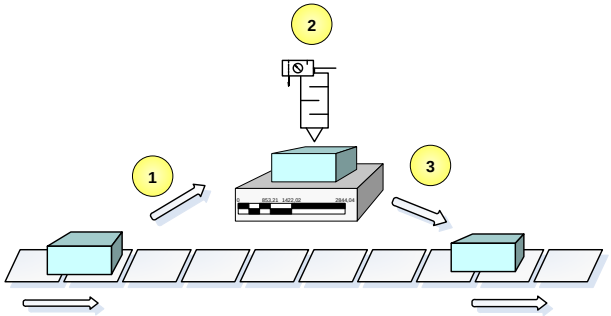
Карпов В.Э.

# Автоматное программирование

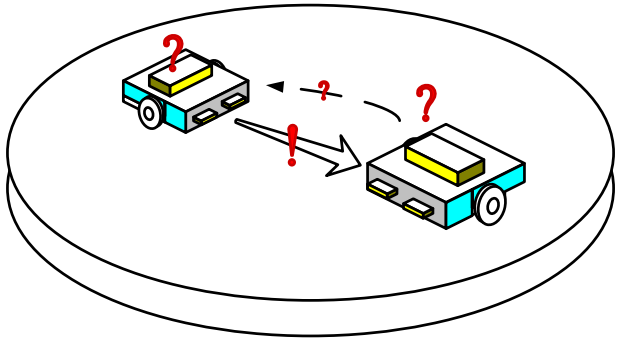
# 3 типичные «робототехнические» задачи

Классическая интерпретация в терминах поколений систем управления – жесткое программное, стимул-реактивное и управление в условиях неопределенности.

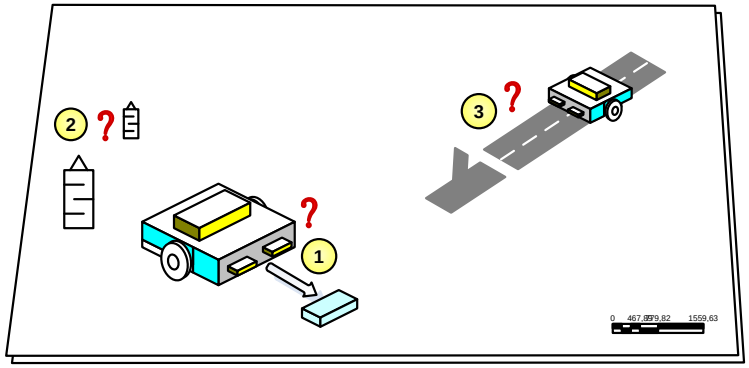
**Задача 1. Фрагмент сборочной лини**




**Задача 2. Робо-сумо**



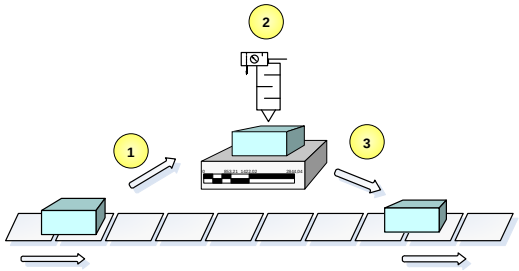
**Задача 3. Соревнования УМНИК-БОТ**

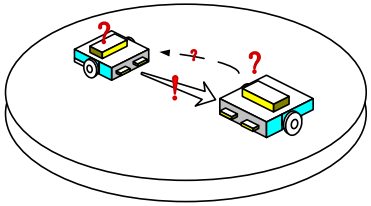


# Принципиальные особенности задач

**Сборочная линия**

- Выполнение жесткой последовательности операций.
- Алгоритм принципиально линеен, т.к. полностью предсказуема среда и условия выполнения операция.
- Это – задача для автоматического устройства.
- Интуитивно представляется, что это не «настоящая робототехническая» задача, хотя реализующее ее устройство может называться промышленным роботом.




**Робо-сумо**

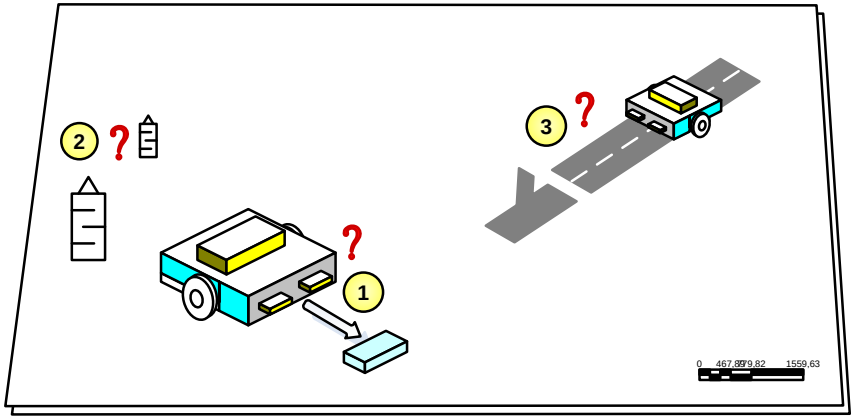
- Робот-соперник активен (перемещается в пространстве, мешает, атакует, убегает и проч.)
- Жесткая, линейная последовательность действий здесь уже не годится.
- Однако сам алгоритм может быть достаточно простым: (1) «сбор информации» - (2) «анализ и принятие решения» - (3) «выполнение действия». Например:

1. *Сбор информации от датчиков*
2. *if(противник_спереди) then атаковать*
3. *if(противник_слева) then повернуть_налево*
4. *if(противник_справа) then повернуть_направо*
5. *…*
6. *goto 1*

Необходим анализ ситуации и, в зависимости от результатов анализа, далее следует выполнение той или иной операции.

# УМНИК-БОТ

- Принципиально иная ситуация.
- «Обычные» управляющие программы жесткой логикой и тем более – с линейной последовательностью действий, уже не годятся.
- УМНИК-БОТ относится к категории сложных поведенческих задач.



# Реальный физический мир

Робот представляет собой реальный физический объект, который должен работать в реальном физическом мире.

**Следствия**:

- Существуют неизбежные ошибки в управлении - робот не может всегда точно и предсказуемо отрабатывать свои действия.
- Очевидно, что существуют помехи и периодически возникают различного рода непредвиденные факторы (различного рода неровности и неоднородности поверхности, неточные показания датчиков и др.).
- Факторы противодействия среды, например, в виде наличия робота-соперника.

=>

Мир, в котором живее робот, сложен для того, чтобы точно формализовать его – мира – условия и поведение робота в нем.

=>

Описывать поведение робота в виде жесткой, линейной схемы крайне **неудобно**.

## Поведенческие задачи

Робот должен уметь **анализировать** текущую ситуацию и, в зависимости от того, как была оценена эта ситуация, выполнять ту или иную процедуру или комплекс действий.

=> Робот должен рассматривать общую задачу как **множество более простых подзадач** и уметь переключаться между ними.

Роботу полезно знать, что именно он сейчас решает или: *в каком состоянии* он сейчас находится.

# Состояние

- Состояние – это то, что определяет ситуацию, в которой находится робот.
- Состояние может определить этап решения задачи, совокупность значений датчиков, некоторую внутреннюю точку отсчета.
- Множество состояний определяет то, что называется памятью в системе управления (в самом широком, «идеологическом» смысле).

**Роботы-сумоисты.** Память не нужна (глубина памяти = 1). Безусловные рефлексы или стимул-реактивным поведение. У них большую роль играет процедура анализа ситуации.

**УУ сборочной линией.** Память нужна (помнить, какой этап операции выполняется в данный момент), но она линейна. Анализ ситуации большой роли не играет.

