GPlan plánovač

Ondřej Sýkora

leden 2006

Obsah

1 Úvod

GPlan je implementace plánovacího systému založená na plánovacím algoritmu Graphplan.

2 Použití

Plánovač se spouští příkazem

```
java -jar GPlan.jar [-v] soubor
```

kde *soubor* je název souboru se zadáním plánovacího problému. Pokud je uveden přepínač *-v*, vypisuje program detailnější zprávy o své činnosti.

Pro spuštění programu je nutné mít nainstalovaný *Java runtime* ve verzi 1.5, neby vyšší.

Pro použití systému je nutné zapsat specifikaci plánovacího problému. Ta se zapisuje ve speciálním formátu podobném zápisu programů v jazyce Prolog.

2.1 Formát vstupního souboru

Názvy predikátů, konstant a proměnných musí začínat písmenem, dále mohou obsahovat libovolný počet písmen, číslic, pomlček a podtržítek. Stejně jako v Prologu konstanty zapisujeme s prvním písmenem malým a proměnné s prvním písmenem velkým.

2.1.1 Komentáře

Vstupní soubor může obsahovat textové komentáře. Komentář vždy začíná znamek procento (%) a končí s koncem řádku.

2.1.2 Počáteční stav

Počáteční stav systému je popsán pomocí samostatných predikátů. Například:

```
ruka-volna.
na-zemi(a).
lezi-na(b,a).
```

2.1.3 Cílový stav

Stejně jako predikáty popisující počáteční stav se zapisují i predikáty, které je nutné splnit pro dosažení cíle. Proti predikátům počátečního stavu se liší zejména v tom, že je předchází slovo goal. Například:

```
goal mam(penize).
goal jsem(doma).
```

Pokud se v zadání plánovacího problému vyskytuje slovo *goals*, všechny predikáty za tímto slovem se považují za cílové, i pokud je nepředchází slovo *goal*. V následujícím příkladu predikáty *ruka-volna* a *lezi-na(b,a)* popisují počáteční stav, zatímco predikáty *mam(penize)* a *jsem(doma)* vyznačují cílový stav.

```
ruka-volna.
lezi-na(b,a).

goals
mam(penize).
jsem(doma).
```

2.1.4 Akce

Zadání akcí se od klasického prologovského zápisu mírně liší, i tak se ale jedná o velmi jednoduchou formu. Akce zapisujeme ve tvaru

$$\{jméno\ akce\}\ ::\ \{předpoklady\} => \{efekty\}.$$

. Předpoklady a pozitivní efekty se zapisují jako seznam predikátů oddělených čárkou, negativní efekty se zapíší stejně jako pozitivní,jen je před ně nutné vložit slovo not. Pomocí predikátu *distinct* lze vynutit různost dvou proměnných.

```
jdi :: jsem-na(X),lokace(Y),distinct(X,Y) => jsem-na(Y),
                                             not jsem-na(X).
vyber-penize :: jsem-na(bankomat),mam(karta) => mam(penize).
```

2.2 Výstup

Při výstavbě plánovacího grafu program vypisuje hlášku po každé vytvořené vrstvě. Pokud plánování selže a plán není nalezen, program toto oznámí a skončí.

V případě, že plán je nalezen, vypíše program posloupnost akcí v plánu. Akce jsou vypsány vždy jedna na řádku v pořadí, v jakém je nutné je provést (první akce nejdříve). Z výpisu jsou vynechány *no-op* akce, které se používají pro přenos predikátů do následující vrstvy.

3 Implementace

Plánovací systém je implementován v jazyce Java, podrobná dokumentace ke kódu je sepsána formou (anglických) komentářů pro javadoc přímo v kódu, v samostatném souboru je přiložena vygenerovaná dokumentace v HTML.

Proti algoritmu popsanému na přednášce tato implementace nepoužívá plně instanciované akce, ale klasickou(predikátovou) reprezentaci. K vytváření konkrétních instancí akcí dochází až při tvorbě akčních vrstev v plánovacím grafu. To vede k výtaznému zkrácení zadání plánovacího problému.

Hlavní část programu je implementována ve třídě *PlanningProblem* v metodě *solve()*. Ta postupně pomocí metody *singleStep()* staví plánovací graf. Pro každou nově přidanou predikátovou vrstvu ověřuje, jestli jsou přítomné všechny cílové predikáty a není mezi nimi žádný mutex. V případě že ne, zkouší provést extrakci plánu pomocí *findSerialPlan*.

Plánovací graf je reprezentován jako posloupnost "dvouvrstev", pracuje se vždy s dvojicí akční vrstva + následující predikátová vrstva. Při výstavbě plánovacího grafu se tato dvojice vytváří "najednou", kontrola ukončovací podmínky probíha jen na predikátové vrstvě. Jednotlivé (dvoj)vrstvy jsou na sobě nezávislé, každá obsahuje kompletní kopii stavu.

Při extrakci plánu se procházejí vrstvy postupně od poslední, vždy se hledá podpora pro předpoklady akcí z předcházející vrstvy. Akce jsou testovány v pořadí, v jakém byly nalezeny. Nalezené no-good kombinace jsou ukládány ve všech (predikátových) vrstvách plánovacího grafu, aby při dalších průchodech byly nalezeny co nejdříve.

3.1 Možná zlepšení

- **Typy pro proměnné**, které by definovaly doménu možných hodnot a při dosazování při vytváření konkrétních instancí akcí by hodnoty proměnných mohly být voleny jen z příslušné domény. Toto je možné obejít přidáním pomocných predikátů určujích doménu, tím ale dochází ke zvětšení plánovacího grafu a zvýšení náročnosti výpočtu.

- Podpora pro **neměnné predikáty**, tedy takové, které jsou platné po celou dobu výpočtu a není je nutné reprezentovat v plánovacím grafu ani pro ně ověřovat mutexy.

- **Heuristiky v prohledávacích procedurách**, které by mohly přispět k rychlejšímu nalezení řešení.

- **Inkrementální reprezentace** vrstev v plánovacím grafu.