Umělá inteligence I

Marta Vomlelová

marta@kti.mff.cuni.cz
http:\\ kti.mff.cuni.cz\~marta
S303

Konzultace: **úterý 16:00 –17:00** či po domluvě e–mailem

Literatura

- E. A. Bender: Mathematical Methods in Artificial Intelligence, 1996, IEEE Computer Society Press, California
- S. Russell, P. Norvig: Artificial Intelligence; A Modern Approach, 1995
- V. Mařík, O. Štěpánková, J. Lažanský a kol.: Umělá Inteligence (1). Academia, Praha
- V. Mařík, O. Štěpánková, J. Lažanský a kol.: Umělá Inteligence (2). Academia, Praha

a další.

Osnova

- Úvod
- Prohledávání stavového prostoru (A*)
- Hry dvou hráčů (Minimax, AO*)
- Automatické dokazování (rezoluce)
- Nemonotónní logiky
- Plánování
- (Nejistota v UI)

Co je to umělá inteligence?

- Rich and Knight 1991: ”The study of how to make computers do things at which, at the moment, people are better”.
- Winston 1992: ”The study of the computations that make it possible to perceive, reason and act”.

Základní cíle UI

- Usuzování
  Máme obecné znalosti + specifická fakta, co jsme z toho schopni odvodit?
  Např. medicínské znalosti + symptomy pacienta, jakou má nemoc?
- Plánování
  Máme znalost prostředí a současné situace, a konkrétní cíl.
  Zajímá nás, jak toho cíle dosáhnout.
- Hledání řešení problému obecně (prohledávání všech možností).

Usuzování za nejistoty

- nemonotónní logiky
  Ptáci (většinou) létají. Tweedy je pták. Tedy odvodím, že létá. Pokud dostanu další informaci, že je to tučňák, tak ruším předchozí odvozenou znalost.
- fuzzy systémy
  funkce náležení "někdo je více malý, někdo spíš není malý, ale hranice není ostrá"
- pravděpodobnostní modely
  Ojeté auto bude na 50% vpořádku, je levnější, nové je dražší a má záruku – které je výhodnější koupit?
  počítám podmíněnou pravděpodobnost jevu dáno znalosti, resp. při dané funkci užitku vybírám řešení s maximální očekávanou hodnotou

Umělý člověk

- robotika
  Asimo, vesmírné sondy – i otázka HW, nutná maximální spolehlivost
- porozumění přirozenému jazyku
  Turingův test, Eliza (1965 Weutenbaum)
- zpracování obrazu

Vnitřní podoba s člověkem

- neuronové sítě (1943 Warren Mc Culloch, Hebb)
- genetické algoritmy
- kognitivní věda se zabývá tím, jak probíhají inteligentní procesy v člověku.

(My se člověkem jen inspirujeme či od něj přebíráme cíle.)
Vývoj létání zpočátku čerpal ze studia létání ptáků, pak šel vlastní cestou.

Učení

- měmím dlouhodobý model světa na základě zkušenosti (série nových pozorování)
  každý se učí
  - neuronové sítě
  - rozhodovací stromy, množiny pravidel
  - logické programy (ILP)
  - ...

Multiagentní systémy

- komunikace
- rozdělování úkolů
- sdílení znalostí
- integrace znalostí – pokud mi agent říká to, nakolik mu mám věřit?

Model světa — stavový prostor

Stavový prostor

- množina stavů $\mathcal{S}$
- množina akcí Φ, tj. parciálních funkcí na stavovém prostoru $\phi_A : \mathcal{S} \hookrightarrow \mathcal{S}$ (*akce nemusí být definovaná na celém* $\mathcal{S}$)

Úloha

- jeden stav **počáteční** s_0
- množina koncových stavů $\mathcal{F}$

Řešení úlohy

- posloupnost akcí - operátorů $(\phi_1, \ldots, \phi_n)$ taková, že $\phi_n(\phi_{n-1}(\phi_{n-2}(\ldots \phi_1(s_0))))$ je definováno a je prvkem množiny cílových stavů $\mathcal{F}$.

Pozn: tento model je základní, složitější jazyky (např. logika) nám umožní komprimovanější reprezentaci.