Автоматное программирование, как метод решения сложных поведенческих задач

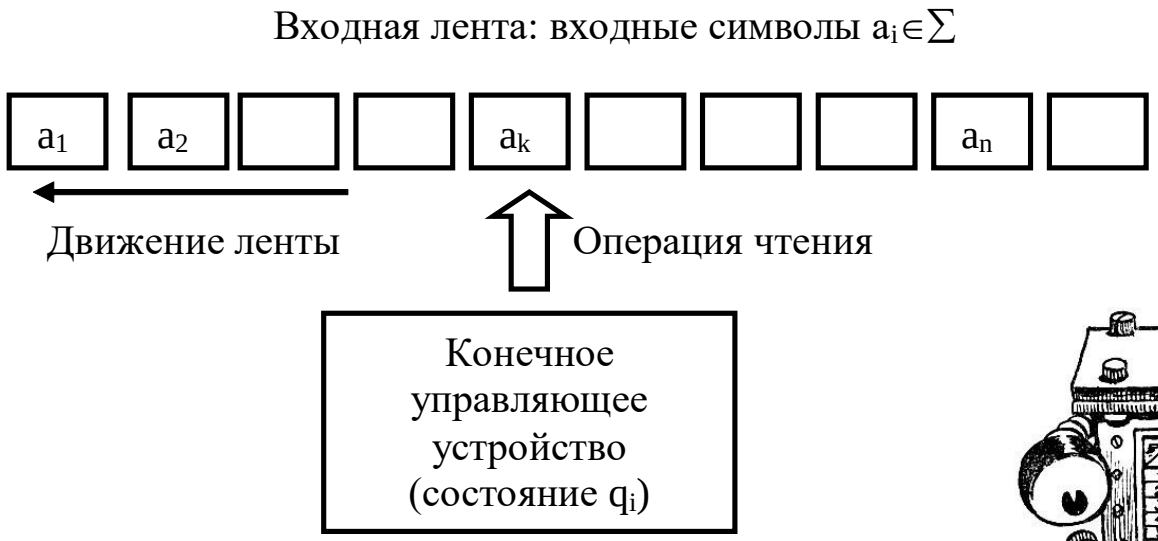
# Автомат

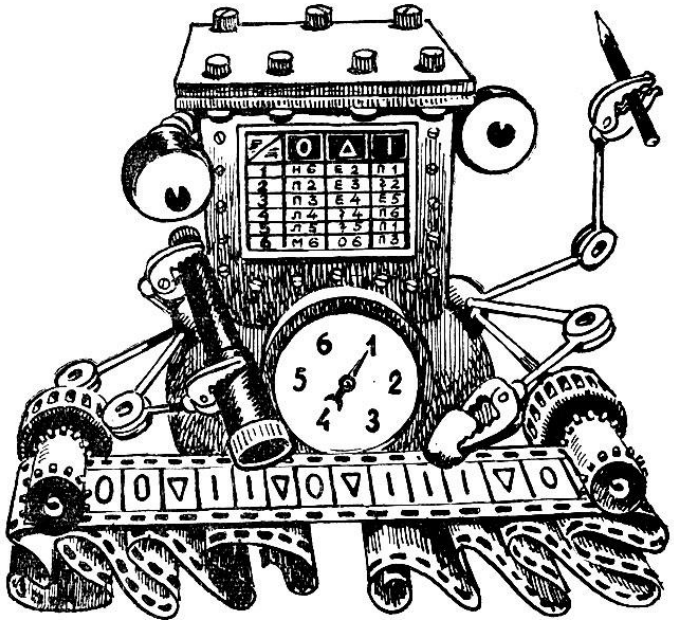
$$A = (\Sigma, Q, q_0, T, P)$$

- $\sum$ – входной алфавит (конечное множество, называемое также входным словарем);
- Q – конечное множество состояний;
- $q_0$ – начальное состояние ($q_0 \in Q$);
- T – множество терминальных (заключительных) состояний, $T \subset Q$;
- P – подмножество отображения вида $Q \times \sum \to Q$, называемое функцией переходов. Элементы этого отображения называются *правилами* и обозначаются как $q_i a_k \to q_j$

где $q_i$ и $q_j$ – состояния, $a_k$ – входной символ: $q_i, q_j \in Q$, $a_k \in \sum$.

# Автомат и Машина Тьюринга



- Автомат, как распознающая система

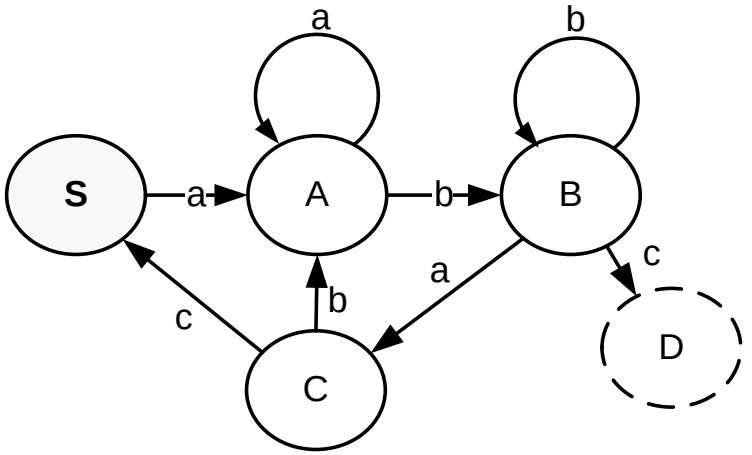
# Задача анализа

Автомат распознает такие последовательности, как:

*(a, b, c), (a, a, a, b, c), (a, a, a, a. b, b, b, c), ($a^n$, bm, a, b, $b^k$, c), ($a^n$, $b^m$, a, c, $a^r$, $b^s$, c)*

и т.п. При этом автомат в конце оказывается в состоянии *D*.

Последовательности вида *(b, c), (an, $b^m$, a)* и проч. не приводят автомат в *D*.Такие последовательности для автомата являются «неправильными», т.е. он их не распознает.



