سیستم‌های چندعاملی

درس ۵

معماری‌های عامل
در سیستم‌های چندعاملی

Agents Architectures in Multiagent Systems

کاظم فولادی قلعه
دانشکده مهندسی، پردیس فارابی
دانشگاه تهران

http://courses.fouladi.ir/mas

معماری عامل‌ها

معماری عامل‌های هوشمند
Architectures for Intelligent Agents

معماری انتزاعی (مجرد)
Abstract Architectures

- عامل‌های واکنشی محض
- عامل‌های واکنشی با حالت

معماری انضمامی (مجسم)
Concrete Architectures

- عامل‌های مبتنی بر منطق
- عامل‌های واکنشی
- عامل‌های باور، مطلوب، قصد *(BDI)*
- عامل‌های با معماری لایه‌ای

فرمول‌بندی انتزاعی عامل استاندارد

حالت‌های محیط
States of Environment

مجموعه حالات محیط:
عامل در هر لحظه در یکی از این حالت‌ها قرار دارد

$$S = \{s_1, s_2, \ldots\} \text{ of } environment\ states.$$

کنش‌های عامل
Actions of Agent

مجموعه کنش‌های عامل:
کنش‌هایی که عامل توانایی انجام آنها را دارد

$$A = \{a_1, a_2, \ldots\} \text{ of } actions.$$

عامل
Agent

عامل به عنوان یک تابع:
نگاشت دنباله‌ی حالت‌ها به کنش‌ها

$$Agent : S^* \rightarrow A$$

فرمول‌بندی انتزاعی محیط

تابع محیط

رفتار محیط با تابع env مدل می‌شود:

$$env : S \times A \rightarrow 2^S$$

$env(s,a)$ مجموعه‌ی همه‌ی حالت‌های ممکن در نتیجه‌ی انجام کنش a در حالت s

* اگر خروجی این تابع همیشه تک عضوی باشد، محیط قطعی وگرنه غیرقطعی است.

تاریخچه

HISTORY

تاریخچه
History

دنباله‌ای از حالت‌های محیط و کنش‌های عامل

$$h: s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \ldots \xrightarrow{a_{u-1}} s_u \xrightarrow{a_u} \ldots$$

حالت آغازین

$$action: S^* \rightarrow A$$ تابع عامل

$$env: S \times A \rightarrow 2^S$$ تابع محیط

$$\forall u \in N, a_u = action((s_0, s_1, \ldots, s_u)) \text{ and}$$

$$\forall u \in N \text{ such that } u > 0,\ s_u \in env(s_{u-1}, a_{u-1})$$

فرمول‌بندی انتزاعی عامل استاندارد

رفتار مشخصه، خصوصیت تغییرناپذیر

رفتار مشخصه
Characteristic Behavior

رفتار مشخصه‌ی یک عامل در یک محیط:
مجموعه‌ی همه‌ی تاریخچه‌هایی که در شرایط تعریف صدق می‌کنند.

خصوصیت تغییرناپذیر
Characteristic Behavior

خصوصیت تغییرناپذیر برای یک عامل در یک محیط:
خصوصیتی که در همه‌ی تاریخچه‌ها برقرار است.

نمادگذاری برای مجموعه‌ی همه‌ی تاریخچه‌های یک عامل در محیط

$$hist(agent, environment)$$

دو عامل معادل

عامل‌های معادل رفتاری
Behaviorally Equivalent Ag's

دو عامل **معادل رفتاری** در یک محیط هستند
اگر و فقط اگر مجموعه‌ی همه‌ی تاریخچه‌های آنها برابر باشد.

دو عامل **معادل رفتاری** (ساده) هستند
اگر و فقط اگر در همه‌ی محیطها معادل رفتاری باشند.

عامل‌های بی‌پایان

NON-TERMINATING AGENTS

عامل‌های بی‌پایان *Non-terminating Agents*	عامل‌هایی که تعامل آنها با محیط هیچ‌گاه خاتمه نمی‌یابد. (تاریخچه‌ی این عامل‌ها نامتناهی است.)

معماری‌های انتزاعی

معماری عامل‌های هوشمند
Architectures for Intelligent Agents

معماری انتزاعی (مجرد)
Abstract Architectures

- عامل‌های واکنشی محض
- عامل‌های واکنشی با حالت

معماری انضمامی (مجسم)
Concrete Architectures

- عامل‌های مبتنی بر منطق
- عامل‌های واکنشی
- عامل‌های باور، مطلوب، قصد (*BDI*)
- عامل‌های با معماری لایه‌ای

عامل‌های واکنشی محض

PURELY REACTIVE AGENTS

عامل‌های واکنشی محض
Purely Reactive Agents

عامل واکنشی محض، در مورد آنچه باید انجام بدهد کاملاً بر اساس حال (بدون ارجاع به تاریخچه‌ی خود) تصمیم‌گیری می‌کند.

$$action : S \rightarrow A$$

برای هر عامل واکنشی محض، یک عامل استاندارد معادل وجود دارد (و برعکس):

$$action : S^* \rightarrow A$$

عامل‌های واکنشی محض

مثال

PURELY REACTIVE AGENTS: EXAMPLE

مثال: یک ترموستات می‌تواند به عنوان یک عامل واکنشی ساده دیده شود:

Assume w/o l.o.g. that the thermostat's environment is in one of two states: "too cold" or "temperature OK."

Then $action : S \rightarrow A$

$$action(s) = \begin{cases} \text{heater off} & \text{if } s = \text{temperature OK} \\ \text{heater on} & \text{if } s = \text{too cold} \end{cases}$$

عامل‌های واکنشی محض

روند طراحی تابع عامل

برای حرکت به سمت طراحی تابع تصمیم‌گیری *action*،
مدل انتزاعی عامل خود را تکمیل می‌کنیم:

آن را به زیرسیستم‌هایی تقسیم می‌کنیم.

↓

تصمیم‌های طراحی را بیشتر با توجه به زیرسیستم‌ها انجام می‌دهیم:

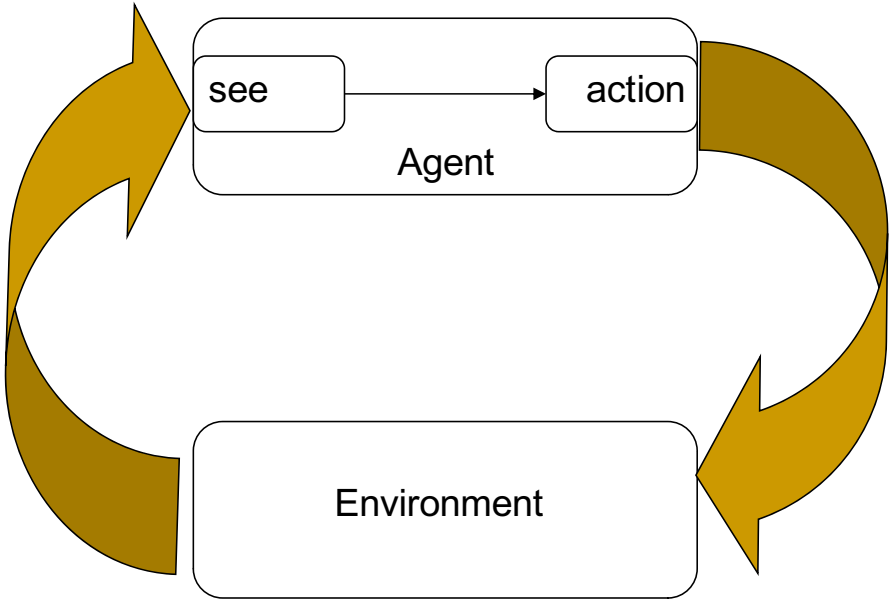
ساختمان‌های داده	کنترل
ساختمان‌داده‌ها و عملیات قابل انجام روی آنها	جریان کنترل بین زیرسیستم‌ها

یک معماری عامل، وضعیت‌های داخلی یک عامل را مشخص می‌کند.

عامل‌های واکنشی محض

روند طراحی تابع عامل: تفکیک تابع تصمیم‌گیری عامل

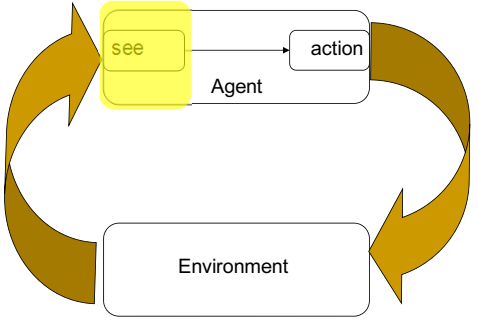
تفکیک تابع تصمیم‌گیری عامل به زیرسیستم‌های ادراک و کنش



عامل‌های واکنشی محض

روند طراحی تابع عامل: تفکیک تابع تصمیم‌گیری عامل: تابع دیدن

تفکیک تابع تصمیم‌گیری عامل به زیرسیستم‌های ادراک و کنش



تابع دیدن
See Function

تابعی که خروجی آن یک ادراک (یک ورودی ادراکی عامل) است.

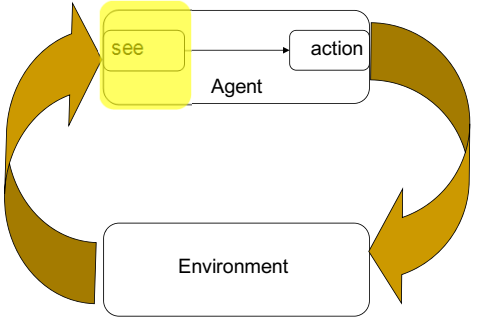
$$action : S^* \rightarrow A \quad \Rightarrow \quad \begin{cases} see : S \rightarrow P \\ action : P^* \rightarrow A \end{cases}$$

P مجموعه‌ی ادراک‌های عامل است.

عامل‌های واکنشی محض

روند طراحی تابع عامل: تفکیک تابع تصمیم‌گیری عامل: تابع دیدن: مثال (ترموستات)

تفکیک تابع تصمیم‌گیری عامل به زیرسیستم‌های ادراک و کنش



x = گزاره‌ی «دمای اتاق مناسب است»
y = گزاره‌ی «خیابان شلوغ است»

اگر فقط همین دو واقعیت مورد نظر باشد، مجموعه‌ی حالت‌های محیط S دقیقاً چهار عنصر دارد:

$$s_1 = \{\neg x, \neg y\}$$
$$s_2 = \{\neg x, y\}$$
$$s_3 = \{x, \neg y\}$$
$$s_4 = \{x, y\}$$

تابع see (دیدن) برای ترموستات دو ادراک در محدوده‌ی خودش دارد: p_1 (دمای سرد) و p_2 (دمای مناسب):

$$see(s) = \begin{cases} p_1 & \text{if } s = s_1 \text{ or } s = s_2 \\ p_2 & \text{if } s = s_3 \text{ or } s = s_4 \end{cases}$$

حالت‌های تمایزناپذیر

INDISTINGUISHABLE STATES

حالت‌های تمایزناپذیر
Indistinguishable States

حالت‌های تمایزناپذیر برای یک عامل:
دو حالت متفاوت محیط که توسط عامل برابر دیده می‌شود.

Suppose we have two environment states, s_1, $s_2 \in S$, such that $s_1 \neq s_2$ but $see(s_1) = see(s_2)$.

So s_1 and s_2 are *indistinguishable* by the agent.

