Продукционная модель

П р о д у к ц и о н н а я модель предполагает такой способ организации вычислительного процесса, при котором программа преобразования некоторой информационной структуры ***C*** задается в виде системы правил вида:

Условие $\rightarrow$ ***Действие***,

где ***Условие*** специфицирует некоторые требования к текущему состоянию структуры ***C***, а ***Действие*** содержит описание тех операций над ***C***, которые надо выполнить, если ***C*** удовлетворяет этим требованиям.

Продукционная модель

- Формальные системы
  - Системы подстановок
  - Формальные грамматики
- Программные системы

Системы подстановок

Системы подстановок служат для обработки слов, заданных в некотором алфавите.

К ним относятся:

(1) **системы продукций Поста** (именно этим системам мы обязаны появлению термина "система продукций", который получил со временем более широкое употребление) и

(2) **нормальные алгоритмы Маркова**.

Системы продукций Поста

Системы Поста определяются алфавитом S и набором правил-продукций вида:

$$a_i W \rightarrow W b_i \quad (i = 1,...,m), \qquad (1)$$

где a_i и b_i - некоторые слова в алфавите S.

Правило *(1)* применимо к слову d, если d начинается с a_i. В этом случае правило вычеркивает в d префикс a_i и приписывает к нему справа слово b_i.

Например, применяя к слову $ababc$ продукцию $abW \rightarrow Wd$, получим слово $abcd$, к которому еще раз может быть применена та же продукция.

Нормальные алгоритмы Маркова

Нормальные алгоритмы Маркова определяются алфавитом S и последовательностью подстановок вида:

$a_i \to b_i$

$a_i \to . b_i$,

где a_i и b_i - некоторые слова в алфавите S, а подстановка, отмеченная точкой является заключительной.

Нормальные алгоритмы Маркова

$a_i \to b_i$
$a_i \to . b_i$

(n+1)-й шаг обработки начального слова d_0 :

Пусть в последовательности подстановок имеется первая, у которой левая часть a_i хотя бы раз входит в слово d_n (здесь d_n - слово, полученное в результате первых *n* шагов), тогда в d_n вместо самого левого вхождения a_i подставляется b_i, порождая слово d_{n+1}.

Нормальные алгоритмы Маркова

$a_i \to b_i$
$a_i \to . b_i$

Если использованная подстановка не была заключительной, осуществляется переход к шагу *(n+2)*, в противном случае d_{n+1} объявляется результатом работы системы.

Если на шаге (*n+1*) не нашлось ни одной применимой подстановки, то результатом будет слово d_n.

Формальные грамматики

Формальные грамматики были введены Хомским, предложившим описывать их четверкой

$$< V, T, P, Z >,$$

где V - алфавит, $T \subseteq V$ - алфавит терминальных символов, P - конечный набор правил подстановки, Z - начальный символ.

Накладывая различные ограничения на вид правил подстановки, получают грамматики различных классов.

Формальные грамматики, как и системы Поста, не предусматривают порядка применения правил и являются недетерминированными системами.

Формальные грамматики

С самого начала формальные грамматики использовались при анализе формальных и естественных языков, что стимулировало их постоянное усложнение и развитие в качестве инструментальных средств программирования.

Первым продукционным языком программирования был язык Флойда, основанный на грамматике и предназначавшийся для синтаксического анализа.

Сравнение формальных систем

В **системе продукций Поста** фиксируется место вхождения левой части правила (начало обрабатываемого слова), но не оговаривается порядок применения правил, что делает систему Поста **недетерминированной** и порождает в общем случае не один, а множество процессов обработки, дающих, возможно, различные результаты.

В **алгоритмах Маркова** правила применимы к любому участку слова (хотя эта свобода и ограничена самым левым вхождением подслова a_i), но при этом **порядок просмотра правил фиксирован**. Вследствие этого система Маркова является **детерминированной** и порождает для каждого конкретного входного слова единственный процесс, приводящий к однозначному результату.

