II. Agentní systémy, základní architektury, jejich modely a realizace

Podklady k přednáškám kurzu AGS

František Zbořil ml.
zborilf@fit.vutbr.cz

Obsah přednášky

- Modely agentních systémů
  - Reaktivní agenti, základní, s vnitřním stavem
  - Agenti se znalostí, kognitivní agenti
  - Proaktivní agenti, praktické rozhodování
  - Hybridní agenti
- Realizace agentů v softwarové podobě, programy
- Realizace agentů v hardwarové podobě, architektury
  - Situovaný automat
  - Subsumpční architektura
  - Vrstvené architektury

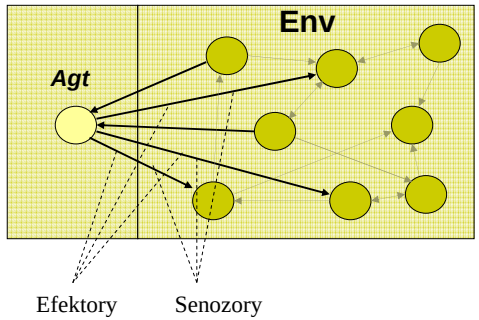
Systém, model

- **S**=(**U**,**R**), kde **U** je universum zahrnující všechny prvky systému, **R** je charakteristika
- Časová množina **T** je množina všech časových okamžiků, ve kterých dochází ke změně systému **T**=$\{t_0,t_1,t_2 \ldots\}$
- Chování systému s okolím u_O je

$$\chi: \sigma_I(u_O)^{\mathbf{T}} \rightarrow \sigma_O(u_O)^{\mathbf{T}}$$

Agent a prostředí (model)

- Agent je jeden z prvků agentního systému a okolí je podsystém **Env**=**AS**-{*Agt*}



Env je systém, kterému tvoří okolí agent Agt, stejně tak část prostředí je okolím pro agenta Agt. Vstupním okolím agenta jsou prvky, které vnímá svými senozory a výstupním prvky prostředí, které může ovlivnit svými efektory.

Agetní systém (model)

- Modely agentního systému budou vždy dvojice **AS**=<**Agt**, **Env**> kde *Agt* je agent a **Env** prostředí. Jak agent, tak prostředí jsou opět struktury.
- V modelech budou použity následující množiny a funkce
  - **E** : množina stavů prostředí
  - **P** : množina vjemů
  - χ : funkce chování prostředí ,
  - **A** : množina akcí prováděných agentem
  - ***S*** : funkce vjemu agenta
  - **C**: množina vnitřních stavů agenta
  - ξ : funkce chování agenta
  - **T** : časová množina, $\mathbf{T}= \mathbf{T}_{\mathbf{ENV}} \cup \mathbf{T}_{\mathbf{AGT}}$

Agent a prostředí

- *Act* je stav na výstupních branách agenta, $Act \in \mathbf{A}$
- *See* je stav na vstupních branách agenta $See \in \mathbf{P}$
- Množina stavů prostředí **E** zahrnuje všechny možné konfigurace prostředí **Env**, tedy stavy jejich vstupního a výstupního prostoru a vnitřní stavy prvků prostředí
- Chování prostředí je tedy χ: $(\mathbf{A}\times\mathbf{E})^{\mathbf{Tenv}} \rightarrow (\mathbf{P}\times\mathbf{E})^{\mathbf{Tenv}}$
- … a chování agenta ξ: $(\mathbf{P}\times\mathbf{C})^{\mathbf{Tagt}} \rightarrow (\mathbf{A}\times\mathbf{C})^{\mathbf{Tagt}}$

Prostředí (model)

- Prostředí bude vnímáno z pohledu agenta jako okolí reprezentované jedním prvkem, které může nabývat stavů z množiny

$$\mathbf{E}=\{e_1,e_2 \ldots\}$$

- Statické, deterministické

$$\chi : \mathbf{E}\times\mathbf{A} \rightarrow \mathbf{E}$$

- Dynamické, deterministické

$$\chi : \mathbf{E}\times\mathbf{A} \rightarrow \mathbf{E}, \chi : \mathbf{E} \rightarrow \mathbf{E}$$