Příklad: přelévání vody

- Jsou dáný dvě nádoby, větší A o obsahu a litrů, B obsahu b litrů
- na začátku jsou obě prázdné (počáteční stav)
- cílem je nádoba A prázdná a v B má $2 \cdot (a - b)$ litrů vody
- k dispozici je neomezený zdroj vody
- nádoby nemají označené míry

Přelévání vody – reprezentace

- Stav: dvojice $\langle c_A, c_B \rangle$ množství vody v nádobách A,B
- počáteční stav $\langle 0, 0 \rangle$
- jediný cílový stav $\langle 0, 2 \cdot (a - b) \rangle$
- přechody:
  - vylití A: $\langle c_A, c_B \rangle \longrightarrow \langle 0, c_B \rangle$
  - vylití B: $\langle c_A, c_B \rangle \longrightarrow \langle c_A, 0 \rangle$
  - naplnění A, B
  - přelití A do B:

$$\langle c_A, c_B \rangle \longrightarrow \langle max(c_A - (b - c_B), 0), min(c_A + c_B, b) \rangle$$

$\vdots$

Př. Devítka

start

8	1	3
2	6	4
7		5

cíl

1	2	3
8		4
7	6	5

Na volnou pozici mohu přesunout libovolné hranou sousedící číslo, a tím přejít do jiného stavu. Existuje cesta ze startu do cíle?

Navrhněte reprezentaci. Jaké budou akce?

Př. Splnitelnost okrajových podmínek

1	1	1	2	1,1	2	1	1	-	5
1,1									
1,1,1									
4,1									
1,1,1									
1,1,1									

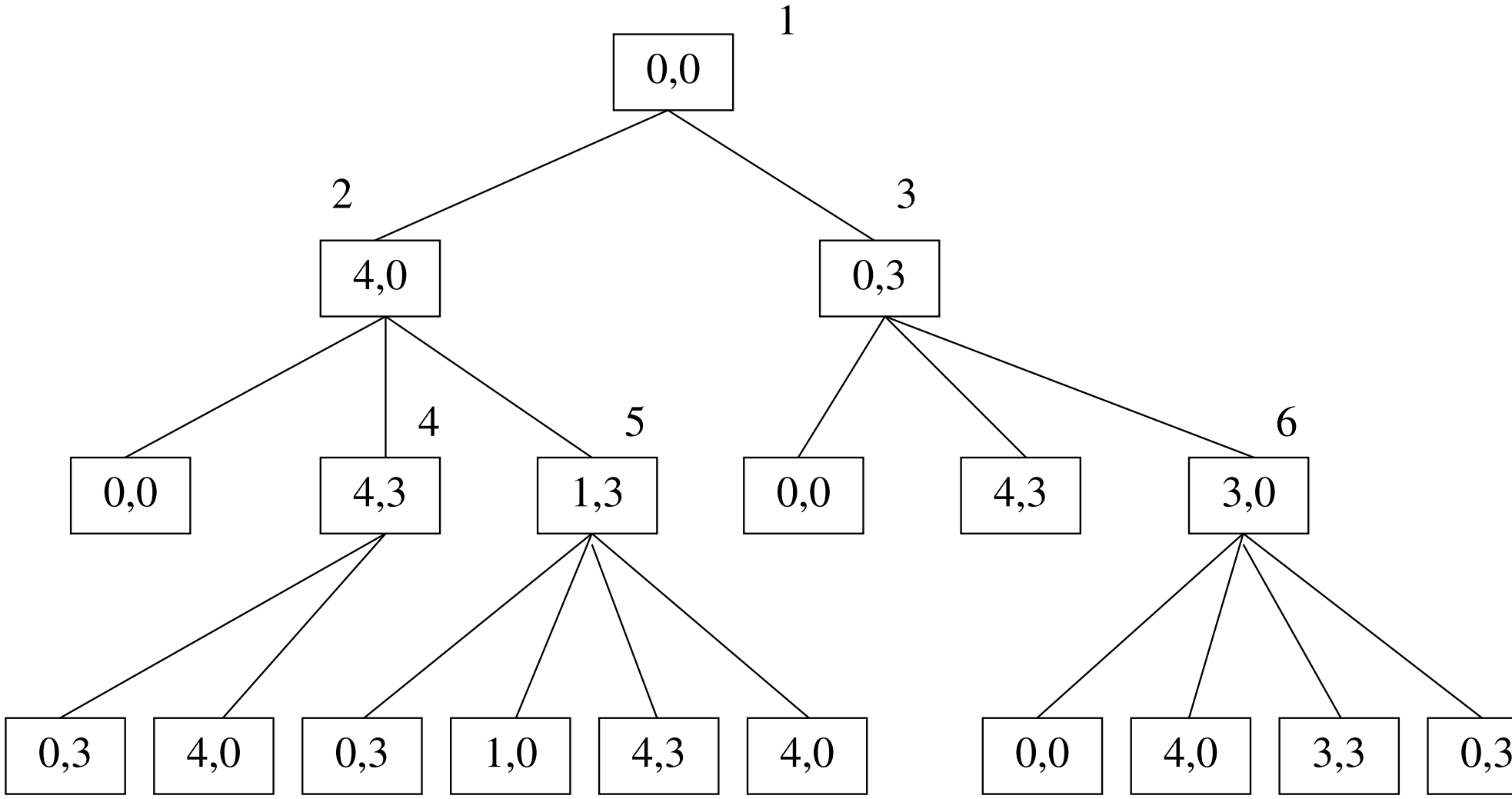
Cílem je zaplnit některá políčka tak, aby souhlasila čísla na stranách: např. 4,1 značí volné místo, čtyři plné bloky za sebou, volné místa, jeden plný blok a pak už jen volno.
Volno na začátku a na konci nemusí být, mezi může být libovolný počet volných bloků.

