Úvod

- Začalo v 90. letech
- V jednu chvíli to bylo populární jako deep neural networks nyní
- Budeme se bavit o agentech
    - Co jsou?
    - Kde se nachází?
    - …
- Důležitým typem jsou *reasoning agents* (uvažující agenti)
    - BDI architektura
        - Believes
        - Desires
        - Intensions
- Agents + Environment = Multi-agent system
    - Agenti mezi sebou komunikují
- Jak agenti komunikují?
    - Speech acts
        - Filozofie + lingvistika
    - Ontologie
- Kooperace agentů
- Soupeření agentů
- Agent
    - V 90. letech zformalizováno
    - Počítačový systém schopný pracovat nezávisle pro uživatele
- Jak vytvořit agenty nebo systém agentů?
    - Navíc s
        - Kooperací
        - Koordinace
        - Vyjednávání
- **Existuje hromada definic agenta**
    - Často závisí k čemu agenta potřebujeme, proto jsou některé definice jsou k tomu přizpůsobené
- Agent
    - Mechanismus selekce akce → **architektura agenta**
        - Agent musí být schopen vybrat si akci a provést ji
        - SW architektura, která umožňuje proces selekce agenta
        - Specifikuje jak může být agent dekomponován na moduly
        - Jaká je interakce mezi moduly
        - Musí být schopna “zodpovědět” otázku, co agent udělá podle jeho stavu (a dat ze senzorů)
- Agent (def 1)
    - Je systém **situovaný v prostředí**
    - Detekuje prostředí pomocí senzorů
    - Koná podle toho co bylo zdetekováno

    - Systém dokáže ovlivnit prostředí
- Agent (def 2)
    - Agent **je** systém **situovaný v prostředí** schopný provádět autonomní akce, aby dosáhl svého cíle
- Prostředí
    - Ovlivňuje složitost provedení akcí agentem
    - Dělení podle informací, které agent získá z prostředí
        - Fully observable
            - Agent ví všechno o prostředí
            - Agent dokáže získat celý stav prostředí z dat senzorů
        - Partially observable
            - Agent nezná celý stav prostředí
    - Dělení podle toho, kdy se mění prostředí
        - Statické
            - Prostředí se mění pouze akcemi agenta
        - Dynamické
            - Prostředí se může měnit i jinými způsoby
            - Př. agent je řízení teploty v místnosti a někdo otevře okno
    - Dělení podle toho, jaký výsledek akce může nastat
        - Deterministické
            - Provedení stejné akce má za následek stejný výsledek
        - Nedeterministické
            - Výsledek jedné akce je podmnožina stavů prostředí
            - Často v partially observable prostředích nebo dynamických prostředích
    - Podle počtu stavů
        - Diskrétní
            - Konečný počet stavů
            - Obvykle budeme uvažovat
        - Spojité
            - “Nekonečný počet stavů”
                - Reálně nám senzory stejně jsou schopny vrátit konečný počet stavů
            - Souvisí s robotikou
- Nejlepší by bylo, aby všechna prostředí byla fully observable, statická, deterministická a diskrétní
    - Bohužel málo kdy případ pro zajímavá prostředí
    - Většinou alespoň jedna podmínka neplatí
- Proces výběru akce agentem = **deliberation**
    - Agent má na výběr z množiny akcí
    - Chceme, aby vybral tu správnou (nejlepší)
        - Plánování
        - Výběr založený na reakci
- Př. termostat v místnosti (je to agent)

    - Prostředí je místnost
    - Máme 2 vjemy: “cold” a “ok”
    - Máme 2 akce: “on” a “off”
    - Máme rozhodovací jednotku
        - Proměnit hodnoty ze senzorů do on/off
        - Jakoby pravidlo = mechanismus výběru akce
            - Moc cold → on
            - “Ok” → off
    - Prostředí není statické → někdo může otevřít okno
- Př. SW daemon xbiff
    - Checkuje emaily a zobrazí ikonu na obrazovce, když je nový email
    - Opět to je agent
    - Prostředí je komplikovanější
        - Dostává signály od OS
- Inteligentní agenti by měli být
    - Proaktivní
        - Má vlastní cíle, které se aktivně snaží splnit
    - Sociální
        - Schopen komunikace s ostatními agenty
    - Reaktivní
        - Detekuje (hodnoty) prostředí pomocí senzorů a koná v závislosti na změnách
    - Automní
        - Schopnost samostatně pracovat
    - *Někdy se tomuto v literatuře říká Weak Agency*
        - Minimální koncept agenta
- Daniel Dennet
    - Filozof zaměřující se na filosofii mysli
    - **Materialistický** přístup k filosofii → rozumí matematice a počítačům
    - Jak lidská mysl vymyslí popis systému, který nezná
        - Intentional (intenzionální) system
    - Lidé si obvykle vytváření vlastní teorii podle toho, co vypozorovali
        - Metafory – myš, okno, tlačítko u PC
    - Stance (postoje osoby snažící se popsat systém)
        - Physical stance
            - Popis pomocí zákonů přírodních věd (fyzika, matematika, …)
            - Pro experty
        - Design stance
            - Můžeme rozumět pouze části nebo vůbec nerozumíme principu fungování
            - Budík
                - Nevím přesně jak funguje
                - Ale rozumím **proč** byl vyroben a **jak** ho používat
            - Důležitá je **funkce**

        - Intentional stance
            - Nemám dostatek informací, abychom dokázali pochopit systém jako u design stance
            - Vytvoříme si vlastní teorii jak systém funguje a budeme predikovat chování systému
            - **Přesvědčení** a **úmysly**
            - Př. v kameře jsou skřítci, kteří umějí velmi rychle nakreslit obrázek
            - Dáváme vlastnosti jako intentions a desires neživým věcem jako počítači
                - Ačkoliv víme, že to tak není
                - Počítač nemá emoční problémy…
            - Je to ale pro nás užitečné, získáme nějaké porozumění, i když je třeba smyšlené, umožní nám to ale **predikovat chování systému**

(!) Abstraktní architektury agentů

- Pokusíme se formalizovat abstraktní pohled na agenta a jeho interakci s prostředím
- Množina stavů prostředí $E = \{e_1, e_2, \ldots\}$
- Prostředí se nachází v **jednom** z **konečného** množství stavů
    - Pokud by prostředí nemělo konečný počet stavů, tak z nich uděláme konečný počet
- Množina akcí agenta $A = \{a_1, a_2, \ldots\}$
- Agent má k dispozici konečné množství akcí
- Pokusíme se modelovat jak agent interaguje (vzájemně na sebe působí) s prostředím
    - Prostředí je v počátečním stavu
    - Agent vybere akci (v závislosti na stavu prostředí)
    - Prostředí může změnit stav do jednoho ze stavů z $F \subset E$
    - Prostředí odpoví změnou do $f \in F$
        - My nevíme, do kterého, výběr f může být nedeterministický
    - Iteruje se: agent vybere novou akci (podle stavu prostředí), …
- Run (běh) agenta
    - Posloupnost alternujících stavů prostředí a akcí agenta
    - $r = e_0, a_0, e_1, a_1, e_2, a_2, \ldots, a_{n-1}, e_n$
    - R … množina všech konečných alternujících posloupností stavu a akce
    - $R_A \subset R$ je množina všech (konečných) alternujících posloupností taková, že končí libovolnou **akcí** z A
    - $R_E \subset R$ je množina všech (konečných) alternujících posloupností taková, že končí libovolným **stavem** z E
    - Pokud by agent neprováděl akci nebo se prostředí neměnilo, tak to stejně modelujeme jako alternující posloupnost – předchozí akce/stav bude stejná/ý s následující
- Vliv agenta na prostředí je modelován **state transformer funkcí**

    - $T: R_A \rightarrow 2^E$
    - Prostředí mají historii
        - Další stav prostředí nezávisí pouze na poslední akci agenta, ale i na **předchozích** akcích a stavech
    - Prostředí jsou nedeterministická
        - Nevím v jakém stavu bude prostředí po provedení akce agenta
        - Další stav může být libovolný z množiny možných stavů, ve kterých se prostředí může nacházet
            - Určena podle historie akcí agenta a předchozích stavech
    - Pokud $T(r) = \emptyset$, říkáme, že run (běh) skončil
- Prostředí
    - Definujeme jako $Env = (E, e_0, T)$
        - E … množina stavů prostředí
        - e_0 … počáteční stav
        - T … state transformer funkce
- Agent
    - $Ag: R_E \rightarrow A$
    - Agent je **deterministický** → agent vždy odpoví jednou akcí
    - AG … množina všech agentů
    - Systém
        - Definujeme jako dvojici agenta a prostředí
        - (Ag, Env)
    - $R(Ag, Env)$ … množinu běhů agenta Ag v prostředí Env
- $(e_0, a_0, e_1, a_1, e_2, \ldots)$ je běh agenta Ag v prostředí $Env = (E, e_0, T)$ právě když
    - e_0 je počáteční stav Env
    - $a_0 = Ag(e_0)$
    - Pro každé $u > 0 : e_u \in T((e_0, a_0, \ldots, a_{u-1})) \ \wedge \ a_u = Ag((e_0, a_0, \ldots, e_u))$
- Dva agenti Ag_1, Ag_2 jsou funkčně ekvivalentní vzhledem k prostředí Env právě když $R(Ag_1, Env) = R(Ag_2, Env)$
- Dva agenti Ag_1, Ag_2 jsou jednoduše funkčně ekvivalentní právě když jsou funkčně ekvivalentní vzhledem ke všem prostředím Env
- Čistě reaktivní agent (neboli tropistický agent)
    - $Ag: E \rightarrow A$
    - Reaguje **bez ohledu** na historii, jen na základě aktuálního stavu prostředí
    - **Ke každém čistě reaktivnímu agentovi existuje ekvivalentní standardní agent** (nemusí platit obráceně)
    - Příklad (termostat)
        - $Ag(e) = off$ pokud $e =$ "$teplota\ OK$"
        - $Ag(e) = on$ jinak
- Agent se stavem