Сравнение формальных систем

Формальные грамматики, служащие для описания синтаксиса и семантики различных языков, являются как раз тем классом формальных систем, который в своем развитии привел к появлению современных систем продукций (СП). Первым продукционным языком программирования был язык Флойда, основанный на грамматике и предназначавшийся для синтаксического анализа.

Программные системы

Программные продукционные системы - системы, имеющие программную реализацию. Такие системы далее называются системами продукций (СП).

Программная СП состоит из трех основных частей: *базы данных* (рабочей памяти), *множества правил-продукций* и *интерпретатора*.

Структура программной СП

Структура программной СП

База данных представляет собой рабочую память (различной организации), над которой работает множество правил.

Правила могут иметь произвольную сложность, но структура у них прежняя: левая часть - условие применимости, а правая часть - действие, которое данное правило выполняет.

Интерпретатор - поисковый процесс, состоящий, по крайней мере, из двух фаз: выбора продукции и ее применения.

Выбор продукций

Задача выбора продукции сводится к двум подзадачам:

(1) максимально ограничить число продукций, условия применимости которых будут проверяться,

(2) из полученного множества продукций выбрать одну – с истинным условием применимости.

Задача (1) – часто решается путем активации нужного подмножества правил-продукций.

Задача (2) возникает в связи с тем, в выделенном подмножестве продукций могут оказаться истинными условия более чем одной продукции.

Конфликтное множество продукций

Множество продукций, у которых условия применения истинны в данный момент, называется ***конфликтным.***

Процедура выбора продукции из такого множества называется ***процедурой разрешения конфликта.***

Способы выбора продукций

Основные **способы выбора продукций** из конфликтного множества:

- случайный выбор;

- выбор по статическому критерию

(например, первой применяется продукция с самыми жесткими требованиями - продукция с самым длинным списком условий), либо на продукциях задан полный порядок или иерархия, при этом первой применяется самая "старшая" продукция и т.п.);