Prohledávání stavového prostoru

- Stavový prostor lze reprezentovat orientovaným grafem
- uzel reprezentuje stav
- hrana reprezentuje přechod mezi stavy
- řešení úloh pak lze formulovat jako hledání *přijatelné* cesty v orientovaném grafu z počátečního uzlu do některého cílového stavu.

Algoritmus **Prohledávání stavového prostoru**

OPEN $\leftarrow\{s_0\}$, CLOSED $\leftarrow \{\}$
    if ($s_0 \in \mathcal{F}$) return *úspěch*
    do
        if(OPEN={}) return *neúspěch*
        STATE$\leftarrow$ dejAOdstranPrvni(OPEN)
        STATES$\leftarrow$odstranUzavrene(Expanduj(STATE))
        přidejVše(STATES, OPEN) , přidej(STATE, CLOSED)
    until(STATES $\cap\mathcal{F} \neq \{\}$)
    return *úspěch*


1
0,0
2
3
4,0
0,3
4
5
6
0,0
4,3
1,3
0,0
4,3
3,0
0,3
4,0
0,3
1,0
4,3
4,0
0,0
4,0
3,3
0,3

Způsoby prohledávání stav. prostoru

- **Do šířky –** uzly k expanzi řadíme do fronty
  Náročné na paměť
  Varianta: vždy prodloužit cestu minimální ceny
- **Do hloubky –** uzly řadíme do zásobníku
  Sejde z cesty a je mimo; nutno ošetřit cykly
- **IDFS** do hloubky s omezením max. hloubky, iterativně prohlubovat
- Dvousměrné prohledávání od poč. stavu i od cíle
- **Informované prohlevávání** - pořadí prohledávání na základě ”další” informace, odhadu vzdálenosti stavu od cíle, vyjádřeného heuristikou $h(uzel)$.

Cenová funkce (rozšíření úlohy)

V následujících algoritmech předpokládáme zadanou
cenovu jednotlivých přechodů (hran) $c(u, v)$
a hledáme **řešení nejmenší ceny**

$$min_{u \in \mathcal{F}} g(u)$$

kde $g(u)$ je kumulovaná cena za průchod od startu do uzlu u, tj.

$$g(u) \equiv \sum_{(v_i, v_{i+1}) \in \text{cesta}} c(v_i, v_{i+1})$$

Prohledávání do šířky modifikujeme výběrem uzlu s nejnižší kumulovanou cenou.

Algoritmus větví a mezí (Branch and bound)

- dokud nenajde nějaké řešení, prohledává (do hloubky či heuristicky) jako dřív
- zná-li řešení ceny $g(u_1)$, pak ze seznamu OPEN odstraňuje všechny uzly s $g(u) \geq g(u_1)$.

Informované prohledávání

Expanduje uzel s nejmenší $f(x)$ Tj. OPEN setříděný od nejmenšího $f(x)$

Algoritmus A

heuristika $h(uzel)$ – odhad cesty do cíle

$h(uzel) = 0$ právě když je $uzel$ cílovým uzlem

a definuji

$$f(uzel) \equiv g(uzel) + h(uzel)$$

$g(u)$ cena cesty od startu, tj. kumulativní $c(,)$

Algoritmus A*

heuristika $h(uzel) \geq 0$ je **dolní** odhad cesty do cíle, tzv. optimistická heuristika

Heuristiky u Devítky

Možné heuristiky jsou např.