حالت‌های تمایزناپذیر

کلاس‌های هم‌ارزی برای حالت‌های محیط

INDISTINGUISHABLE STATES

حالت‌های تمایزناپذیر
Indistinguishable States

حالت‌های تمایزناپذیر برای یک عامل:
دو حالت متفاوت محیط که توسط عامل برابر دیده می‌شود.

دو حالت محیط برای یک عامل معادل هستند، هرگاه توسط آن به یک صورت دیده شوند.

$$\text{For } s, s' \in S, \text{ we write } s \equiv s' \text{ if } see(s) = see(s').$$

$\equiv$ یک رابطه‌ی هم‌ارزی روی مجموعه‌ی حالت‌های محیط S است
که مجموعه‌ی S را به مجموعه‌هایی ناتهی و دوبه‌دو متمایز (کلاس‌های هم‌ارزی) افراز می‌کند.

هرچه $\equiv$ بزرگ‌تر باشد، ادراک عامل کارآیی کمتری دارد:

$|\equiv| = |S|$: عامل می‌تواند هر حالتی را تمییز دهد.

$|\equiv| = 1$: عامل توان ادراکی ندارد.

عامل‌های واکنشی با حالت داخلی

AGENTS WITH STATE

در عامل واکنشی با حالت داخلی، تابع تصمیم‌گیری عامل به صورت تابعی مدل می‌شود که دنباله‌ی حالت‌های محیط یا ادراکات را به کنش‌ها نگاشت می‌دهد.

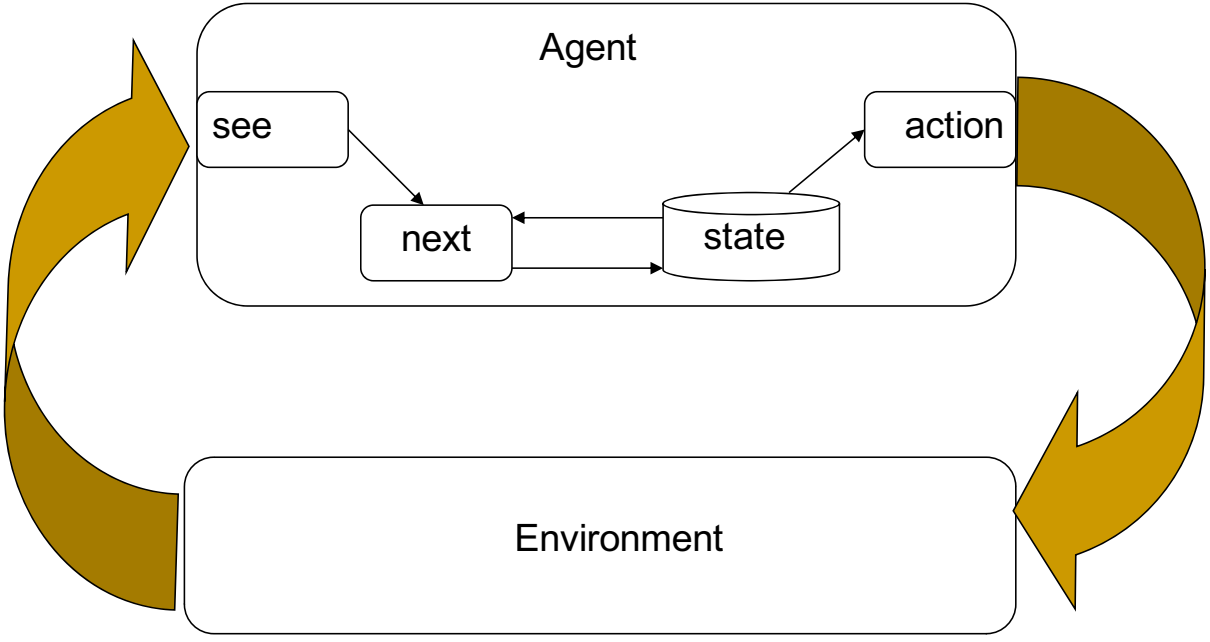
$$action : S^* \rightarrow A$$

(این به صورتی غیرمستقیم، تأثیر تاریخچه را بر روی تصمیمات نشان می‌دهد.)

نیاز به ساختمان‌داده‌ای که بتواند اطلاعات در مورد محیط و تاریخچه‌ی آن را ثبت کند.

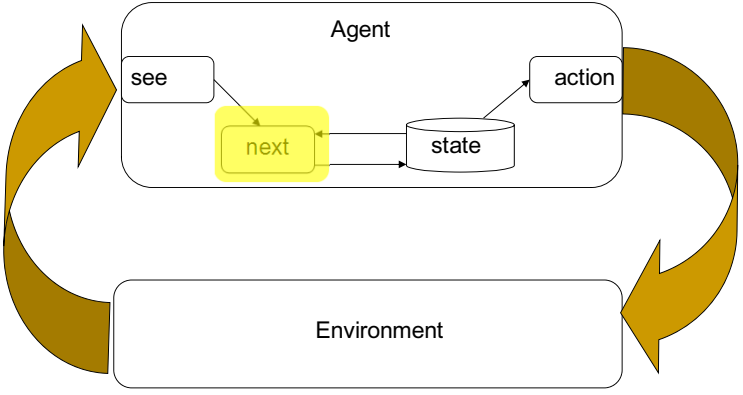
عامل‌های واکنشی با حالت داخلی

فرآیند طراحی



عامل‌های واکنشی با حالت داخلی

فرآیند طراحی: تابع بعدی



Let I be the set of internal agent states.

We now have the functions

see: $S \rightarrow P$ (unchanged)

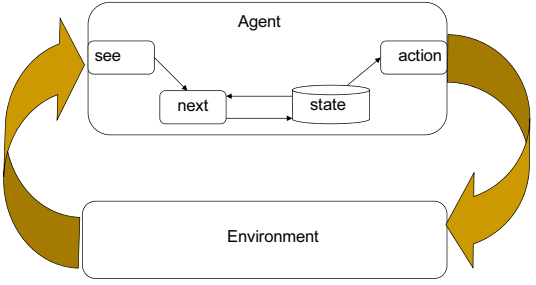
action: $I \rightarrow A$ (domain changed from P^* to I)

next: $I \times P \rightarrow I$ (new)

عامل‌های واکنشی با حالت داخلی

فرآیند طراحی: حلقه‌ی کنترلی عامل

AGENT'S CONTROL LOOP



The agent starts in some initial state i_0.

It then observes its environment state s and generates a percept $see(s)$.

The agent's state is then updated to $next(i_0, see(s))$, and the action the agent selects is

$action(next(i_0, see(s)))$

After performing this action, the agent enters another cycle.

هر عامل مبتنی بر حالت می‌تواند به یک عامل استاندارد معادل رفتاری با آن تبدیل شود.

معماری‌های انضمامی

معماری عامل‌های هوشمند
Architectures for Intelligent Agents

معماری انضمامی (مجسم)
Concrete Architectures

- عامل‌های مبتنی بر منطق
- عامل‌های واکنشی
- عامل‌های باور، مطلوب، قصد *(BDI)*
- عامل‌های با معماری لایه‌ای

معماری انتزاعی (مجرد)
Abstract Architectures

- عامل‌های واکنشی محض
- عامل‌های واکنشی با حالت

معماری عامل‌های هوشمند
Architectures for Intelligent Agents

معماری انضمامی (مجسم)
Concrete Architectures

عامل‌های مبتنی بر منطق *Logic-Based Agents*	تصمیم‌گیری از طریق استنتاج/استنباط منطقی تحقق می‌یابد.
عامل‌های واکنشی *Reactive Agents*	تصمیم‌گیری از طریق نگاشت مستقیم از وضعیت به کنش پیاده‌سازی می‌شود.
عامل‌های باور، مطلوب، قصد *Belief-Desire-Intension Agents (BDI)*	تصمیم‌گیری وابسته به دستکاری ساختمان‌داده‌های بازنمایی‌کننده‌ی باورها، مطلوب‌ها و قصدهای عامل است.
عامل‌های با معماری لایه‌ای *Layered-Architecture Agents*	تصمیم‌گیری از طریق لایه‌های نرم‌افزاری متعدد تحقق می‌یابد که هر یک در سطوح متفاوتی از انتزاع در مورد محیط استدلال می‌کنند.

معماری‌های مبتنی بر منطق

عامل‌های مبتنی بر منطق *Logic-Based Agents*	تصمیم‌گیری از طریق استنتاج/استنباط منطقی تحقق می‌یابد.

رویکرد «سنتی» به ساخت سیستم‌های *AI* (*Symbolic AI*)

- رفتار هوشمند با دادن یک بازنمایی نمادین از محیط عامل و رفتار مطلوب عامل تولید می‌شود.
- سپس این بازنمایی در قالب نمادین دستکاری می‌شود.

ملزومات معماری مبتنی بر منطق

بازنمایی نمادین *Symbolic Representation*	دستکاری نحوی *Syntactic Manipulation*
فرمول‌های منطقی	استدلال منطقی (اثبات قضیه)

معماری‌های مبتنی بر منطق

حالت: بازنمایی نمادین

حالت‌ها	
حالت‌های داخلی *Internal States*	**حالت‌های محیط و ادراکی** *Environment / Percept States*
به صورت مجموعه‌ای از باورها دیده می‌شوند. در قالب مجموعه فرمول‌های منطقی (مثلاً منطق مرتبه اول) $\Delta = \{temp(roomA, 20), heater(on), ...\}$	مربوط به محیط و حسگرهای عامل

$(\Rightarrow)$

بستار یک مجموعه حالت با عملگر ایجاب:

$$closure(\Delta, \Rightarrow) = \{\varphi | \varphi \in \Delta \vee \exists \psi . \psi \in \Delta \wedge \psi \Rightarrow \varphi\}$$

(همه‌ی حالت‌های قابل استنتاج از یک مجموعه حالت)

معماری‌های مبتنی بر منطق

اثبات: دستکاری نحوی

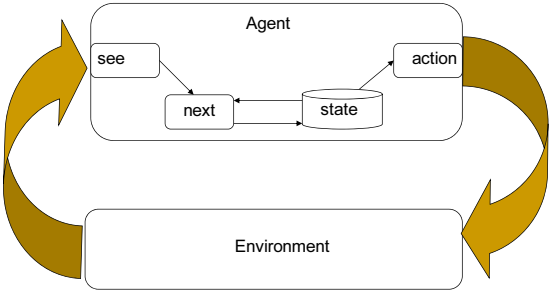
L	مجموعه‌ی جملات در منطق مرتبه اول *(FOL)*
$D = 2^L$	مجموعه‌ی همه‌ی پایگاه‌های داده (حالت‌های داخلی ممکن)
The members of D are written $\Delta, \Delta_1, \ldots$.	
ρ	مجموعه‌ی قواعد استنباط (مدل یک فرآیند تصمیم‌گیری عامل)

$$\Delta \vdash_{\rho} \varphi$$

یعنی: فرمول φ می‌تواند با استفاده از قواعد ρ از پایگاه داده‌ی Δ اثبات شود.

معماری‌های مبتنی بر منطق

تابع عامل



$$see: S \rightarrow P \quad \text{(as before)}$$
$$next: D \times P \rightarrow D \quad \text{(with } D \text{ in place of } I\text{)}$$
$$action: D \rightarrow A \quad \text{(with } D \text{ in place of } I\text{)}$$

برنامه‌نویس عامل، ρ و Δ را به‌گونه‌ای کد خواهد کرد که،
اگر $Do(a)$ قابل استخراج باشد،
آن‌گاه a بهترین کنشی باشد که می‌تواند انجام شود.