- I … množina interních stavů agenta
- Per … množina vjemů agenta
- see … funkce vnímání agenta
  - $see : E \to Per$
- $action : I \to A$
- $next : I \times Per \to I$
- **Každého agenta se stavem lze převést na ekvivaleního standardního agenta**
- Práce
  - Agent začne v interním stavu i_0
  - Vnímá stav prostředí e a generuje vjem $see(e)$
  - Aktualizuje si vnitřní stav na $next(i_0, see(e))$
  - Vybere akci $action(next(i_0, see(e)))$
  - Provede ji a začne další cyklus

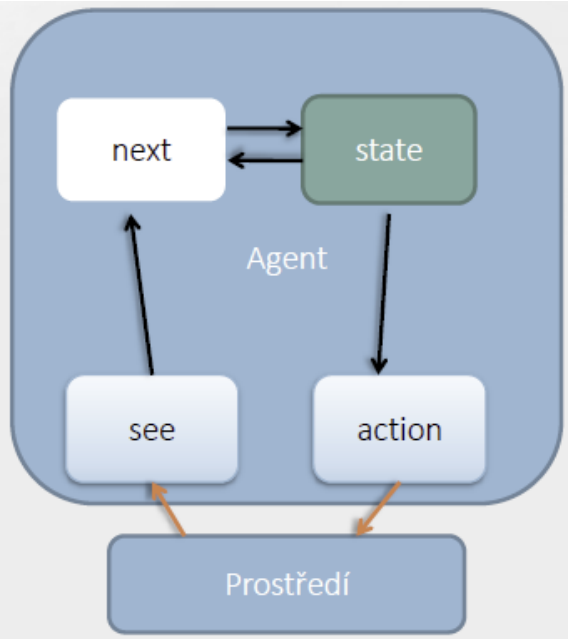


Schéma agenta se stavem

- Co má agent dělat a jak mu to říct
  - Nechceme agenta pevně naprogramovaného
  - Chceme mu říct **co** má dělat, **ne** jak to má dělat
  - Typicky definujeme problém/cíl nepřímo pomocí míry úspěšnosti agenta
  - Účelová funkce – ohodnocení stavů prostředí
    - $u : E \to \mathbb{R}$
  - Agent má za úkol dostat prostředí do stavů s **vysokou** hodnotou účelové funkce
- Účelová funkce běhu
  - Jak transformovat účelovou funkci pro stav na celý běh
  - Chceme $u : R \to \mathbb{R}$

    - Musíme nějak agregovat hodnoty účelové funkce pro stavy do běhu
    - Můžeme založit na
        - Hodnotě účelové funkce nejhoršího stavu, ve který agent navštívil
        - Průměr hodnot účelové funkce stavů navštívených agentem
        - *Není jasné, která varianta je lepší*
- Místo maximalizace zisku (hodnoty účelové funkce) můžeme **maximalizovat očekávanou hodnotu zisku**
    - $P(r \mid Ag,\ Env)$ … pravděpodobnost, že nastane běh r, když agent Ag je umístěn do prostředí Env
    - Platí, že $\sum_{r \in R(Ag,Env)} P(r \mid Ag,\ Env) = 1$
        - Suma přes všechny běhy
    - Optimální agent Ag_{opt}
        - $$Ag_{opt} = \underset{Ag \in AG}{\operatorname{argmax}} \sum_{r \in R(Ag,Env)} u(r) \cdot P(r|Ag, Env)$$
        - Ag_{\,opt} = \underset{Ag \in AG}{\mathrm{argmax}} \quad \underset{r \in R(Ag,Env)}{\sum} u(r) \cdot P(r \mid Ag, Env)
        - Neříká nám to nic o tom jak ale takového agenta sestrojit
        - Někdy je dokonce obtížné zkonstruovat samotnou účelovou funkci
- Predikátová specifikace úlohy
    - Speciální varianta účelové funkce
    - $u : R \to \{0, 1\}$
    - Běh je úspěšný, pokud $u(r) = 1$
    - Definujeme predikátovou specifikaci F
        - $F(r)$ je splněna právě když $u(r) = 1$
    - Prostředí úlohy je $(Env\ ,\ F)$
        - Env … prostředí
        - $F : R \to \{0, 1\}$
    - Prostředí úlohy
        - Definujeme $RF(Ag\ ,\ Env)$ … splňující běhy agenta Ag v prostředí Env
            - $RF(Ag\ ,\ Env) = \{\, r \mid r \in R(Ag\ ,\ Env)\ \wedge F(r) \,\}$
        - Kdy agent Ag vyřeší úlohu $(Env\ ,\ F)$?
            - Pesimista
                - $RF(Ag\ ,\ Env) = R(Ag\ ,\ Env)$
                - Každý běh splní F
            - Optimista
                - $\exists r \in R(Ag\ ,\ Env)$ takový, že $F(r)$ je splněno
                - Alespoň jeden běh splní F
            - Realista
                - Změříme pravděpodobnost splnění F
                - $P(F \mid Ag\ ,\ Env) = \sum_{r \in RF(Ag,Env)} P(r \mid Ag\ ,\ Env)$
- Pohled na typy úloh ze života
    - Achievement task

        - Cílem je dosáhnout libovolném stavu z G
        - G … množina cílů
            - $G \subset E$
        - $F(r)$ platí, pokud alespoň jeden ze stavů z G je v běhu r
        - **Agent je úspěšný, pokud všechny jeho běhy skončí stavem z G**
        - Př: libovolná úloha z umělé inteligence (hledáme řešení)
    - Maintenance task
        - Agent naopak musí zabránit některým stavům prostředí
        - B … špatné stavy
            - $B \subset E$
        - $F(r)$ je neplatí, pokud se libovolný ze stavů z B se objeví v běhu r
        - Př: hry, B jsou prohrávající stavy, prostředí je oponent
    - Kombinace předchozích dvou
        - Dosáhni stavu z G, ale vyhni se stavům z B

Deduktivně uvažující agent

- Klasický způsob jak můžeme dělat umělou inteligenci
    - Symbolická reprezentace prostředí a chování
    - Používáme matematickou logiku
    - Používáme logickou dedukci a theorem proving pro manipulaci symbolické reprezentace
- Moderní způsob jak dělat umělou inteligenci
    - Dnes obvykle nazýváno pojmem computational intelligence
    - Neuronové sítě, evoluční algoritmy, …
- Dále se budeme zabývat klasickým způsobem jak dělat umělou inteligenci
- Dedukce
    - Obecné → specifické
    - “Všichni lidé jsou smrtelní, Sokrates je člověk → Sokrates je smrtelný”
- Indukce
    - Specifické → obecné
    - Policejní vyšetřování, máme nějaký důkaz (předpoklad), potom prohlásíme pachatele za vraha
        - Nemůžeme si být 100% jisti
- **Dedukce a indukce jsou dnes běžně používány přesně obráceně**
    - Sherlock Holmes tvrdí, že používá dedukci, ale ve skutečnosti používá indukci
    - Matematická indukce je ve skutečnosti dedukce
- Agent jako theorem prover
    - L … množina formulí logiky prvního řádu
    - $D = 2^L$ … množina databází formulí L
    - Vnitřní stav DB agenta je $DB \in D$
    - Rozhodování (deliberation) je prováděno pomocí dedukce/odvozováním formulí pomocí pravidel P v dané logice

        - $DB \vdash_P f$ (formuli f odvodíme z DB pomocí dedukčních pravidel P)
    - $see : E \rightarrow Per$
    - $next : D \times Per \rightarrow D$
    - $action : D \rightarrow A$
    - see a $next$ známe, jak uděláme $action$?
        - Pro každou akci máme predikát $Do(a)$
        - Pro všechny akce zkoušíme jestli formule $Do(a)$ je validní formule, která může být odvozena z databáze faktů DB, pokud takovou akci najdeme, tak ji vrátíme
        - Pokud žádná formule nejde odvodit, tak se snažíme najít libovolnou akci a, takovou, že negace $Do(a)$ nejde odvodit, pokud takovou akci nalezneme, tak ji vrátíme

```
Function action(DB:D) returns action A
begin
   for each a ϵ A do
      if DB ⊢P Do(a) then return a
   for each a ϵ A do
      if DB ! ⊢P ¬Do(a) then return a
return null
end
```