-выбор по динамическому критерию

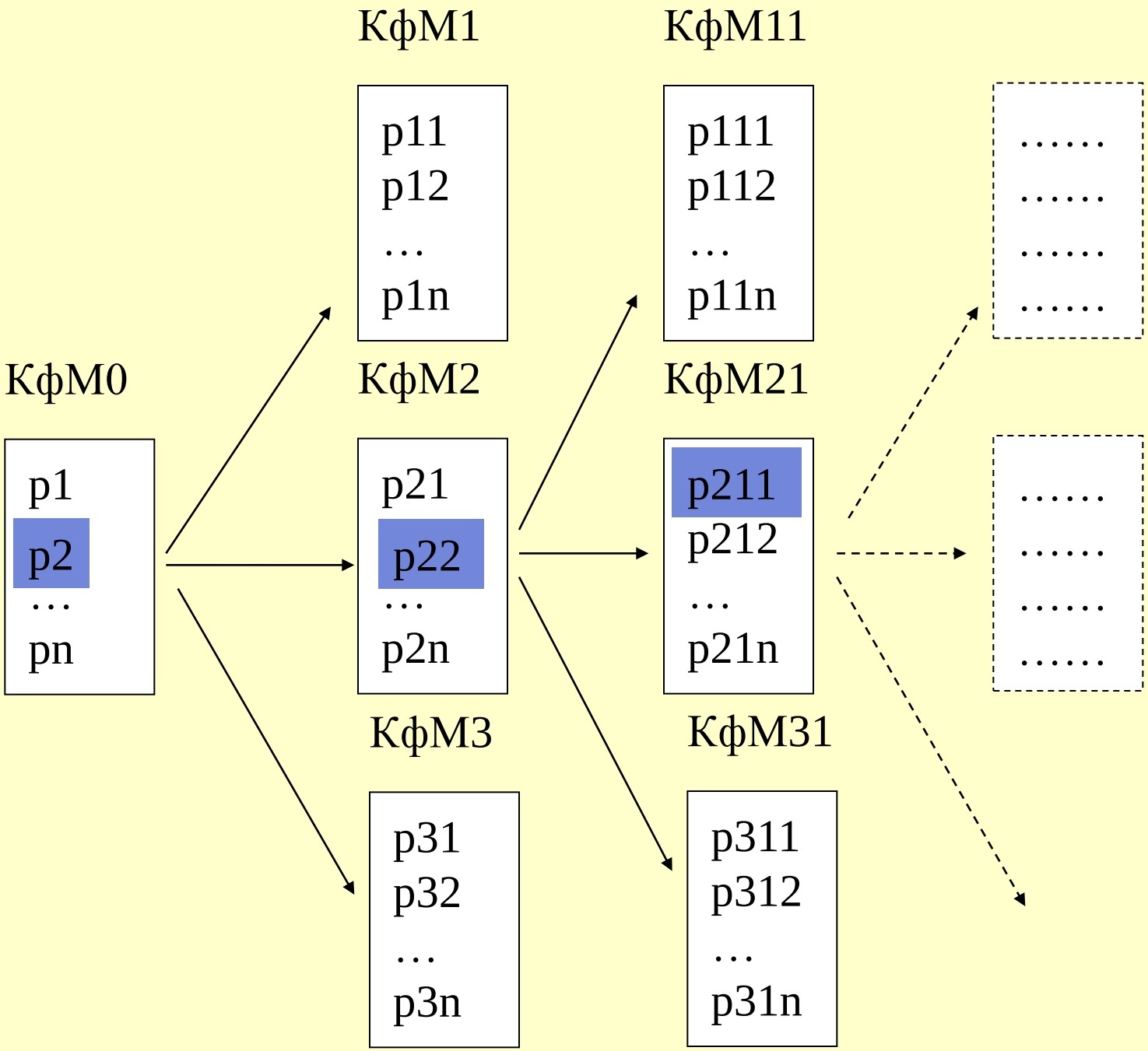
(например, приписыванием правилам и/или компонентам базы данных динамически вычисляемых весов (приоритетов); в этом случае первой для исполнения выбирается продукция с наивысшим приоритетом, или продукция, условия (образец) которой удовлетворяется на данных, имеющих максимальный приоритет).

Управляющие стратегии

Выделяется два класса **управляющих стратегий** применения продукций: ***безвозвратная*** и ***пробная***.

При использовании **безвозвратной** стратегии на каждом шаге вычислений из конфликтного множества для выполнения выбирается одна из подходящих продукций, и в дальнейшем вернуться к этой точке вычислений и применить другую продукцию невозможно.

При **пробной** стратегии (***бэктрекинге)*** обеспечивается возможность возврата к уже пройденной точке вычислений и применения другой (альтернативной) продукции из конфликтного множества.


КфМ1
КфМ11
p11
p12
…
p1n
p111
p112
…
p11n
……
……
……
……
КфМ0
КфМ2
КфМ21
p1
p2
…
pn
p21
p22
…
p2n
p211
p212
…
p21n
……
……
……
……
КфМ3
КфМ31
p31
p32
…
p3n
p311
p312
…
p31n

Стратегии применения СП

По способу применения продукций выделяют два вида систем продукций: п р я м ы е и о б р а т н ы е.

В СП, работающей в **прямом направлении**, образцом для поиска служит левая часть продукции (***Условие***). Задача решается в направлении от исходного состояния к целевому. Продукции, применяясь к текущему состоянию, порождают новые состояния.

В **обратных СП** задача решается в обратном направлении - от цели к начальному состоянию. Каждый шаг обратного движения, т.е. применение продукции в обратном направлении, когда ***Условие*** (применения) и ***Действие*** меняются местами, производит подцелевое состояние, из которого целевое может быть получено при прямом движении.