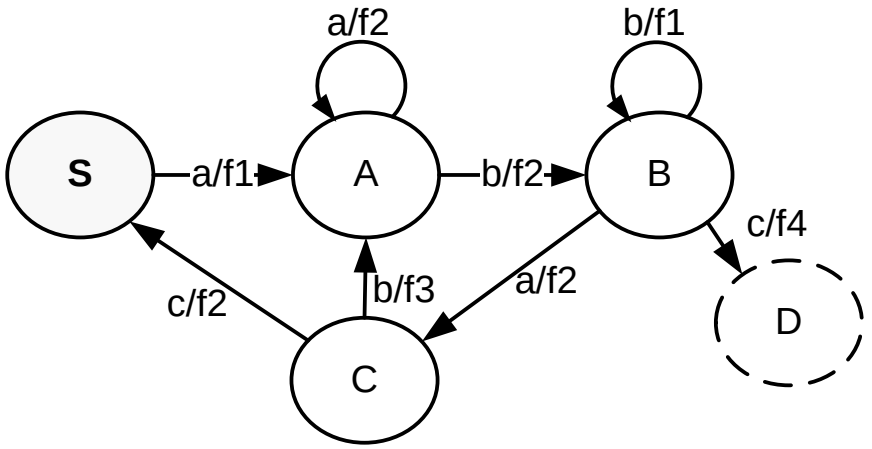
# Автоматы с выходами

$$A_{\text{вых}} = (\Sigma, Q, q_0, T, P, Y, \lambda)$$

- Автомат Мили

$$\lambda : Q \times \Sigma \to Y$$

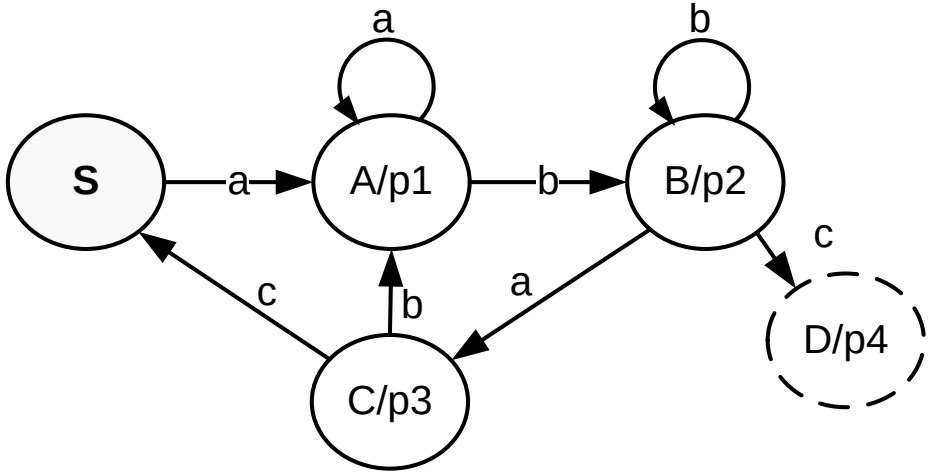
$$y(t+1) = \lambda(q(t), x(t))$$




- Автомат Мура

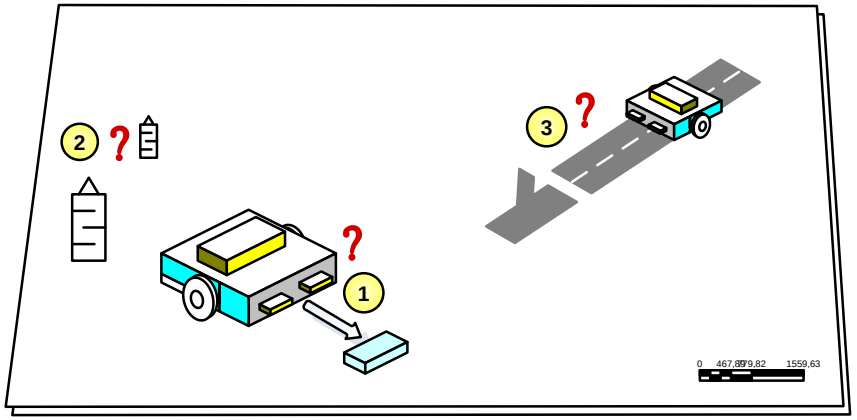
$$\lambda : Q \to Y$$

$$y(t+1) = \lambda(q(t))$$



# Задача регламента УМНИК-БОТ

- «Эвакуация»
- «Кнопка»
- «Свеча»
- «Флаг»




***Робот умеет (базовое ПО):***

- совершать действия вида движения в заданном направлении, развороты;
- ехать по линии, ориентируясь по датчикам;
- считывать значения с установленных на роботе дальномеров;
- реагировать на препятствия;
- обнаруживать источники инфракрасного излучения («свечи»);
- включать вентилятор;
- поднимать флаг и т.п.

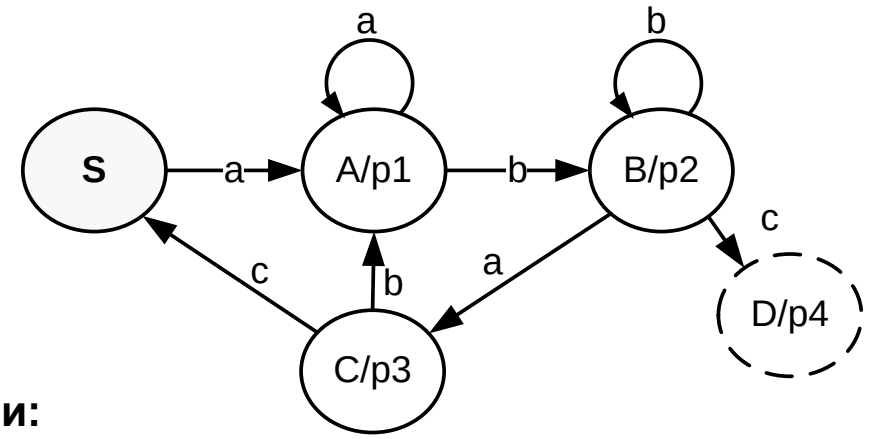
# Управляющий автомат

**Работа автомата** заключается в том, чтобы:

1. Проанализировать ситуацию.
2. Определить, в какое состояние он должен перейти
3. Определить, что ему делать в этом состоянии.

**Память** в виде состояний должна определить:

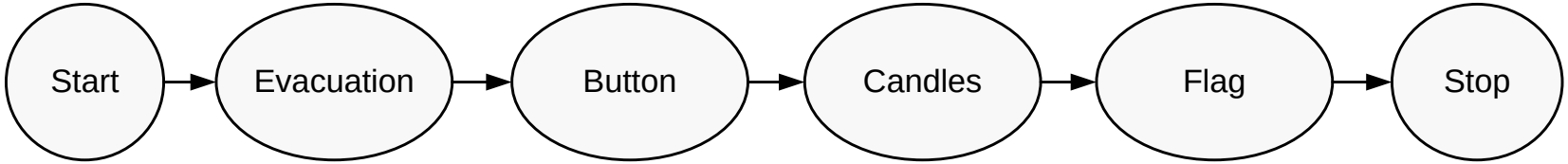
1. Что делать в случае изменившихся условий,
2. На какую задачу переключиться,
3. Куда вернуться после отработки рефлексов и проч.




**Две основные стратегии решения задачи:**

1. Жесткая стратегия, при которой последовательность решаемых задач строго зафиксирована.
2. **Ситуационная** стратегия, при которой робот выбирает, какую задачу следует решать в данный момент времени, исходя из анализа текущей ситуации, его приоритетов, планов и проч.

# Фиксированная стратегия



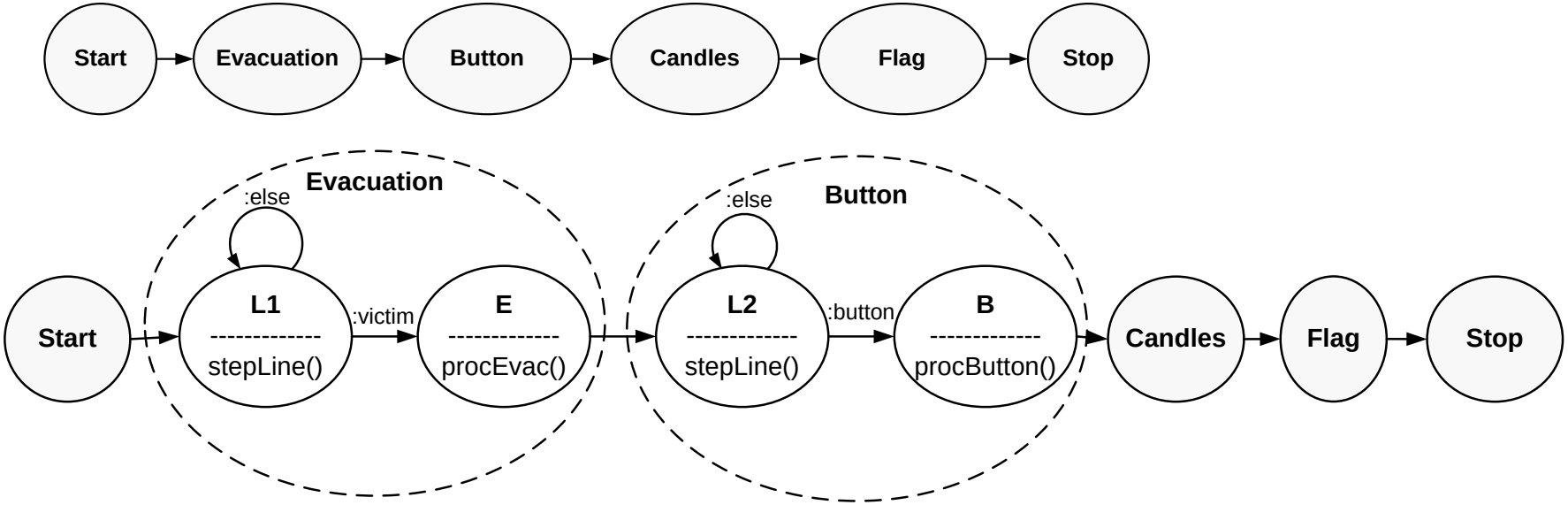

**Действия**:

- Движение по линии.
- Процедура эвакуации.
- Процедура нажатия на кнопку.
- Поиск свечи.
- Задувание свечи.
- Поиск зоны приема.
- Процедура поднятия флага
- …

**Обнаруживаемые объекты:**

- Брусок («пострадавший») (“:victim”)
- Кнопка (“:button”)
- Препятствие спереди (“:obstacle”)
- Свеча (“:candle”)
- …

# «Уточнение» автомата




Множество *Q* состояний автомата – это вершины *Start, L1, L2* и т.д.,

множество $\sum$ (входной алфавит автомата) – это сигналы *“:victim”, “:button”, “:obstacle”, “:candle”* и проч.