- Dynamické, nedeterministické

$$\chi : \mathbf{E}\times\mathbf{A}\times\mathbf{T} \rightarrow \mathbf{E}\times\mathbf{T}, \chi : \mathbf{E}\times\mathbf{T} \rightarrow \mathbf{E}\times\mathbf{T}$$

Reaktivní agenti

- Reaktivní agent adekvátně reaguje na daný stav prostředí
- Většinou nemá symbolickou reprezentaci prostředí (model) a neprovádějí na základě tohoto modelu odvozování ohledně svého dalšího chování – svět sám o sobě je nejlepším modelem sebe samotného
- Čistě reaktivní agent, reaktivní agent s vnitřním stavem, subsumpční architektura

Model reaktivního agenta

- Budeme nejprve uvažovat prostředí diskrétní, statické a deterministické

  $\mathbf{Env_1}=<e_{10},\mathbf{E_1},\chi_1>$,

  $\mathbf{E_1}=\{e_{11},e_{12}\ \ldots\ e_{1i}\}$, $\chi_1: \mathbf{E_1}\times\mathbf{A_1}\rightarrow\mathbf{E_1}$
- Reaktivní agent

  $\mathbf{Agt_1}=<\mathbf{A_1}, \mathbf{P_1}, \boldsymbol{\xi}_1, \boldsymbol{See_1}>$

  $\mathbf{A_1}=\{a_{11},a_{12}\ \ldots\ a_{1i}\}$, $\mathbf{P_1}=\{p_{11},p_{12}\ \ldots\ p_{1i}\}$

  $\boldsymbol{See_1}: \mathbf{E_1}\rightarrow\mathbf{P_1}$, $\boldsymbol{\xi}_1: \mathbf{P_1}\rightarrow\mathbf{A_1}$

Reaktivní agent, příklad

- Prostředí tvoří šachovnice n×m
- Stav prostředí je dán pozicí agenta na šachovnici
- Agent vnímá nejbližší stěnu (s prioritou sever/jih) a může se pohybovat vodorovně a svisle
- $\mathbf{E_1}$={[1,1],[1,2],…[1,n],[2,1],[2,2]…[2,n]… [m,1] … [m,n]}
- $\mathbf{P_1}$={*north*, *south*, *west*, *east*, *null*}
- $\mathbf{A_1}$={*go_north*, *go_south*, *go_west*, *go_east*}

Reaktivní agent, příklad

- $\boldsymbol{S_1}$([a,b])=*north* a<m-a ∧ a<=b ∧ a<=n-b;
- $\boldsymbol{S_1}$([a,b])=*south* m-a<a ∧ m-a<=b ∧ m-a<=n-b;
- $\boldsymbol{S_1}$([a,b])=*west* b<n-b ∧ b<a ∧ b<m-a;
- $\boldsymbol{S_1}$([a,b])=*east* n-b<b ∧ n-b<a ∧ n-b<m-a;
- $\boldsymbol{S_1}$([a,b])=*null* else
- $\boldsymbol{\xi}_1$(*north*)=*go_south*; $\boldsymbol{\xi}_2$(*south*)=*go_north*;
- $\boldsymbol{\xi}_1$(*west*)=*go_east*; $\boldsymbol{\xi}_2$(*east*)=*go_west*;
- χ_1([a,b], *go_north*)=[max(a-1,0),b]
- χ_1([a,b], *go_south*)=[min(a+1,m),b]
- χ_1([a,b], *go_west*)=[a,max(a-1,0)]
- χ_1([a,b], *go_east*)=[a,min(a+1,n)]

Běh

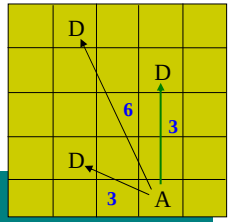
- Agent vykonáváním akcí ovlivňuje přostředí, jednání agenta lze vyjádřit posloupnost akce/stav prostředí

  $(e_0,a_{t1},e_{t1},a_{t2},e_{t2}\ \ldots\ a_{tn},e_{tn}\ldots)$
- Běh $\mathbf{R}^{e'}$je posloupnost, která začíná ve stavu a končí ve stavu *e'*
- Pro uvedený příklad, rozměry šachovnice 5×5 a e_{10}=[1,1] existuje jeden běh $\mathbf{R_1}^{[3,3]}$