معماری‌های مبتنی بر منطق

تابع عامل: شبه کد

```
1.  function action(Δ : D) : A
2.  begin
3.      for each a ∈ A do
4.          if Δ |=ρ do(a) then
5.              return a
6.          end if
7.      end for
8.      for each a ∈ A do
9.          if Δ |≠ρ ¬do(a) then
10.             return a
11.         end if
12.     end for
13.     return noop
14. end function action
```

معماری‌های مبتنی بر منطق

مثال (۱ از ۴)

Example: a robotic agent to clean a room.

The room is a 3 × 3 grid; the robot starts at square (0,0).

Percepts:

dirt (there's dirt beneath it)

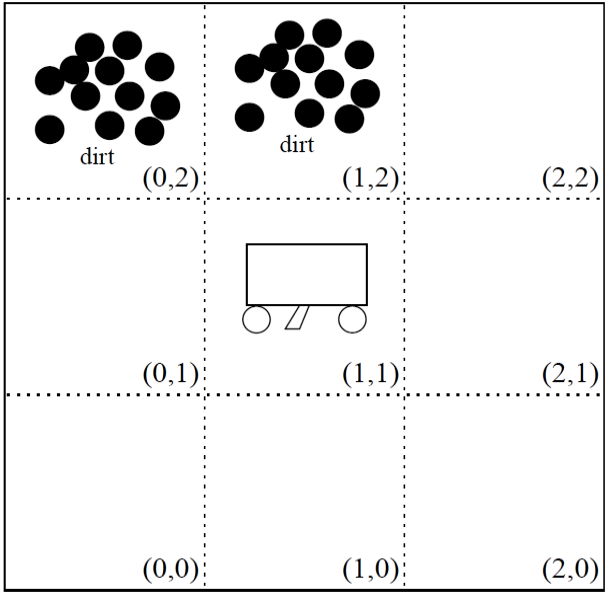
null (no special information)

Actions:

forward (one square)

suck

turn (90^0 right)



معماری‌های مبتنی بر منطق

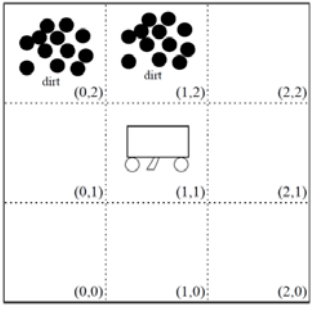
مثال (۲ از ۴)

Three domain predicates:

In(*x*,*y*): agent is at (*x*,*y*)

Dirt(*x*,*y*): there is dirt at (*x*,*y*)

Facing(*d*): agent is facing direction *d*



تابع *next*

* باید به اطلاعات ادراکی از محیط نگاه کند
* و یک پایگاه داده‌ی جدید شامل این اطلاعات تولید کند
* همچنین باید اطلاعات قدیمی یا نامربوط را حذف کند
* و مکان و جهت جدید عامل را بفهمد.

معماری‌های مبتنی بر منطق

مثال (۳ از ۴)

$old(\Delta)$ بیانگر اطلاعاتی در Δ که باید حذف شود.

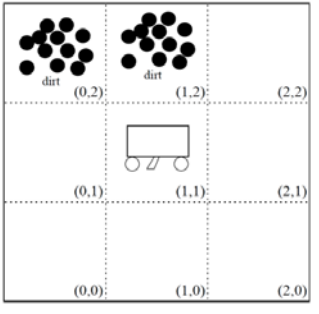
$$old(\Delta) = \{P(t_1,\ldots,t_n) : P \in \{In, Dirt, Facing\} \wedge P(t_1,\ldots,t_n) \in \Delta\}$$

$new(\Delta, p)$ مجموعه‌ی محمول‌های جدید که باید به پایگاه داده اضافه شود.

$$new : D \times P \rightarrow D$$

در این صورت:

$$next(\Delta, p) = (\Delta - old(\Delta)) \cup new(\Delta, p)$$



قواعد استنباط، عامل را مجبور می‌کند که رفتاری به صورت قاعده‌ی

$$\phi(\ldots) \rightarrow \psi(\ldots)$$

داشته باشد.

اگر مقدم قاعده ϕ با پایگاه داده‌ی عامل مطابقت یافت،
آن‌گاه تالی قاعده ψ می‌تواند نتیجه گرفته شود (با نمونه‌سازی متغیرها).

معماری‌های مبتنی بر منطق

مثال (۴ از ۴)

قاعده‌ی دارای بالاترین اولویت عبارت است از:

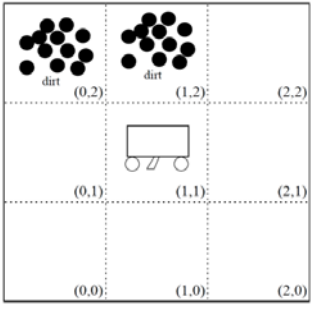
$$In(x,y) \wedge Dirt(x,y) \rightarrow Do(suck)$$

اگر شرایط این قاعده برقرار نشود،
آن‌گاه عامل در اتاق حرکت خواهد کرد (با فرض اینکه ترتیب مشاهده‌ی مربع‌ها ثابت است):

$$(0,0), (0,1), (0,2), (1,2), (1,1), \ldots$$

مثلاً: قواعد برای جابجایی عامل تا مربع $(0,2)$

$$In(0,0) \wedge Facing(north) \wedge \neg Dirt(0,0) \rightarrow Do(forward)$$
$$In(0,1) \wedge Facing(north) \wedge \neg Dirt(0,1) \rightarrow Do(forward)$$
$$In(0,2) \wedge Facing(north) \wedge \neg Dirt(0,2) \rightarrow Do(turn)$$
$$In(0,2) \wedge Facing(east) \rightarrow Do(forward)$$



معماری‌های مبتنی بر منطق

نقاط ضعف

SHORTCOMINGS OF LOGIC-BASED AGENTS

نقاط ضعف معماری‌های مبتنی بر منطق

رسیونالیته‌ی حسابی *Calculative Rationality*	تصمیم‌پذیری منطق *Decidability of Logic*	کدگذاری ادراکات *Encoding of Percepts*
عامل منطقی در محیطهای دینامیک (پویا) دارای رسیونالیته‌ی حسابی است.	منطق مرتبه اول نیمه‌تصمیم‌پذیر است.	کدگذاری ادراکات بسیار دشوار است (مثل داده‌های بصری)

انتقاد به روی‌کردهای نمادین/مبتنی بر منطق

مسئله‌ی «**انجام‌ناپذیری**» در مورد روی‌کردهای نمادین/ منطقی،
منجر به این شده است که برخی پژوهشگران، فرضیات این روی‌کردها را رد کنند.

می‌گویند: تغییرات اندک در روی‌کرد نمادین
برای ساخت عامل‌هایی که باید در محیط‌های دارای قید زمانی عمل کنند، کافی نیست.

برخی از **روی‌کردهای** این پژوهشگران:

رد کردن بازنمایی‌های نمادین و تصمیم‌گیری بر اساس دستکاری نحوی بر روی این بازنمایی‌های نمادین؛

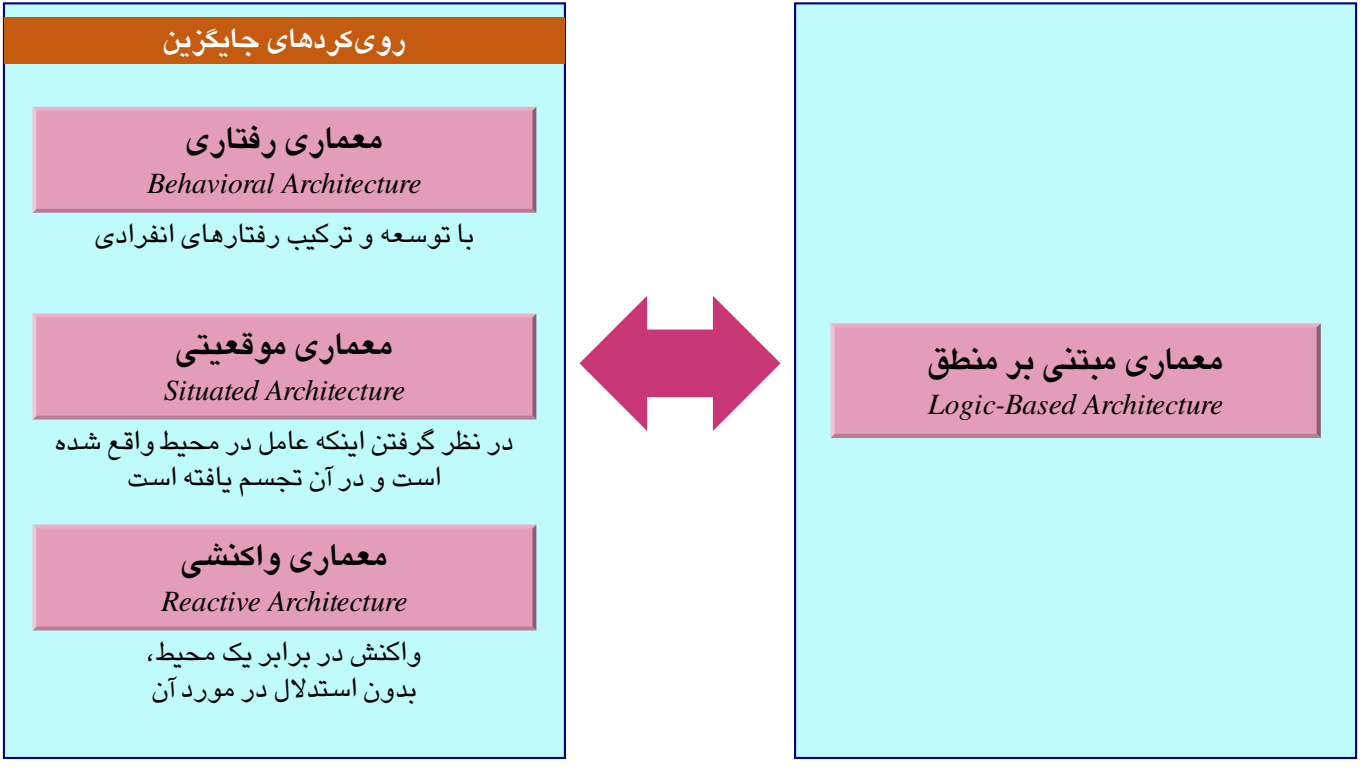
ایده ۱: رفتار هوشمند ذاتاً به محیط عامل متصل است
رفتار هوشمند نامجسم *(disembodied)* نیست،
بلکه حاصل تعامل *(interaction)* عامل و محیطش است.

ایده ۲: رفتار هوشمند از تعامل رفتارهای ساده‌ی گوناگون **بروز** *(emerge)* می‌کند.

