Алгоритмы принятия решений
Decision making

Ст. преп. каф. ПМП Пучкин Максим Валентинович

Определения

- Decision making – the ability of a character to decide what to do. Given a set of knowledge, we need to generate a corresponding action from a set of possible actions.
- Методы принятия решений для игровых агентов, как правило, на основе некоторой входной информации (input);
- Под решением понимаем выбор одного элемента из некоторого конечного (или бесконечного?) множества – множества стратегий, множества ходов, множества состояний и проч.
- Таким образом, формализуем:

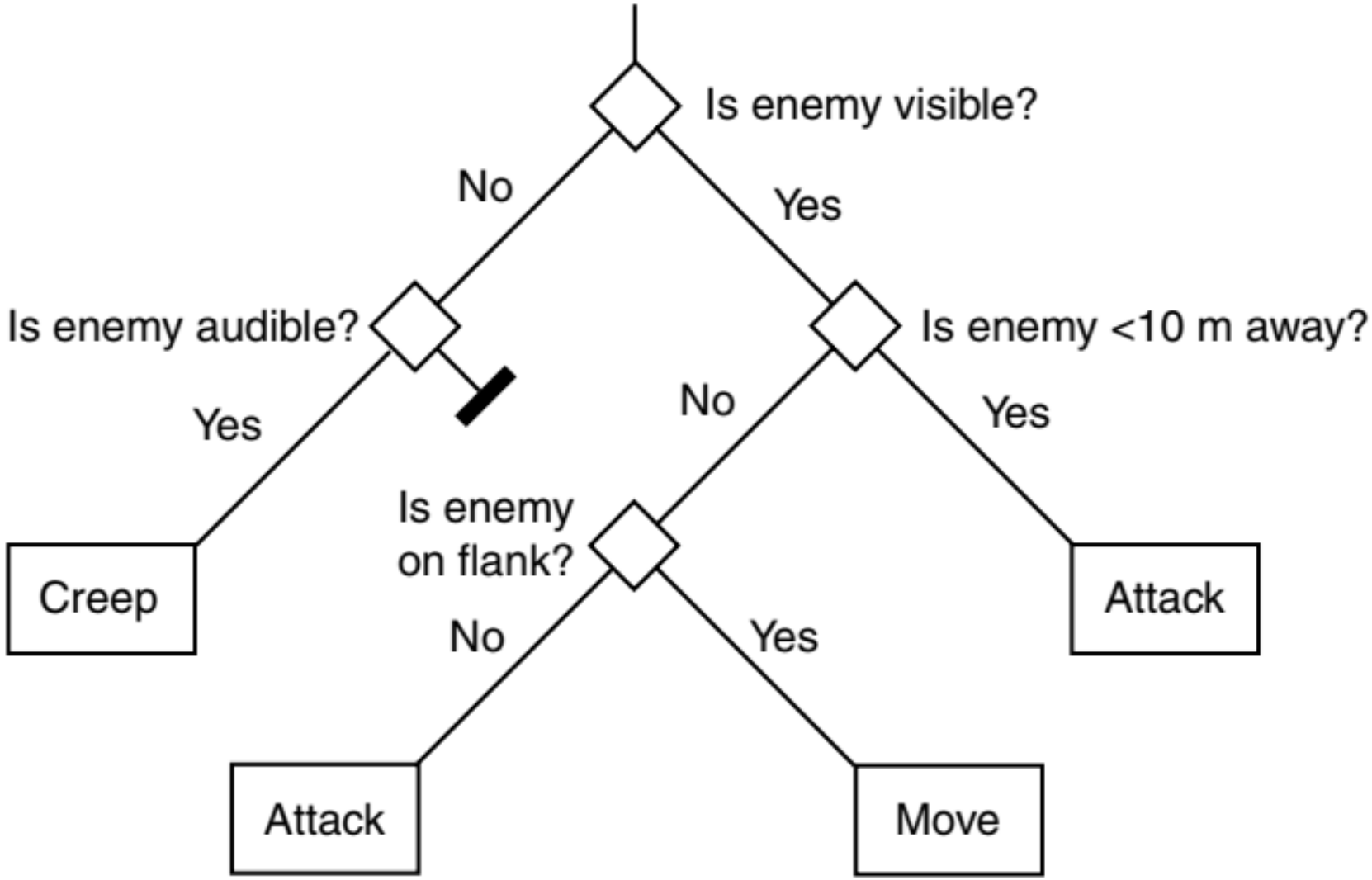
$$Decision: \ States \times Input \rightarrow Reactions$$

- *States* – множество возможных состояний агента, к которым можно отнести и управляющие воздействия от планировщика верхнего уровня.
- *Reactions* – множество возможных реакций агента.

Подходы

- Много *if … then* – а почему бы нет?
- Decision tree – деревья принятия решений. Довольно легко строятся, могут модифицироваться в процессе игры. Высокая эффективность;
- Finite state machine (FSM) – конечные автоматы. Эффективны, легко реализуемы и модифицируемы, соответствую человеческой логике.
- Expert system (Rule-based system) – «тяжёлая артиллерия». Используются редко – нужны «движки», навыки, а отличия от FSM на простых задачах невелики. Умеют рассуждать разумно.
- Artificial neural networks – сложно, непонятно, и долго считают.

Decision trees



Decision trees

- Дерево с начальным узлом – root.
- Бинарные – в каждом узле да/нет, но в общем случае могут быть и ветвящимися.
- Листья – отдельный тип, отвечающий за конкретное решение.
- Формально могут быть не деревьями – вопрос оптимизации, впрочем, принципиально реализация не отличается.
- Типы вершин – логические, перечислимые, диапазоны (ООП нам в помощь).
- Знания, как правило, ссылаются на глобальную область, а вот собственно деревья для каждого агента свои.
- Недостаток – начинаем каждый раз от корня (ну, не такой уж и недостаток).

Как построить?

- Ручками (или карандашами) – обычно строятся вручную, благо само представление в виде дерева позволяет уменьшить число ошибок.
- Можно строить автоматически – например, как в игре «Животные», однако возникает вопрос устойчивости к ошибкам.
- Вопросы эффективности – балансировка. Дерево должно быть сбалансированным, но можно ли его балансировать? Подумать!
- При построении используется различные виды вершин, что, строго говоря, добавляет сложностей – для диапазонных типов, к примеру, нужно вносить разбиения.
- Сложность определения релевантных признаков.

Пример

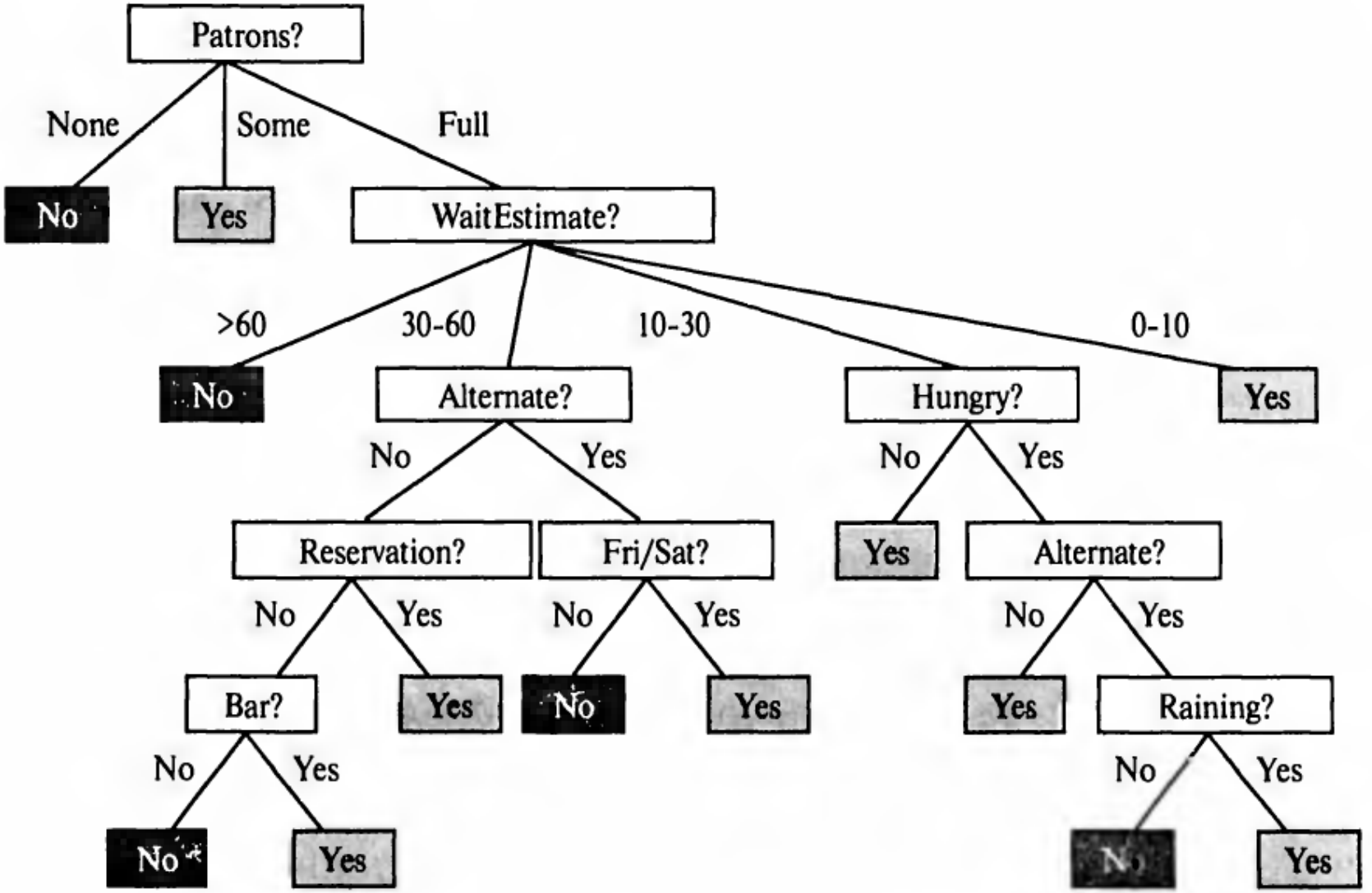


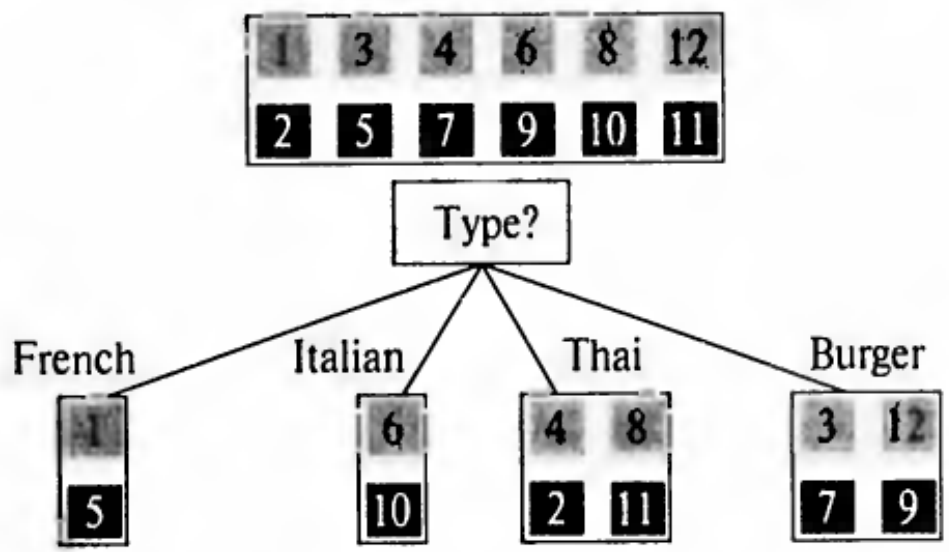
Рис. 18.2. Дерево решений для определения того, следует ли подождать, пока освободится столик