- h_1 počet špatně umístěných kostiček
- h_2 součet vzdáleností kostiček od jejich správných pozic

Algoritmus **Algoritmus A***

OPEN $\leftarrow \{\langle s_0, f(s_0)\rangle\}$, CLOSED $\leftarrow \{\}$
LOOP:
    If(OPEN=$\{\}$) return *neúspěch*
    STATE$\leftarrow$ dejAOdstraňMinimum(OPEN)
    CLOSED $\leftarrow$ CLOSED $\cup$ {STATE}
    If(STATE$\in \mathcal{F}$) return *úspěch*
    STATES$\leftarrow$ expanduj(STATE)
    forAll $\langle uzel, rodic\rangle$ in STATES
        Case *uzel* not in OPEN, CLOSED:
            add($\langle N, f(N), rodic\rangle$, OPEN)
        Case *uzel* in OPEN or CLOSED with $f'(N) > f(N)$:
            remove($\langle N, f'(N), rodic\rangle$, OPEN)
            add($\langle N, f(N), rodic\rangle$, OPEN)
goto LOOP

Přípustná a monotónní heuristika

Nechť h^* značí optimální cenu cesty z uzlu u do cíle, h značí náš heuristikou daný odhad téhož.

- **Přípustná heuristika** je heuristika h, pokud pro každý uzel u platí $h(u) \leq h^*(u)$, tj. h je dolní odhad ceny cesty do cíle.

Přípustnost A*

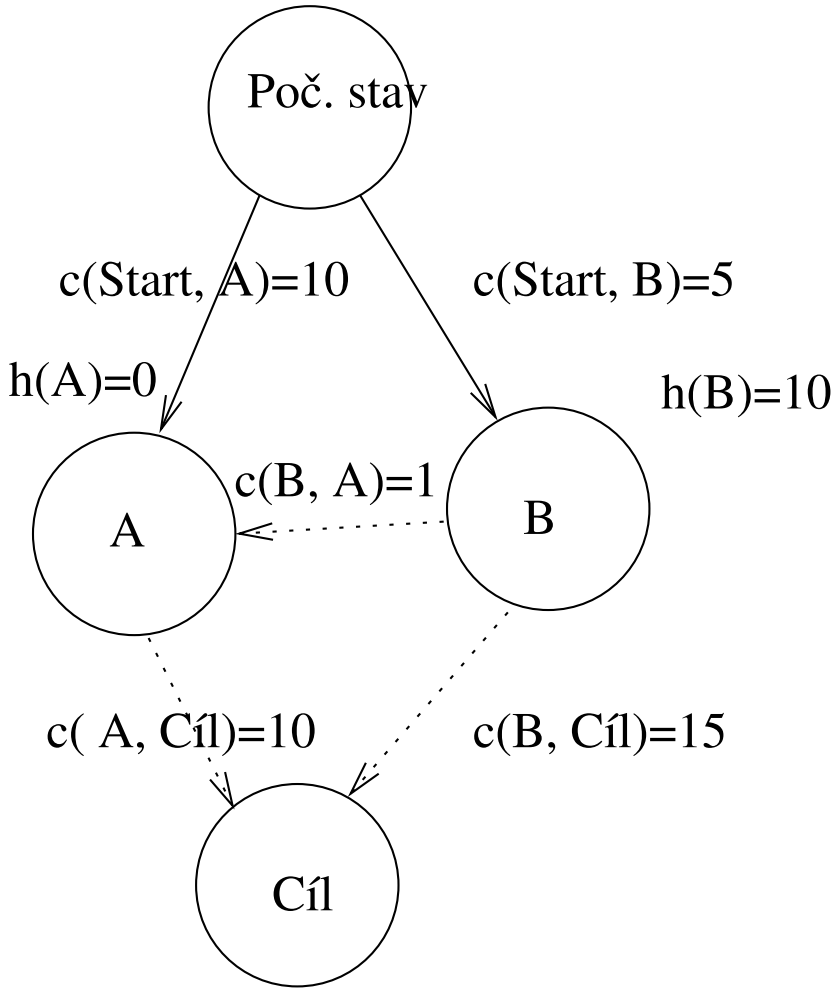
- **Přípustný (=úplný) algoritmus**: Jestliže je cena každé akce kladná a v grafu existuje cesta k cíli, pak algoritmus po konečném počtu kroků vydá optimální řešení.
- **Věta**: Je–li zhora omezený počet větvení v každém kroku a cena libovolného kroku aspoň $\epsilon > 0$, pak je Algoritmus A* přípustný.