Стратегии применения СП

В качестве примера рассмотрим СП, включающую несколько простых правил:

(1) $y \,\&\, w \rightarrow x$

(2) $u \,\&\, z \rightarrow y$

(3) $r \rightarrow z$

В данном случае стрелка ($\rightarrow$) означает, что если верно то, что написано слева, то верно то, что написано справа.

Эта СП работает над БД, содержащей следующие факты:

r, s, t, u, v, w,

где, например, факт *r* означает “зажигание исправно”,

а факт *s* – “шумы в коробке передач”.

Первое правило (1) может интерпретироваться следующим образом: «Если имеет место перегрев двигателя (*y*) и есть шумы в двигателе (*w*), то перебит маслопровод (*x*)".

Прямой вывод

Прямой вывод предполагает использование правил для вывода новых фактов из имеющихся.

Для этого Интерпретатор по очереди просматривает все правила с целью выяснения, являются ли факты в левой части правил истинными. Если у очередного правила левая часть истинна, то добавляется факт из правой части правила к хранимым фактам (в базу фактов).

Затем Интерпретатор переходит к следующему правилу и повторяет тот же процесс. Проверив все правила, он начинает проверку правил сначала. Эта работа продолжается до тех пор, пока в базу фактов добавляются новые факты.

Прямой вывод

Вернемся к нашему примеру.
Сначала может примениться только одно правило
(3) $r \rightarrow z$,
которое добавляет z в базу фактов.
После этого может примениться правило
(2) u & z $\rightarrow y$,
так как факт u задан как истинный с самого начала, а факт z
только что был выведен.
В результате в базу фактов будет добавлен факт y .
Теперь в базе фактов есть и y, и w , поэтому может сработать правило
(1) $y \,\&\, w \rightarrow x$,
которое выведет факт x .
После этого работа Интерпретатора завершится.

База правил:
(1) $y \,\&\, w \rightarrow x$
(2) $u \,\&\, z \rightarrow y$
(3) $r \rightarrow z$

База фактов:
r, s, t, u, v, w,

Обратный вывод

При этом способе **вывод начинается** не с посылок правил, а сразу с интересующего нас **заключения** (будем называть его целевым или **целью**). Как правило, такое заключение не находится среди известных фактов, поэтому его истинность нужно доказать.
Механизм вывода в этом случае состоит в нахождении тех правил, которые содержат данный факт в качестве заключения.
Затем просматриваются посылки этих правил.
Если они уже хранятся в базе фактов, то цель доказана.
Если посылки не являются истинными фактами, то делается проверка: не являются ли они заключениями других правил. Если это так, то предпринимается попытка доказать истинность уже посылок этих правил.
Этот процесс продолжается до тех пор, пока в качестве посылок не окажутся факты, которые являются истинными. Если такие факты найдутся, значит, цель доказана, в противном случае целевое заключение не доказано.

Обратный вывод

Предположим, нам необходимо доказать истинность факта ***x*** на основе представленных выше правил и фактов.

База правил:
(1) $y \,\&\, w \rightarrow x$
(2) $u \,\&\, z \rightarrow y$
(3) $r \rightarrow z$

База фактов:
r, s, t, u, v, w,

Факт *x* не задан явно в базе фактов, поэтому цель *x* активирует правило
(1) $y \,\&\, w \rightarrow x$,
w есть в базе фактов, тогда нужно доказать подцель *y*.
Подцель *y* активирует правило
(2) $u \,\&\, z \rightarrow y$
u есть в базе фактов, тогда нужно доказать подцель *z*.
Подцель *z* активирует правило
(3) $r \rightarrow z$,
r есть в базе фактов, тогда подцель *z* истинна, а значит истинна подцель *y*, а тогда истинна и цель *x*.

Выбор стратегии вывода

Выбор стратегии вывода зависит от решаемой задачи.

Прямой вывод применим в тех ситуациях, когда **пространство** возможных **решений необозримо**, в то время как **количество исходных данных невелико**.
Например, имеется огромное число способов сборки сложного компьютера из модульных компонентов, набор которых ограничен. В связи с этим прямой вывод чаще всего применяется в задачах планирования и проектирования.