- Výhody a nevýhody
    - Není příliš vhodné pro praktické aplikace
    - Matematicky elegantní
    - Složité implementovat

(!!) BDI

- Budeme se bavit o prakticky uvažujících agentech (**Practical reasoning agents**)
- Praktické uvažování
    - Inspirováno lidským rozhodováním
    - Dvě části
        - Deliberation
            - Čeho chceme dosáhnout
            - Př. chci dokončit školu
        - Means-End reasoning
            - Jak toho chceme dosáhnout
            - Př. vytvořím si studijní plán jak dodělám školu
- BDI
    - B - beliefs
        - Reprezentují znalosti agenta

        - Anglicky to je beliefs a ne knowledge, protože reprezentuje **subjektivní** znalosti agenta
        - Fakta a odvozovací pravidla jako v Prologu
    - D - desires
        - Cíle (může jich být více)
        - Reprezentují motivační stav agenta
    - I - intentions
        - "Stav světa", kterého chce agent dosáhnout
            - Jeden z desires, kterého se snažíme dosáhnout
        - Vedou agenta k akcím, které provede, aby dosáhl tohoto "stavu světa"
        - Omezuje proces rozhodování agenta
- Obvykle označujeme písmenky B, D, I aktuální množiny beliefs, desires a intentions jednoho agenta
- Abychom dosáhli našeho intention, vyprodukujeme **plán**
    - Obvykle se ho držíme dokud nedosáhneme intentionu nebo zjistíme, že ho není možné splnit
- Deliberation (rozhodování)
    - Funkce generování desires (možností)
        - $options: 2^B \times 2^I \rightarrow 2^D$
    - Funkce výběru z možností
        - $filter: 2^B \times 2^D \times 2^I \rightarrow 2^I$
    - Funkce aktualizace domněnek
        - $brf: 2^B \times Per \rightarrow 2^B$ (Per jako perception, co agent vnímá) (brf jako belief refresh function)

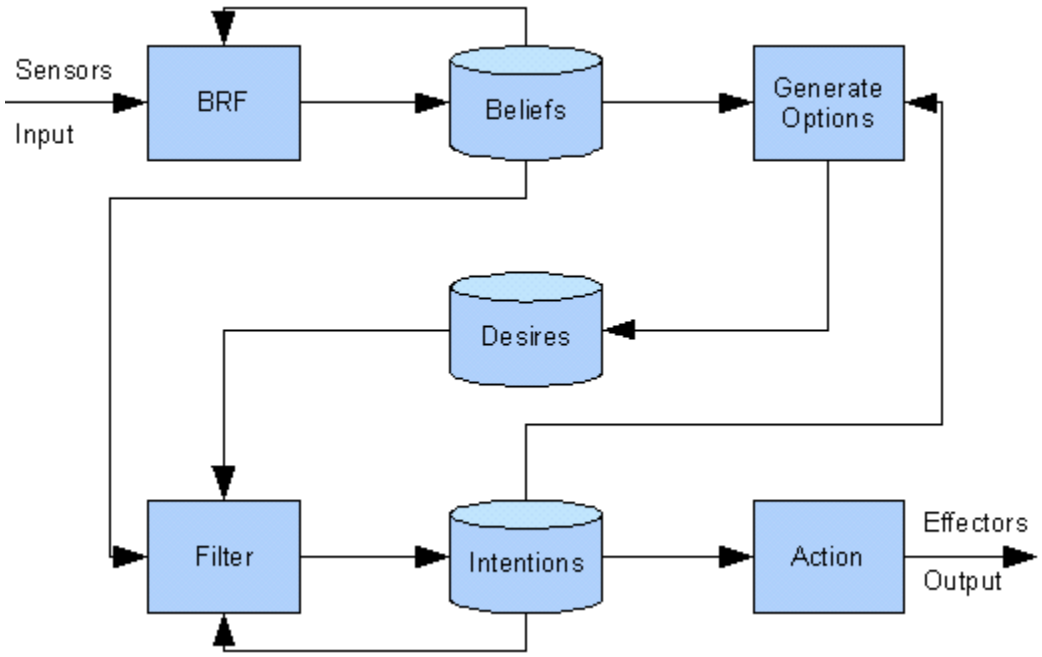


Schéma BDI agenta

- Plán
  - Jak dosáhnu cíle (intention) pomocí akcí, které mám k dispozici
  - Vstup
    - Cíl (nějaký intention)
    - Aktuální B
    - (Víme jaké akce máme k dispozici)
  - Výstup
    - Plán = sekvence akcí
    - Provedení akcí musí vést do cíle
- STRIPS
  - Článek z roku 1971 popisující algoritmus
  - První úspěšná formulace plánů a plánování
  - Považují B za množinu logických formulí
  - Akce má 2 části
    - Preconditions
      - Podmínka, abychom vůbec mohli použít akci
    - Effects
      - Jak popsat, co dělá akce
      - Add
        - Fakt bude přidán do B
      - $Delete$
        - Fakt bude smazán z B
- Definice plánů
  - Množina akcí $A_c = \{ a_1, a_2, \ldots, a_n \}$
  - Deskriptor akce a je $[P_a, D_a, A_a]$
    - P_a je množina **preconditions** akce a
    - D_a je množina formulí, které budou smazány po akci a
    - A_a je množina formulí, které budou přidány po akci a
    - D_a, A_a označujeme za množiny **efektů**
  - Plánovací problém potom definujeme jako $[B_0, O, G]$
    - "Chceme se dostat z B_0 do G přes akce popsané v deskriptorech O"
    - B_0 … počáteční beliefs agenta
    - $O = \{ [P_a, D_a, A_a] : a \in A_c \}$
    - G … množina formulí reprezentující cíl
- Plán p je posloupnost akcí $(a_1, a_2, \ldots, a_n)$ splňující kde $a_i \in A_c$
  - p pro $[B_0, O, G]$ určuje posloupnost databází $B_0, B_1, \ldots, B_n$ takovou, že $B_i = (B_{i-1} \setminus D_{a_i}) \cup A_{a_i}$ pro $i = 1, \ldots, n$

- p je přípustný pro $[B_0, O, G] \Leftrightarrow B_{i-1} \models P_{a_i}$ pro $i = 1, \ldots, n$
- p je korektní pro $[B_0, O, G] \Leftrightarrow p$ je přípustný pro $[B_0, O, G] \wedge B_n \models G$

- **Pro $[B_0, O, G]$ najdi korektní plán, nebo řekni, že neexistuje**
- Další definice k plánu p
  - $Plan$ … množina plánů (nad A_c)
  - $pre(p)$ … podmínka plánu p
  - $body(p)$ … tělo plánu p
  - $empty(p)$ … booleovská funkce určující, jestli je plán p prázdný
  - $execute(p)$ … procedura plnící plán p
  - $hd(p)$ … první akce v těle plánu p
  - $tail(p)$ … druhá až poslední akce v těle plánu p
  - $sound(p, I, B)$ … plán p je korektní plán pro intensions I a beliefs B
- Agentova plánovací funkce
  - $plan: 2^B \times 2^I \times 2^{A_c} \rightarrow Plan$

Implementation

- B:= B_0; I:= I_0
- while true do
  - v:= see(); B:= brf(B,v); D:= options(B,I); I:= filter(B,D,I);
  - p = plan(B,I,Ac);
  - while not (empty(p) or succeed(I,B or impossible (I,B)) do
    - a:= hd(p); execute(a); p:= tail(p);
    - v:= see(); B = brf(B,v);
    - if reconsider(I,B) then D:= options(B,I); I:= filter(B,D,I) endif
    - if not sound(p,I,B) then p = plan(B,I,Ac);
  - endwhile
- endwhile

```
1.  B ← B0;          /* B0 are initial beliefs */
2.  I ← I0;          /* I0 are initial intentions */
3.  while true do
4.    get next percept ρ via sensors;
5.    B ← brf(B, ρ);
6.    D ← options(B, I);
7.    I ← filter(B, D, I);
8.    π ← plan(B, I, Ac); /* Ac is the set of actions */
9.    while not (empty(π) or succeeded(I, B) or impossible(I, B)) do
10.     α ← first element of π;
11.     execute(α);
12.     π ← tail of π;
13.     observe environment to get next percept ρ;
14.     B ← brf(B, ρ);
15.     if reconsider(I, B) then
16.       D ← options(B, I);
17.       I ← filter(B, D, I);
18.     end-if
19.     if not sound(π, I, B) then
20.       π ← plan(B, I, Ac)
21.     end-if
22.   end-while
23. end-while
```

Figure 2.1 Overall control loop for a BDI practical reasoning agent.

