <—————— ЛЮБИТ <—————— ?
кого? что делать?

Тривиальность рассмотренного примера убеждает в необходимости режима возвратов при реализации неформальных процедур.

Логический вывод

Важность логического вывода становится очевидной уже при рассмотрении простейших информационно-логических процедур.

Предположим, что некоторая база данных содержит сведения об отношениях **"o — ОТЕЦ y" и "x — МАТЬ y"**.

Чтобы обработать запросы типа:

ИВАНОВ А.И. — ДЕД ПЕТРОВА В.А.?

ПЕТРОВ В.А. — ВНУК ИВАНОВА А.И.?

необходимо либо ввести в базу данных также и сведения об отношениях **"x — ДЕД y" и "x — ВНУК y"**, либо объяснить системе, как из отношений **ОТЕЦ**, **МАТЬ** извлечь искомую информацию.

Реализация первой возможности связана с неограниченным ростом избыточности базы данных.

Вторая возможность при традиционном алгоритмическом подходе требует написания все новых и новых программ для реализации новых типов запросов.

Логический вывод позволяет расширять возможности "общения" наиболее просто и наглядно. Так, для приведенных типов запросов системе достаточно будет сообщить три правила:

1. **x—ДЕД y если x—ОТЕЦ a и a—РОДИТЕЛЬ y**

2. **x—РОДИТЕЛЬ y если x—ОТЕЦ y или x—МАТЬ y**

3. **x—ВНУК y если y—ДЕД x**

Эти правила содержат естественные и очевидные определения понятий **ДЕД**, **РОДИТЕЛЬ**, **ВНУК**.

Поясним в чем состоит логический вывод для запроса "**A—ДЕД B?**" в предположении, что в базе данных имеются факты: **A—ОТЕЦ Б** и **Б—МАТЬ B**. При этом для упрощения опустим тонкости, связанные с падежными окончаниями.

Пользуясь определением 1 система придет к необходимости проверки существования такого индивидуума a, что факты **A—ОТЕЦ a и a—РОДИТЕЛЬ B** истинны. Если такой a существует, то **A—ДЕД B**, если не существует такого a, то **A** не является дедом **B**.

Зависимость продукций

Мы могли бы использовать в таблице решений только конкретные факты, например правила **ДОМ --> ДОМА**, **МАМА --> МАМЫ** и т. д., и динамичность соответствующей таблицы решений была бы восстановлена — подобные правила можно было бы вводить в произвольном порядке!

Однако цена подобной "динамичности" окажется непомерно высокой — полный отказ от обобщенных правил.

Продукционные системы с исключениями

Если отношение "**правило—исключение**" встроено в систему, она сама может понять, что преобразование **ПАЛКА --> ПАЛКЫ** незаконно.

При этом система должна руководствоваться простым принципом:

если применимо исключение, общее правило запрещено.

Соответствующие системы будем называть системами с исключениями.

Отношение "**общее правило — исключение**" безусловно полезно для понимания системой уместности правил. Можно сказать, что это отношение устанавливает автоматически (по умолчанию) наиболее типичное для неформальных процедур взаимодействие правил:

— исключение "вытесняет" общее правило.

— при пересечении разрешены оба правила.

Разумеется, возможны ситуации, когда необходимо поступать наоборот:

— исключение не запрещает общего правила

— при пересечении одно из правил запрещено.

Пусть дано, например, общее правило **x --> p1** и его исключение **Ax --> p2**.

Таким образом, для произвольного слова необходима реакция **p1**.

Для слова же, начинающегося с буквы **A**, исполняется реакция **p2** — по умолчанию для таких слов реакция **p1** незаконна.

Предположим, однако, что по условию конкретной задачи для слов, начинающихся с **A**, реакция **p1** также допустима. В этом случае введение нового правила **Ax --> p1** снимает запрет на реакцию **p1** в ситуации **Ax**.

Аналогичный способ годится для пересечения правил.

Таким образом, аппарат исключений позволяет устанавливать произвольные способы взаимодействия правил, в том числе и отличные от взаимодействия по умолчанию.

При развитии продукционной системы с исключениями программист сосредотачивает свое внимание на выявлении новых правил и на обобщении уже имеющихся.

Аппарат исключений освобождает программиста от решения трудоемких вопросов согласования правил — распознавание и интерпретация исключений осуществляется автоматически.