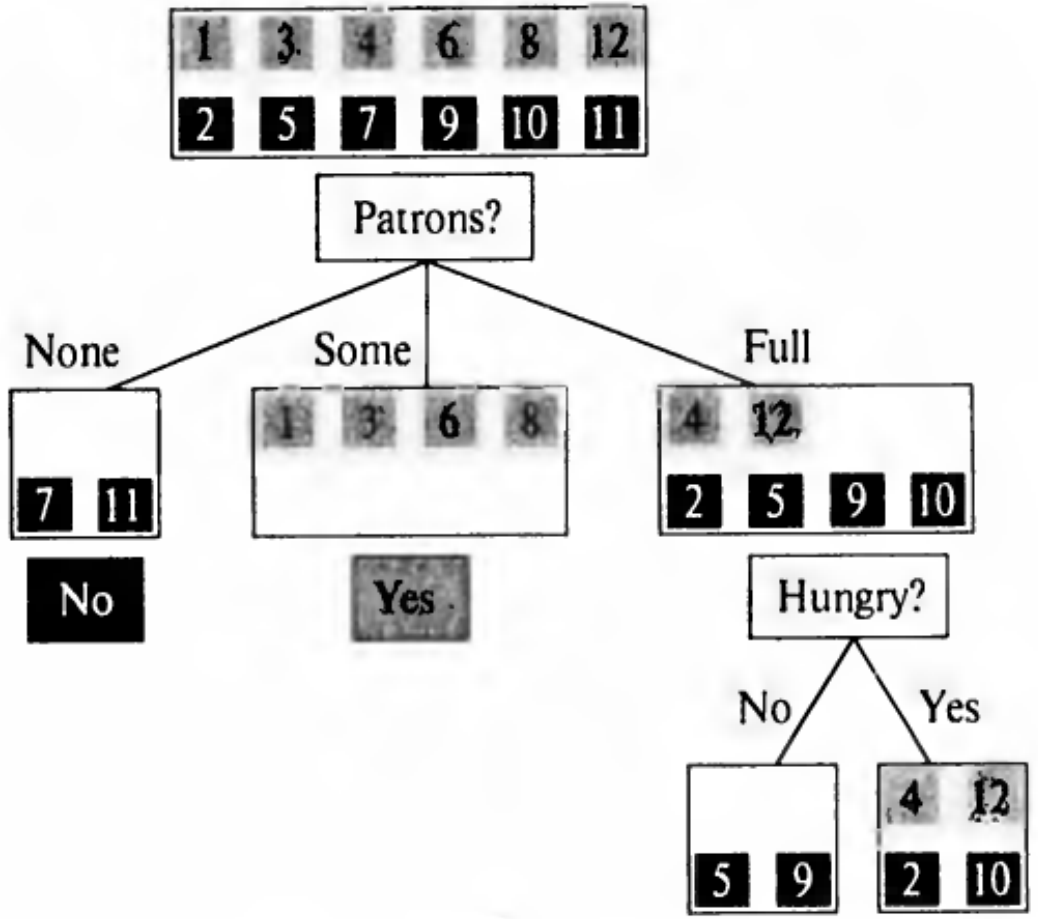
Стюарт Рассел, Питер Норвиг. Искусственный интеллект – современный подход, 2 изд. Стр. 871

Обучающая выборка

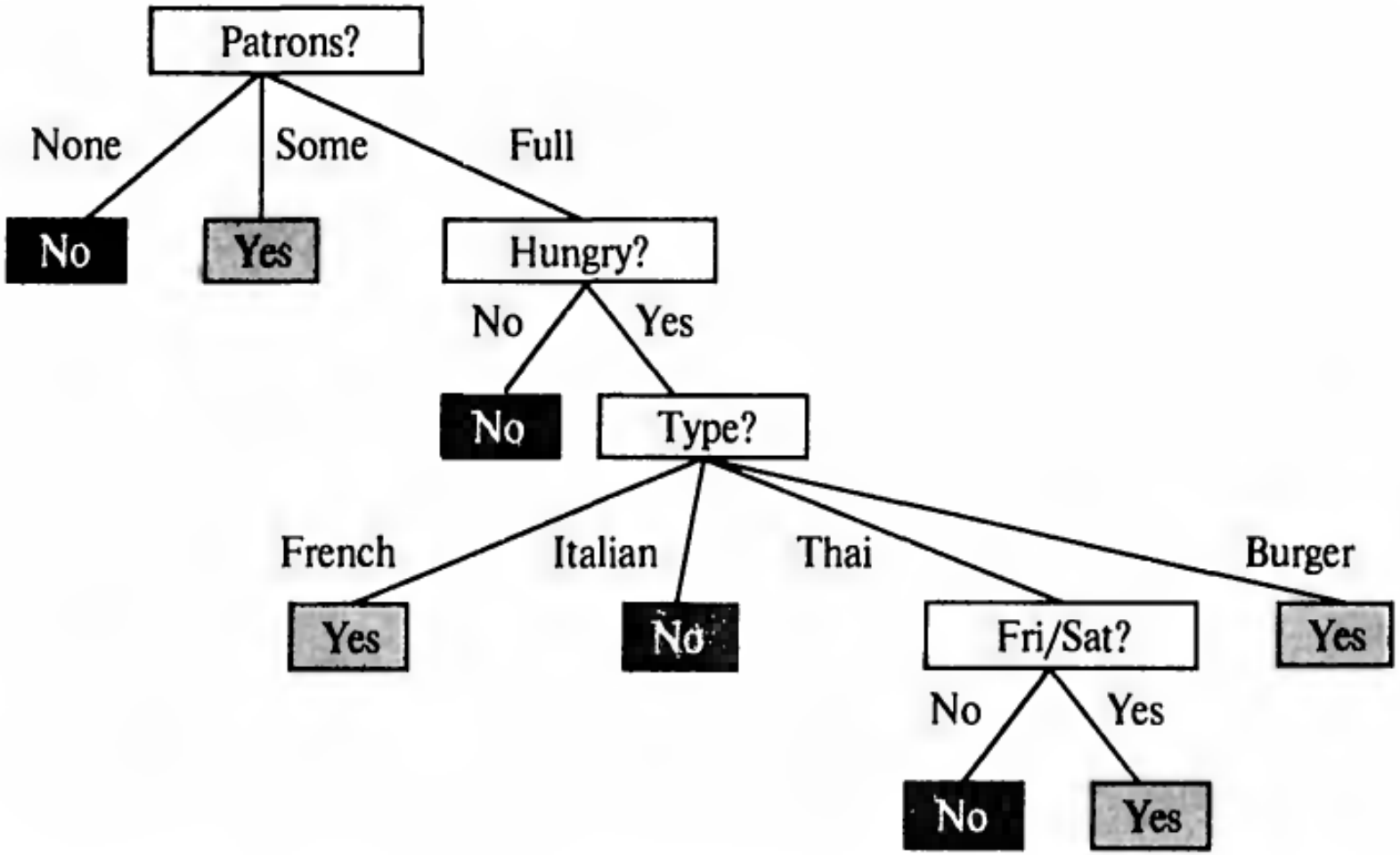
	Атрибуты										Цель
Пример	Alt	Bar	Fri	Hun	Pat	Price	Rain	Res	Type	Est	WillWait
X_1	Yes	No	No	Yes	Some	$	No	Yes	French	0-10	Yes
X_2	Yes	No	No	Yes	Full	$	No	No	Thai	30-60	No
X_3	No	Yes	No	No	Some	$	No	No	Burger	0-10	Yes
X_4	Yes	No	Yes	Yes	Full	$	Yes	No	Thai	10-30	Yes
X_5	Yes	No	Yes	No	Full	$	No	Yes	French	760	No
X_6	No	Yes	No	Yes	Some	$	Yes	Yes	Italian	0-10	Yes
X_7	No	Yes	No	No	None	$	Yes	No	Burger	0-10	No
X_8	No	No	No	Yes	Some	$	Yes	Yes	Thai	0-10	Yes
X_9	No	Yes	Yes	No	Full	$	Yes	No	Burger	760	No
X_{10}	Yes	Yes	Yes	Yes	Full	$	No	Yes	Italian	10-30	No
X_{11}	No	No	No	No	None	$	No	No	Thai	0-10	No
X_{12}	Yes	Yes	Yes	Yes	Full	$	No	No	Burger	30-60	Yes