- Commitment of agents (jak jsou agenti urputní a trvající na cílech)
  - Kdy se vzdát cíle (a změnit ho)
  - Blind commitment
    - Agent si drží svůj intention až do té doby než ho dosáhne
  - Open-minded
    - Agent si drží svůj intention až do té doby než
      - Cíl je splněn
      - Zjistí, že ho není možné splnit
      - Ztratí důvod (motivaci) k jeho dosažení

    - Single-minded
        - Agent si drží svůj intention až do té doby co si myslí, že je možné ho dosáhnout
        - Př. robot ví, že nemá dostatek baterie
    - Kdy by se agent měl zastavit a znovu přehodnotit jeho intention?
        - Klasické dilema
            - Uvažování stojí čas a prostředí se během něho může změnit
            - Agent, který nepřehodnocuje intentions dost často, může plnit intention, který už nemá smysl plnit
            - Agent, který přehodnocuje intentions pořád, nemá dost času na jejich plnění a tak třeba nic nesplní
        - Extrémní řešení dilematu
            - Bold agent
                - Přehodnocuje intentions pouze po skončení provádění aktuálního plánu
            - Cautions agent
                - Přehodnocuje po každé akci plánu
        - Level of boldness
            - Po kolika provedených akcích mám přehodnotit intentions

BDI v praxi

- PRS (procedural reasoning system)
    - První implementace BDI architektury
    - Vzniklo na Stanfordově univerzitě v 80. letech
    - Dneska hromada derivátů
        - AgentSpeak/Jason
        - JADEX
        - …

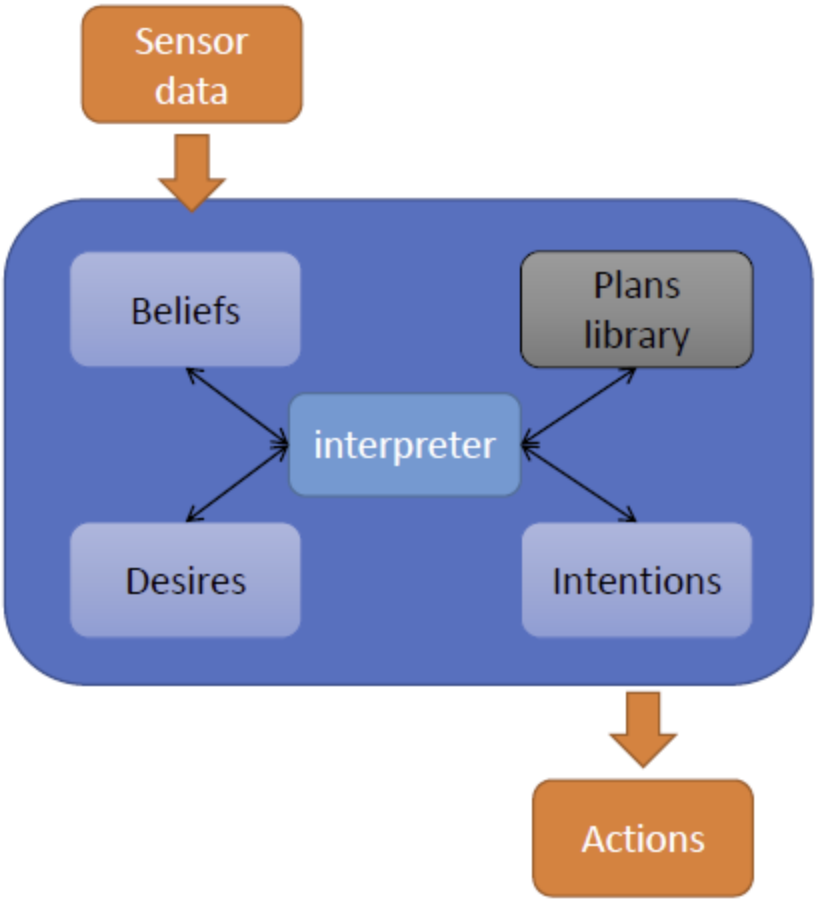


Schéma PRS agenta

- Plány v PRS
  - Agent má k dispozici **knihovnu předpřipravených plánů**
  - Plán
    - Tělo
      - Akce k vykonání
      - Nemusí se jednat o lineární sekvenci akcí
        - Achieve goal f
        - Achieve goal f or g
        - Keep achieving f until g
    - Cíl
      - Konečný stav, kterého chceme dosáhnout
    - Context
      - Podmínky, které musí být splněny, aby plán mohl být spuštěn
- Plánování v PRS
  - Inicializace
    - Máme nějaké initial beliefs B_0 a initial cíl G_0
  - Máme stack of intentions
    - Obsahuje více intentions (aktuálně rozpracované)

        - Funkce
            - Interpretr vezme intention na vrcholu stacku a hledá plány, které mají tento intention jako cíl, z těchto plánů ty, které mají splněn kontext se stanou možnosti (desires)
            - ~~Interpretr vezme intention na vrcholu a hledá v plánech, zda se jejich cíl shoduje~~
            - ~~Z nich jen některé mají splněn kontext vzhledem k aktuálním beliefs → vybereme právě ten~~
                - ~~Pouze tyto reprezentují aktuální options/desires~~
            - ~~Updatujeme desires a intentions~~
- Deliberation in PRS (rozhodování v PRS)
    - Deliberation – výběr intention z desires
    - Původní PRS měl meta-plány
        - Plány o plánech
        - Popisují jak modifikovat intentions
        - Komplikované, opuštěné
    - Utility plánování (klasický přístup z umělé inteligence)
        - Každý plán je ohodnocen nějakou číselnou hodnotou očekávaného zisku
        - Vybereme ten s největší
        - Vybraný plán se spustí, což může způsobit přidání dalších intentions
        - Pokud plán selže, agent vybere jiný intention z možných a pokračuje

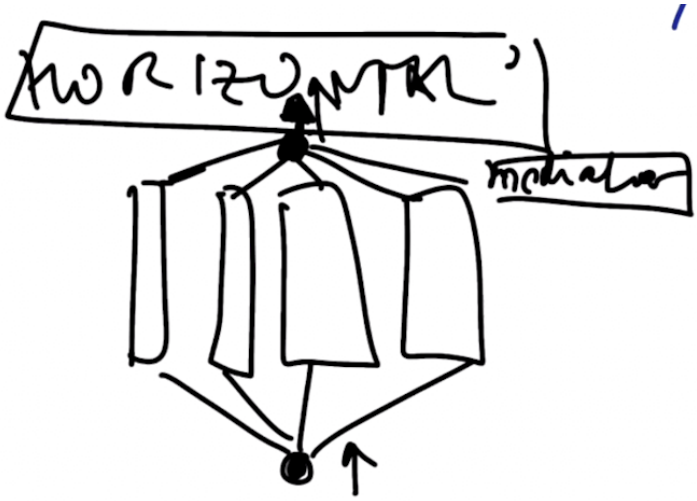
(!) Reaktivní architektury

- Přístupy podle reaktivity
    - Plánování / theorem proving (0% reaktivita)
    - BDI / PRS
    - Hybridní architektury
    - **Reaktivní** (100% reaktivita)
        - Sem patří Brooksova subsumpční architektura
- Reaktivní přístup
    - (Už jsme viděli reaktivní příklad s termostatem)
    - Populární v 80-90. letech
    - Alternativní paradigma pro umělou inteligenci
    - Přírodou inspirované paradigma
    - Co je důležité pro reaktivní přístup?
        - **Interakce**
            - Inteligentní chování se odvíjí od prostředí, ve kterém se agent nachází
        - **Embodiment**
            - Koncept z psychologie
            - Inteligentní chování není jen logická operace uvnitř mozku, ale je produktem agenta a jeho “těla”
        - **Emergence**
            - Inteligentní chování vzniká interakcí agentů v prostředí

            - Komplexní chování skupiny agentů dostáváme z jednoduchého chování jednotlivých agentů
- Reaktivní agenti
    - Na co se dává důraz při jejich designu
    - **Behaviorální**
        - Důraz na (vývoj a kombinace jednotlivých) chování
    - **Situované**
        - Agent v prostředí, je embodied (ztělesněn)
        - Robot, ne jenom algoritmus
    - **Reaktivní**
        - Agent reaguje hlavně na prostředí
        - Agent neprovádí uvažování
    - **Subsymbolická reprezentace**
        - (Doteď jsme dělali symbolickou reprezentaci)
        - Snaží se udělat reprezentaci jednodušší
        - Agent je seznam jednoduchých IF THEN pravidel
        - Spojeno s neuronovými sítěmi
        - *Někdy se používají i konečné automaty*
- **Brooksova subsumpční architektura agenta**
    - Asi nejúspěšnější z reaktivních přístupů
    - Článek napsal Rodney Brooks v roce 1991
    - Inteligentní chování lze vytvořit bez symbolické reprezentace
    - **Inteligence je emergentní vlastností určitých komplexních systémů**
        - Komplexní = mnoho jednoduchých agentů
        - *Dalo by se nazvat kontroverzním názorem – spousta lidí s tím nesouhlasí*
    - Prakticky se jedná o množinu jednoduchých chování
    - Architektura
        - Agentova inteligence je realizována jednoduchým **na cíl orientovaným chováním**
        - Každé chování
            - Má jednoduchou strukturu IF THEN
            - Je mechanismus selekce akce
            - Dostane vjemy a transformuje je na akce
            - Má na starosti nějaký úkol
            - Soupeří s ostatními o kontrolu nad agentem
            - Funguje paralelně
        - Všechna chování jsou v **subsumpční hierarchii** jsou definována v určitém pořadí
            - Pořadí určuje priority
            - Tomu vlastně říkáme subsumpční hierarchie
            - Subsumpční hierarchie = nějaký mechanismus pro zajištění priorit
        - Subsumpční mechanismus reaguje na vstupy (z prostředí)
            - Vybere se pravidlo, které odpovídá situaci