Důkaz:

- Omezení větvení a ceny zaručí, že je jen konečně uzlů s $f(n) < f^*$.
- 1. Nechť G optimální cílový stav s cenou $g(G) = f^*(G)$, G_2 není optimální, tedy $g(G_2) > f^*(G)$.
  2. Protože h je přípustná, $f^*(G) \geq f(n)$ pro každý uzel n na optimální cestě do G.
  3. Protože $g(G_2) > f^*(G)$, je i $f(G_2) > f^*(G)$ (přípustnost h)
  4. a uzel G bude vybrán pro expanzi (tj. konec algoritmu) před tím, než by byl expandován G_2.

Příklad nutnosti přepočtu uzlu na CLOSED

Následující příklad ukazuje, že je občas nutné ”vytáhnout” uzel z CLOSED a znovu přepočíst.



Po Startu budeme expandovat uzel A, protože má menší součet ceny

od startu plus heuristika ($10 + 0 < 5 + 10$). Pak expandujeme uzel B a najdeme kratší cestu do A. Heuristika je přípustná, protože vždy dává dolní odhad.
Funkce f není monotónní.

Monotonie

- **Monotónní heuristika** je heuristika h, pokud pro každé dva bezprostřední následníky u_1, u_2 platí $h(n_1) - h(n_2) \leq c(n_1, n_2)$ a $h(cilovyStav) = 0$ pro každý cílový stav.
- **Věta**: Jestliže heuristická složka h funkce $f = g + h$ užitá v alg. A* při prohledávání stavového grafu splňuje na něm podmínku monotonie, pak vybral-li A* v nějakém kroku pro expanzi uzel n, je $g(n) = g^*(n)$.

Chyba heuristiky

- Informovanější heuristika je ta, která expanduje méně uzlů, tj. h_1 je informovanější než h_2 pokud $h_1(n) \geq h_2(n)$ pro každé n.
- např. u Devítky je h_2 součet vzdáleností políček od správné pozice informovanější než h_1 počet špatně umístěných políček.
- Pozn.: Aby počet prohledávaných stavů nerostl exponenciálně, musí chyba heuristiky být malá, tj. $|h(n) - h^*(n)| \leq O(\log h^*(n))$. Informovanější heuristika má menší chybu.

Heuristiku můžeme často nalézt vypuštěním některých omezení z pravidel - např. povolením libovolného tahu dostaneme h_1, povolením překrývání kostiček, ale tahu jen vedle dostaneme h_2 atd.

Toho využívá program `ABSOLVER`, který bere pravidla tahů zapsaná jako implikace a generuje heuristiky vynecháním premis. Našel i první užitečnou heuristiku pro Rubikovu kostku.

Poznámky: maximum přes několik heuristik, statistický odhad, features (např v šachách), čas ke spočtení heuristiky, IDA^*[a], SMA^* Simplified Memory-bounded A*

Graceful Decay of Admissibility POkud h někdy (ale málokdy) nadhodnocuje h^* o hodnotu větší než δ, potom algoritmus A

[a] Iterative deepening A*. Prohledává do hloubky, vylučuje uzly s f(uzel) větší než prahová hodnota f(initState), pokud neuspěje, zvýší práh o minimum toho, o kolik byl práh překročen

zřídkakdy nachází řešení, jehož cena je větší o více než δ ve strovnání s cenou skutečně optimální cesty.

Pozn2: gradientní metody, metoda simulovaného žíhání

-

Heuristika přidaná k akcím

Heuristiky pro přelévání vody:

- Nevyprázdňuj nikdy obě nádoby,
- nenaplňuj obě nádoby,
- nenaplňuj nádobu, je–li druhá prázdná a není–li naplňovaná prázdná,
- nevyprázdňuj nádobu, je–li druhá plná,
- nepřelévej, pokud by potom byla jedna nádoba plná a druhá prázdná.

Produkční systémy

Produkční systém je definován

- souborem produkčních pravidel
  tvaru {`situace S`}→{`akce A`}
  s interpretací: Nastala–li situace S, proveď akci A.
- bází dat
  popisující okamžitý stav řešené úlohy; obvykle bývá vhodným způsobem členěna
- řídící strategií, která určuje, jak pravidla na bázi dat aplikovat.

Produkční systém pracuje v cyklech
rozpoznání situace → vykonání akce

Řídící mechanismus určuje, v jakém pořadí aplikovat pravidla na bázi dat.