# Особенности программной реализации

Используемые переменные:

Q – номер или имя состояния автомата

**X** – входной символ

1. Setup() -- Инициализация системы
2. Q := Start -- Робот вначале находится в состоянии Start
3. **X** := ReadSensors() -- Чтение информации от датчиков (формирование вектора входных символов **X**)
4. Q’ := F(Q, X) -- Определение нового состояния Q’, исходя из того, в каком состоянии находился автомат и каков входной символ.
5. Q := Q' -- Переход в это состояние

6. ExecProc(Q) -- Выполнить процедуру, связанную с этим -- состоянием Q (у нас – автомат Мура)

7. goto 3

# П.3. Чтение информации от датчиков

- Формируем не один-единственный входной символ *x*, а некоторое множество входных сигналов ***X***. Это – существенное отличие от того, что говорилось в определении автомата.

**N двоичных датчиков**
**$|\Sigma| = 2^N$**

- Вектор ***X***. Множество переменных, отвечающих за регистрацию того или иного события, значения сигнала и т.п. Флаги (предикаты)

- *X[0]* - признак того, что обнаружен брусок (сигнал ":victim" на графе),
- *X[1]* - признак ":button" и т.п.

# П.4. Определение нового состояния

- Функция переходов *F(Q, X)* и матрица переходов ***M***.

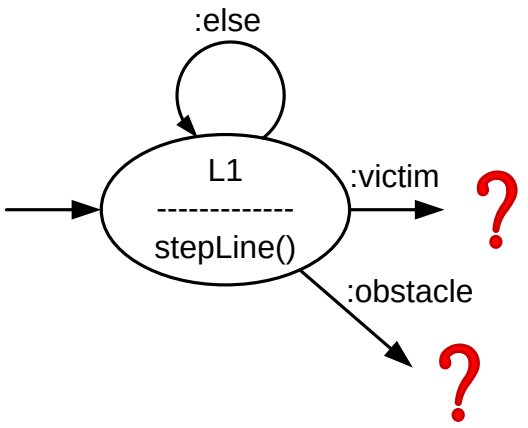
***M:***

| | Start | L1 | E | L2 | … |
|---|---|---|---|---|---|
| :obstacle | | | | | |
| :victim | | E | | | |
| :candle | | | | | |
| :button | | | | B | |
| … | | L1 | | | |
| … | | | L2 | | |

- Неоднозначность переходов.
  - Приоритеты?
  - Эвристики?
  - Зависимость от контекста?

**X[ ]**

| | |
|---|---|
| :obstacle | **1** |
| :victim | **1** |
| :candle | **0** |
| :button | **0** |
| | **…** |



# Реентерабельные процедуры

- **Выполнение процедуры, связанной с текущим состоянием Q.**
- Особенность – в цикличности выполнения
- Нельзя уйти в выполнение некоторой «долгой» процедуры.
- Цикличность: шаги 3-7 будут выполняться с большой частотой (100 Гц). Эта цикличность означает т.н. **реентерабельность** процедур, т.е. повторность входа в процедуры.

# Пример

- Реентерабельная процедура (процедура с повторным входом), – это некоторая последовательность операций, которую можно выполнять, используя внешний цикл. Например:

```
1. void stepLine (void)
2. {
3.   if(SensLeft && SensRight) GoFwd();
4.   else
5.   if(SensLeft) GoLeft();
6.   else
7.   if(SensRight) GoRight();
8. }
```

# Требования к реентерабельным процедурам

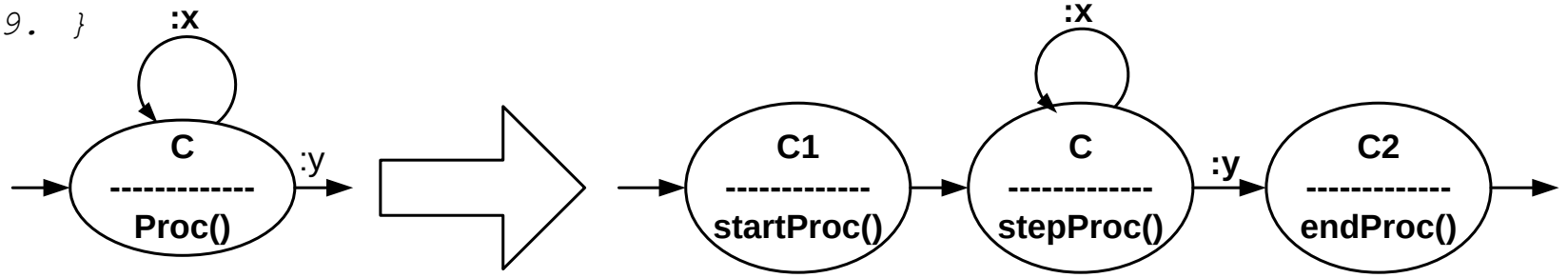
- Процедура должна осуществлять управление путем ее многократного вызова (мы вызываем ее циклически извне).
- Процедура должна быть «короткой».

Иногда оказывается, что реализовать чисто реентерабельную процедуру либо сложно, либо вовсе невозможно.

Тогда действие (или поведенческая процедура) в общем случае дополняется «прологом» (процедурой инициализации) и «эпилогом» (завершающей процедурой):

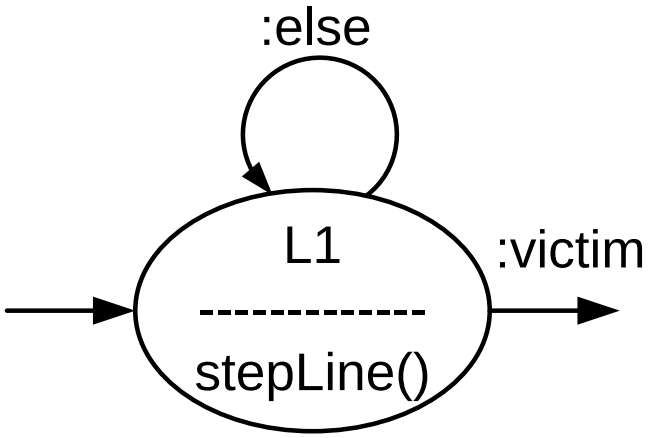
*startProc() – stepProc() – endProc().*

```
1. void Proc(void)
2. {
3.   Init(); // Пролог. Выделится в процедуру startProc()
4.   while(…)
5.   {
6.     Ctl(); // Основная часть. Выделится в процедуру evalProc()
7.   }
8.   Finish(); // Эпилог. Выделится в процедуру endProc()
9. }
```



# Переходы по условию «иначе»

- Автомат перейдет в следующее состояние, когда придет символ *“:victim”*.
  В противном случае он останется в текущем состоянии *L1*.
  Именно это и означает пометка *“:else”* на дуге.




- На самом деле *“:else”* должен означать некий вполне определенный входной символ, пусть он и является для нас фиктивным.

***M:***

| | Start | **L1** | E | L2 | … |
|---|---|---|---|---|---|
| :obstacle | | | | | |
| :victim | | E | | | |
| :candle | | | | | |
| :button | | | | B | |
| … | | | L2 | | |
| **:else** | | L1 | | L2 | |

# Функция F(Q, X) основного алгоритма

```
1.  int F(int q, int X[])
2.  // q – номер текущего состояния
3.  // X – вектор входных сигналов
4.  {
5.    for(int i=0;i<DIM_X;i++)
6.      if(X[i]!=0 && M[i][q]!=0) // Нашли «нормальный» переход
7.        return M[i][q];
8.    // Не нашли. Возвращаем переход по условию «иначе»
9.    return M[DIM_X][q];
10. }
```