                - Pro každé chování, které je možné aplikovat, se nejdřív zkontroluje, jestli existuje chování s vyšší prioritou
                    - Pokud ne tak se vybere toto
                - Pokud se nic nevybere, tak se nic neprovede
        - Jednoduché, je možné akcelerovat pomocí HW
- **Roboti na Marsu** (příklad Brooksovy subsumpční architektury)
    - (Steelsův průzkumník Marsu)
    - *Cílem: Prozkoumat vzdálenou planetu a odebrat vzorky vzácné horniny. Poloha vzorků není předem známa, ale ví se, že bývají seskupeny*
    - Máme hejno robotických průzkumníků, kteří jsou agenti podle Brooksovy subsumpční architektury
    - Máme nějakou stanici, která vysílá signál (víme kde je)
    - Každý robot má k dispozici radioaktivní drobečky pro nepřímou komunikaci mezi sebou
        - Dokáže si zapamatovat takto cestu (jako kdyby házel drobečky chleba)
    - Máme několik pravidel a jejich **pořadí je důležité**
        - R_1: pokud detekuji překážku, pak změním směr
        - R_2: pokud nesu vzorek a jsem u základny, pak ho položím
        - R_T: **pokud nesu vzorek a nejsem u základny, pak jdu směrem stoupajícího signálu k základně**
        - R_4: pokud vidím vzorek, tak ho zvedni
        - R_5: if true jdi náhodně (výchozí pravidlo na konci – agent nemá nic lepšího na práci)
    - Nepřímá komunikace mezi agenty (pravidlo R_T v detailu)
        - R_3: pokud nesu vzorek a nejsem u základny, polož 2 drobečky a jdi stoupajícím směrem signálu k základně
        - $R_{4.5}$: pokud cítím drobeček, potom zvedni 1 drobeček a jdi stoupajícím směrem po drobečcích
    - **Priorita**: $R_1 > R_2 > R_3 > R_4 > R_{4.5} > R_5$
        - R_3 nahrazuje R_T
    - Simulace: https://www.youtube.com/watch?v=93LwvuxDbfU
    - Dobrá prezentace: https://cgi.csc.liv.ac.uk/~trp/COMP310_files/COMP310-Chapter5.pdf
- Hybridní architektury
    - Snaží se zachovat vlastnosti Brooksovy subsumpční architektury
    - Kompletně reaktivní agent ani kompletně plánovací agent není nejlepší
    - Architektura se skládá z **layerů**
        - Reaktivní layer
        - Plánovací layer
    - Layery kooperují
    - Layery budou mít nějakou prioritu mezi sebou

- Horizontální architektura
  - Layery pracují paralelně
  - Layery se mohou navzájem ovlivňovat
  - Potřebujeme nějaký mediator
    - Řeší konflikty mezi layery
    - Potencionální bottleneck



- Vertikální architektura
  - Layery jsou položeny horizontálně na sobě
  - Dvě verze
    - One-pass
      - Senzory posílají informací nejnižšímu layeru
      - Layer zpracuje informace a pošle nějaké další informace dalšímu layeru
      - Poslední layer vyprodukuje nějakou akci

    - Two-pass
      - Informace se od nejnižšího layeru dostane k nejvyššímu stejně jako v one-pass

    - Potom ale nejvyšší layer pošle informace zpátky dolů
    - Postupně se informace dostane zpátky až do nejnižšího layeru, který vyprodukuje akci

Ontologie

- Přesouváme se od návrhu agentů k jejich komunikaci
- Jak multiagentní systém komunikuje?
    - **Konkrétněji, jak si navzájem agenti mezi sebou porozumí**
- Potřebujeme popsat sémantiku (význam)
- Ontologie je společným slovníkem, který navíc popisuje vztahy mezi jednotlivými objekty (a třídami objektů) v prostředí. Pokud se dva agenti dohodnou, že budou používat stejnou ontologii, oba vědí, co ostatní agent myslí libovolným výrazem/zprávou
- Formálněji se ontologie skládá z jedinců (zhruba ekvivalentních konstantám v logice prvního řádu), tříd (odpovídajícím unárním predikátům) a binárních vlastností mezi třídami nebo jedinci (binární predikáty)
- Nejčastější binární relací mezi třidami je relace, které vyjadřuje, že jedna třída je podtřídou druhé. Můžeme například mít třídu zvířat a třídu psů a můžeme specifikovat, že třída psů je podtřída všech zvířat. Navíc můžeme mít objekt (jedince) Rex a můžeme říct, že je to jeden objekt z třídy pes. Nyní, kdykoliv se agenti dozví nějakou informaci o všech zvířatech (např. že dýchají), mohou odvodit, že se tato informace týká i všech psů a tedy i Rexe
- Relace podtřídy je ta nejjednodušší a snadno může být vyjádřena i v libovolném programovacím jazyce. Vztahy v ontologiích ale mohou být mnohem obecnější. Můžeme si například vytvořit ontologii o univerzitě. Na univerzitě máme dva typy lidí – studenty a učitele. Navíc máme několik přednášek. Potom si můžeme definovat vztah, který vyjadřuje, že student navštěvuje přednášku a jiný, který definuje, že učitel učí přednášku.

- Každá třída v ontologii může mít své vlastní vlastnosti (např. lidé mají jména) a každá vlastnost má definovaný typ, který je buď jeden ze základních typů (string, integer, …) nebo nějaký vlastní typ
- Kromě přímých definicí tříd můžeme také definovat třídy pomocí výrazů. Například můžeme definovat třídu pro velkou přednášku jako přednášku, kterou navštěvuje alespoň 50 studentů
- Ontologické jazyky
    - XML (eXtensible Markup Language)
        - Není k tomu určen, často se ale používá
        - Pochází z webu
        - Hlavní výhoda – definice nových tagů → tagy pak přirozeně reprezentují slovník
    - RDF (Resource Definition Framework)
        - Standardní reprezentace znalostí pro web
        - Reprezentace relací jako trojic subjekt-predikát-object (SVO)
    - OWL (Web Ontology Language)
        - Dnes dvě nekompatibilní verze: verze 1 vs verze 2
        - Také pochází ze sémantického webu
        - Sbírka několik formalismů na popis ontologií
- **Deskripční logika**
    - Dotazy a podmínky (a celá definice ontologie) jsou v ontologiích často specifikované v tzv. deskripční logice
    - Je extenzí výrokové logiky, ale je slabší než logika predikátová
    - Speciálně je rozhodnutelná, tedy o každé formuli v ní zapsané umíme rozhodnout (často velmi efektivně) jestli je pravdivá, nebo ne
    - Terminologie deskripční logiky se trochu liší od terminologie ontologií. V deskripční logice se místo pojmu “třída” používá “koncept” a místo “vlastností” (binárních relací) se používá pojem “role”
    - Axiomy v deskripční logice se rozdělují na tzv. ABox a TBox. V ABoxu se typicky specifikují informace o konceptech a rolích mezi nimi. TBox potom obsahuje informace o jedincích (do jaké třídy patří, jaké mají vztahy s jinými jedinci apod.). Dalo by se také říct, že ABox popisuje obecné vlastnosti, které platí v nějakém světě, TBox potom udává konkrétní jedince (objekty), kteří se v tomto světě vyskytují a jaké jsou vztahy mezi nimi

Komunikace agentů

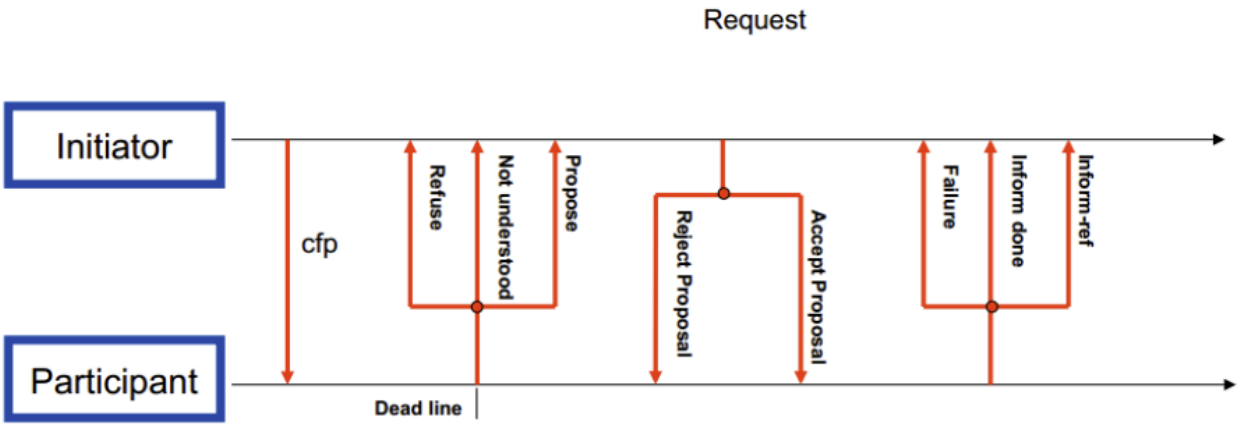
- Jak agenti mezi sebou **komunikují**
- Narozdíl od ontologie, teď nebudeme řešit porozumění, ale **komunikaci**
- Komunikace určuje, kdy může agent poslat zprávu jinému agentovi a jaká je očekávaná odpověď
- Existuje řada standardů, nejznámější je však standard FIPA vydávaný Foundation for Intelligent Physical Agents
- Speech acts