- přímý režim řízení *(data–driven strategy)*– od počátečního stavu k některému cílovému stavu
- zpětný režim řízení *(goal–driven strategy)*– od cílového stavu k počátečnímu. Oba režimy lze kombinovat.

Přelévání vody jakožto produkční systém

- báze dat – dvojice $\langle c_A, c_B \rangle$ jako dříve
- pravidla:

	$\{c_A > 0\} \rightarrow \{\text{vylej A}\}$
	$\{c_B > 0\} \rightarrow \{\text{vylej B}\}$
	$\{c_A < a\} \rightarrow \{\text{naplň A}\}$
	$\{c_B < b\} \rightarrow \{\text{naplň B}\}$
	$\{c_A > 0, c_B < b\} \rightarrow \{\text{přelij obsah A do B}\}$
	$\{c_A < a, c_B > 0\} \rightarrow \{\text{přelij obsah B do A}\}$

Jak se liší (takto definovaný, OPS-5, TOPS) produkční systém od Prologu?

Přelévání vody se specifičtějšími pravidly

- báze dat – dvojice $\langle c_A, c_B \rangle$ jako dříve
- pravidla:

	$\{c_A > 0 \& c_B > 0 \& c_B \neq b\} \rightarrow \{\text{vylej A}\}$
	$\{c_B > 0 \& c_A > 0 \& c_A \neq a\} \rightarrow \{\text{vylej B}\}$
	$\{c_A < a \& c_B \neq b \& (\& c_B \neq 0 \vee c_A = 0)\} \rightarrow \{\text{naplň A}\}$
	$\{c_B < b \& c_A \neq a \& (\& c_A \neq 0 \vee c_B = 0)\} \rightarrow \{\text{naplň B}\}$
	$\{c_A > 0 \& c_B < b \& c_A + c_B \neq b\} \rightarrow \{\text{přelij obsah A do B}\}$
	$\{c_A < a \& c_B > 0 \& c_A + c_B \neq b\} \rightarrow \{\text{přelij obsah B do A}\}$

Tím jsme omezili prostor k prohledávání, tím — snad — i dobu potřebnou k řešení.

Anytime algoritmus

Prohledávání většinou trvá dlouho. Často (např. v systémech pracujících v reálném čase) potřebujeme odpověď v určitém okamžiku, který nemusí být znám předem.

- **Anytime (přerušitelný) algoritmus** je algoritmus který je schopen poskytnout odpověď kdykoli ho uživatel přeruší, pokud mu byl poskytnut určitý minimální čas pro běh.
- Na anytime algoritmus se nehodí prohledávání do hloubky; hodí se heuristické best–first prohledávání a iterativně prohlubované prohledávání.
- Z libovolného algoritmu $\Xi(x)$, kde parametr x ovlivňuje čas běhu $t(x)$ tak, že t je rostoucí funkce, umíme udělat anytime algortmus $\Xi * (\overrightarrow{x})$, kde $\overrightarrow{x}$ je námi zvolená posloupnost $x_1 < x_2 < \ldots$.

Konstrukce anytime algoritmu

- Inicializace: Dáme $k = 1$ a $out = 0$.
- Krok: Provádíme výpočet $\Xi(x_k)$. Jsme–li přerušeni, vrátíme out, po úspěšném konci pokračujeme dále
- Iterace: Položíme $out = \Xi(x_k)$ a $k = k + 1$ a pokračujeme dalším krokem.

O co je anytime verze pomalejší?

- Pokud byl algoritmus $\Xi*(\overrightarrow{x})$ přerušen v čase $T \geq t(x_1)$, pak vrátí stejný výstup jako $\Xi(x_k)$ kde

  $$t(x_1)+t(x_2)+\ldots+t(x_k) \leq T < t(x_1)+t(x_2)+\ldots+t(x_k)+t(x_{k+1}).$$

- Pokud $\Xi(x)$ zlepšuje výstup při delším čase běhu, pak existuje posloupnost $\overrightarrow{x}$ s následující vlastností:
  Je–li $\Xi*(\overrightarrow{x})$ přerušen v čase T, pak $\Xi(x)$ potřebuje čas větší než $\frac{T}{4}$ k tomu, aby dosáhlo lepšího výsledku.

Posloupnost $\overrightarrow{x}$ může závisí na neznámé funkci t. Pak můžeme volit např. x_k tak, aby očekávaný čas běhu $\Xi(x_k)$ byl dvojnásobkem času $\Xi(x_{k-1})$.