  $\mathbf{R_1}^{[3,3]}$=([1,1], *go_south*, [1,2], *go_east*,[2,2], *go_south*, [2,3], *go_east*, [3,3])

Reaktivní agent v dynamickém a nedeterministickém prostředí, příklad

- $\mathbf{Env}_2$=<e_{20},$\mathbf{E}_2$,χ_2,$\mathbf{T}_{2ENV}$>, $\mathbf{Agt}_2$=<A_2,$\mathbf{P}_2$,ξ_2,See_2,$\mathbf{T}_{2AGT}$>
- Tileworld – obdobné prostředí, ale se vznikajícími ‘dírami’, agent má za úkol vzniklé díry zaplňovat dlaždicemi



- E_2 – stav musí zahrnovat i všechny díry
- $\mathbf{A}_2$, $\mathbf{P}_2$ a $\mathbf{S}_2$ jsou stejné jako v předchozím případě
- ξ(north)=go_north ξ(south)=go_south atd.
- S_2 určí směr jako největší vzdálenost k nejbližší díře (priorita N a S)
- χ_2 :
  - 1, k transformacím se po provedení akce může přidat novou množinu děr
  - 2, může dojít ke změně prostředí i pokud agent nekoná

Reaktivní agent v dynamickém a nedeterministickém prostředí, příklad

- Dynamické prostředí by mělo pro některé (nebo všechny) stavy definovanou funkci chování prostředí (funkce přidej díru mapuje výsledný stav pro nějaký vstupní stav a souřadnice nové díry):

  $\forall e \in \mathbf{E}\ \chi_2(e)$=add_hole(e,[rand(n),rand(m)]) a
  $\forall e \in \mathbf{E}\ \chi_2(e,a)$=add_hole($\chi_1$(e,a),[rand(n),rand(m)])

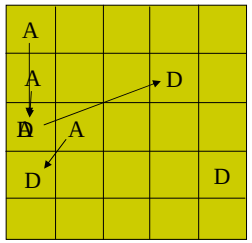
- Nedeterministické prostředí navíc může mít pro stejný stav v různých okamžicích různé výsledné stavy

  $\forall e \in \mathbf{E}\ \forall t \in \mathbf{T}_{env}\ \chi_2(e,t)$=add_hole(e,[rand(n),rand(m)]) a
  $\forall e \in \mathbf{E}\ \forall t \in \mathbf{T}_{agt}\ \chi_2(e,a,t)$=add_hole($\chi_1$(e,a),[rand(n),rand(m)])

- **add_hole přidá ‘díru’ do reprezentace prostředí a naopak vymaže díru z šachovnice, pokud na ní stojí agent*

Příklad běhu agenta

- T=(1,2,3,4,5,6 …), e_0
- e_0=([1,1],([3,1],[2,4],[4,5]))
- ξ(**S**())= ξ(*south*)=*go_south*
- χ(e_0, *go_south*,1)=e_1
- e_1=([2,1],([3,1],[2,4],[4,5]))
- ξ(**S**())= ξ(*south*)=*go_south*
- χ(e_1, *go_south*,2)=e_2
- e_2=([3,1],([2,4],[4,5]))
- ξ(**S**())= ξ(*east*)=*go_east*
- χ(e_1, *go_south*,3)=e_3
- e_3=([3,2],([2,4],[4,5],**[4,1]**)
- ξ(**S**())= ξ(*south*)=*go_south* …



Reaktivní agent v dynamickém prostředí

- Agent je schopen přizpůsobit své chování i měnícímu se prostředí – reaguje na jeho změny
- Jeho chování ve vysoce dynamickém prostředí *může* být efektivnější, než pokud by následoval předem sestavený plán
- Na druhou stranu agent nemá zadání konkrétního cíle (podcíle) a nemusí nikdy dosáhnout cílů, pro které byl sestaven a které je možné dosáhnout

Program řízení agenta (program)