    - Některé části jazyka mají charakter akcí, protože mění stav světa stejně jako naše fyzické akce
        - Prohlašuji vás mužem a ženou
        - Deklaruji válku Rusku (pokud daná osoba má na toto potřebnou moc)
    - John Searle je zformalizoval a rozdělil
    - Locutionary acts
        - Co bylo řečeno
        - Př. Make me a tea
    - Illocutionary act
        - Co bylo myšleno
        - Př. She asked me to make a tea
    - Perlocutionary act
        - Co se opravdu stalo
        - Efekt speech actu
        - Př. She made me to make her a tea
- Kategorie speech acts
    - Staré
        - Request, promise, advice, promise
    - Nové
        - Assertives
            - Informuje
        - Directives
            - Požadavek
        - Comissive
            - Sliby
        - Expressive
            - Emoce (děkuji apod.)
        - Declarations
            - Mění stav věcí (vyhlášení války)
- Plánovací teorie speech acts
    - Modelovat speech acts jako plánovací systém
    - Definovali operátory, akce, preconditions, postconditions
    - Používají STRIPS systém
- Jazyky pro komunikaci agentů
    - KQML (Knowledge Query Manipulation Language)
        - Dnes jeho proměněnou verzi známe ho jako ACL, který standardizovala FIPA
        - Nepoužívá se
        - “Obálka dopisu”
        - Obsahuje illocutionary část zprávy
    - KIF (Knowledge Interchange Format)
        - Formalismus nad logikou 1. řádu
        - Stále se používá
    - **FIPA ACL** (Foundation of Physical Agents, Agents Communication Language)

        - Založen na logice 1. řádu
        - Zjednodušené a standardizované KQML
        - Praktická implementace v prostředí JADE

(!) Contract net, BBS

- **Koherence**
    - Jak výkonný je systém jako celek
- **Koordinace**
    - Jak dobře agenti minimalizují režijní náklady (overhead) jako synchronizace ap.
- ContractNET / CNET
    - Contract Net protokol
    - Protokol pro kooperaci při řešení problémů agentů
    - Smith and Davis, 1977
    - Snaží se napodobovat reálný život (simuluje kontrakty)
    - Metafora pro sdílení úkolů přes kontrakty
    - **Recognition**
        - Agent nemůže vyřešit problém sám
        - Zjistí, že potřebuje pomoc dalších agentů
        - (Pomoc získá simulováním kontraktů)
    - **Announcement**
        - Agent broadcastuje oznámení úkolu včetně
            - Podmínek (deadline, price)
            - Popis (specifikace)
    - **Bidding**
        - Ostatní agenti začnou bidding proces potom co “uslyší” announcement
        - Rozhodnou se jestli se budou účastnit a dají nabídku
            - Nabídka obsahuje kolik času to bude trvat, co dokáže splnit, …
    - **Awarding, Expediting**
        - Agent, který oznámil nabídku si vybere “vítěze” a uzavře s ním kontrakt
- FIPA ACL CNET protocol
    - Initiator
        - 1. Odesílá CFP (call-for-proposal = announcement) ostatním participants
        - 3. Odešle reject / accept na proposal
    - Participant(s)
        - 2. Odešle refuse / not understood / propose na CFP
        - 4. Úkol budťo dokončí a pošle done (+ referenci) nebo selže a pošle failure



- BBS (Blackboard system)
  - Lidé se snažili přijít s alternativami k CNETu
  - Funkce
    - Několik agentů sedí u tabule
      - Agentům, kteří sledují tabuli, se říká experti
    - Na tabuli mohou zapsat nebo z ní číst
    - Oznámení úkolů i výsledků se sdílí přes tabuli
    - Pokud expert vidí, že může úkol vyřešit, tak si ho vezme
    - Úkol ze označí jako “právě řešen”
      - Nějak dobu ho řešící agent řeší
    - Pokud řešící agent přijde na to, že vyžaduje vyřešení podúkolu, tak ho přidá na blackboard
  - BB (blackboard) = sdílená paměť
  - **Použití hlavně při distribuovaném řešení problému**
  - **Vyžaduje společný jazyk pro interakce**
  - Arbitr
    - Organizátor, který je odpovědný za běh Blackboard systému
    - Vybírá úkoly a vhodné agenty, kteří se mohou ucházet o úkol
- Př. BB War Game
  - Tabule
    - Hashovací tabulka – mapuje požadované schopnosti na úkoly
    - Zveřejňují se na ni otevřené mise – úkoly
  - Experti
    - Řešiči jednotlivých úkolů, hieararchická struktura
    - Mají seznam schopností a efektivity
      - Velitel rozumí úkolu $ATTACK\text{-}CITY$
      - Voják rozumí úkolu $ATTACK\text{-}LOCATION$
  - Velitel hledá cíle typu $ATTACK\text{-}CITY$ transformuje je na $ATTACK\text{-}LOCATION$
  - Vojáci hledají cíle typu $ATTACK\text{-}LOCATION$ a začnou provádět misi
    - Mohli by ještě třeba vytvářet další úkolu typu (letecká podpora, …)
- BBS výhody a nevýhody
  - Jednoduchý mechanismus pro koordinaci a kooperaci agentů

    - Experti nemusejí vědět o ostatních expertech, se kterými kooperují
    - Někdy se používá i pro samotnou komunikaci mezi agenty
    - Agenti musejí typicky sdílet stejnou architekturu a u tabule je nával
        - Je nutné řešit nějak pomocí distribuovaných hashovacích tabulek

(!) Jak by ContractNet vypadalo s jinýma aukcema

????

Interakce agentů

- Začneme se základy teorie her, která se nám pro multiagentní systémy hodí
- Agent i má strategii S_i
- S_i je dominantní strategií agenta i pokud dává stejný, nebo lepší výsledek než jakákoliv jiná strategie agenta i proti všem ostatním strategiím agenta j
- S_i a S_j jsou v Nashově ekvilibriu, pokud
    - Hraje-li agent i strategii S_i , agent j dosáhne nejlepšího výsledku se strategií S_j
    - Hraje-li agent j strategii S_j , agent i dosáhne nejlepšího výsledku se strategií S_i
    - *Najít ekvilibria pro n agentů a m strategií trvá m^n*
- Nashova věta
    - Každá hra s konečným počtem strategií má Nashovo ekvilibrium ve smíšených strategiích
- Řešení je Pareto-optimální
    - Když neexistuje jiná strategie, která by zlepšila agentův výnos bez zhoršení výnosu jiného agenta
- Social welfare (společenské dobro)
    - Suma výnosů všech agentů
    - *Výhodné jen při kooperaci, typicky, když jsou agenti součástí jednoho týmu a mají jednoho vlastníka*
- Rozhodování jako hra
    - Oba agenti ovlivňují výsledek
    - Změna stavu prostředí
        - $e : A_i \times A_j \rightarrow O$
            - $e(a_1, a_2) = o$
                - a_1 … akce agenta i
                - a_2 … akce agenta j
                - o … outcome (nějaké číslo)

(!) Iterated prisoner's dilemma

- Obyčejné vězňovo dilema

    - Vězňovo dilema je hra pro dva hráče. Je založena na příběhu dvou vězňů: dva kamarádi spolu páchali zločiny tak dlouho, až je jednou chytili. Neměli ale dost důkazů a ačkoliv je podezřívali z mnoha věcí, dokázat jim mohli jen tu poslední. Posadili je tedy každého do jedné místnosti a nabídli jim nižší trest za to, že svého kamaráda zradí
    - Každý vězeň tedy má dvě možnosti, buď spolupracovat s tím druhým (cooperate), nebo ho zradit (defect). Pokud oba **spolupracují**, nic moc jim nedokážou a dají jim **menší** trest. Pokud jeden spolupracuje a druhý ho zradí, tak zrádce odejde **bez** trestu a spolupracující dostane **dlouhý** trest. Konečně, pokud se zradí oba navzájem, dostanou oba trest, který je sice větší, než kdyby spolupracovali, ale **menší** než ten, co by dostal spolupracující, kdyby spolupracoval jen jeden z nich
    - Nahradíme-li délku trestu body (tj. kratší trest = více bodů) a vězně hráči, můžeme se na hru podívat z pohledu teorie her. Dostáváme hru s následující matici odměn:

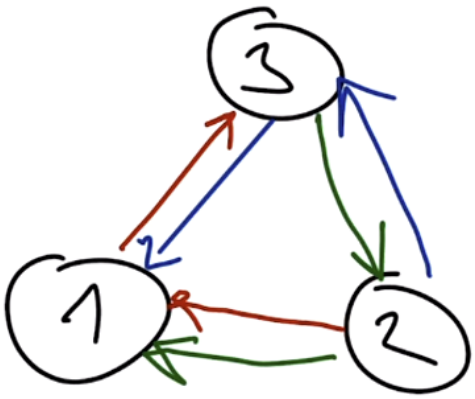
	C	**D**
C	3	0
D	5	1

    - Racionální hráč bude nejspíš uvažovat nějak takto:
        - Mohu buď spolupracovat, nebo zradit
        - Když budu předpokládat, že mě protihráč zradí, vyplatí se mi ho zradit ($1 > 0$)
        - Když budu předpokládat, že protihráč bude spolupracovat, vyplatí se mi ho zase zradit ($5 > 3$)
        - Zradím tedy svého protihráče
    - Stejným způsobem bude uvažovat i protihráč, a oba navzájem se tedy zradí a dostanou jeden bod. Pokud se vězňovo dilema hraje jen jednou, nemohou racionální hráči dosáhnout jiného výsledku (pokud se předtím nedomluví a nevěří si)
- Iterované vězňovo dilema
    - Mnohem zajímavější situace ale nastane, když se vězňovo dilema hraje **opakovaně** mezi dvěma hráči, kteří (oba) mají za úkol maximalizovat svůj bodový zisk. V takové situaci totiž existuje možnost oplatit zradu v předchozím kole a existuje tedy i možnost, aby oba hráči spolupracovali
    - Pro iterované vězňovo dilema a pro vznik spolupracujících strategií je ale důležité, aby hráči **nevěděli**, kolikrát se bude vězňovo dilema opakovat (jinak se vyplatí v posledním kole zradit, díky tomu se ale vyplatí zradit i v předposledním kole, atd.)
    - Strategie pro iterované vězňovo dilema

        - **Always Defect** - hraje vždycky zradu nezávisle na druhém hráči
        - **Always Cooperate** - vždy spolupracuje nezávisle na tom, jak hraje druhý hráč
        - **Random** - hraje náhodně
        - **Tit-for-tat** - v prvním kole spolupracuje, v dalších hraje to, co hrál protihráč v předcházejícím kole, tj. zradí, pokud zradil (oplácí zradu), a spolupracuje, pokud spolupracoval

Hlasování

- Motivováno volbami (election)
- Hlasování je efektivní způsob jak provádět **rozhodnutí**
    - Máme n agentů, kteří hlasují a mají na výběr z m možností
    - Agenti mají nějaké preference
    - Cílem je agregovat preference agentů do jednoho výsledku → **social choice**
    - Příklad
        - Máme agenty A, B a C
        - Preference
            - $A: o_2 > o_1 > o_3$
            - $B: o_3 > o_2 > o_1$
            - $C: o_2 > o_3 > o_1$
        - Celková preference = social welfare
            - $o_2 > o_3 > o_1$
        - Vítěz = social choice
            - o_2



        - Preference jsou lineární, ale graf už lineární být nemusí
        - Obecně je tento problém těžký
- Ballot
    - Preference agenta
    - Obvykle nedostáváme kompletní preference agenta, ale pouze jeho preferovaného kandidáta
        - Volím X jako prezidenta

- Preference ballot
    - Kompletní seznam preferencí agenta
- Voting schemes
    - Majority scheme
        - Vítěz má více než 50% hlasů
        - Ballot není preferenční
        - *Málokdy uspěje, v praxi nefunguje pro více než 2 kandidáty*
    - Plurality scheme
        - Vítěz je ten co má nejvíce hlasů
        - Ballot není preferenční
        - Př. do voleb na prezidenta nebo senátu v ČR se kombinuje s majority scheme
            - Takto se dnes často “opravuje” plurality
        - Problém tohoto schématu je vidět na tomto příkladě
            - o_1 dostane 10 hlasů
            - o_2 dostane 8 hlasů
            - o_3 dostane 7 hlasů
            - Výherce je o_1 s 10 hlasy, ale 15 agentů pro něj nevolilo
    - Plurality s eliminací / Instant runoff
        - Kombinujeme první kolo a druhé kolo dohromady
        - Voliči dávají preferenční ballot
        - Děláme plurality
        - Najdeme nejhoršího a toho vyřadíme
            - Ten, co je nejméně krát v preferencích voličů jako první
        - Jeho hlasy se přesunou na voličovu další volbu → děláme plurality znovu, než má jeden kandidát majoritu hlasů
        - Výhody
            - Eliminujeme taktické hlasování
        - Používá se v
            - Australian House of Representatives
            - Olympic committee
                - Pro výběr země, kde bude další olympiáda
                - *Toto říkají, ale pravděpodobně používají k hlasování úplatky*
        - Jinak se v praxi moc nepoužívá (zejména kvůli tomu, že voliči musí dát kompletní preference, což se jim moc nelíbí)
- Condorcetův paradox
    - Při více kandidátech často dochází k paradoxu, že je zvolen kandidát, který byl pro většinu špatný
    - Př.
        - o_1 dostane 40% hlasů
        - o_2 dostane 30% hlasů

        - o_3 dostane 30% hlasů
        - 60% lidí nevolilo o_1
    - Vede k taktickému hlasování → nehlasuju podle svých preferencí, ale cíleně proti ~~Milošovi~~ někomu tím, že volím kandidáta, který pro mě není nejvhodnější, ale má šanci ho porazit
    - Př.
        - Kámen, nůžky, papír scenario
        - $Volič_1 : A > B > C$
        - $Volič_2 : B > C > A$
        - $Volič_3 : C > A > B$
        - Social welfare je cyklická
        - Neexistuje Condorcetův vítěz
            - Vítěz při porovnání se všemi kandidáty
- Condorcetův vítěz by měl být vítěz hlasování
    - Je to velmi silná vlastnost
- Další hlasovací systémy
    - Sequential majority elections
        - Condorcetova metoda
            - Simulujeme volby všech párů kandidátů
            - Trvá dlouho, stojí moc peněz → nikdo nepoužívá
        - Tree metoda
            - Pavouk jako v tenise
            - Někdy se používá → většinou v méně seriózních záležitostech
- Borda count
    - Další rodina hlasovacích mechanismů
    - Používalo se ve Francii do té doby než Napoleon změnil státní řízení
    - Každý volič dává kompletní preference na všechny kandidáty
    - Pro hlasování N kandidátů má moje číslo jedna $N - 1$ bodů a moje poslední preference má 0 bodů
    - Social welfare spočítáme tak, že každému kandidátovi spočítáme jeho body
    - Kandidát s maximálním součtem je social choice
    - Používají ve
        - Slovinsku na volení parlementu
        - Islandu na volení parlamentu
        - Akademických institucích
    - Deriváty
        - Borda combinations
            - Black method
                - Condorcetova metoda, pokud neexistuje Concorcetův vítěz, tak zvol Borda vítěze
            - Baldwin method

        - Spočítáme Borda skóre a eliminujeme kandidáty s nejméně body
        - Znovu spočítáme Borda skóre pro zbylé kandidáty a iterujeme
    - Nanson method
        - Spočítáme Borda skóre a eliminujeme kandidáty s podprůměrným počtem bodů
        - Znovu spočítáme Borda skóre pro zbylé kandidáty a iterujeme
    - *Všechny tyto metody dávají Condorcetova vítěze, pokud existuje*
- Vlastnosti hlasovacích procedur
    - Paretova vlastnost
        - Pokud všichni agenti mají v preferencích, že $X > Y$, tak $X >_{SW} Y$
            - SW značí social welfare
            - Platí v majority a Borda counting
            - Neplatí pro sequential majority
    - Condorcet winner
        - Kandidát, který porazí všechny ostatní při pairwise porovnání
        - Je to silná vlastnost
        - Často není k dispozici
        - Platí pouze v Condorcetově metodě a v Borda combinations
    - IIA (independence of irrelevant alternatives)
        - $X >_{SW} Y$ by měl záviset pouze na pořadí X a Y v jednotlivých hlasováních
        - Př. porušení
            - Máme dalšího kandidáta Z, pokud ho smažu, nebo přídám jiného dalšího kandidáta W, celkové pořadí X a Y se nesmí změnit
    - Unrestricted domain, or universality
        - Social welfare by měl záviset na všech hlasech
        - Neplatilo v minulém století
            - Ženy nemohli volit
    - Dictatorship
        - Systém, kde výsledek určuje jeden volič (diktátor)
- **Arrow's theorem**
    - (Pro 3 a více kandidátů) Žádný volební systém založený na pořadí **nedokáže** vytvořit z individuálních preferencí uspořádání za zachování všech následujících podmínek
        - Universality
        - Non-dictatorship
        - Paretovy vlastnosti
        - IIA
- Arrow's theorem (zjednodušená verze – důsledek)
    - Ve volbách s více než 2 kandidáty existuje pouze jedna hlasovací procedura splňující Paretovu vlastnost a IIA → **dictatorship**

