UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY
KATEDRA APLIKOVANEJ INFORMATIKY

Úvod do Umelej Inteligencie
Backward-Chaining & Wumpus

Október 30, 2025

BACKWARD-CHAINING

Budeme implementovať logickú inferenciu pomocou Backward-Chaining algoritmu, ktorú následne použijeme na zistenie bezpečných políčok v hre.

Knowledge Base:

Aby bola inferencia pomocou B-CH možná, tak naša KB bude pozostávať iba z klauzúl. Tie budeme v záujme jednoduchšej implementácie spracovávať ako implikácie s n predpokladmi ($premises$) a práve jedným záverom ($conclusion$), napr.:

$$(A \wedge B) \implies C$$
$$(C \wedge \neg B \wedge D) \implies F$$

Všimnime si, že ľubovoľnú implikáciu $(A \wedge B) \implies C$ môžeme prepísať ako $(\top \wedge A \wedge B) \implies C$, kde $\top$ je znak pre $True$. Vďaka tomu môžeme do KB zapisovať fakty, t.j. literály ktoré platia, vo forme implikácií s nula predpokladmi:

$$\text{fakt: } A \quad \equiv \quad \text{implikácia: } \top \implies A$$

Jednotlivé implikácie v KB sa môžu volať aj klauzuly, vety či pravidlá, záver ($conclusion$) je potom "hlava pravidla".

Backward-Chaining:

Algoritmus B-CH dostane na vstupe literál q, ktorý chceme dokázať, t.j. cheme ukázať že $KB \models q$. Postupne sa na neho pokúsi aplikovať všetky pravidlá, pričom ďalej dokazuje predpoklady týchto pravidiel. V skratke:

1. vstup: literál q
2. nájdi všetky pravidlá $R_1, R_2, \ldots$ ktoré majú q ako záver
3. dokáž **aspoň jedno** pravidlo R_i tým, že pomocou B-C dokážeš **všetky** jeho predpoklady
4. ak sa (3) podarilo, tak $KB \models q$, inak $KB \not\models q$

Treba si dať pozor na to, že pri dokazovaní sa dá zacykliť pokiaľ máme cyklické pravidlá, napr. pokiaľ chceme dokázať A a naša KB je:

$$A \implies B$$
$$B \implies A$$

Výsledný program, ktorý s týmto počíta, by mal mať dosť blízko k back-trackingu alebo prehľadávaniu grafu do hĺbky.

Program:

Program je rozdelený na tri súbory:
Súbor **logic.py** neupravujete, obsahuje pripravené triedy pre literály a klauzuly:

- $Literal$: jednoduchý literál, môže/nemusí byť negovaný ($\neg A$ alebo A)
- $Fact$ a $Implication$: triedy pre klauzuly

Súbor **kb.py** obsahuje triedu KB, reprezentáciu samotnej Knowledge base. Najdôležitejšie časti sú:

- $self.clauses$: list klauzúl, ktoré sú v KB
- $self.tell(clause)$: funkcia, ktorá "povie" KB nejakú znalosť. V skutočnosti iba pridá klauzulu (implikáciu/fakt) do $self.clauses$
- $self.ask(goal\_literal)$: funkcia, ktorá sa KB "spýta" na nejakú znalosť. Sem budete dorábať samotný B-CH. **Pozor**: argument funkcie je **Literal**, nie **Fact** (klauzula)!

Príklady sú na konci tohto súboru, naozaj si ich pozrite! Ak nerozumiete nejakej časti pripraveného kódu, pýtajte sa.

WUMPUS

Súbor **wumpus.py** obsahuje konzolovú implementáciu Wumpus World (z prednášky). Akcie sú $\{move[up/down/left/right], shoot[up/down/left/right], climb, pickgold\}$, cieľom je nájsť zlato, zabiť Wumpusa a vyliezť z jaskyne. V prípade mierumilovného agenta Wumpusa zabije iba ak je to nevhnutné na nájdenie zlata, ktoré tam určite je. Je implementovaný interaktívny hráč
choose_action_interactive, ktorý vám vždy napíše bezpečné ťahy a spýta sa na ďalšiu akciu - napr. "$moveup$", "$moveU$", "$shootdown$", "$climb$".
Po správnej implementácii úlohy pomocou tohoto súboru môžete otestovať, či hráč dokáže nájsť zlato (a vyhnúť sa dierám a wumpusovi), ak robíte bezpečné ťahy (získané z B-CH).

Úloha 1: Dorobte $KB.ask(goal\_literal)$, kde implementujete B-C algoritmus na inferenciu bez cyklických pravidiel.
Úloha 2: Dorobte $KB.ask(goal\_literal)$ tak, aby inferoval aj v systémoch s cyklickými pravidlami.
Úloha 3: Dorobte funkciu $choose\_action\_automatic$ v súbore wumpus.py tak, aby agent automaticky prehľadal mapku, získal zlato a úspešne vyliezol z jaskyne. Nemusíte počítať s okrajovými prípadmi, t.j. pri testovaní môžete očakávať, že sa v mapke nachádza práve jedno zlato, hráč má práve jeden šíp a ku zlato je možné dostať sa bezpečnými ťahmi (buď so zabitím wumpusa alebo bez).