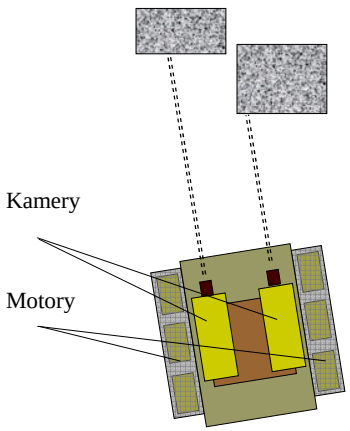
- Řídící program agenta bude zapsán v imperativním PASCAL pseudo-jazyce
- Při zápisu algoritmu se budou předpokládat dva výrazy `See()` a `Do(a)`. `See()` je funkce, která zobrazuje vjem v aktuálním stavu prostředí, příkaz `Do(a)` znamená provedení akce `a` agentem

Reaktivní agent, řídící program

```
while 1 do
        percept=see();
        action=select(percept);
        if action!=null then do(action)
end

function select(percept):TAction
        action:=null;
        switch(percept)
                case p1: action=aca
                case p2: action=acb
                                :
                case pn: action=acx
        return(action)
end
```


Robot s reaktivní architekturou



- Dvě kamery umístěny vedle sebe a vnímající prostředí ve směru pohybu robota zjišťují vzdálenost překážky
- Rozdíly rychlostí pravého a levého pásu jsou závislé na naměřených rozdílech mezi levou a pravou kamerou
- S přibližující se překážkou se robot zpomaluje, v případě rozdílu je rychlejší ten pás, který je na straně bližší překážky.
- Robot je schopen se vyhýbat překážkám a tedy chová se inteligentně jen na základě popudů z prostředí a bez toho, aby uchovával vnitřní stavu

Reaktivní agent s vnitřním stavem

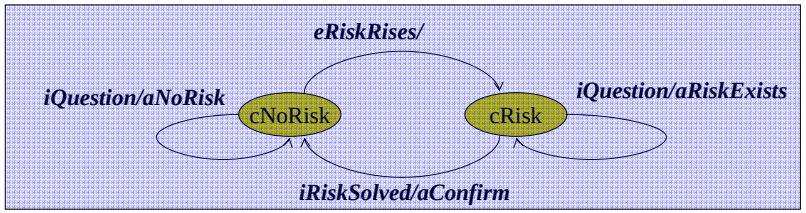
- $\mathbf{Env}_3$ bude modelem nedeterministického, dynamického prostředí
- Agent je struktura

$$\mathbf{Agt}_3=<\mathbf{A}_3,\mathbf{P}_3,\mathbf{C}_3,\boldsymbol{\xi}_3,\boldsymbol{See}_3,\ c_0,\mathbf{T}_{\mathbf{AGT}}>$$

- C_3 je množina vnitřních stavů a c_0 je počáteční stav.
- Agent je potom stavový, sekvenční systém.
- Nejprve budeme uvažovat vnitřní stav z konečné množiny stavů $\mathbf{C}_3=\{c_{31}, c_{32}, c_{33}, \ldots\}$
- Chování agenta bude dáno nejen vjemem, ale i vnitřním stavem
- *souvislost s Moorovým nebo Mealyho automatem?

Reaktivní agent s vnitřním stavem, příklad

- **A**={aNoRisk,aRiskExists,aConfirm}
- **P**={eRiskRises,iQuestion,iRiskSolved}, ***See*** pro tento případ vynecháme
- **C**={cNoRisk,cRisk}, c_0=cNoRisk
- ξ(iQuestion,cNoRisk)=(aNoRisk,cNoRisk)
- ξ(iQuestion,cRisk)=(aRiskExists,cRisk)
- ξ(eRiskRises,cNoRisk)=(-,cRisk)
- ξ(iRiskSolved,cRisk)=(aConfirm,cNoRisk)



Reaktivní SW agent s vnitřním stavem

```
state=c0
while 1 do
  percept=see();
state=next_state(state,percept)
  action=select(percept,state);
  if action!=null do(action)
end
```