معماری‌های واکنشی

به‌عنوان نمونه‌ای از جایگزین‌های معماری‌های مبتنی بر منطق

REACTIVE ARCHITECTURES



معماری سابسامشن

SUBSUMPTION ARCHITECTURE

معماری سابسامشن
Subsumption Architecture

ارائه شده توسط *Rodny Brooks*

ویژگی‌ها

۱) تصمیم‌گیری عامل از طریق رفتارهای انجام وظیفه *(task accomplishing behavior)*

هر رفتار می‌تواند به صورت یک تابع *action* جداگانه مطرح شود.
ماژول‌های انجام وظیفه، بازنمایی نمادین پیچیده ندارند و هیچ استدلال نمادینی وجود ندارد:
$situation \rightarrow action$

۲) امکان فایر شدن تعداد زیادی رفتار به‌طور همزمان (رفع تناقضات با سلسله‌مراتب)

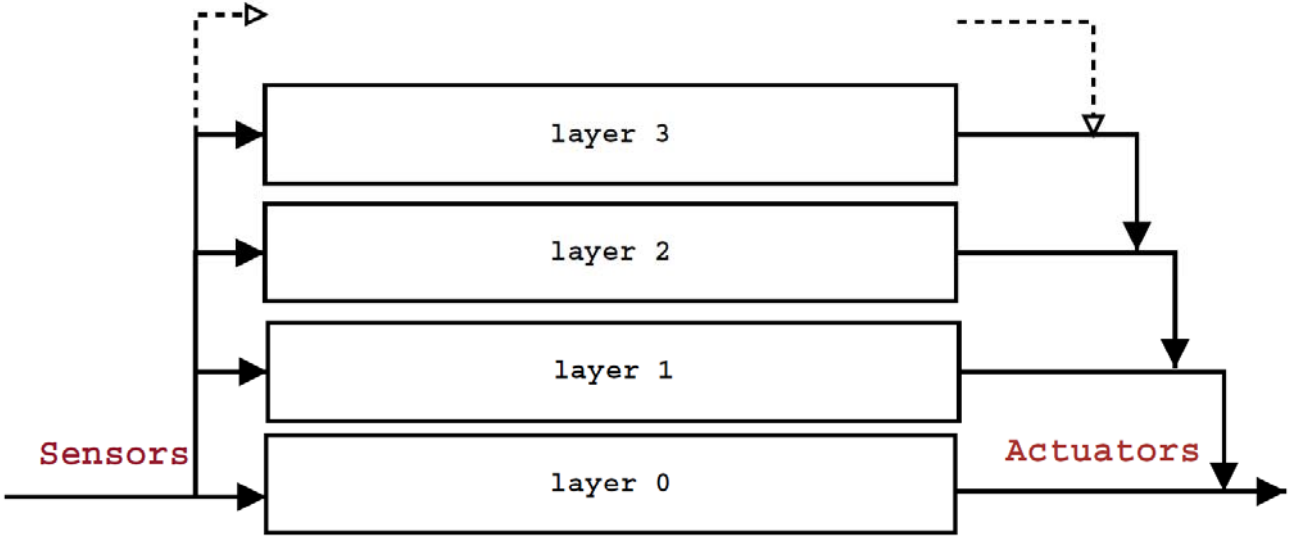
باید مکانیزمی برای انتخاب بین اعمال مختلف فایر شده توسط کنش‌ها وجود داشته باشد.
راه حل: سلسله‌مراتب سابسامشن *(subsumption hierarchy)* با رفتارهای مرتب شده در لایه‌ها:

- لایه‌های پایین‌تر می‌توانند لایه‌های بالاتر را منع کنند.
- لایه‌های زیرین یک لایه از لایه‌های بالاتر اولویت بیشتری دارند. (مثل اجتناب از مانع در ربات‌ها)
- لایه‌های بالاتر رفتارهای انتزاعی‌تر را بازنمایی می‌کنند.

معماری سابسامشن

نمودار

SUBSUMPTION ARCHITECTURE



معماری سابسامشن

صورت‌بندی

FORMALIZATION OF SUBSUMPTION ARCHITECTURE

تابع see تغییر نمی‌کند، اما ادراک شدیداً با کنش جفت می‌شود (داده‌های خام حسگری خیلی تبدیل نمی‌شود).

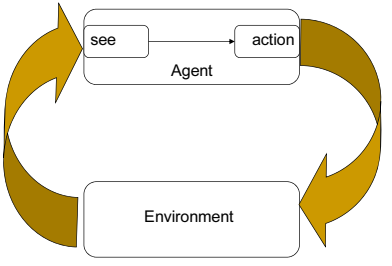
تابع تصمیم $action$ از طریق یک مجموعه از رفتارها به‌همراه یک رابطه‌ی $inhibit$ بین رفتارها پیاده می‌شود.

- یک رفتار یک زوج (c,a) است که در آن $c \subseteq P$ (شرایط) و $a \in A$ (کنش).
- رفتار (c,a) فایر می‌شود وقتی که محیط در حالت $s \in S$ باشد اگر و فقط اگر $see(s) \in c$.
- فرض می‌کنیم $Beh = \{(c,a): c \subseteq P \text{ and } a \in A\}$ مجموعه‌ی همه‌ی چنین قواعدی باشد.
- و رابطه‌ی $R \subseteq Beh$ مجموعه‌ی قواعد رفتار عامل باشد: رابطه‌ی «جلوگیری: $inhibition$» (ترتیب کامل $\prec$)

$$b_1 \prec b_2$$

b_1 از b_2 جلوگیری می‌کند.

b_1 در سلسله‌مراتب پایین‌تر است، پس اولویت بیشتری دارد.



معماری سابسامشن

روال انتخاب کنش: شبه کد

SUBSUMPTION ARCHITECTURE

```
1.  function action(p : P) : A
2.  var fired : ℘(R)
3.  begin
4.          fired := {⟨c, a⟩ | ⟨c, a⟩ ∈ R ∧ p ∈ c}
5.          for each ⟨c, a⟩ ∈ fired do
6.                  if ¬(∃⟨c′, a′⟩ ∈ fired ∧
                           ⟨c′, a′⟩ ≺ ⟨c, a⟩)
7.                  then return a
8.                  end if
9.          end for
10.         return noop
11. end function action
```

معماری سابسامشن

روال انتخاب کنش: پیچیدگی محاسباتی

SUBSUMPTION ARCHITECTURE

```
1.  function action(p : P) : A
2.  var fired : ℘(R)
3.  begin
4.          fired := {⟨c, a⟩ | ⟨c, a⟩ ∈ R ∧ p ∈ c}
5.          for each ⟨c, a⟩ ∈ fired do
6.                  if ¬(∃⟨c′, a′⟩ ∈ fired ∧
                        ⟨c′, a′⟩ ≺ ⟨c, a⟩)
7.                  then return a
8.                  end if
9.          end for
10.         return noop
11. end function action
```

پیچیدگی محاسباتی این تابع $action$ برابر با $O(n^2)$ است.

n = بزرگ‌ترین مقدار بین تعداد رفتارها و تعداد ادراک‌ها

پیچیدگی محاسباتی بهتر:
از طریق کدگذاری منطق تصمیم‌گیری در سخت‌افزار
$\Leftarrow$ زمان‌تصمیم‌گیری ثابت $O(1)$

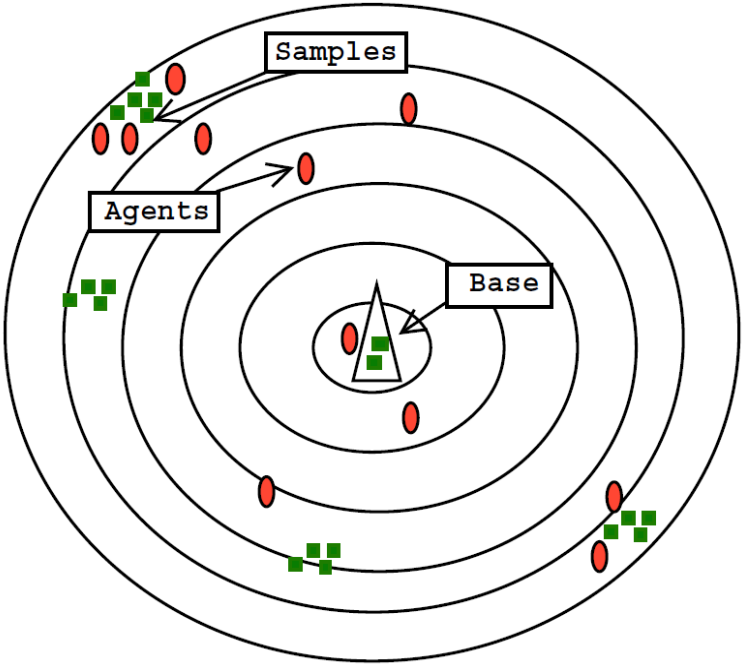
سادگی محاسباتی، احتمالاً بزرگ‌ترین نقطه‌قوت معماری سابسامشن است.

معماری سابسامشن

مثال

SUBSUMPTION ARCHITECTURE

Case Study on Foraging Robots [Steels, 1990; Drogoul and Feber, 1992]



قیدها:

- عدم امکان تبادل پیام
- فقدان نقشه‌ی عامل‌ها
- وجود موانع
- میدان گرادیان
- خوشه‌بندی نمونه‌ها

معماری سابسامشن

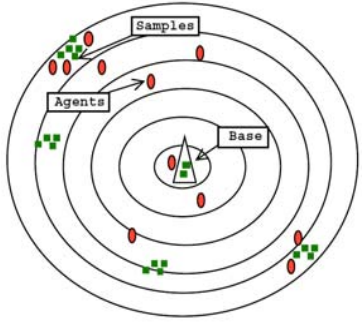
مثال

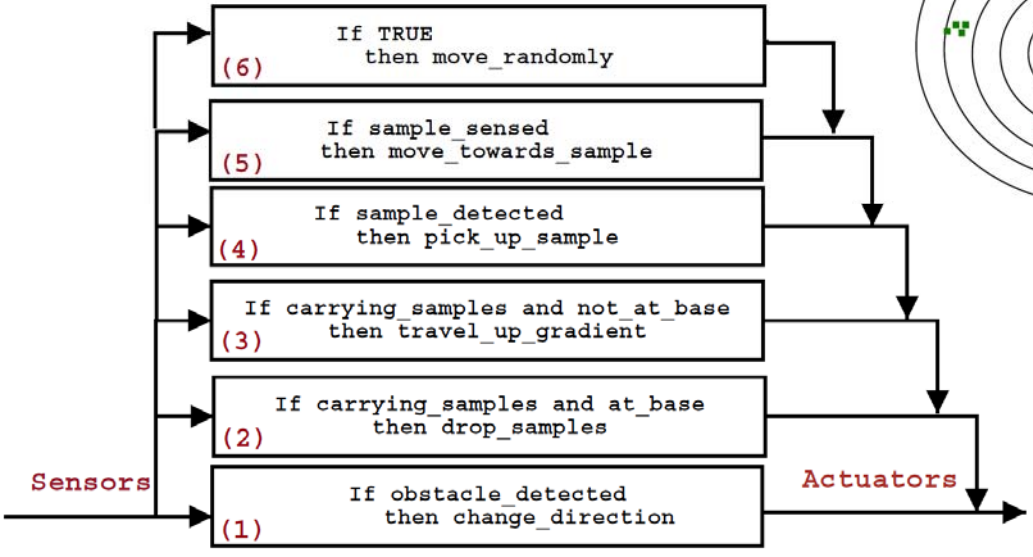
SUBSUMPTION ARCHITECTURE

Case Study on Foraging Robots [Steels, 1990; Drogoul and Feber, 1992]

Subsumption ordering:

$(1) \prec (2) \prec (3) \prec (4) \prec (5) \prec (6)$





معماری «باور ـ مطلوب ـ قصد» BDI

BDI (BELIEF-DESIRE-INTENTION) ARCHITECTURE

معماری «باور ـ مطلوب ـ قصد» *BDI*
BDI (Belief-Desire-Intention) Architecture

ریشه‌های این معماری در حوزه‌ی فلسفی «استدلال کاربردی» *Practical Reasoning* قرار دارد.

فرآیند تصمیم‌گیری لحظه به لحظه که در آن اعمال برای نزدیک شدن به اهداف انجام می‌شوند.