Анализ

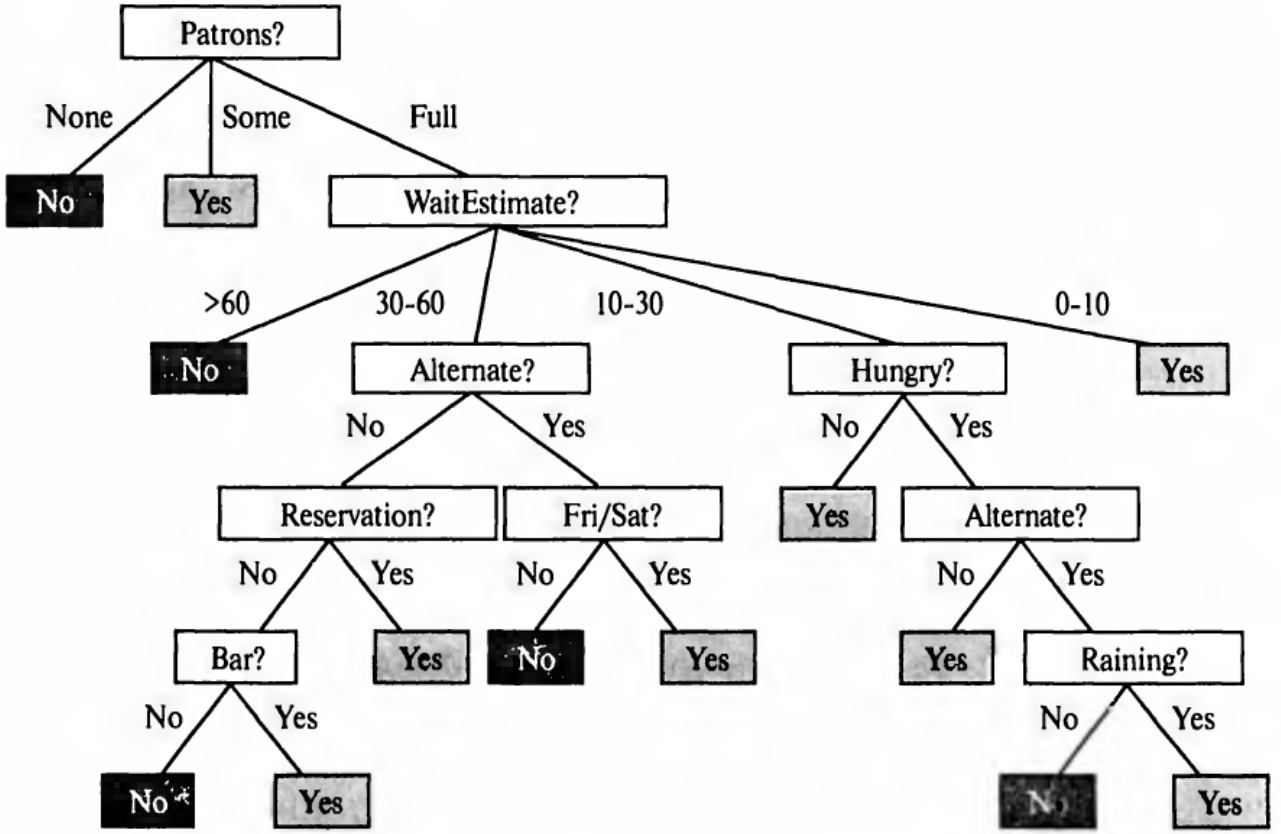

1 3 4 6 8 12
2 5 7 9 10 11
Type?
French
Italian
Thai
Burger
1
5
6
10
4 8
2 11
3 12
7 9

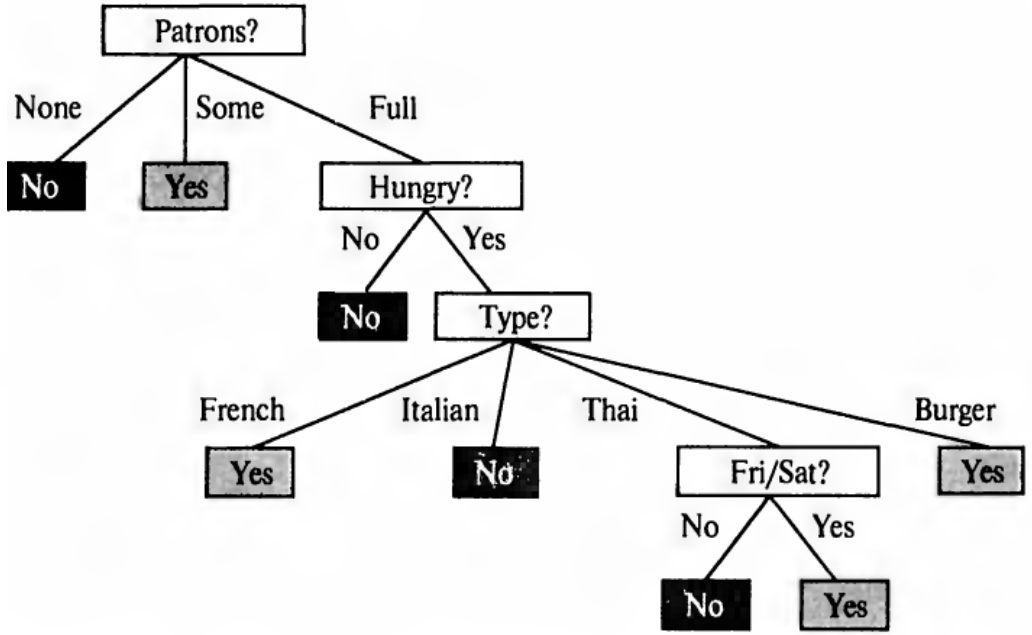

1 3 4 6 8 12
2 5 7 9 10 11
Patrons?
None
Some
Full
7 11
No
1 3 6 8
Yes
4 12
2 5 9 10
Hungry?
No
Yes
5 9
4 12
2 10

Решение


Patrons?
None
Some
Full
No
Yes
Hungry?
No
Yes
No
Type?
French
Italian
Thai
Burger
Yes
No
Fri/Sat?
Yes
No
Yes
No
Yes

Сравнение


Patrons?
None
Some
Full
No
Yes
WaitEstimate?
>60
30-60
10-30
0-10
No
Alternate?
Hungry?
Yes
No
Yes
No
Yes
Reservation?
Fri/Sat?
Yes
Alternate?
No
Yes
No
Yes
No
Yes
Bar?
Yes
No
Yes
Yes
Raining?
No
Yes
No
Yes
No
Yes


Patrons?
None
Some
Full
No
Yes
Hungry?
No
Yes
No
Type?
French
Italian
Thai
Burger
Yes
No
Fri/Sat?
Yes
No
Yes
No
Yes

Выбор признаков

Для бинарных – на основе количества информации (теория Шеннона и Уивера).

$$I\big(P(v_1), P(v_2), \dots, P(v_n)\big) = \sum_{i=1}^{n} -P(v_i)\log_2 P(v_i) \text{ бит}$$

$$I\left(\frac{p}{n+p}, \frac{n}{n+p}\right) = -\frac{p}{n+p}\log_2\frac{p}{n+p} - \frac{n}{n+p}\log_2\frac{n}{n+p}$$

Если есть некоторый атрибут *A*, имеющий *v* различных значений, и делящий обучающее множество *E* на $E_1..E_v$. Каждое подмножество E_i включает p_i положительных примеров, и n_i отрицательных.

Выбор признаков

Остаток информации после перехода по *i*-му атрибуту:

$$Remainder(A) = \sum_{i=1}^{v} \frac{p_i + n_i}{p + n} I\left(\frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i}\right)$$

Приращение информации после анализа *i*-го атрибута

$$Gain(A) = I\left(\frac{p}{n + p}, \frac{n}{n + p}\right) - Remainder(A)$$

Таким образом, изначально выбираем тот атрибут, который обеспечивает наибольшее приращение информации при анализе. Далее – аналогично.

ID3

Рассмотренный алгоритм описан Куинланом – Iterative Dichotomizer 3. Имеет некоторые недостатки – в частности, работа с диапазонами (непрерывные параметры).

Был улучшен – версия C4.5, CART

А потом дело дошло и до See5/C5.0
https://www.lshnk.me/2018/09/29/%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D1%8C%D1%8F-%D0%BF%D1%80%D0%B8%D0%BD%D1%8F%D1%82%D0%B8%D1%8F-%D1%80%D0%B5%D1%88%D0%B5%D0%BD%D0%B8%D0%B9-%D0%BD%D0%B0-c/

На текущий: ID3, CART, C4.5, C5.0, NewId, ITrule, CHAID, CN2 и другие.

GINI

Ещё один распространённый критерий: *Gini index* (Corrado Gene – итальянский статистик). Считается по формуле:

$$G_i = 1 - \sum_{k=1}^{n} p_{i,k}^2$$

где $p_{i,k}$ – отношение сущностей класса *k* среди сущностей *i*-го узла в обучающей выборке. Например, для 43 (13/30) имеем *Gini impurity* равным 1 – (13/43)^2 – (30/43)^2 ≈ 1 – 0.09 – 0.49 ≈ 0.42. Для «чистого» узла индекс равен 0 – это означает, что все элементы одного класса.

$$J(k, t_k) = \frac{m_l}{m} G_l + \frac{m_r}{m} G_r$$

Здесь m – количество сущностей при разделении, а t – порог для признака. Минимизируем. Рекурсивно повторяем. CART.

Проблемы

Компрометация – классическая проблема обучения, когда алгоритм учится не проблеме, а обучающей выборке.

Overfitting – переобучение. В принципе, верно и для деревьев решений, но выражается несколько по-другому.

Нерелевантные признаки – например, броски монетки в зависимости от дня недели.

Требуется способ проверки значимости атрибутов, и отбрасывание нерелевантных даже в том случае, если они могут влиять на классификацию. А для этого необходимо оценивать значимость атрибутов – например, по тем же самым значениям приращения информации.

Finite State Machines

Finite State Machine – абстракция, имеющая один вход, и, формально, один выход.

Чаще всего рассматривают работу на строках: на вход автомату подаются символы, на выходе формируется отклик. В простейшем случае выход – допускает (accepts) или отвергает (rejects) автомат строку.

Приоритетная область исследования – парсинг строк, синтаксические и лексические анализаторы …, область исследования – теория конечных автоматов.

Автомат в каждый момент времени находится в одном состоянии из множества возможных, и это множество конечно.

Детерминированы – каждое переход из одного состояния в другое однозначно определяется текущим состоянием и информацией на входе автомата.

Определение FSM

Формальное определение:

$$A = (Q, q_0 \in Q, A \in Q, \Sigma, \delta)$$

Q – (states), множество состояний (узлы графа);

q_0 – (initial state), начальное состояние;

A – (accepting state), допускающее состояние;

Σ – (alphabet), входной алфавит;

$\delta: Q \times \Sigma \to Q$ – функция переходов. Может быть определена не на всех элементах множества $\{Q \times \Sigma\}$, в таком случае подразумеваем, что автомат останавливается (неполная функция переходов).

Конечность автомата – два условия, нет самопроизвольных переходов между состояниями, и функция переходов не подразумевает случайность.

Представление и оптимизация FSM

Представление может быть задано таблицами, диаграммами переходов, UML или графами (что-то тут не то!).