***M:***

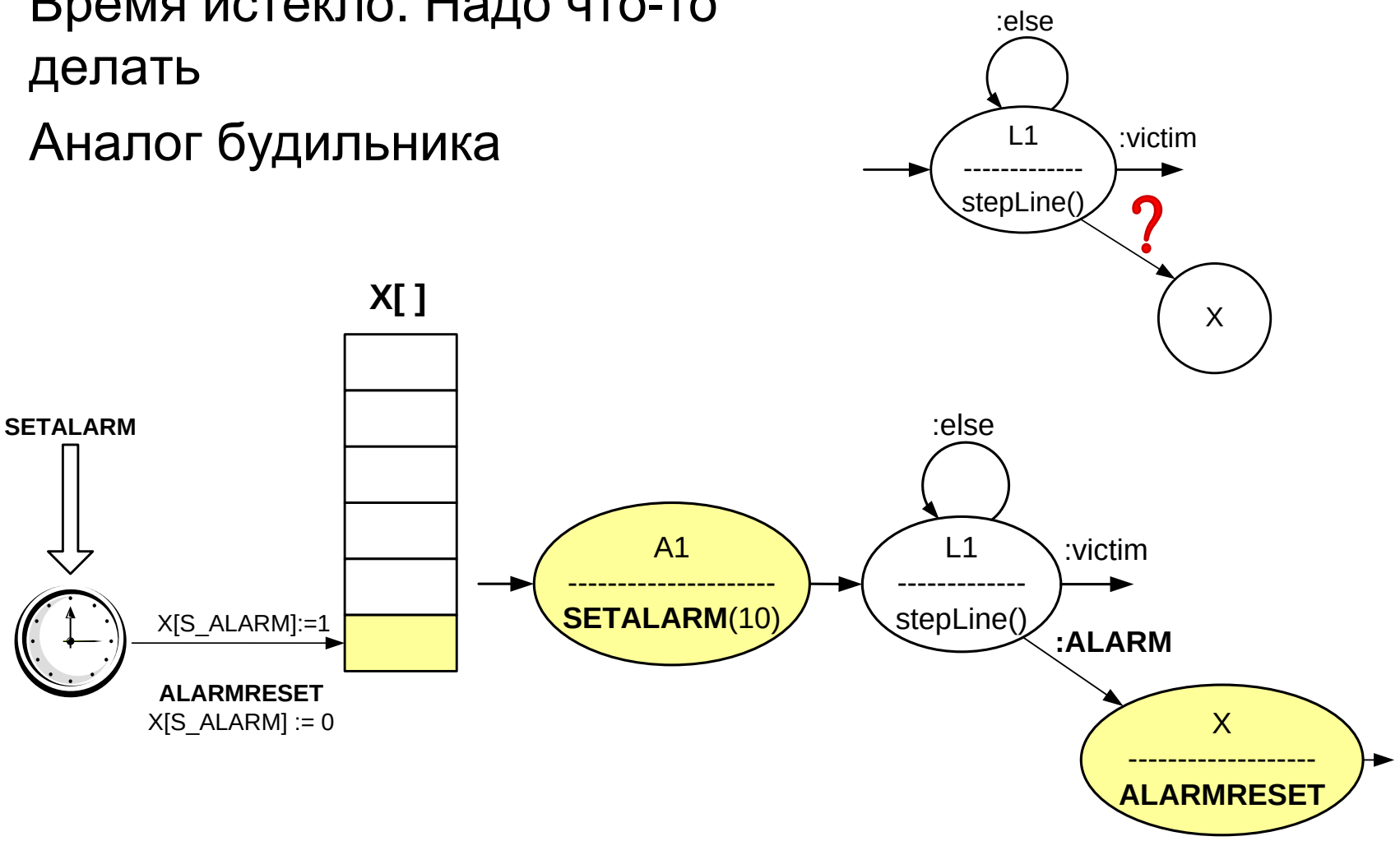
| | Start | **L1** | E | L2 | … |
|---|---|---|---|---|---|
| :obstacle | | | | | |
| :victim | | E | | | |
| :candle | | | | | |
| :button | | | | B | |
| … | | | L2 | | |
| **:else** | | L1 | | L2 | |

Вектор входных сигналов X содержит DIM_X элементов

Количество строк в матрице переходов ***M*** равно (DIM_X+1), т.к. необходима еще одна строка для фиктивного перехода по условию «иначе».