Обратный вывод применяется в тех задачах, где **число возможных решений невелико**, но присутствуют **большие объемы исходных данных**. К таким задачам относятся задачи классификации и диагностики, в которых число видов (например, животных) или диагностируемых ситуаций (например, заболеваний, технических неисправностей) невелико. Поэтому обратный вывод чаще всего применяется при диагностике и классификации.

Классификация систем продукций

Рассмотрим деление СП по тому, как они решают проблему активации продукций.

- Простые системы продукций
- Управляемые системы продукций
  - СП с независимым управляющим языком
  - Иерархические СП
  - Последовательные СП
  - Параллельно-последовательные СП

В управляемых СП предусмотрены средства структуризации множества продукций и/или их активации.

Простые системы продукций

В **простых СП** активными считаются все продукции.
Такой способ активации правил применяется

– в СП с **небольшим количеством правил**,

– в СП, в которых структуризация множества продукций и их принудительная активация **противоречат принципам**, положенным в основу данной СП, или используемой ею стратегии выбора продукций.

Это относится, например, к

– **семейству языков OPS**, использующих специальный алгоритм быстрого сопоставления образцов (*Rete-алгоритм*, эффективность которого не зависит от количества правил) и

– **языку Пролог**, множество правил которого образует единую систему утверждений и выключение хотя бы одного правила из этой системы, нарушит ее логическую целостность.

СП с независимым управляющим языком

«Чистая» система продукций:

p1
p2
…
pn

Стратегия активации правил:

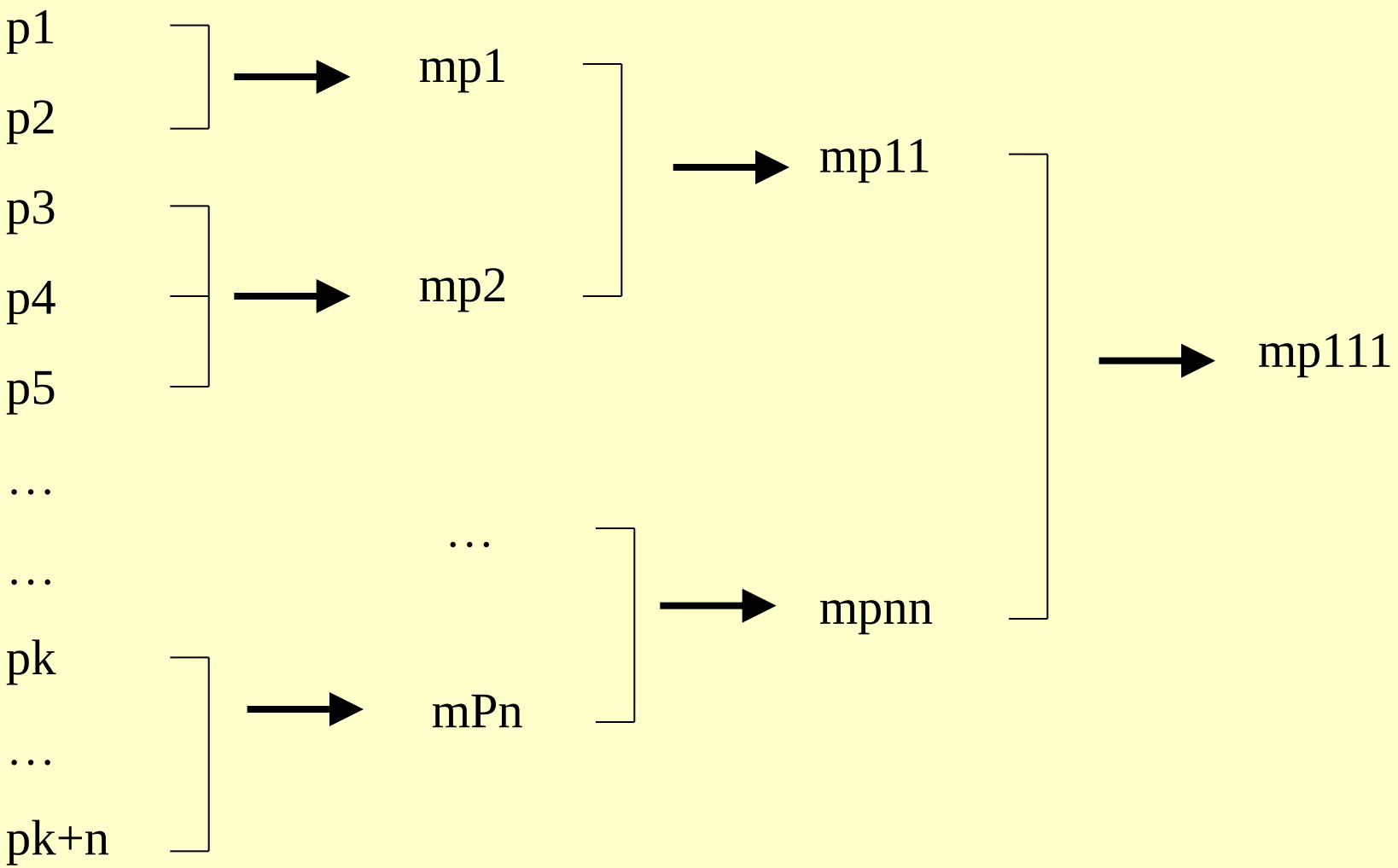
$p1, (p2,p3)^2, (p4,p5)^*, (p6)^3, p7$

Иерархическая СП

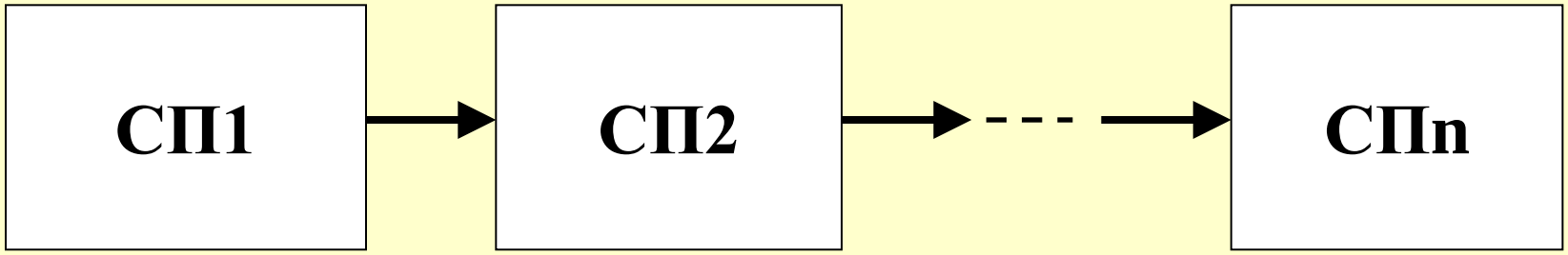
При этом подходе используются **метапродукции**, т.е. продукции, содержащие информацию о других продукциях и активирующих (дезактивирующих) на основе этих знаний и анализа текущего состояния БД другие продукции.

СП могут включать несколько уровней метапродукций. Поэтому такие СП называются **иерархическими**.