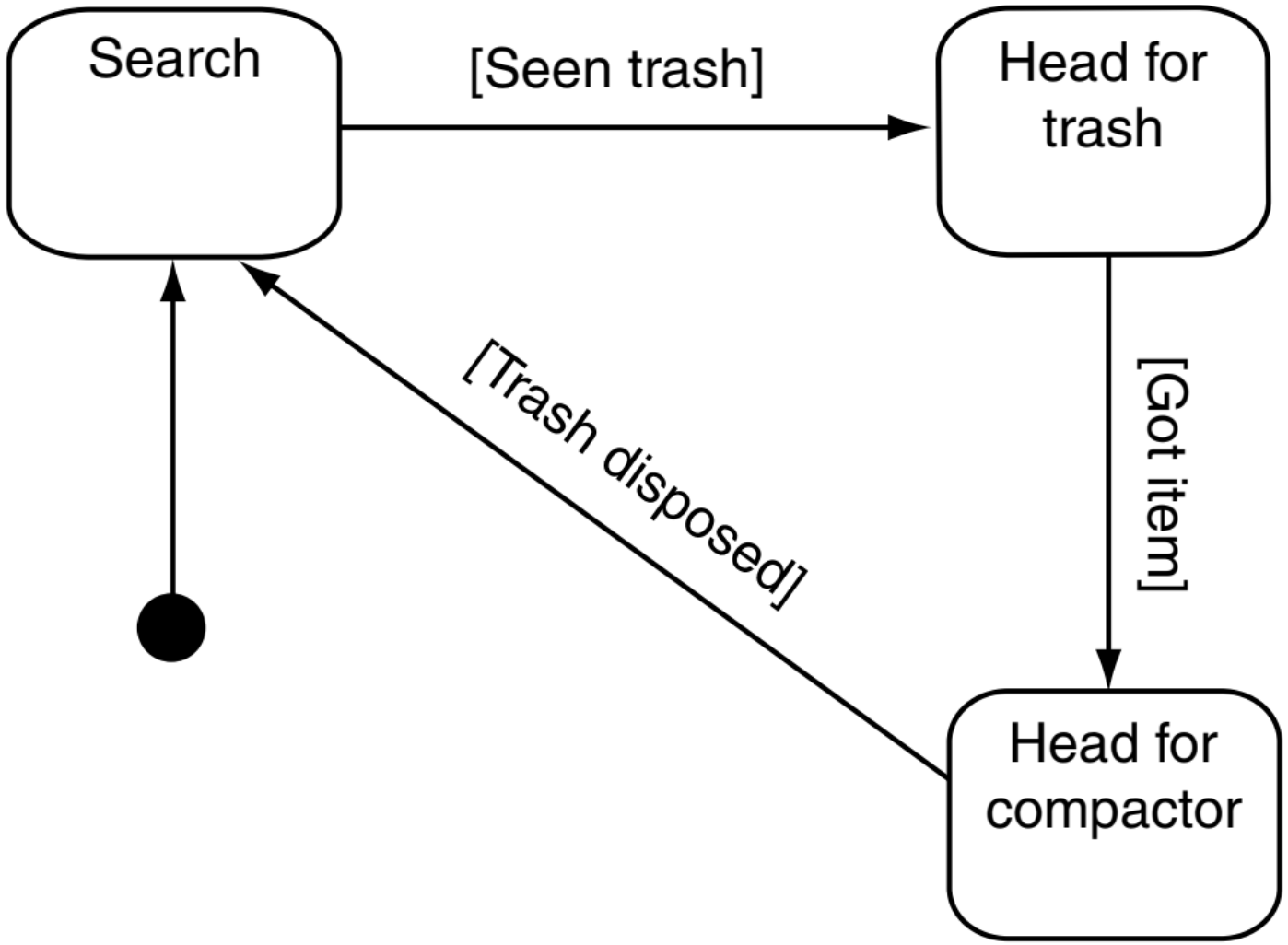
В игровых системах, равно как и в относительно небольших комплексах, представление может быть неявным – определяем множество состояний, и переходы.

Оптимизация подразумевает «упрощение» автомата – построение минимального или редукция к минимальному, эквивалентному с точки зрения функционирования (функции конечного состояния эквивалентны). Методы – алгоритмы Хопкрофта, Бжозовского и другие.

Для моделирования систем поведения ни способ представления, ни оптимизация особой роли не играют.

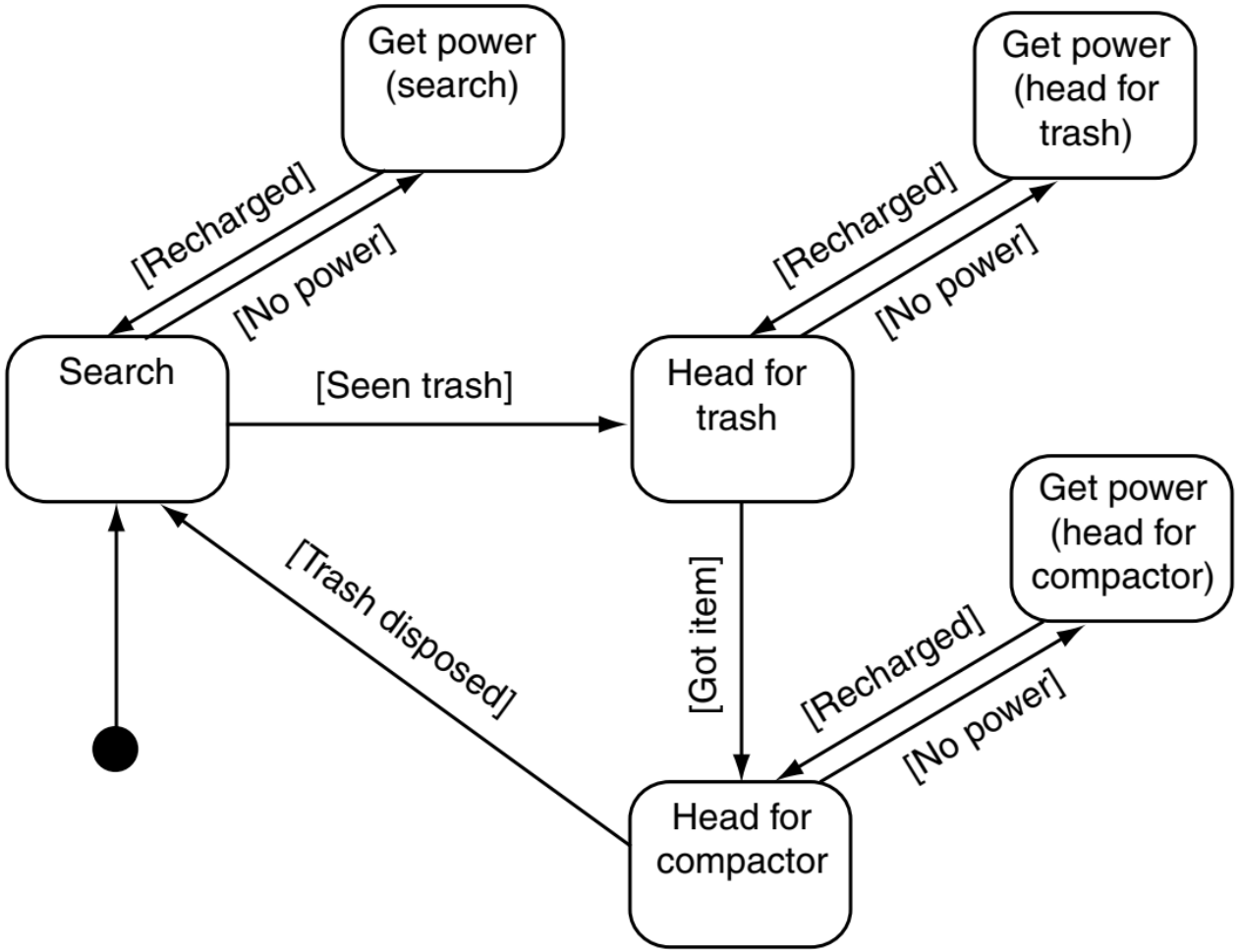
В игровом AI существенной проблемой FSA является сложность обучения (для большинства задач).

Пример


Search
[Seen trash]
Head for trash
[Got item]
Head for compactor
[Trash disposed]

Hierarchical FSM

Некоторые аспекты поведения может быть сложно реализовать – так называемые «Alarm behaviors» являются неплохим примером.

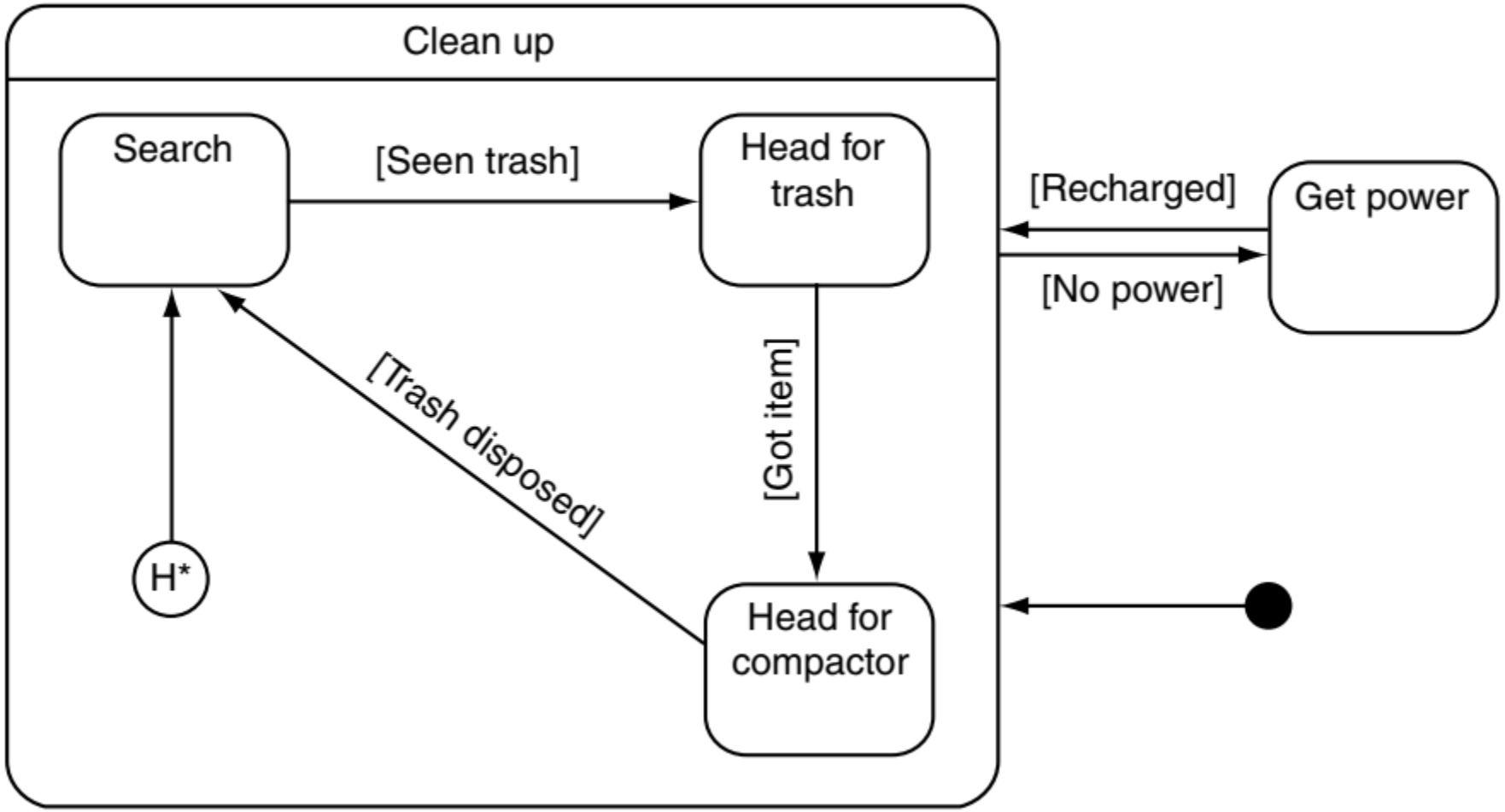


Hierarchical FSM

Решением является реализация многоуровневого автомата:

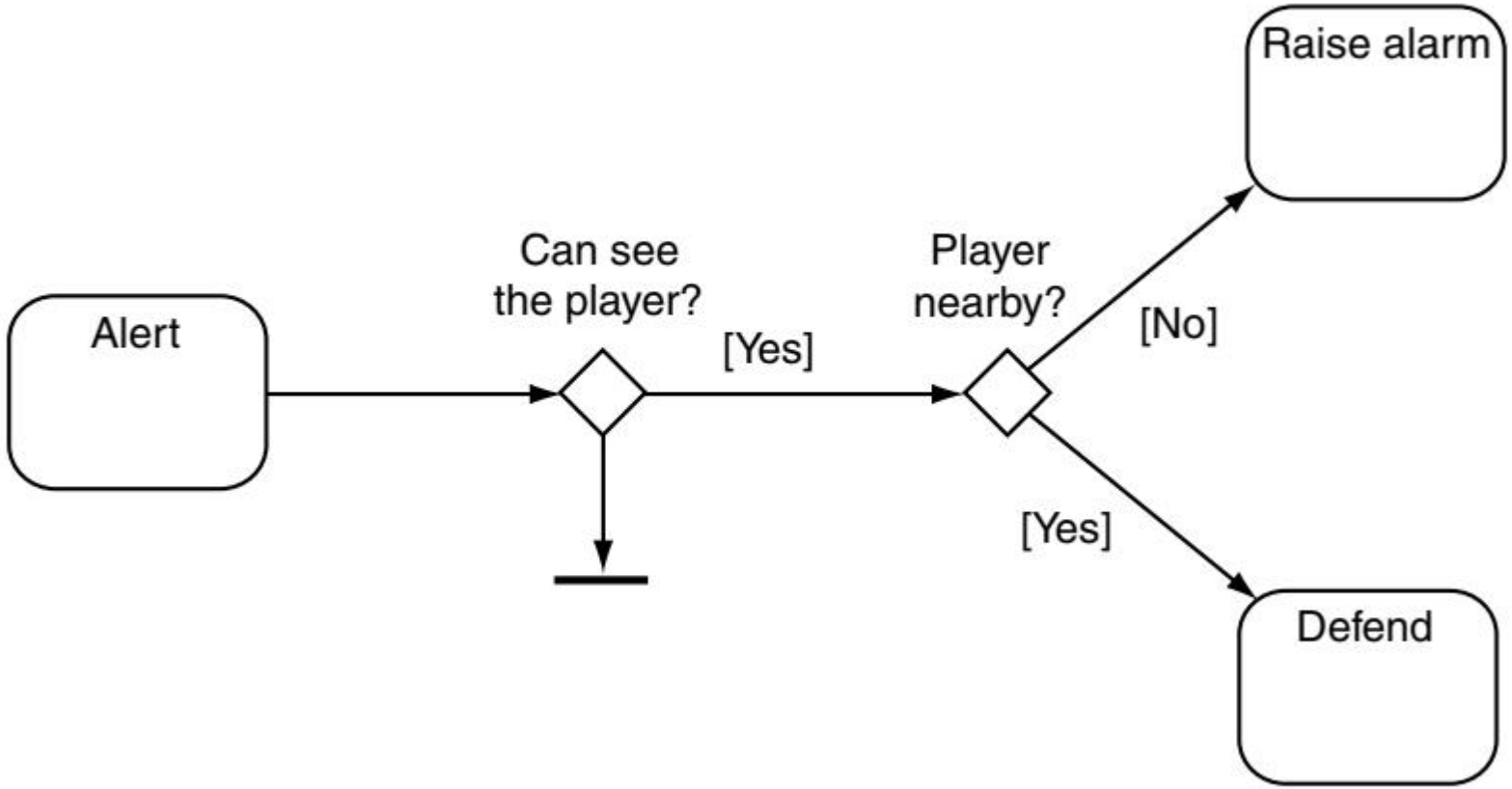
- FSM могут быть вложенными, в некоторых состояниях возможен переход во вложенные FSM;
- Получаем эффект «комбинированного состояния», при котором общее состояние системы является комбинацией нескольких состояний;
- Для каждого «вложенного» состояния необходимо указывать начальную точку входа – состояние, в которое попадаем изначально, а в дальнейшем текущая точка может запоминаться, возврат происходит в эту точку;
- Можно реализовывать переходы «между уровнями», либо только внутри иерархии.

Пример HFSM


Clean up
Search
[Seen trash]
Head for trash
[Recharged]
Get power
[No power]
[Got item]
[Trash disposed]
H*
Head for compactor

Комбинирование Decision-Trees и FSM

Довольно удобно реализуется



Behavior Trees

Похожи на HFSM, но основной строительный блок – задачи (*task*).

В общем случае задача – некоторый код, получающий определённое время на своё выполнение, и после возвращающий статус (успешно или нет).

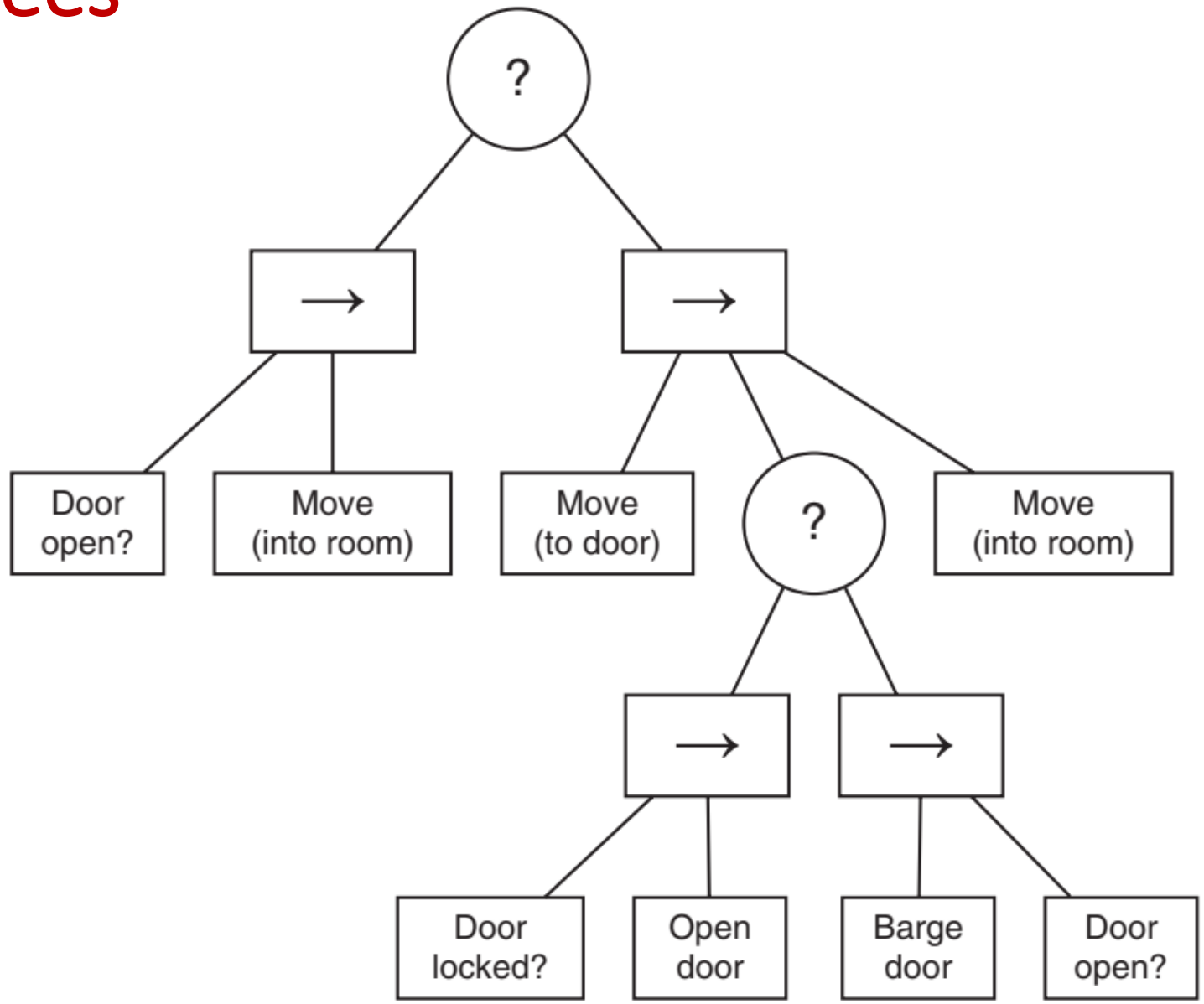
Виды задач могут быть следующие:

– Conditions (условия). Просто проверка на выполнение некоторого условия, возможно, довольно сложного – близость к противнику, наличие или отсутствие чего-либо, и тому подобное. Листья.