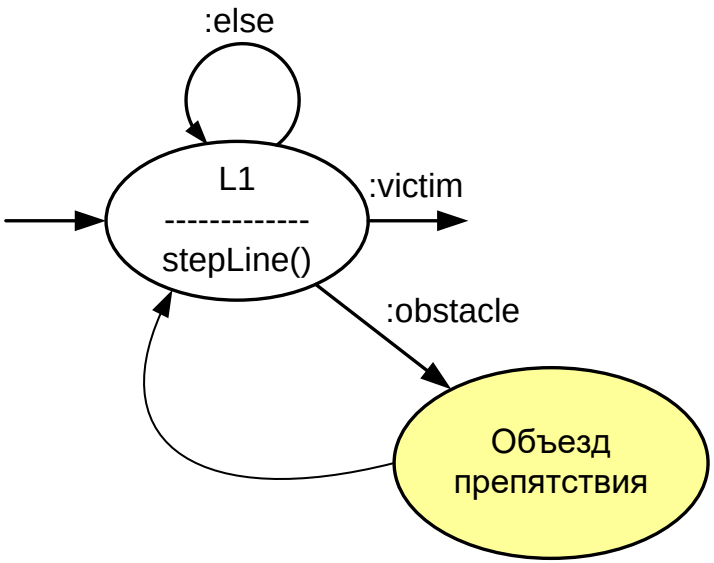
# Время

- Время истекло. Надо что-то делать
- Аналог будильника



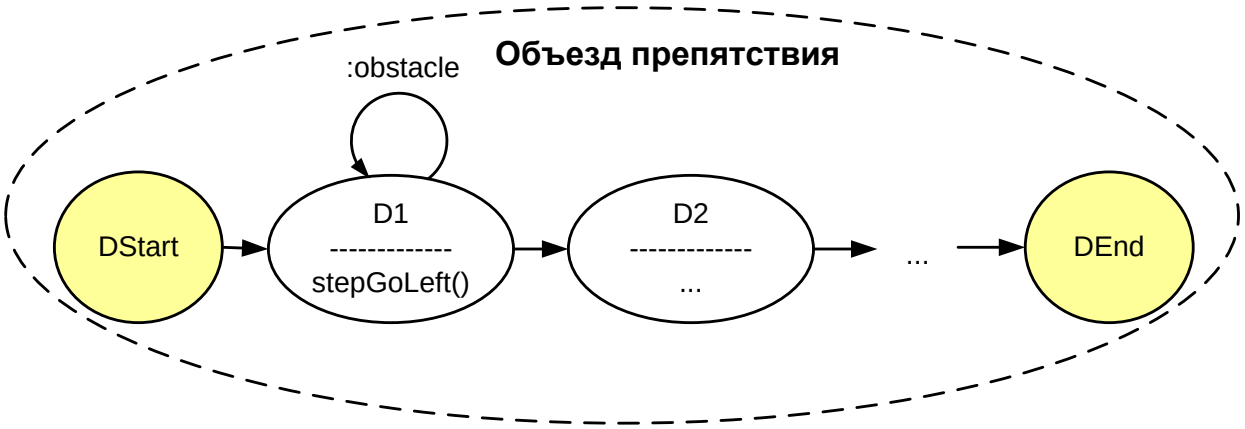
# Вызовы подпрограмм

- Вызов CALL
- Возврат RETURN

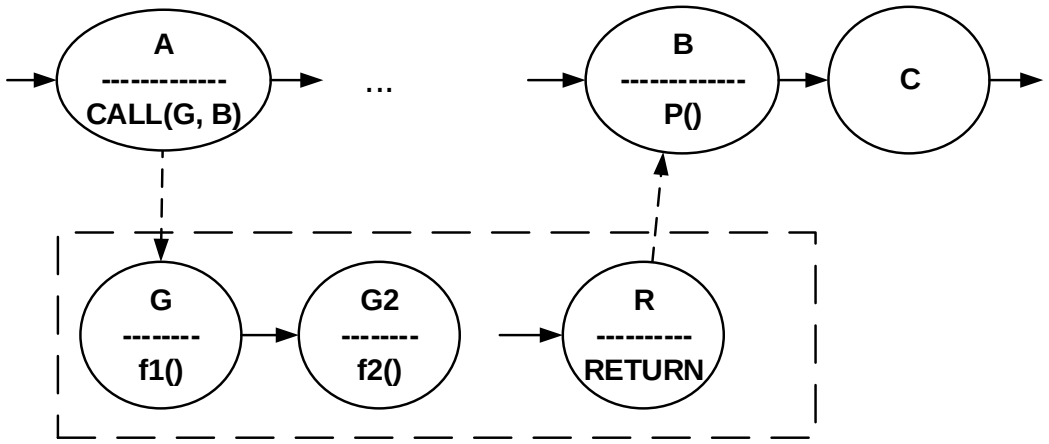



***CALL(GStart, RetCond)***

Рекурсии



# Пропуск вызова/перехода




1. Setup()
2. Q := Start
3. X := ReadSensors()
4. ExecProc(Q)
5. Q := F(Q, X)
6. goto 3

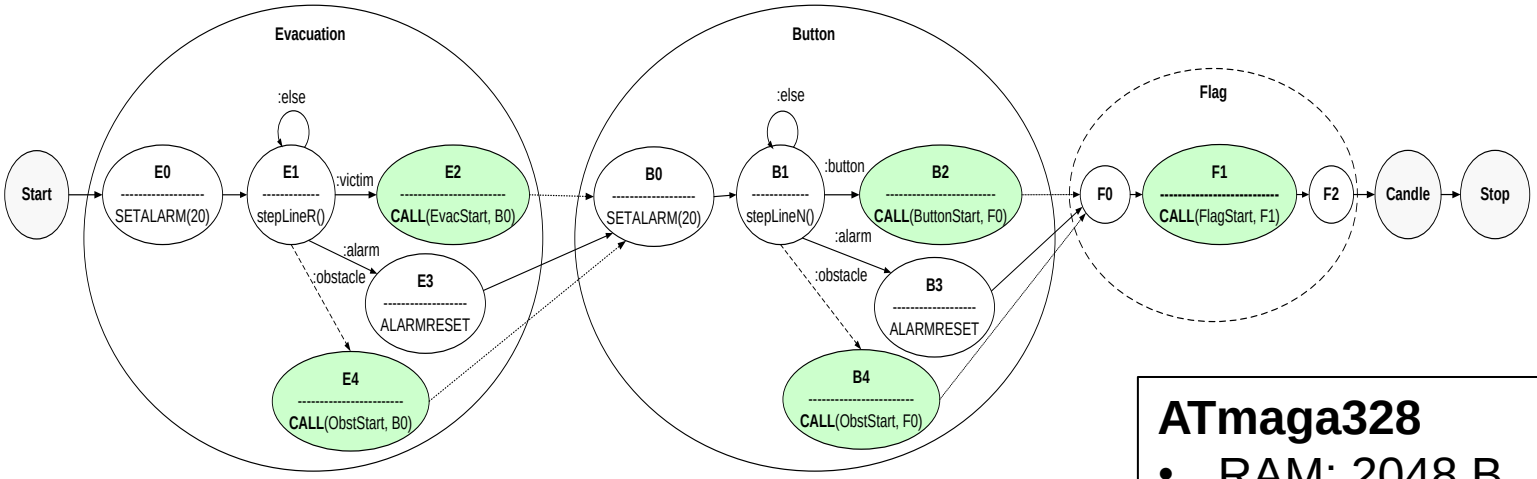
[ (4): G, (5): G2/f2 ]
=> f1() не срабатывает

…

[ (4): B, (5): C ] =>
P() не срабатывает

1. Setup()
2. Q := Start
3. **X** := ReadSensors()
4. **if ExecProc (Q) = 1 then**
5. **Q := F(Q, X)**
6. goto 3

# От матрицы переходов к списку рёбер




**ATmaga328**
- RAM: 2048 B
- FLASH: 32 K
- EEPROM: 1024 B

Разреженный граф

- |Q| = 32, |X|= 14 =>

Размер матрицы переходов = 32*14 = 448

*Список рёбер {<src, dest, condition>}*

32 дуги (ребра): 32*3=96 элементов

# Фрагменты программы (1/3)

```
1. // Ребро графа
2. struct TArc
3. {
4.   byte src, dest; // источник, приемник
5.   int cond;       // условие перехода (пометка дуги)
6. };

7. // Собственно список рёбер (фрагмент):
8. #define NumArcs 32
9. TArc arc[NumArcs] =
10.{
11.  {qStart,qE0, sigALWAYS},
12.  // Evacuation
13.  {qE0,qE1, sigALWAYS},
14.  {qE1,qE1, sigELSE},
15.  {qE1,qE2, sigVictim},
16.  {qE1,qE3, sigAlarm},
17.  {qE1,qE4, sigObstacle},
18.  {qE3,qB0, sigALWAYS},
19.    ...
20.};
```

# Фрагменты программы (2.3)

```
1.  int FM(void)
2.  {
3.    for(int i=0;i<NumArcs;i++)
4.      if(arc[i].src==Q)
5.      {
6.        int cond = arc[i].cond;
7.        if(X[cond] && cond!=sigELSE) return arc[i].dest; // Что-то сработало
8.      }
9.    // Не нашли. Ищем ELSE
10.   for(int i=0;i<NumArcs;i++)
11.     if(arc[i].src==Q && arc[i].cond==sigELSE) return arc[i].dest;
12.   // Не нашли вообще ничего. Никуда не будем переходить.
13.   return Q;
14.}
```

# Фрагменты программы (3/3)

```
1.  int ExecProc(int q)
2.  {
3.    switch(q)
4.    {
5.      case qStart: break;
6.      case qE0: sysSETALARM(5); break;
7.      case qE1: stepLine(); break;
8.      case qE2: sysCALL(qEvacStart, qB0); return 0;
9.      case qE3: sysALARMRESET(); break;
10.     case qE4: sysCALL(qObstStart, qB0); return 0;
11.     ...
12.     case qB1: stepLine(); break;
13.     case qB2: sysCALL(qObstStart, qC0); return 0;
14.     case qB3: sysALARMRESET(); break;
15.     ...
16.   }
17. return 1;
18. }
```

# Особенности АП

**Сильные стороны АП**

- Простота и, как следствие, надежность. Код строится из небольших по объему «строительных» элементов.
- Наглядность алгоритма. Сложность алгоритма поведения перенесена из программного кода вовне. Особенности психологии образного человеческого восприятия.
- Гибкость алгоритма. Автоматное представление позволяет постепенно наращивать сложность и гибкость поведения робота, не ломая логику работы всей программы.

**Слабые стороны АП**

- Неизбежное «трюкачество». Теснота формальных рамок. Практические программистские или технические соображения (входной символ *x*, и вектором сигналов ***X***).
- Громоздкость автомата. Реализация того, что было естественно для «обычного» программирования превращается в наращивание структуры автомата. Основная проблема - свести вместе процедурную необходимость и автоматную возможность.
- Неестественность некоторых механизмов. Автоматный эквивалент работы со временем, вызовом подпрограмм и т.п. => неизбежное «трюкачество», и увеличение сложности автомата.

# Принципиальная ограниченность автомата

- *АП не претендует на универсальность. Метод хорош тогда, когда решаемая задача укладывается в рамки его – метода – модели.*

1. КА плохо умеет считать.
2. Возможности его памяти ограничены количеством и структурой связей его состояний. Это означает, что он может реализовывать управление лишь *регулярным* поведением.
3. Слабость представления рекурсивного поведения. КА не может описать и сохранить в памяти **контекст ситуации**.

- *Решение о применении принципов АП принимается тогда, когда мы четко сопоставили возможности модели и решаемую задачу.*

С практической точки зрения:

- ограниченность автомата проявляется главным образом в реализации функции переходов *F(Q, X)*:
  «Трюкачество» с вектором **X** так и не смогло привести к реализации возможности реализации **произвольного условия перехода**.