Softwarový agent s vnitřním stavem

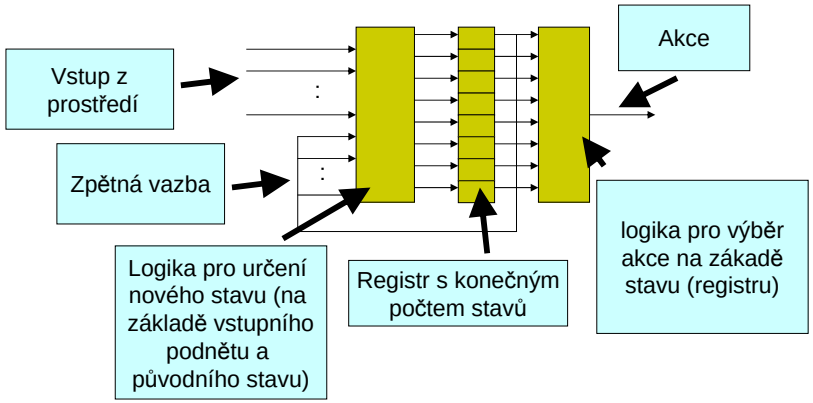
```
function select(percept,state):TAction
action:=null;
switch(state)
      case s0:
            switch(percept)
            case p1: action=acb
            case p2: action=acb
                        :
            case pm: action=acb
      case s1: action=acb
                  :
      case sn: action=acx
return(action)
end
```

… a obdobně funkce next_state

Situované automaty

- Systém nemá vnitřní model prostředí, nicméně jeho vnitřní stav závisí na minulých stavech systému – tedy jedná se o sekvenční systém (systém, jehož stav závisí na sekvenci minulých podnětů na vstupech)
- Kombinačními poli jsou realizovány logické funkce pro přechod mezi stavy a pro výběr akce pro daný stav

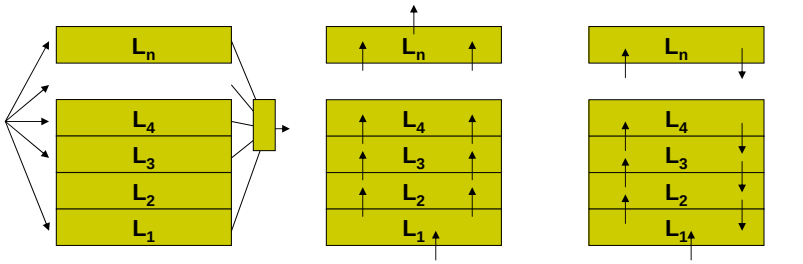
Situované automaty



Vrstvené arichitektury

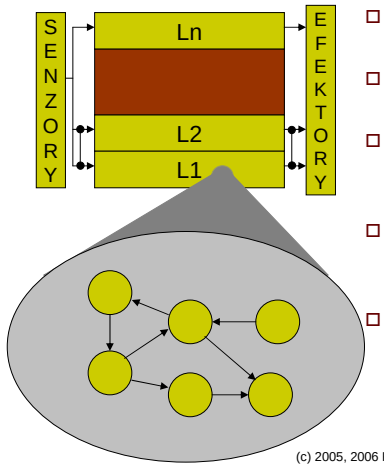
- Dvě nebo více vrstev, každá z vrstev představuje buďto jeden originální vzorek chování (dekompozice podle chování), nebo funkční modul (dekompozice podle funkce)
- Jednotlivé vrstvy jsou propojeny a navzájem komunikují předáváním zpráv
- Vrstvené architektury mohou být horizontálně, nebo vertikálně vrstvené

Vrstvené architektury



Horizontální *Vertikální, jednocestná* *Vertikální, dvoucestná*

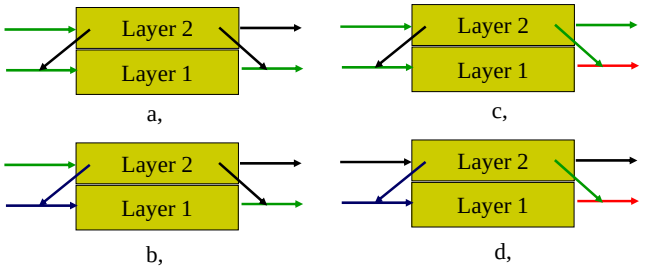
Subsumpční architektura (Subsumption Architecture)