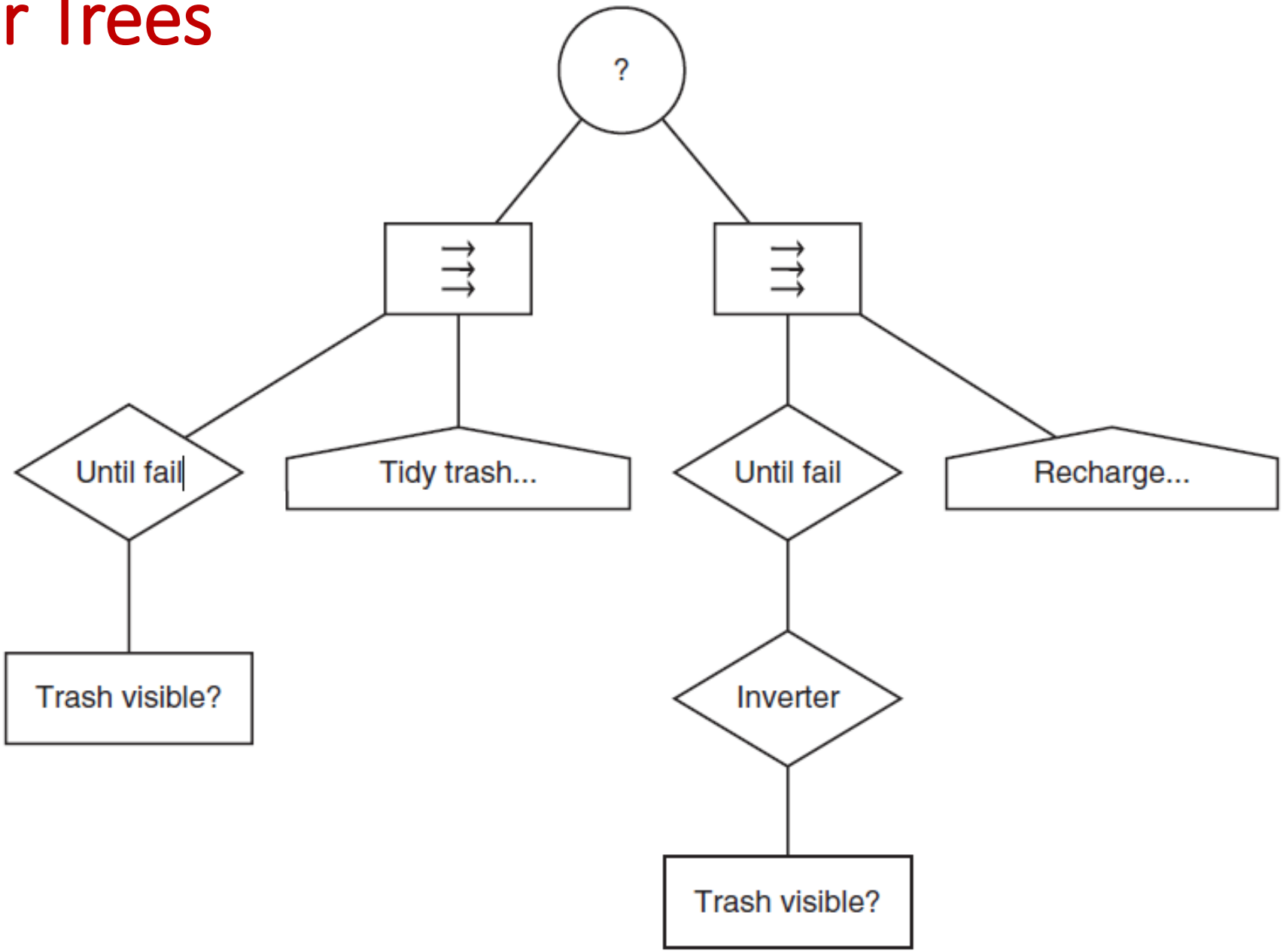
– Actions (действия). Любые действия – сделать что-либо агенту или как-либо изменить состояние мира. Листья.

– Compositions – композиции, внутренние вершины. Могут иметь дочерними другие вершины композиции. Различаются по типу, например Selector и Sequence.

Behavior Trees



Behavior Trees



Blackboard architecture

Архитектура «классной доски» (стратегия решения сложных системных задач с привлечением разнородных источников знаний, взаимодействующих через общее информационное поле) — модель управления, которая успешно применяется для решения этой и других задач, требующих координации различных процессов или источников знания.

«Классная доска» — центральная глобальная база данных, предназначенная для связи независимых асинхронных источников знаний.

Когда возникает необходимость? Система представляется набором модулей – агенты, подсистемы, наборы продукций, скрипты, боты, методы, да и вообще всё, что угодно. Установить детерминированный алгоритм обработки информации не представляется возможным.

Элементы

Под собственно «классной доской» понимают единое пространство для обмена данными – некоторыми сообщениями.

Есть набор Knowledge source, которые публикуют информацию на доске в определённом формате. Опубликованная информация доступна либо всем ассоциированным процессам, которые могут читать с «доски», либо их подмножеству.

Можно разделять источники знаний и процессы – например, процессы представляют экспертов, которые получают знания и «высказывают» своё мнение, то есть процесс двусторонний. Источники знаний, с другой стороны, не читают информацию с доски.

Вся деятельность асинхронная (в общем), управляется некоторым координатором.

Экспертов можно разделять по уровням.

Hearsay - II

Одна из первых реализаций – в системе Hearsay-2, для распознавания речи. Три измерения – время, иерархия и достоверность гипотез.

KS1 Форма волны (временная диаграмма) акустического сигнала.

KS2 Фонемы или возможные звуковые сегменты акустического сигнала.

KS3 Слоги, которые могут быть составлены из фонем.

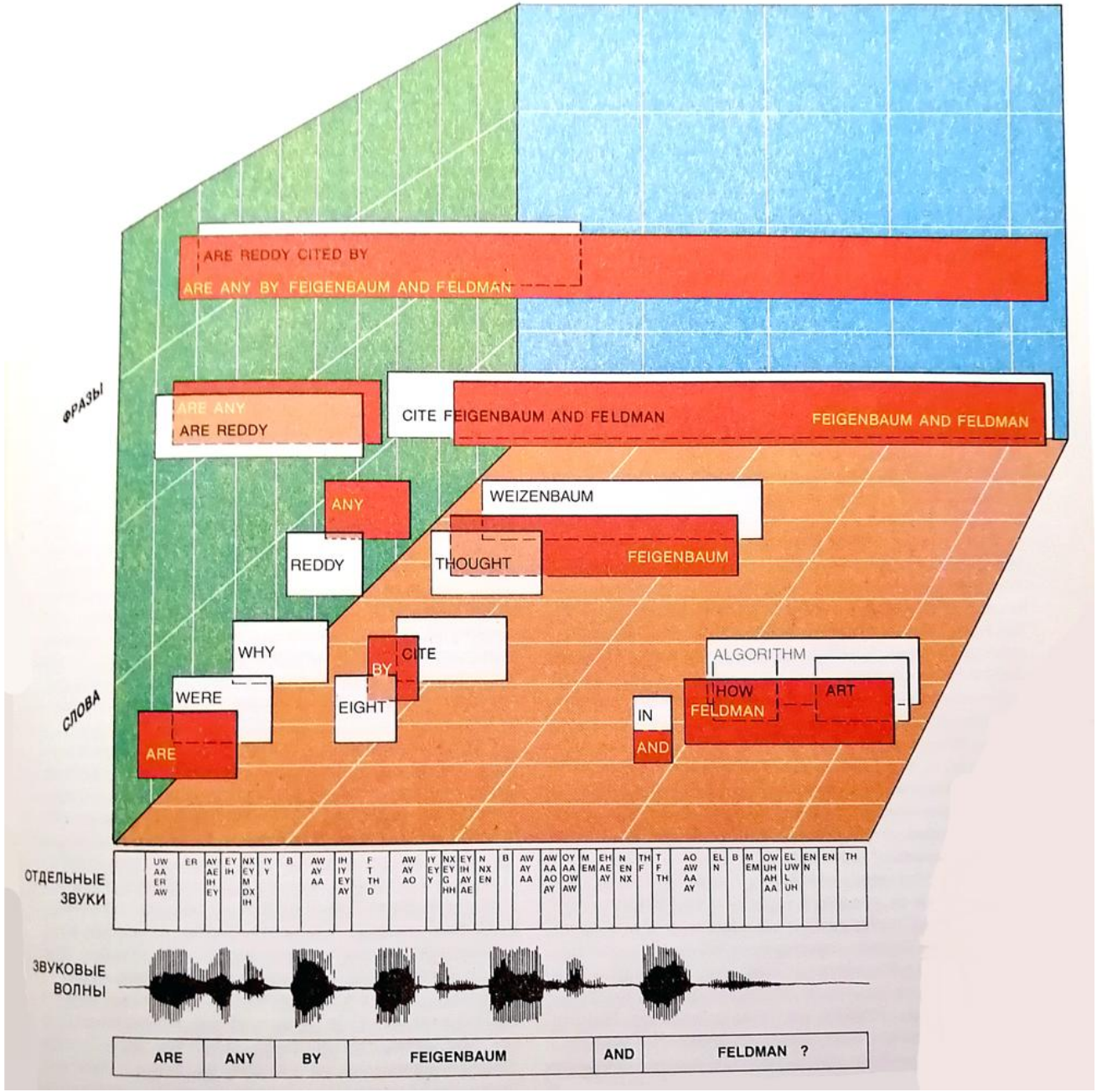
KS4 Возможные слова, результат анализа первого KS.

KS5 Возможные слова, результат анализа второго KS (обычно рассматриваются слова из различных частей данных).

KS6 Генерация возможных последовательностей слов.

KS7 Связывание последовательности слов в фразы.

Hearsay - II


ARE REDDY CITED BY
ARE ANY BY FEIGENBAUM AND FELDMAN
ФРАЗЫ
ARE ANY
ARE REDDY
CITE FEIGENBAUM AND FELDMAN
FEIGENBAUM AND FELDMAN
ANY
WEIZENBAUM
REDDY
THOUGHT
FEIGENBAUM
WHY
BY
CITE
ALGORITHM
СЛОВА
WERE
EIGHT
HOW
ART
FELDMAN
IN
ARE
AND
ОТДЕЛЬНЫЕ ЗВУКИ
ЗВУКОВЫЕ ВОЛНЫ
ARE
ANY
BY
FEIGENBAUM
AND
FELDMAN ?

Hearsay - II

Каждая гипотеза находится на доске на одном из уровней и несет определяющую метку, выбранную из набора, подходящего для этого уровня (например, слово FLYING, слог ING или звук NG). Гипотеза содержит дополнительную информацию, включая ее временные координаты в пределах произнесенного высказывания и рейтинг достоверности.