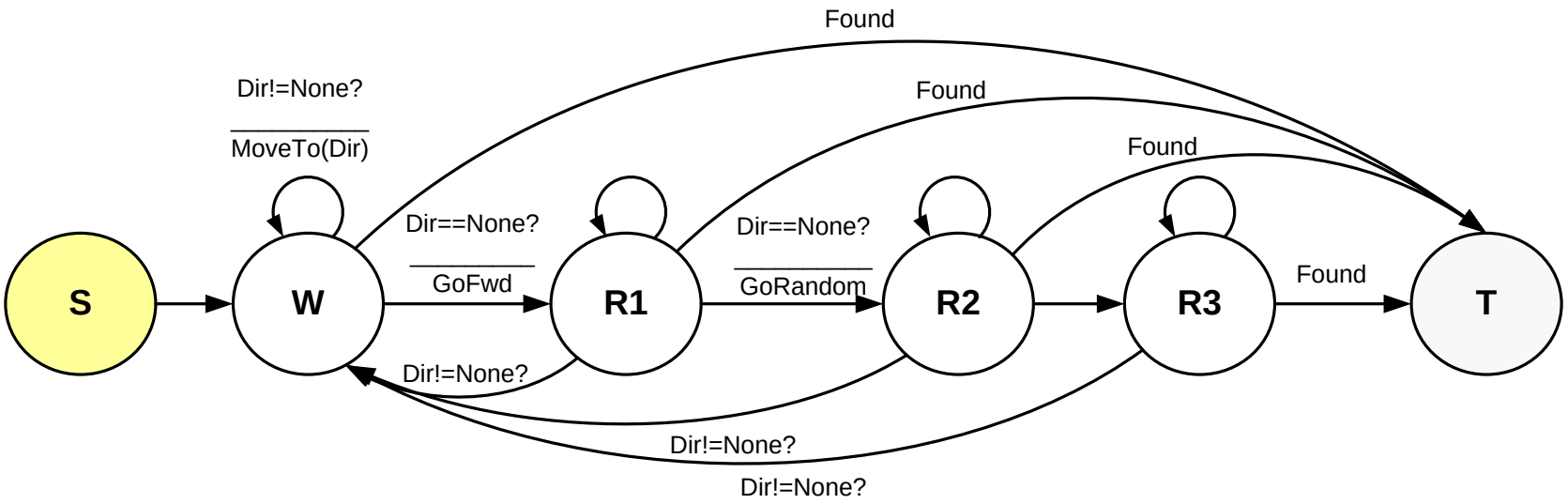
# Автомат Мили и автомат Мура

- Теоретически они равноправны.
- В какой-то степени выполнение процедуры на переходе (автомат Мили) более универсально и удобно, чем в состоянии (автомат Мура).
- Автомат Мили более «лаконичен».
- «Процедурная емкость» автомата Мили выше. Если у обоих автоматов будет по |Q| состояний и |S| входных символов, то количество процедур у автомата Мура будет ограничено величиной |Q| (состояние-процедура), то у автомата Мили их может быть |Q|*|S|.
- Платой за гибкость и емкость автомата Мили является усложнение процедур и формы представления:
- Вместо одной простой матрицы переходов ***M*** надо либо строить дополнительную матрицу выходов (т.е. процедур), либо усложнять саму матрицу ***M***.
- Используя автомат Мура, мы просто переложили эту работу на большее количество элементарных операций.

# Обобщенные автоматы

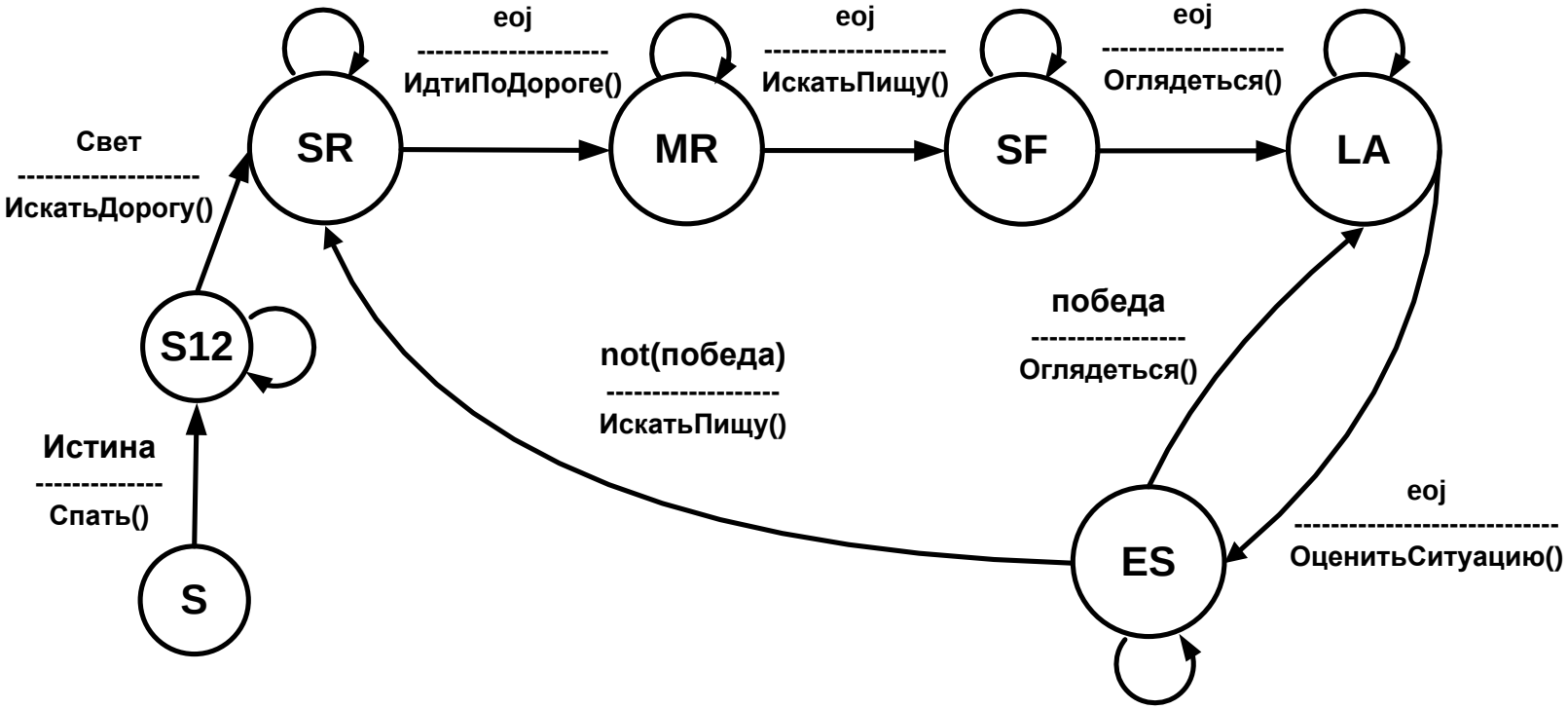
- Обобщенный КА – это параметризированный КА с предикатными условиями переходов. Параметры определяют направленность поведения.

- Пример обобщенного автомата, реализующего поисковую процедуру.