دو فرآیند اصلی در استدلال کاربردی

تأمل *Deliberation*	**استدلال «هدف ـ وسیله»** *Means-Ends Reasoning*
تصمیم‌گیری در مورد اهدافی که می‌خواهیم به آنها دست یابیم.	چگونه می‌خواهیم به این اهداف دست پیدا کنیم؟

قصدها *(Intentions)* نقش مهمی در فرآیند استدلال کاربردی دارند.

معماری «باور ـ مطلوب ـ قصد» BDI

نقش «قصد» در فرآیند استدلال کاربردی

BDI (Belief-Desire-Intention) Architecture

قصدها *(Intentions)* نقش مهمی در فرآیند استدلال کاربردی دارند.

نقش قصدها

محرک استدلال هدف ـ وسیله

تصمیم می‌گیرم که چگونه باید به یک قصد دست پیدا کنم.
اگر یک دسته از کنش‌ها برای دست‌یابی به آن قصد شکست خورد، معمولاً سایر دسته کنش‌ها را آزمایش می‌کنم.

مقید کننده‌ی تأمل‌های آینده

سرگرم گزینه‌های ناسازگار با یک قصد نمی‌شوم.

پافشاری کننده *(Intentions Persist)*

قصدها پافشاری می‌کنند مادامی که من باور دارم به طور موفقیت‌آمیزی به آنها دست می‌یابم،
باور دارم که من نمی‌توانم به آنها دست پیدا کنم، یا اینکه هدف آن قصد دیگر موجود نیست.

مؤثر بر باورها

قصدها بر باورها تأثیر می‌گذارند، زیرا استدلال‌های کاربردی آینده بر پایه‌ی آنهاست.

معماری «باور ـ مطلوب ـ قصد» BDI

قصد

INTENTION

مسئله‌ی اصلی در طراحی عامل‌های مبتنی بر استدلال کاربردی تعادل بین اثرات «قصد» است.

انتخاب قصد جدید
توسط عامل در هر لحظه

در اثر رسیدن به یک باور جدید
یا اینکه باور قبلی دیگر توجیهی نداشته باشد

بده‌بستان
Tradeoff

تعهد به قصد انتخاب شده
توسط عامل برای همیشه

عامل برای بازبینی کافی توقف نکند

- به نظر می‌رسد که بهتر است عامل از یک لحظه تا لحظه‌ی دیگر قصدهایش را بازبینی کند.
  - ولی این بازبینی دارای هزینه است (منابع زمانی و محاسباتی).
  - اما هنوز مشکلات دیگر وجود دارد:
    - اگر عامل برای بازبینی کافی نایستد، به تلاش برای رسیدن به قصدهایی ادامه می‌دهد که حتی واضح است پس از این نمی‌توان به آنها رسید (و یا اهدافی که دیگر دلیلی برای رسیدن به آنها وجود ندارد).
    - عاملی که به طور مداوم قصدهایش را بازبینی می‌کند، زمان کافی برای رسیدن به آنها را نخواهد داشت و بنابراین این ریسک را خواهد داشت که هرگز به آنها نرسد.

لازم است یک نوع بده‌بستان بین درجه‌ی تعهدات و بازبینی آنها صورت گیرد.

معماری «باور ـ مطلوب ـ قصد» BDI

قصد: تمایز بین «عامل‌های محتاط» و «عامل‌های جسور»

INTENTION

عامل‌های محتاط
Cautious Agents

که مدام برای بازبینی می‌ایستند.

بده‌بستان
Tradeoff

عامل‌های جسور
Bold Agents

که هرگز برای بازبینی نمی ایستند.

یک پارامتر γ را به عنوان نرخ تغییرات محیط در نظر می‌گیریم.

- اگر γ کوچک بود (یعنی محیط به کندی تغییر می‌کند): عامل‌های جسور بهتر عمل می‌کنند.
- اگر γ بزرگ بود (یعنی محیط به شدت تغییر می‌کند): عامل‌های محتاط بهتر عمل می‌کنند.

درس: محیط‌های مختلف، استراتژی‌های تصمیم‌گیری متفاوتی را می‌طلبد.

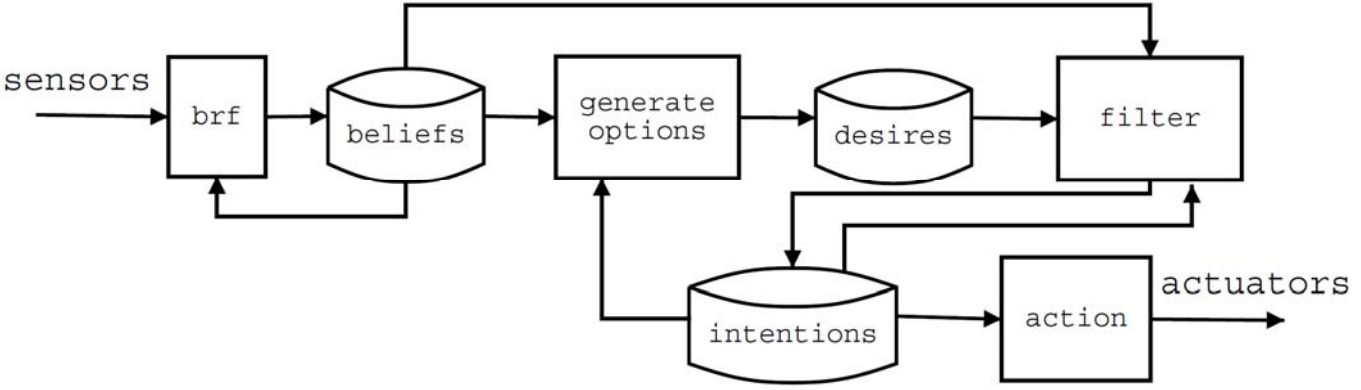
معماری «باور ـ مطلوب ـ قصد» BDI

مؤلفه‌های اصلی عامل *BDI*

مؤلفه‌های اصلی عامل *BDI*		
	مجموعه‌ی باورهای فعلی *A Set of Current **Beliefs***	باورهای فعلی عامل در مورد محیط خود
	تابع بازبینی باور ***Belief Revision Function (brf)***	از روی یک ادراک و باورهای فعلی عامل، یک مجموعه‌ی جدید از باورها را تعیین می‌کند.
	تابع تولید گزینه *Option Generation Function*	گزینه‌ها (مطلوب‌ها)ی موجود عامل را از روی باورها و قصدهای فعلی آن تعیین می‌کند.
	مجموعه‌ی گزینه‌های فعلی (مطلوب‌ها) *A Set of Current **Options (Desires)***	دسته‌های کنش‌های موجود عامل
	تابع فیلتر *Filter Function*	فرآیند تأمل عامل: قصدها را از روی باورها، مطلوب‌ها و قصدها تعیین می‌کند.
	مجموعه‌ی قصدهای فعلی *A Set of Current **Intentions***	تمرکز فعلی عامل‌ها: آن حالت‌هایی از امور که برای اجرای آنها تعهد شده است
	تابع انتخاب کنش (اجرا) *Action Selection Function (Execute)*	یک کنش را بر اساس قصدهای فعلی عامل تعیین می‌کند.

معماری «باور ـ مطلوب ـ قصد» BDI

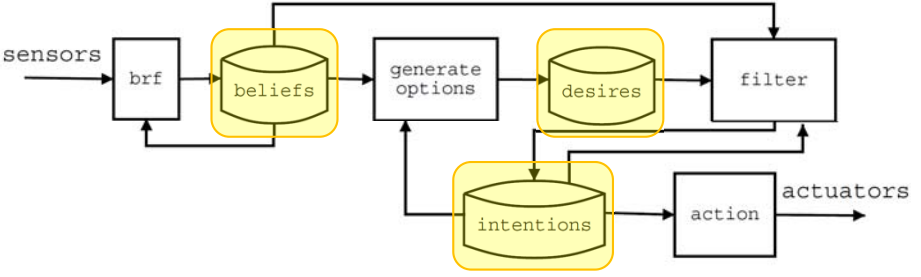
نمودار



معماری «باور ـ مطلوب ـ قصد» BDI

فرمول‌بندی معماری *BDI*: مجموعه‌های «باورها»، «مطلوب‌ها»، «قصدها»

Bel, Des and *Int*



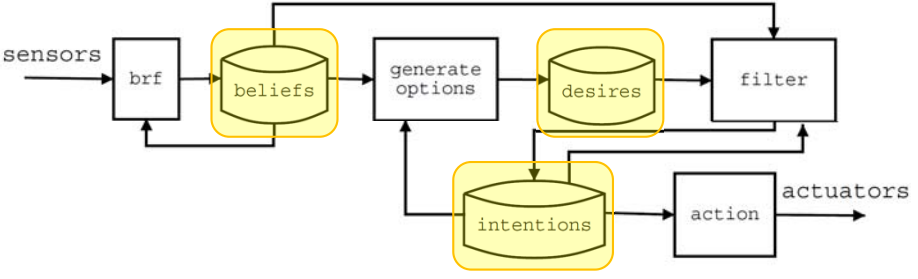
معماری «باور ـ مطلوب ـ قصد» BDI

فرمول‌بندی معماری BDI: بازنمایی حالت داخلی در قالب سه‌تایی «باورها»، «مطلوب‌ها»، «قصدها»

بازنمایی حالت داخلی عامل به صورت سه‌تایی:

$$< B, D, I >, \text{ where}$$

$$B \subseteq Bel, D \subseteq Des \text{ and } I \subseteq Int$$

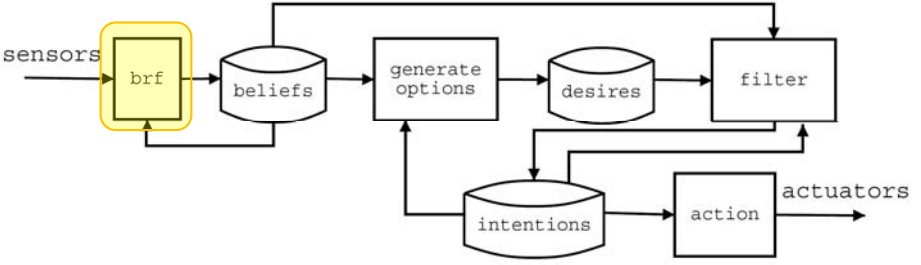


معماری «باور ـ مطلوب ـ قصد» BDI

فرمول‌بندی معماری *BDI*: تابع بازبینی باورها

تابع بازبینی باورها:

$$brf : \wp(Bel) \times P \rightarrow \wp(Bel)$$



معماری «باور ـ مطلوب ـ قصد» BDI

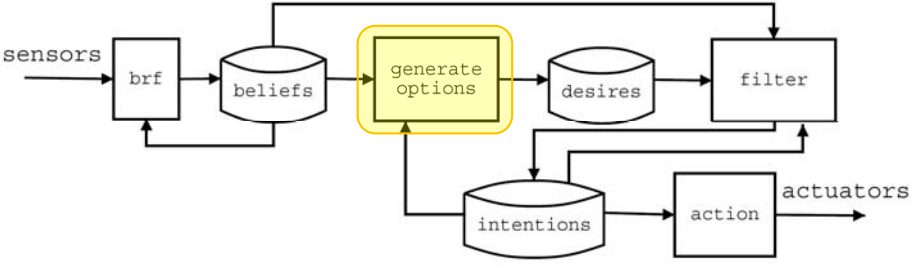
فرمول‌بندی معماری *BDI*: تابع تولید گزینه‌ها

تابع تولید گزینه‌ها:

$$options : \wp(Bel) \times \wp(Int) \rightarrow \wp(Des)$$

این تابع مسئول فرآیند استدلال **هدف ـ وسیله** *means-end reasoning* برای عامل است.