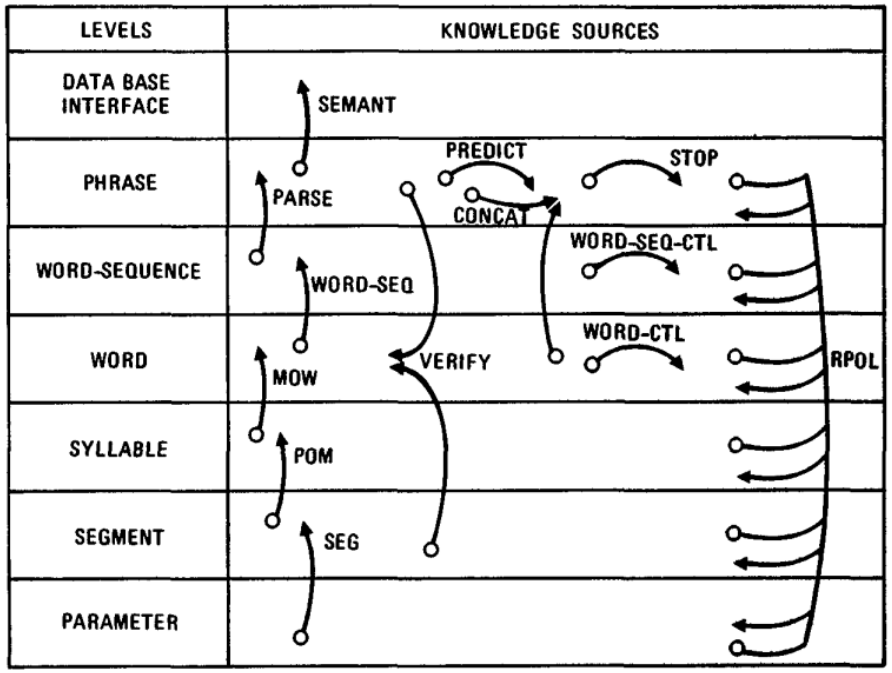


FIGURE 2 The levels and knowledge sources of September 1976. KSs are indicated by vertical arcs with the circled ends indicating the input level and the pointed ends indicating output level.

Hearsay - II

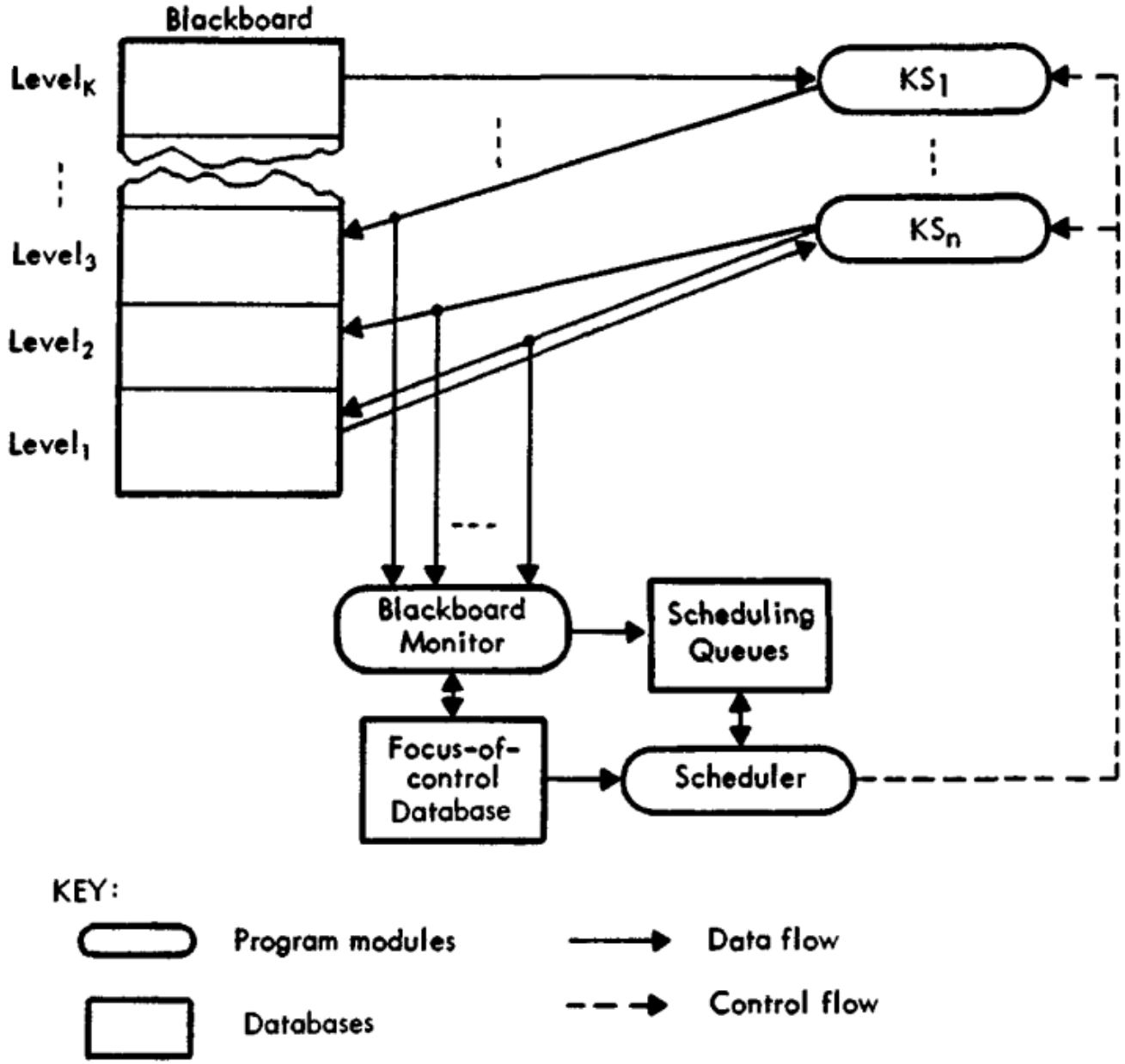
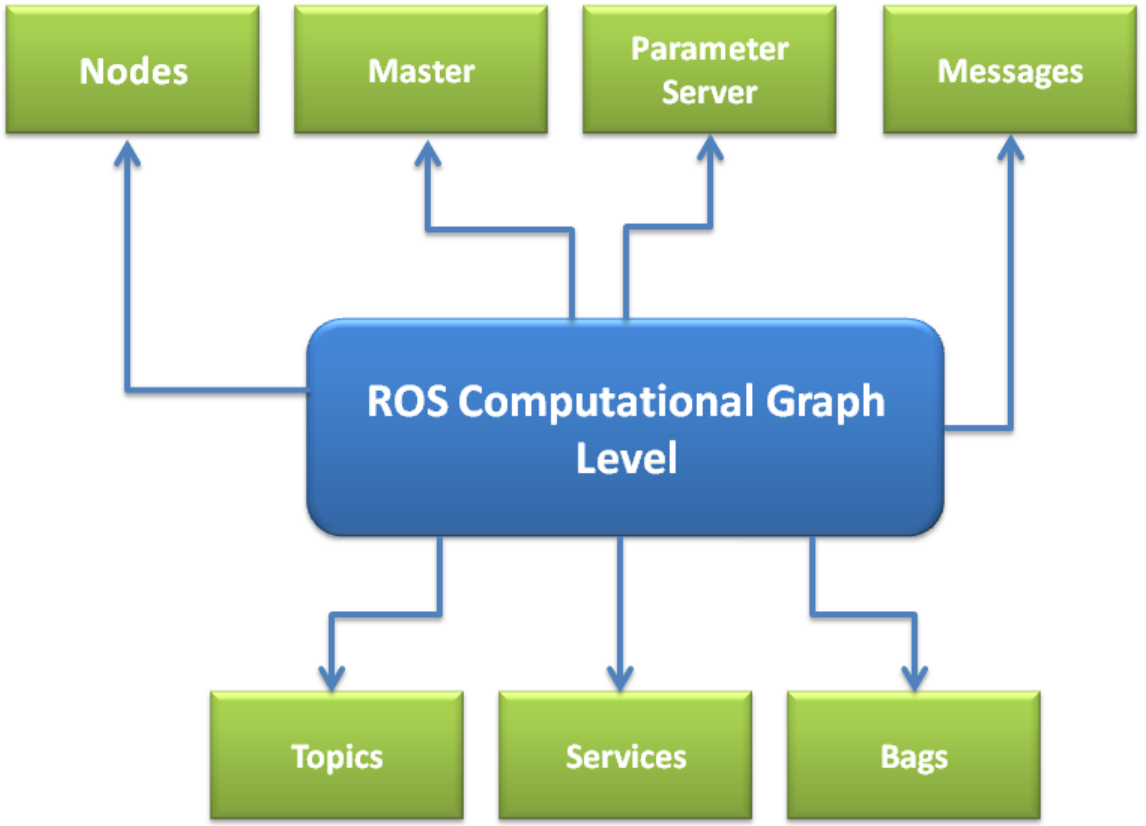


FIGURE 4. Schematic of the Hearsay-II architecture.

Robot operating system

Надстройка над Linux, определяющая инфраструктуру для создания сложных модульных систем.

Позиционируется как платформа для разработки софта для роботов.



Robot operating system

Nodes: процессы, выполняющие вычисления. Используя методы связи ROS, они могут общаться друг с другом и обмениваться данными. Одна из целей узлов ROS – создавать простые процессы, а не большие, со всеми функциями. Узлы ROS имеют простую структуру и легко поддаются отладке.

Master: диспетчер ROS обеспечивает регистрацию имен и поиск остальных узлов. Узлы не смогут находить друг друга, обмениваться сообщениями или вызывать службы без ROS Master. В распределенной системе диспетчер может функционировать на одном компьютере.

Parameter server: база данных, сервер параметров позволяет хранить данные в одном месте. Все узлы могут обращаться к этим значениям и изменять их. Сервер параметров является частью ROS Master.

Robot operating system

Messages: структуры для передачи данных. Существуют стандартные примитивные типы (целые числа, числа с плавающей запятой, логические и т. Д.). Допускается определять собственные типы сообщений.

Topics: каждое сообщение в ROS передается с помощью именованных шин, называемых темами. Публикация в темах и подписки на темы. Каждая тема имеет уникальное имя, и любой узел может получить доступ к этой теме и отправлять данные через нее, если у них есть правильный тип сообщения.

Services: аналоги RPC, если стандартная схема не работает.

Logging: ROS предоставляет систему регистрации для хранения данных, таких как данные датчиков, для дальнейшего использования.

Game AI. F.E.A.R.

STRIPS was developed at Stanford University in 1970, and the name is simply an acronym for the STanford Research Institute Problem Solver. STRIPS consists of goals and actions, where goals describe some desired state of the world we want to reach, and actions are defined in terms of preconditions and effects. An action may only execute if all of its preconditions are met, and each action changes the state of the world in some way.