    - *Fundamentální limit demokratického rozhodování*
    - **Pozor:** neznamená to, že jediný fungující systém ve společnosti je diktátorství

Aukce

- Aukce se již dlouho používají k prodeji a nákupu různého zboží po celém světě
- Typy klasických aukcí
    - Existuje řada různých typů aukcí, nicméně čtyři z nich jsou typické
    - **Aukce s rostoucí cenou (anglická aukce)**
        - Aukce s rostoucí cenou jsou interaktivní, prodávající postupně zvyšuje cenu, uchazeči z aukcí vypadávají vždy, když je cena vyšší než cena, kterou jsou ochotni nabídnout, poslední zbývající uchazeč je ten, kdo aukci vyhraje a zaplatí aktuální cenu
    - **Aukce s klesající cenou (holandská aukce)**
        - Při aukcích s klesající cenou prodávající na začátku oznámí vysokou cenu a postupně ji snižuje, jakmile je zájemce ochoten zaplatit cenu, oznámí ji, vyhraje aukci a zaplatí aktuální cenu
    - **Obálková aukce s první cenou**
        - V obálkových aukcích všichni uchazeči předloží nabídku současně v zapečetěné obálce, uchazeč s nejvyšší nabídkou vyhraje a zaplatí hodnotu nabídky
    - **Obálková aukce se druhou cenou (Vickreyova aukce)**
        - Obálková aukce s druhou cenou je podobná aukci s první cenou, nicméně vítěz neplatí hodnotu své nabídky, ale hodnotu druhé nejvyšší nabídky
- Aukce s druhou cenou se může zdát divná. Proč by chtěl prodejce použít tento typ aukce, když dostane méně peněz než v aukci s první cenou? Ve skutečnosti se ukazuje, že lidé velmi často v dražbě za druhou cenu přihazují o něco více, což rozdíl mezi konečnými cenami snižuje
- **Konečná cena v obálkové aukci za druhou cena odpovídá anglické aukci**
    - Odejít dříve z aukce než je moje vnitřní hodnota nemá smysl
- **Konečná cena v obálkové aukci za první cenu odpovídá holanské aukci**
    - Uchazeč nemusí nabízet svou vnitřní hodnotu

Učení v MAS

- Inteligentní agenti by měli být adaptivní, měli by se učit
- Nejčastější přístup k učení v MAS je **reinforcement learning**
    - Změna chování metodou pokus/omyl podle odměn získaných z prostředí
- Učení jednoho agenta je jednodušší
    - Můžeme použít tradiční reinforcement learning algoritmy jako Q-Learning
- Učení multi-agentů je obtížnější
    - Multi-agent reinforcement learning (MARL)
    - Agenti se učí simultánně
    - Markov Games

        - Je Nash, Pareto optimální
    - Deep learning (MADRL)
        - Aktuálně state-of-the-art
- Markov decision process (MDP)
    - Pomocí tohoto budeme definovat náš problém učení
    - Dva přístupy
    - **Value-based**
        - Postavené na teorii her
        - Q-Learning
    - **Policy-based**
        - Podle vstupu z prostředí vybereme správnou akci → tomuto mechanismu říkáme policy
        - Nějak kódujeme mechanismus selekce akce
        - Můžeme si to představit jako neuronovou síť, která přijímá nějaké vstupy z prostředí a produkuje výstup, kterým je akce
    - Rozhodnutí agenta závisí pouze na aktuálním stavu (ne celé historii)
    - MDP je (S, A, T, R, γ)
        - S … konečná množina stavů agenta
        - A … konečná množina akcí agenta
        - T … funkce přechodu
        - R … funkce odměny
        - γ … discount člen
    - $T: S \times A \times S \to [0, 1]$
        - Mám možnosti jít do jiného stavu zvolením akce z A
        - $T(s_1, a, s_2) = p$
            - s_1 … počáteční stav (stav, ve kterém aktuálně jsem)
            - s_2 … koncový stav (stav, do kterého chci)
            - a … akce, kterou zvolím
            - p … pravděpodobnost že přejdu z s_1 do s_2 akcí a
    - $R: S \times A \times S \to \mathbb{R}$
        - Odpověď z prostředí
        - $R(s_1, a, s_2) = k$
            - s_1 … počáteční stav (stav, ve kterém aktuálně jsem)
            - s_2 … koncový stav (stav, do kterého chci)
            - a … akce, kterou zvolím
            - k … okamžitá odměna, kterou agent dostane při přechodu z s_1 do s_2 akcí a
    - γ
        - Obvykle $\gamma \in [0, 1]$

        - Říká nám "vztah" mezi nejbližší odměnou, kterou dostaneme nyní a dalšími potencionálními odměnami, které můžeme dostat
        - *Chceme aby suma odměn nebyla nekonečná*
    - Co je řešení
        - Agent se snaží maximalizovat očekávanou odměnu v dlouhodobém měřítku
        - Pokud jsou T a R pro nás definovány, tak řešení je triviální matematická optimalizační úloha
        - Obvykle neznáme T
        - Agent se snaží vymyslet policy $\pi : S \rightarrow A$
            - $\pi(s) = a$
            - Pokud jsem ve stavu s, udělej akci a
    - $\pi^* : S \rightarrow A$
        - Optimální policy
        - π^* … maximalizuj očekávanou hodnotu discounted sumy odměn
        - $$\max \mathbb{E}\left(\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1})\right)$$
            - Discounted suma
            - \max{\, \mathbb{E} \left(\underset{t=0}{\overset{\infty}{\sum{}} }\, \gamma^t R(s_t,a_t,s_{t+1}) \right) }
        - První odměna je nejdůležitější, další méně, atd.
            - Proto náme discount člen v sumě
        - Toto je ideální řešení, zatím ale nevíme jak se k tomu algoritmicky dostat
    - Definujeme value funkci $V_\pi : S \rightarrow R$
        - Pro stav nám vrátí odměnu v závislosti na policy
        - Tuto funkci budeme odhadovat – odhadujeme odměnu ve stavu, pokud agent dodržuje policy
- Q-Learning
    - První úspěšný algoritmus pro reinforcement learning
    - Snažíme se odhadnout discounted sumu
    - $Q(s, a)$
        - s … stav
        - a … akce
        - Vrací jaká je discounted suma
    - $Q(s, a) \leftarrow Q(s, a) + \alpha((r + \gamma \cdot max_{a'} Q(s', a')) - Q(s, a))$
        - α … rychlost učení
- Markov Game
    - Q-Learning dohromady s teorií her
    - Zatím jsme se koukali na učení agentů z pohledu jednoho agenta
    - **Markov Game zobecňuje MDP pro N agentů**
    - Chytré pozorování, postavené na tom, že výběr policy je vlastně hra
    - Koukáme se na hru v závislosti na funkci dávající odměnu
        - Můžeme zjišťovat jestli hra má Nashovo ekvilibrium

    - Každý agent má
        - Vlastní funkci R_i dávající odměnu
        - Vlastní policy π
    - Value funkce V_π nezávisí pouze na policy jednoho agenta, ale na policy všech agentů
    - Každý agent se snaží co nejlépe učit
    - Úspěch celého MAS měříme jako úspěch všech agentů z hlediska Nashova ekvilibria
    - Můžeme řešit/měřit Pareto-optimalitu
    - Odměny pro jednotlivé agenty ale **nejsou nezávislé**
        - Odměna jednoho agenta závisí i na tom, co dělají ostatní agenti
- Policy-based learning
    - Dneska spjato hlavně s deep neural networks
    - Zatímco v Q-Learningu jsme optimalizovali $Q(s, a)$, zde optimalizujeme policy π samotnou
        - Policy je to co je důležité, musíme mít dobrou policy π
    - Policy π je parametrizována parametrem θ
        - Hledáme optimální θ^*
        - Na prostor (možných) parametrů θ použijeme gradient descent
    - REINFORCE strategie
        - Použití Monte Carlo simulace
- Actor-critic methods
    - Actor je agent, který optimalizuje policy
    - Critic je použit pro učení value funkce
    - Critic se učí se prostředí a odhaduje performance actora
    - Snaží se optimalizovat, to co dělá Q-Learning
    - A3C model
- Deep learning
    - DQN = Deep Q-Learning network
    - Policy je reprezentována neuronovou sítí
        - Vstupy jsou stavy
        - Výstupy jsou akce
        - Váhy jsou parametry
    - Asynchronous Advantage Actor Critic (A3C)
        - Jeden z nejnovějších v oblasti Deep Reinforcement Learning algoritmů
        - Více workerů aktualizuje informaci o gradientu asynchronně