Иерархическая СП



Последовательная СП

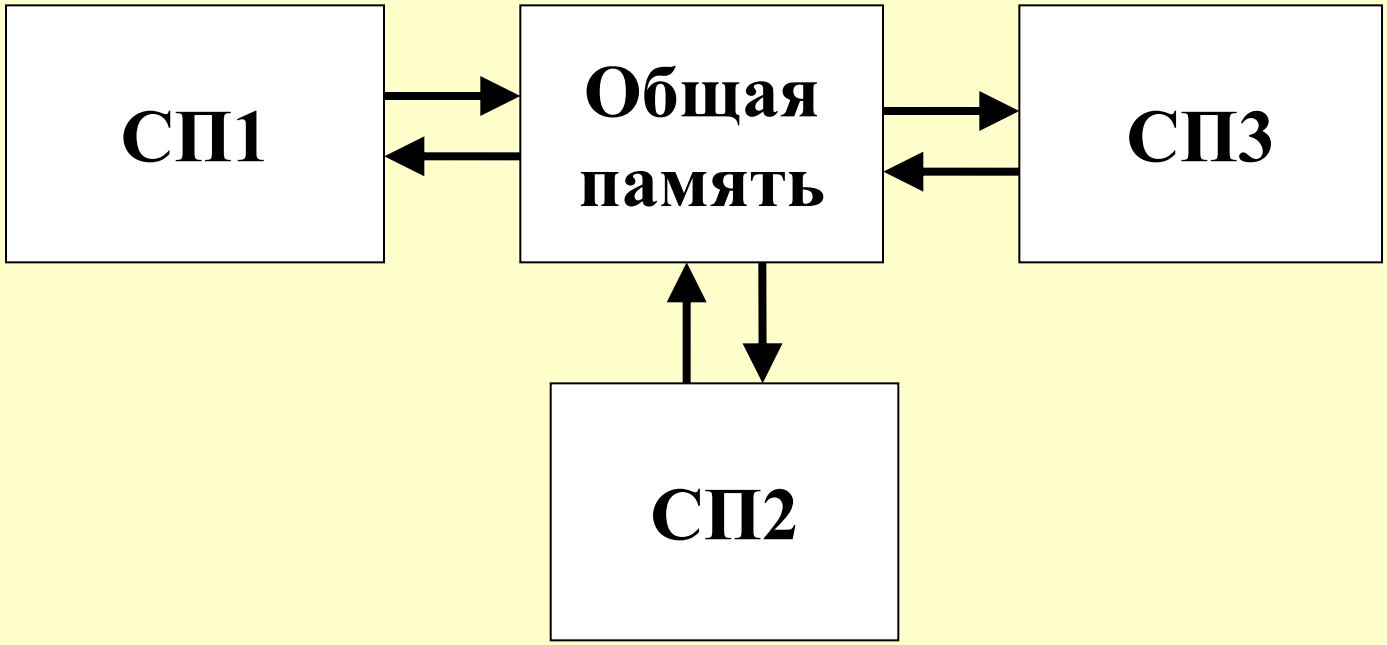


СП разбивается на несколько подмножеств, каждое из которых представляет автономный модуль обработки данных (продукционный модуль).

Каждый модуль (СПi) соответствует определенному уровню знаний или этапу обработки данных.

Пример: система ЗАПСИБ (InBASE) – обработка запросов на естественном языке к реляционной БД.

Параллельно-последовательная СП



Достоинства и недостатки СП

Достоинства:

•**Универсальность** СП, как метода описания широкого класса задач.

•**Естественность спецификации знаний**. Для многих предметных областей естественно представлять знания в виде правил вида ***Условие → Действие***.

•**Высокая и естественная модульность** СП:
(1) каждая продукция представляет собой автономное действие, снабженное индивидуальной функцией управления, самостоятельно определяющей момент выполнения действия;
(2) все множество продукций может естественным образом структурироваться путем разбиения на подмножества, объединяющие продукции, которые относятся к одним и тем же компонентам знаний.

Достоинства и недостатки СП

Недостатки:

•Существенно более **низкая эффективность** вычислительного процесса по сравнению с программированием на традиционных языках.

•Повышенная **сложность контроля** правильности СП-процесса.

•**Сложность отслеживания** непротиворечивости множества правил.

Использование продукционной модели

- Построение компиляторов
- Автоматическая обработка текстов
- Распознавание и синтез речи
- Экспертные системы