Система игрового интеллекта в F.E.A.R. наиболее близка к STRIPS, и оперирует целями и действиями.

Состояния (states) – некоторое описание мира (в отличие от FSM), может рассматриваться как присваивание значений множеству переменных.

Game AI. F.E.A.R.

The design philosophy at Monolith is that the designer's job is to create interesting spaces for combat, packed with opportunities for the A.I. to exploit. For example, spaces filled with furniture for cover, glass windows to dive through, and multiple entries for flanking. Designers are not responsible for scripting the behavior of individuals, other than for story elements. This means that the A.I. need to autonomously use the environment to satisfy their goals.

Но если просто разместить игрового бота в мире и позволить ему увидеть игрока, то не произойдёт ничего – у бота нет целей.

В WorldEdit нужно было назначать Цели (Goals), точнее, наборы целей для игрового A.I.

Базовые алгоритмы поиска соответствуют таковым в задачах поиска (A*, Dijkstra и другим).

Game AI. F.E.A.R.

Goal Sets – набор целей. Он может совпадать у различных ботов, например, у ассасина и солдата, хотя способы достижения целей будут различаться.

Более того, набор целей GDC(Attack, Patrol), приводящие к различным действиям для солдата и ассасина, могут быть назначены и крысе, которая может патрулировать, но не имеет возможности никого убивать – нет соответствующего набора действий.

Action Sets – доступные наборы действий, которые приводят к изменению мира, и могут рассматриваться как операторы/дуги в процессе поиска решения.

Game AI. F.E.A.R.

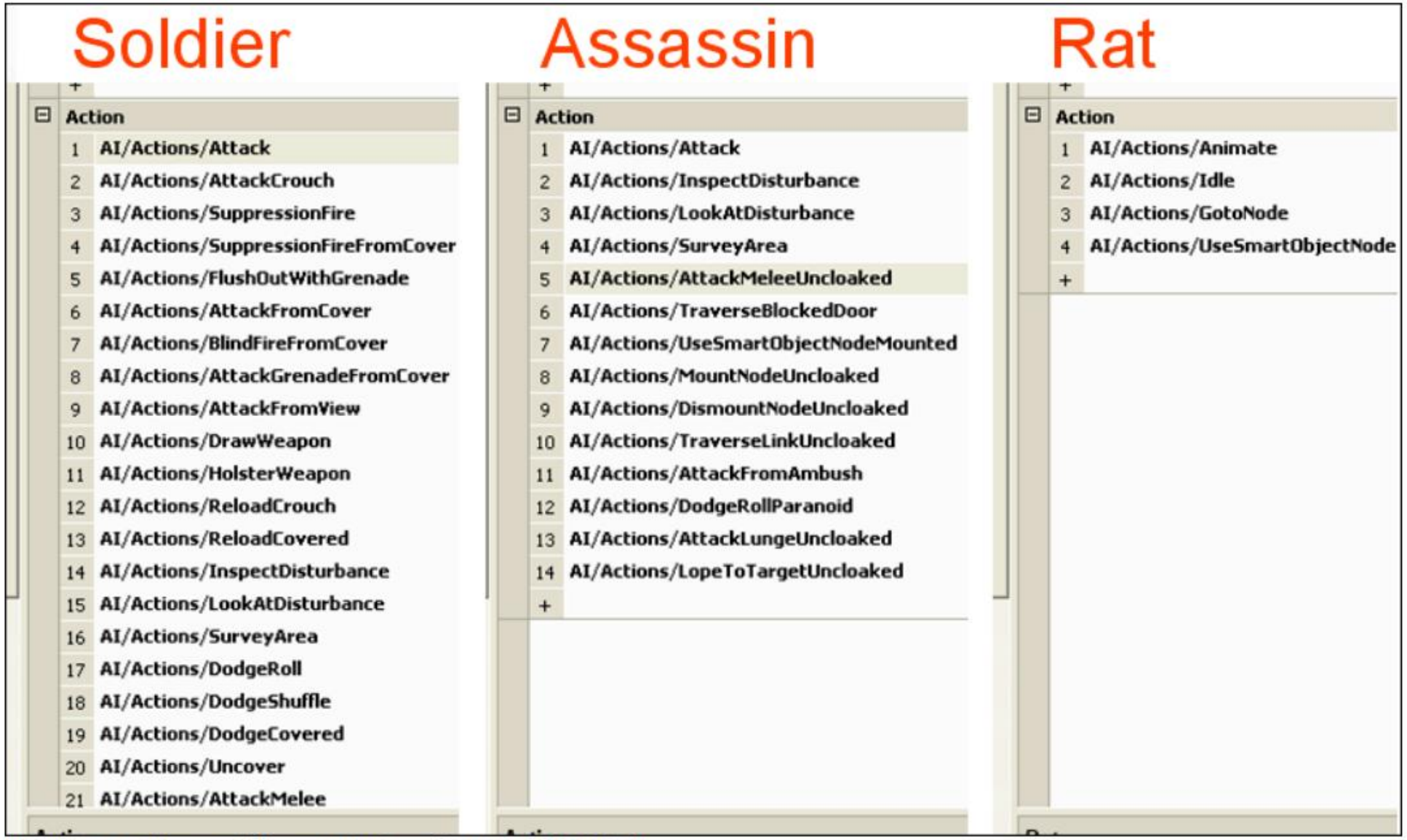


Figure 4: Three different Action Sets in GDBEdit.

Достоинства разделения целей и действий

For example, we had out of shape policemen in NOLF2 who needed to stop and catch their breath every few seconds while chasing. Even though only one type of character ever Three States and a Plan: The A.I. of F.E.A.R. 8 Game Developers Conference 2006 exhibited this behavior, this still required a branch in the state machine for the Chase goal to check if the character was out of breath. With a planning system, we can give each character their own Action Set, and in this case only the policemen would have the action for catching their breath. This unique behavior would not add any unneeded complexity to other characters.

The modular nature of goals and actions benefited us on F.E.A.R. when we decided to add a new enemy type late in development. We added flying drones with a minimal amount of effort by combining goals and actions from characters we already had. By combining the ghost's actions for aerial movement with the soldier's actions for firing weapons and using tactical positions, we were able to create a unique new enemy type in a minimal amount of time, without imposing any risk on existing characters.

Game AI. F.E.A.R.

For example someone could sit down at a desk and do some work. The problem was that only the Work goal knew that the A.I. was in a sitting posture, interacting wit the desk. When we shot the A.I., we wanted him to slump naturally over the desk. Instead, he would finish his work, stand up, push in his chair, and then fall to the floor.

This was because there was no information sharing between goals, so each goal had to exit cleanly, and get the A.I. back into some default state where he could cleanly enter the next goal. Decoupling the goals and actions forces them to share information through some external working space. In a decoupled system, all goals and actions have access to information including whether the A.I. is sitting or standing, and interacting with a desk or some other object. We can take this knowledge into account when we formulate a plan to satisfy the Death goal, and slump over the desk as expected.

Game AI. F.E.A.R.

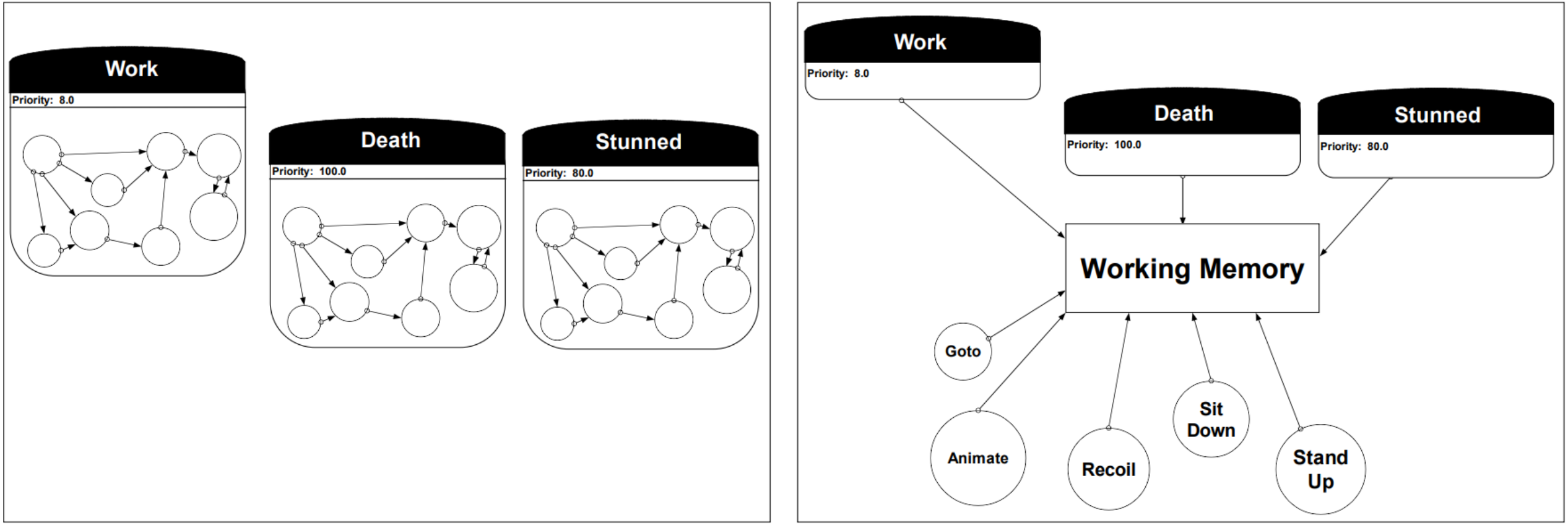


Figure 5: NOLF2 goals as black boxes (left) and F.E.A.R. decoupled goals and actions (right).

Game AI. F.E.A.R.

Разделение целей и действий даёт гораздо более гибкую и удобную систему, которую легче модифицировать и наполнять новыми действиями и целями.

Поведение стрелка в бою можно моделировать не скриптами, а действиями наподобие *Уклоняться, Прятаться, Устраивать засады и прочее*.

Также легко модифицировать различные поведенческие паттерны, добавляя, к примеру, рукопашную атаку, которая сопряжена с уже существующими целями.

FSM для таких задач тяжеловесны, и громоздки.

Game AI. F.E.A.R.

Динамическое планирование.

The third benefit of a planning system is the dynamic problem solving ability that re-planning gives the A.I. Imagine a scenario where we have a patrolling A.I. who walks through a door, sees the player, and starts firing. If we run this scenario again, but this time the player physically holds the door shut with his body, we will see the A.I. try to open the door and fail. He then re-plans and decides to kick the door. When this fails, he re-plans again and decides to dive through the window and ends up close enough to use a melee attack!

Возможно перестраивать планы «на лету»