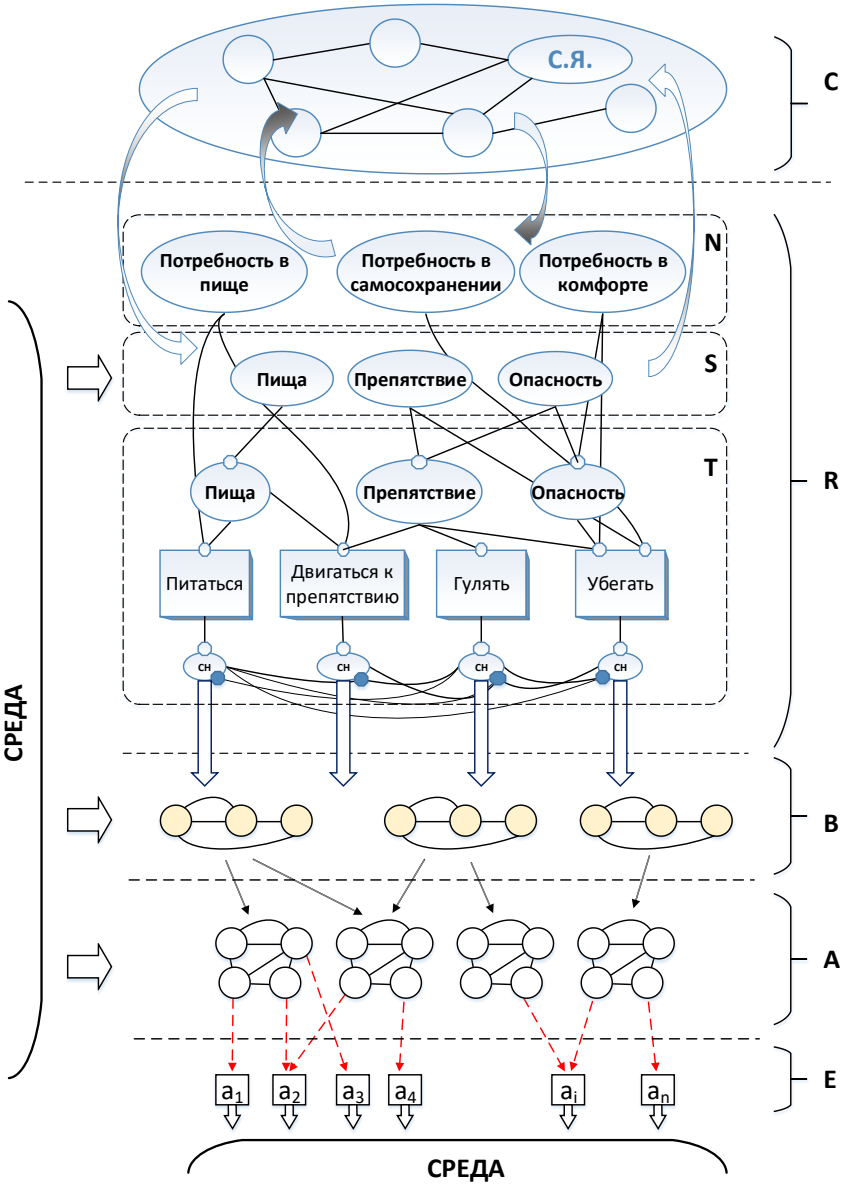


# Поведение. Мета-автоматы

- Суть его работы заключается в том, чтобы, в зависимости от результатов анализа состояния внешней и внутренней среды, активизировать тот или иной автомат, реализующий некоторое действие.
- Пример фрагмента мета-автомата, реализующего одну из моделей пищевого поведения анимата

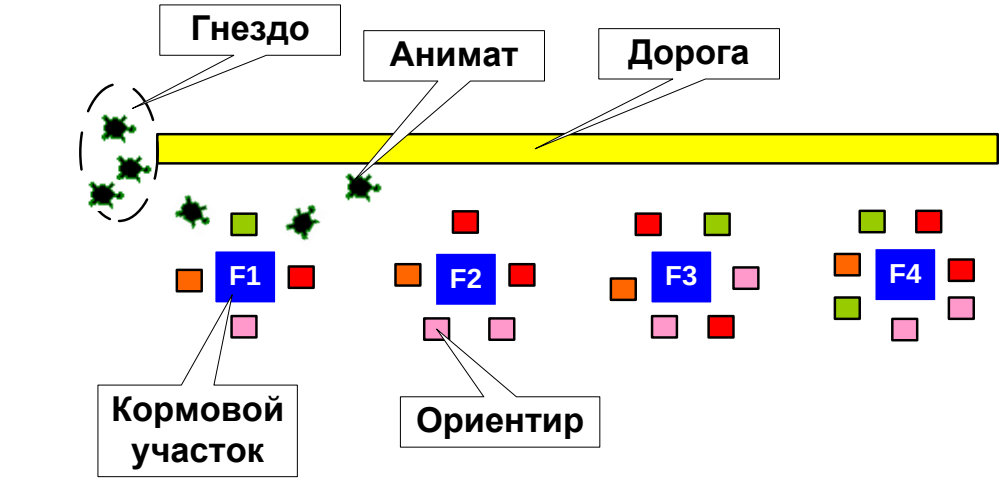


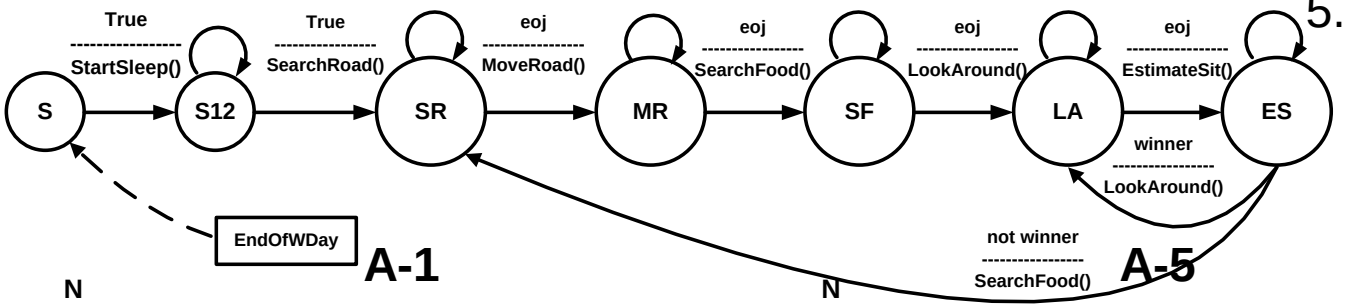
# Архитектура анимата

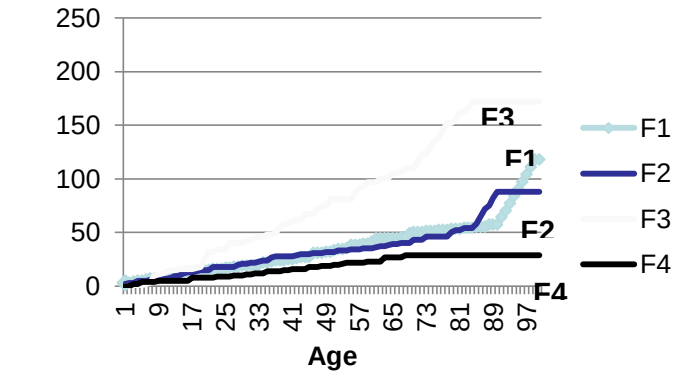


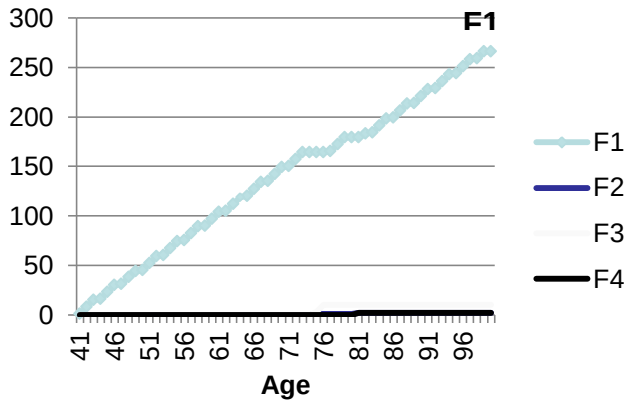

- E. Эффекторный уровень. Движения.
- A. Уровень действий. Реализуем с помощью КА.
- B. Уровень поведения. Реализуем с помощью мета-автоматов.
- R. Регулятивный уровень.
- C. Когнитивный уровень.

# Пример. Пищевое поведение




1. Поиск дороги.
2. Движение по дороге.
3. Поиск кормового участка.
4. Запоминание сцены.
5. Оценка ситуации.

# Заключение

АП — это лишь некая общая парадигма программирования, суть которой заключается в том, что создаваемая программа рассматривается как реализация некоторого управляющего автомата.

Цель – донести эти общие принципы, убедить в том, что для создания сложных поведенческих программ удобно использовать понятие автомата:

(1) представление алгоритма работы системы управления в виде автомата очень наглядно (граф автомата – это весьма удобный для восприятия объект),

(2) стараясь придерживаться **формальных** определений и механизмов, мы заставляем себя писать программу более строго и, главное, строить ее единообразным образом из элементарных составляющих (повышение культуры программирования).

# Что еще

АП – не единственная парадигма:

(1) Более сложные, развитые, «интеллектуальные» модели:

- ситуационное управление,
- интеллектуальное планирование,
- глубокие процедуры анализа, оценки и прогноза

(2) Радикальный минимализм, в котором все сводится к стимул-реактивному поведению, а вместо управляющего автомата используются, например, процедуры обработки прерываний.

В этом смысле АП является «золотой серединой» между сложностью решаемых задач и ограничениями ресурсов бортовой системы управления (и простоты решения).