- Jedna z architektur založených na chování (Brooks 91)
- Vrstvená architektura, jednotlivé vrstvy / moduly reprezentují chování
- v jednotlivých vrstvách jsou dány vzorky chování jako stavové automaty s časováním
- Vrstvy mají priority určené vzestupně, vyšší vrstvy mohou měnit stav vrstev nižších
- vrstvy s vyšší prioritou mohou rovněž ovlivňovat senzory a blokovat akce, které chtějí vykonat vrstvy s nižší prioritou

Subsumpční architektura

- Vrstvy mohou být propojeny mezi sebou z důvodu potlačení (suppresing) vstupu a znehybnění (inhibition) výstupu (Brooks p8)
- Pokud se vyšší vrstva pošle informace po vstupním propojení, je původní informace z prostředí potlačena a zaslaná informace brána jako vstup pro nižší vrstvu (b), vyšší vrstva pokud začne konat, zablokuje výstup nižší vrstvy (c). Znehybnění a potlačení může probíhat současně (d)



Agent se znalostí

- Agentův vnitřní stav *c* reprezentuje nějaký model **M**, většinou model prostředí, ve kterém se agent nachází. Výraz **Know**(*f*) bude značit to, že *f* vyplývá z agentova modelu prostředí
- Pokud je model popsán v jazyce formální logiky, potom sémantika výrazu Know je následující:

$$\mathbf{Know}(f) \textit{ iff } \mathbf{M} \models f$$

Praktické odvozování

- Teoretické odvozování odvozuje platnost formule f v nějakém stavu *s* modelu **M**.

$$\mathbf{M},s \models f$$

- Praktické odvozování odvozuje, co je třeba učinit, aby bylo v modelu **M** ze stavu *s* dosaženo platnosti cíle g resp. stavu systému *s'*, ve kterém je cíl g splněn.

$$\mathbf{M},s,g \mid\sim \mathrm{Do}(\pi) \rightarrow \mathbf{M},\mathbf{T^P}(s,\pi) \models g$$

- π je plán, $\pi \in \mathbf{A}^*$, $\mathbf{T^P}$ je funkce $\mathbf{T^P}: \mathbf{S}\times\mathbf{A}^* \rightarrow \mathbf{S}$

Užitkový model

- Každému ze stavů je dána hodnota, která udává užitek pro agenta být v tomto stavu

$$\boldsymbol{U}: \mathbf{E} \rightarrow \Re$$

- Později, v multiagentních skupinách, je užitek každého ze stavů definován pro každého agenta

$$\boldsymbol{U}: \mathbf{E}\times\mathbf{A} \rightarrow \Re$$

- Předpokládá se, že agent má znalost o užitku stavů.

$$\forall e \in \mathbf{E}(\boldsymbol{U}(e)=n \rightarrow \mathbf{Know}(\boldsymbol{U}(e,n)))$$

- Agent také ví, do jakého stavu se dostane po provedení kterékoli ze svých akcí

$$\forall a \in \boldsymbol{A}(\boldsymbol{T}(e,a)=e' \rightarrow \mathbf{Know}(\boldsymbol{T}(e,a)=e'))$$

Rozhodování agenta se znalostmi o užiktu stavů

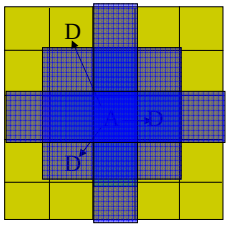
- Racionální agent volí ze stavů, do kterých se může dostat po provedení nějaké akce, ten, který má pro něj největší užitek

$$\xi(e,\boldsymbol{M})=a,\ a=\text{argmax}_{\forall a\in \mathbf{A}}\{\boldsymbol{U}(\boldsymbol{T}(e,a))\}$$

- *Pozn. byl použit zjednodušený zápis bez použití operátoru* ***Know***

Příklad

- Příklady ohodnocení prostředí pro úlohu Tileword



U(e,**M**) = Počet děr v prostředí
Počet děr v okolí do určité vzdálenosti
Součet vzdáleností děr