- وقتی یک عامل تصمیم می‌گیرد به چه دست یابد،
  بررسی می‌کند که چگونه باید به آن دست پیدا کند.
- وقتی یک گزینه به قصد تبدیل شد، به تابع تولید گزینه فیدبک می‌شود.
  - در نتیجه تابع *option* به صورت بازگشتی یک نقشه‌ی سلسله‌مراتبی می‌سازد.
    - که متعهد است به صورت پیشرونده به قصدهای خاص‌تر و خاص‌تر بپردازد،
    - تا به قصدهای مربوط به کنش‌های اجرا شده‌ی میانی برسد.



معماری «باور ـ مطلوب ـ قصد» BDI

فرمول‌بندی معماری *BDI*: تابع تولید گزینه‌ها: ویژگی‌ها

تابع تولید گزینه‌ها:

$$options : \wp(Bel) \times \wp(Int) \rightarrow \wp(Des)$$

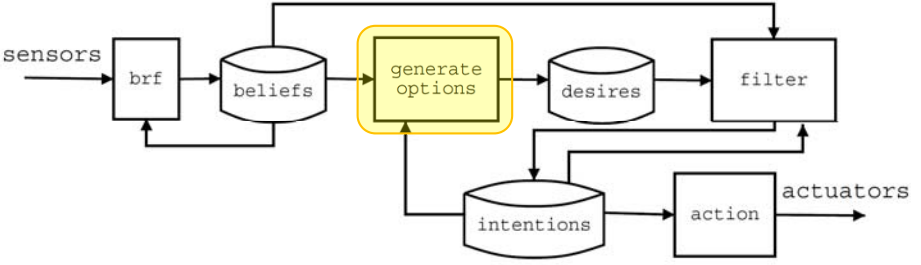
ویژگی‌های تابع تولید گزینه

سازگار بودن
Consistent

گزینه‌هایی را تولید کند که با باورها و قصدهای فعلی عامل سازگار باشد.

فرصت‌طلب بودن
opportunistic

تغییرات دارای برتری که راه‌های جدید برای دست‌یابی به قصدها را پیشنهاد می‌کنند، تشخیص دهد که باعث می‌شود دست‌یابی به قصدهای دست‌نیافتنی قبلی ممکن شود.



معماری «باور ـ مطلوب ـ قصد» BDI

فرمول‌بندی معماری *BDI*: تابع فیلتر

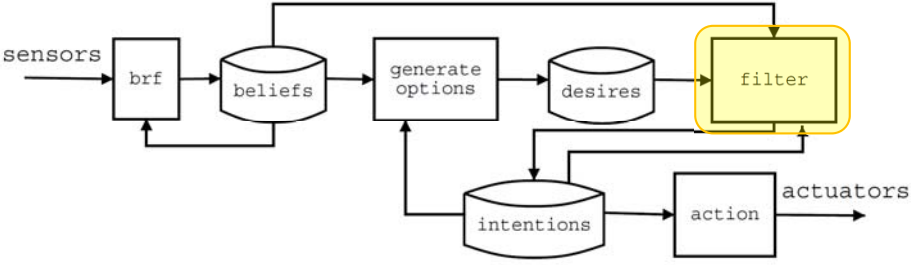
تابع فیلتر:

$$filter : \wp(Bel) \times \wp(Des) \times \wp(Int) \rightarrow \wp(Int)$$

پیاده‌سازی فرآیند تأمل *deliberation*: تصمیم گیری در مورد آنچه باید انجام شود.

- برای تصمیم‌گیری در مورد آنچه باید انجام شود:
  - قصدهای غیرممکن را حذف می‌کند. (یا آنهایی که دست‌یابی به آنها خیلی گران است)
  - قصدهایی که تا کنون به آنها نرسیدیم را حفظ می‌کند.
  - قصدهای جدید را می‌پذیرد: برای رسیدن به قصدهای موجود **یا** به‌کارگیری فرصت‌های جدید

یک قید برای هر تابع فیلتر: $\forall B \in \wp(Bel), \forall D \in \wp(Des), \forall I \in \wp(Int), filter(B, D, I) \subseteq I \cup D$



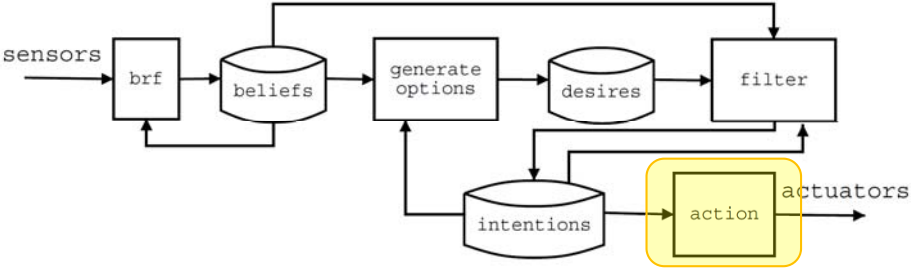
معماری «باور ـ مطلوب ـ قصد» BDI

فرمول‌بندی معماری *BDI*: تابع اجرا

تابع اجرا:

$$execute : \wp(Int) \rightarrow A$$

برای قصدهای قابل اجرا قصدی را برمی‌گرداند که به طور مستقیم قابل اجراست.



معماری «باور ـ مطلوب ـ قصد» BDI

فرمول‌بندی معماری *BDI*: بازنمایی قصدها

بازنمایی قصدها *(intentions)*

بازنمایی مجموعه‌ای
Set Representation

به صورت یک مجموعه
(بدون ساختار)
(در عمل بسیار ساده است.)

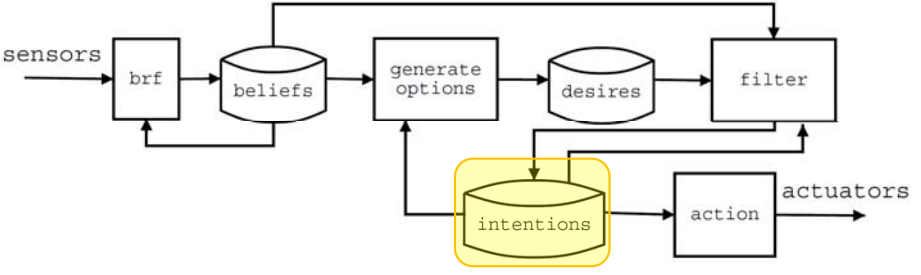
بازنمایی مجموعه‌ای مرتب
Ordered Set Representation

به هر قصد
یک عدد اولویت نسبت می‌دهیم.

بازنمایی پشته‌ای
Stack Representation

قصدها را با یک پشته نمایش می‌دهیم:

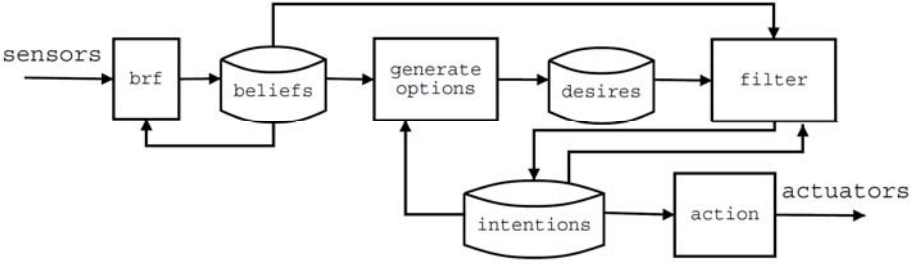
* وقتی یک قصد پذیرفته می‌شود، آن را روی پشته *push* کنیم.
* وقتی قصد دست‌نیافتی می‌شود، آن را از روی پشته *pop* کنیم.



معماری «باور ـ مطلوب ـ قصد» BDI

تابع تصمیم‌گیری در *BDI*: شبه کد

```
1.    function action(p : P) : A
2.    begin
3.          B := brf(B, p)
4.          D := options(D, I)
5.          I := filter(B, D, I)
6.          return execute(I)
12.   end function action
```



معماری «باور ـ مطلوب ـ قصد» BDI

یک مفسر انتزاعی برای *BDI*

AN ABSTRACT BDI INTERPRETER

```
1.    function BDI_interpreter
2.    begin
3.          initialize_state()
4.          repeat
5.              options := option_generator(event_queue)
6.              selected_options := deliberate(options)
7.              update_intentions(selected_options)
8.              execute()
9.              get_new_external_events()
10.             drop_succesful_intentions()
11.             drop_impossible_intentions()
12.         end repeat
13.   end function action
```

Ref.: the OASIS system [Ljungberg and Lucas, 1992]

معماری «باور ـ مطلوب ـ قصد» BDI

ملاحظات

REMARKS

مزایا

جذاب بودن این معماری:

۱) شهودی است.

۲) یک تجزیه‌ی کارکردی واضح ارائه می‌دهد:
که زیرسیستم‌های مورد نیاز را نشان می‌دهد.

معایب

دشواری اصلی معماری:

چگونگی پیاده‌سازی این توابع به صورت کارآمد

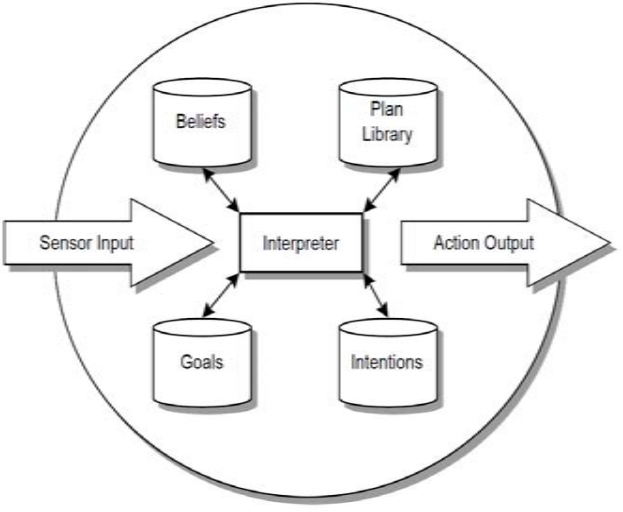
معماری «باور ـ مطلوب ـ قصد» BDI

نمونه چارچوب‌های کاری پیاده‌سازی *BDI*

BDI Implementation Frameworks

PRS (Practical Reasoning System)

Rao and Georgeff



JADEX BDI agent system

University of Hamburg

`http://sourceforge.net/projects/jadex`

معماری‌های لایه‌ای

LAYERED ARCHITECTURES

معماری لایه‌ای
Layered Architectures

طراحی زیرسیستم‌ها با سلسله مراتبی از لایه‌های تعامل کننده

در معماری لایه‌ای، حداقل دو لایه داریم:

لایه‌ی **رفتار کنشی** *Active*	لایه‌ی **رفتار پیش‌کنشی** *Pro-active*
مانند رفتارهای *event-driven*	مانند رفتارهای *goal-directed*

معماری‌های لایه‌ای

انواع لایه‌بندی

LAYERED ARCHITECTURES

انواع لایه‌بندی (بر اساس جریان کنترل بین لایه‌ها)	
لایه‌بندی افقی *Horizontal Layering*	**لایه‌بندی عمودی** *Vertical Layering*