Kognitivní agent

- Kognitivní agent je schopen zpracovávat vstupní informace. (z latinského *cogito*, myslet)
- V umělé inteligenci je kognitivní systém chápán jako systém, který uchovává znalosti o prostředí a je schopný se během své existence učit.
- Kognitivní agenti budou mít v čase měnící se reprezentaci prostředí a bude jednat na základě této reprezentace -> praktické rozhodování

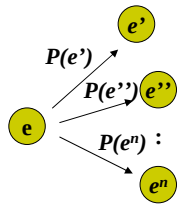
Kognitivní agent, nedeterministické prostředí

- Agent zná (z předchozí zkušenosti) pravděpodobnostní rozložení nad možnými stavy po provedení nějaké akce
- Užitek provedení akce *a* ve stavu *e* je

$$\mathbf{U}(a)=\sum \mathrm{p}(e'|e,a)\mathbf{U}(e')$$

- A racionální agent vybírá akci s maximálním užitkem

$$\xi(e,\boldsymbol{M})=a,$$
$$\mathbf{U}(a)=\boldsymbol{argmax}_{\forall a'\in \mathbf{A}}\{\mathbf{U}(a')\}$$



Proaktivní agent

- Proaktivní agent je iniciativní vzhledem ke svým cílům
- Vnitřní stav je struktura <**KB**,π> a plánování je proces, který na základě znalostí a cílů (uložených v **KB**, nebo definovaných při návrhu agenta) vytvoří plán
- Plán je posloupnost akcí $\pi \in \mathbf{A}^*$, které *mohou* vést ke splnění zvoleného cíle

Proaktivní agent, řídící smyčka

```
while 1 do
   percept:=see();
   knowledge:=updatekb(knowledge,percept);
   if(plan!=empty)
        perform_action(head(plan));
        plan:=tail(plan);
   else
        plan:=construct_plan(knowledge);
```

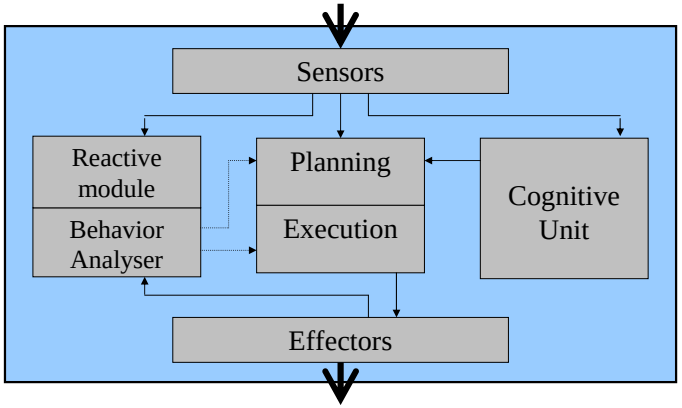

Proaktivní agent v reálném prostředí

- Změna prostředí může vést k tomu,
  - že cíl není dosažitelný sestaveným plánem
  - zvolený plán není optimální
  - je výhodnější následovat jiný cíl
- Racionální agent v reálném prostředí -> Měl by být schopný sledovat vhodnost existujícího plánu pro aktuální stav prostředí a případně dokázat rozhodnout o jeho přehodnocení

Hybridní architektury

- Většinou se jedná o vrstvené architektury, využívají kombinace reaktivního a proaktivního přístupu.
- Reaktivní přístup může být využit u hybridního agenta v kombinaci s proaktivním -> reaktivní složka kontroluje chování agenta řízeného plánem a na základě porovnání chování reaktivních a proaktivních složek

Hybridní agent, příklad – hybridní architektura pro přehodnocování plánu



Hybridní agent, příklad – hybridní architektura pro přehodnocování plánu

1.1δ=0, n=1, KB=initial knowledge
1.2 Create a plan π
2. While not Empty(π) do
3. Take first action from the plan α=Head(π)
4. Execute(α)
5. Update knowledge about the environment *e*
    6.1 Find αbest as an answer of the reactive module to *e*
    6.2 Update KB with *e*
    6.2 Compute δ=δ+U(αbest)-U(α)
    6.3 If δ/n>threshold then Reconsider(π)
    6.4 n=n+1
7. π=Tail(π)
8. Goto 1.2