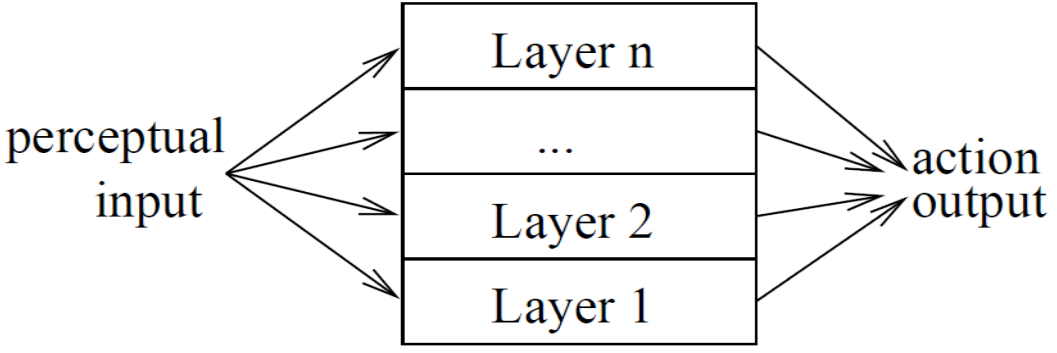
معماری‌های لایه‌ای

انواع لایه‌بندی: لایه‌بندی افقی

LAYERED ARCHITECTURES

انواع لایه‌بندی (بر اساس جریان کنترل بین لایه‌ها)

لایه‌بندی عمودی *Vertical Layering*	**لایه‌بندی افقی** ***Horizontal Layering***



معماری‌های لایه‌ای

انواع لایه‌بندی: لایه‌بندی افقی: ویژگی‌ها

LAYERED ARCHITECTURES

انواع لایه‌بندی (بر اساس جریان کنترل بین لایه‌ها)

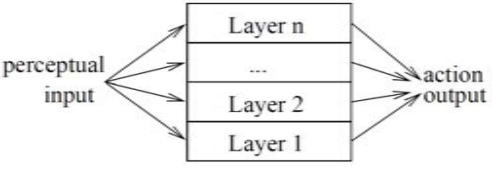
لایه‌بندی عمودی *Vertical Layering*	**لایه‌بندی افقی** *Horizontal Layering*

لایه‌های نرم‌افزاری هر یک مستقیماً به ورودی‌های حسگر و خروجی‌های کنش‌گر متصل هستند.

هر لایه به تنهایی می‌تواند یک عامل در نظر گرفته شود.

مزایا

* موازی سازی
* سادگی
* مقاوم بودن تا حدودی



معایب

اگر عامل نیاز به بروز n رفتار مختلف داشته باشد،
آن گاه باید n لایه‌ی مختلف بسازیم.

رفتار کلی سیستم باید سازگار باشد:

* نیاز به میانجی *(mediator)* برای اطمینان از انسجام *(coherence)* کلی.
* نیاز به کنترل متمرکز: گلوگاه تصمیم‌گیری عامل
* تعامل میان لایه‌ها نمایی: $O(m^n)$
m = تعداد کنش‌ها در هر لایه

معماری‌های لایه‌ای

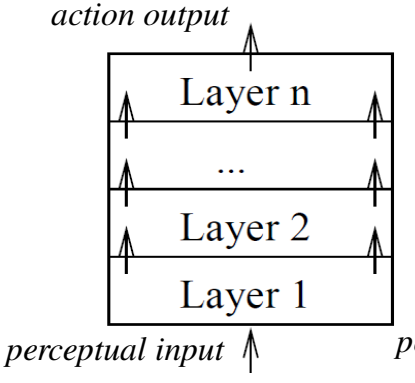
انواع لایه‌بندی: لایه‌بندی عمودی

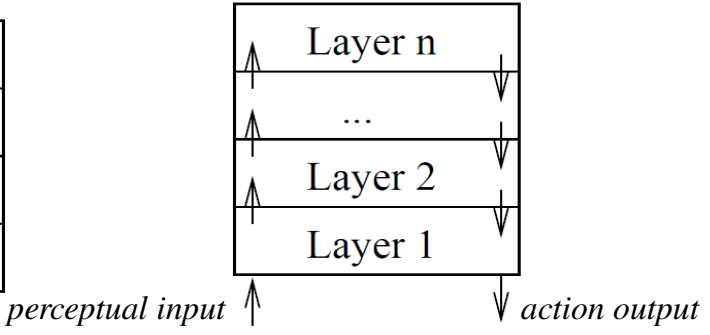
LAYERED ARCHITECTURES

انواع لایه‌بندی (بر اساس جریان کنترل بین لایه‌ها)

لایه‌بندی عمودی
Vertical Layering

لایه‌بندی افقی
Horizontal Layering





کنترل تک‌گذره
One Pass Control

کنترل دوگذره
Two Pass Control

معماری‌های لایه‌ای

انواع لایه‌بندی: لایه‌بندی عمودی: ویژگی‌ها

LAYERED ARCHITECTURES

انواع لایه‌بندی (بر اساس جریان کنترل بین لایه‌ها)

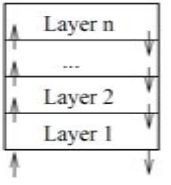
لایه‌بندی عمودی
Vertical Layering

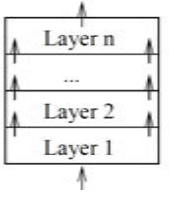
لایه‌بندی افقی
Horizontal Layering

ورودی‌های حسگر و خروجی‌های کنش‌گر هر یک حداکثر با یک لایه در ارتباط هستند.

کنترل دوگذره
Two Pass Control

جریان اطلاعات از لایه‌ی اول تا آخر می‌گذرد و جریان کنترل در جهت معکوس به لایه‌ی اول می‌رسد و خروجی کنش تولید می‌شود.





کنترل تک‌گذره
One Pass Control

جریان کنترل به صورت ترتیبی از لایه‌ها می‌گذرد تا به لایه‌ی آخر برسد که در آنجا خروجی کنش تولید می‌شود.

مزایا

* حداکثر $m^2(n-1)$ تعامل بین لایه‌ها

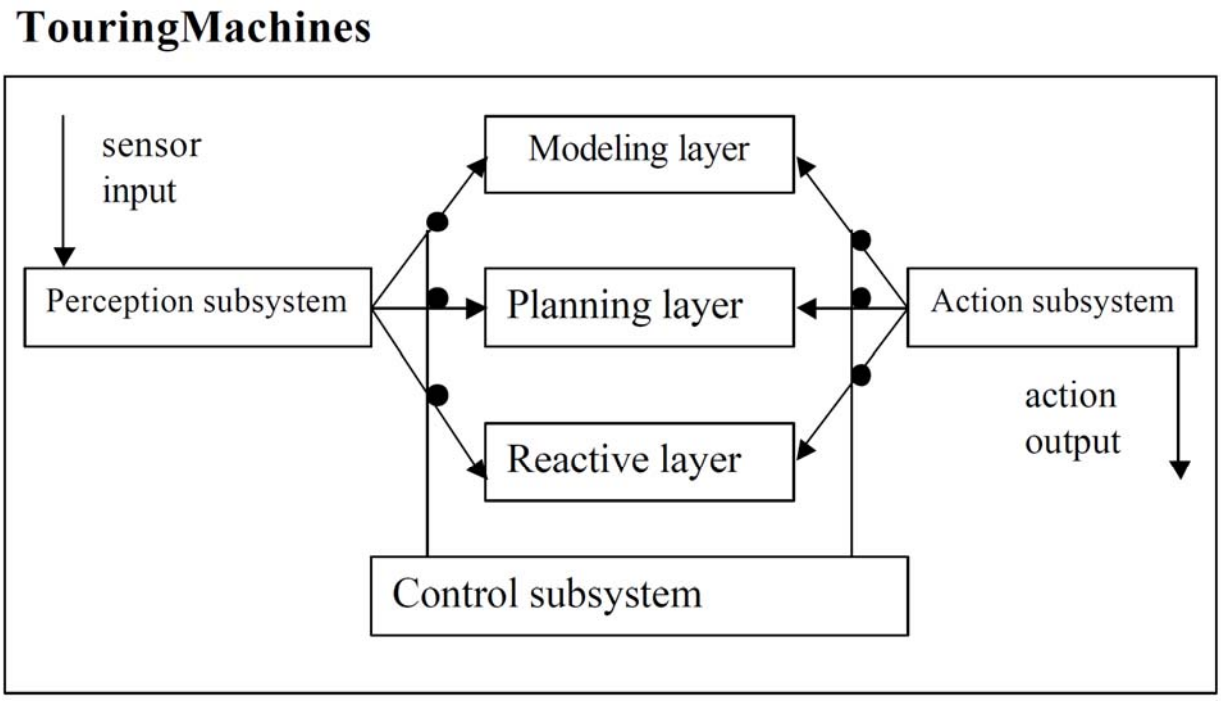
معایب

* جریان کنترل باید از همه‌ی لایه‌ها عبور کند.
* عدم تحمل‌پذیری نقص:
اگر یک لایه خراب شود، کل سیستم از کار می‌افتد.

معماری‌های لایه‌ای

انواع لایه‌بندی: لایه‌بندی افقی: مثال (معماری ماشین‌های تورینگ)

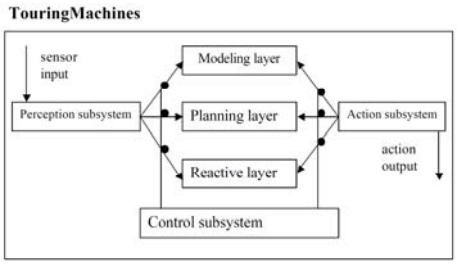
TOURING MACHINES ARCHITECTURE



معماری‌های لایه‌ای

انواع لایه‌بندی: لایه‌بندی افقی: مثال (معماری ماشین‌های تورینگ)

TOURING MACHINES ARCHITECTURE



دارای سه لایه برای تولید کنش

هر لایه به صورت مداوم «پشنهادهایی» برای کنش‌های قابل انجام ارائه می‌دهد.

لایه‌ی واکنشی
Reactive Layer

به تغییرات محیط پاسخ سریع می‌دهد.

لایه‌ی طرح‌ریزی
Planning Layer

به رفتار پیش‌کنشی دست می‌یابد.

لایه‌ی مدل‌سازی
Modeling Layer

موجودیت‌های مختلف در دنیا را بازنمایی می‌کند.
(خود عامل و دیگر عامل‌ها)

تداخل بین عامل‌ها را پیش‌بینی می‌کند و اهداف جدید برای رفع این تداخل‌ها طراحی می‌کند.

اهداف جدید به لایه‌ی طرح‌ریزی فرستاده می‌شوند.

سه لایه در یک سیستم کنترلی قرار می‌گیرند

که تصمیم می‌گیرد کدام لایه باید کنترل را در اختیار بگیرد.

زیرسیستم کنترل
Control Subsystem

به صورت مجموعه‌ای از قواعد کنترلی در دو قالب:

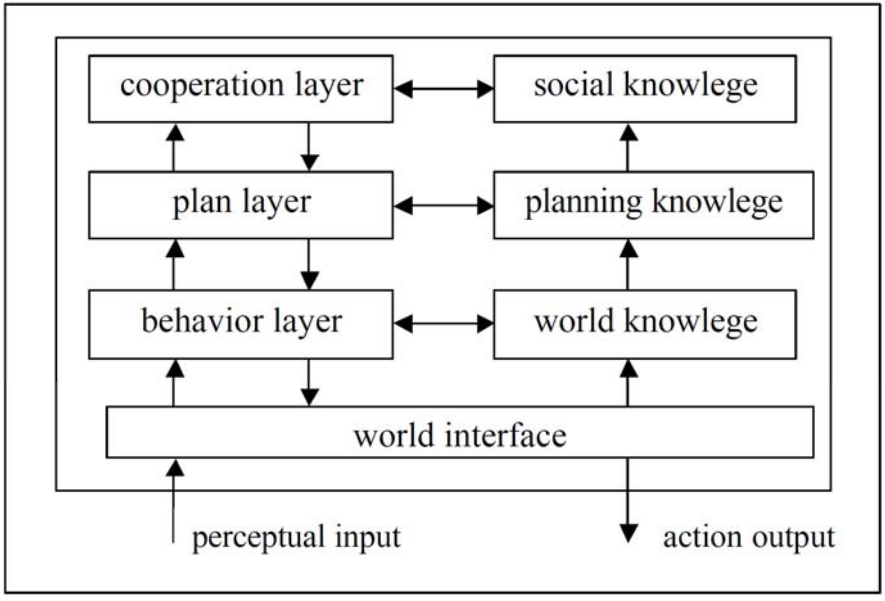
سرکوب *Suppress*	**سانسور** *Censor*
از انتقال اطلاعات حسگری به لایه‌های کنترلی جلوگیری می‌کند.	خروجی‌های کنش از لایه‌های کنترلی می‌آید.

معماری‌های لایه‌ای

انواع لایه‌بندی: لایه‌بندی عمودی: مثال (معماری اینتررراپ)

INTERRRAP ARCHITECTURE

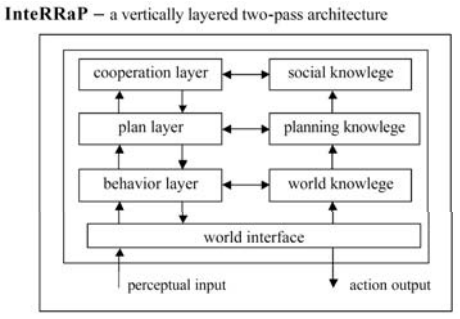
InteRRaP – a vertically layered two-pass architecture



معماری‌های لایه‌ای

انواع لایه‌بندی: لایه‌بندی عمودی: مثال (معماری اینتررراپ)

INTERRRAP ARCHITECTURE



دارای سه لایه برای تولید کنش

لایه‌ی پایینی: مبتنی بر رفتار
Lower Layer: Behavior-Based

با رفتار واکنشی سروکار دارد.

لایه‌ی میانی: طرح‌ریزی محلی
Middle Layer: Local Planning

با طرح‌ریزی روزمره برای رسیدن به اهداف سروکار دارد.

لایه‌ی بالایی: طرح‌ریزی همکارانه
Upper Layer: Cooperative Planning

با تعاملات اجتماعی سروکار دارد.

هر لایه یک پایگاه دانایی (*KB*) دارد

شامل بازنمایی‌های دنیا متناسب با همان لایه

لایه‌ها برای دست‌یابی به انسجام با هم تعامل می‌کنند

دو تعامل اصلی بین لایه‌ها:

فعال‌سازی پایین به بالا
Bottom-Up Activation

وقتی یک لایه‌ی پایینی کنترل را به یک لایه‌ی بالایی انتقال می‌دهد، زیرا خودش برای اداره‌ی وضعیت فعلی شایستگی ندارد.

اجرای بالا به پایین
Top-Down Execution

وقتی یک لایه‌ی بالاتر از امکانات فراهم شده در یک لایه‌ی پایینی استفاده می‌کند تا به یک هدف برسد.

معماری‌های لایه‌ای

جمع‌بندی

معماری‌های لایه‌ای متداول‌ترین کلاس عمومی از معماری‌های عامل است.

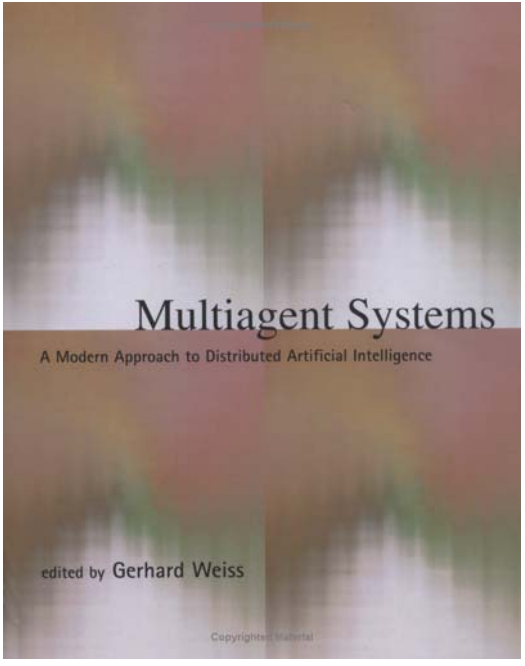
مزایا	معایب
❋ این معماری‌ها، یک تفکیک طبیعی از کارکردها را بازنمایی می‌کنند.	❋ فقدان وضوح مفهومی و معنایی (که در روی‌کردهای بدون لایه‌بندی وجود دارد) ❋ پیچیدگی ممکن در تعاملات میان لایه‌ها (این مشکل تا حدی با معماری‌های عمودی دوگذره حل می‌شود.)

معماری‌های عامل در سیستم‌های چندعاملی

۲

منابع

منبع اصلی



Gerhard Weiss (ed.),
Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence,
MIT Press, 1999.
Chapter 1

1 Intelligent Agents

Michael Wooldridge

1.1 Introduction

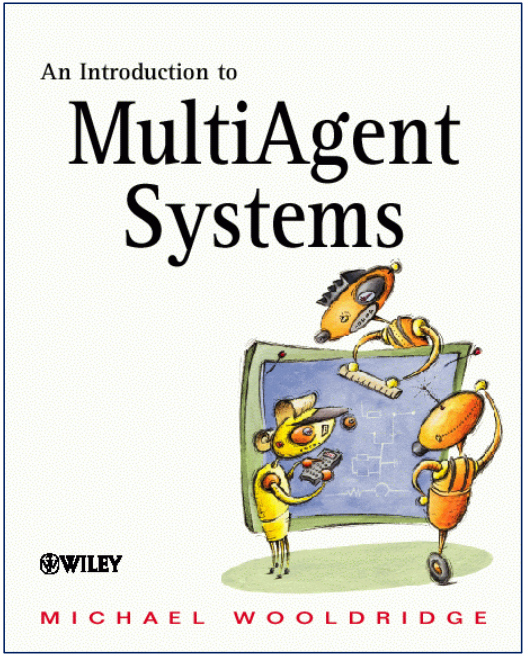
Computers are not very good at knowing what to do: every action a computer performs must be explicitly anticipated, planned for, and coded by a programmer. If a computer program ever encounters a situation that its designer did not anticipate, then the result is not usually pretty – a system crash at best, multiple loss of life at worst. This mundane fact is at the heart of our relationship with computers. It is so self-evident to the computer literate that it is rarely mentioned. And yet it comes as a complete surprise to those encountering computers for the first time.

For the most part, we are happy to accept computers as obedient, literal, unimaginative servants. For many applications (such as payroll processing), it is entirely acceptable. However, for an increasingly large number of applications, we require systems that can *decide for themselves* what they need to do in order to satisfy their design objectives. Such computer systems are known as *agents*. Agents that must operate robustly in rapidly changing, unpredictable, or open environments, where there is a significant possibility that actions can *fail* are known as *intelligent agents*, or sometimes *autonomous agents*. Here are examples of recent application areas for intelligent agents:

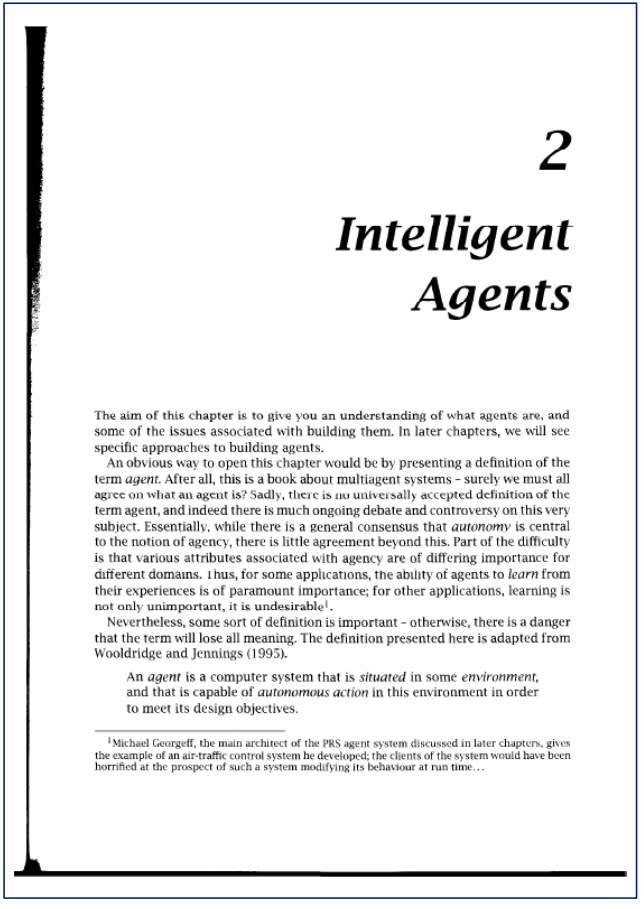
- When a space probe makes its long flight from Earth to the outer planets, a ground crew is usually required to continually track its progress, and decide how to deal with unexpected eventualities. This is costly and, if decisions are required *quickly*, it is simply not practicable. For these reasons, organisations like NASA are seriously investigating the possibility of making probes more autonomous – giving them richer decision making capabilities and responsibilities.
- Searching the Internet for the answer to a specific query can be a long and tedious process. So, why not allow a computer program – an agent – do searches for us? The agent would typically be given a query that would require synthesising pieces of information from various different Internet information sources. Failure would occur when a particular resource was unavailable, (perhaps due to network failure), or where results could not be obtained.

This chapter is about intelligent agents. Specifically, it aims to give you a thorough

منبع اصلی



Michael Wooldridge,
An Introduction to Multiagent Systems,
John Wiley & Sons, 2002.
Chapters 2, 3, 4, 5



2

Intelligent Agents

The aim of this chapter is to give you an understanding of what agents are, and some of the issues associated with building them. In later chapters, we will see specific approaches to building agents.

An obvious way to open this chapter would be by presenting a definition of the term *agent*. After all, this is a book about multiagent systems – surely we must all agree on what an agent is? Sadly, there is no universally accepted definition of the term agent, and indeed there is much ongoing debate and controversy on this very subject. Essentially, while there is a general consensus that *autonomy* is central to the notion of agency, there is little agreement beyond this. Part of the difficulty is that various attributes associated with agency are of differing importance for different domains. Thus, for some applications, the ability of agents to *learn* from their experiences is of paramount importance; for other applications, learning is not only unimportant, it is undesirable[1].

Nevertheless, some sort of definition is important – otherwise, there is a danger that the term will lose all meaning. The definition presented here is adapted from Wooldridge and Jennings (1995).

> An *agent* is a computer system that is *situated* in some *environment*, and that is capable of *autonomous action* in this environment in order to meet its design objectives.

[1] Michael Georgeff, the main architect of the PRS agent system discussed in later chapters, gives the example of an air-traffic control system he developed; the clients of the system would have been horrified at the prospect of such a system modifying its behaviour at run time…