Použití frameworku Squander

Vedoucí: Ing. Jiří Daněček

Oponent: Ing. Martin Bloch, CSc.

Bc. Martin Kožený

České vysoké učení technické v Praze - Fakulta Elektrotechnická

23. ledna 2012

Zadání a cíle

- prostudovat framework Squander
- popsat význam jednotlivých komponent
- implementace sady algoritmů, které je obtížné implementovat imperativně
- pro vybrané algoritmy porovnat implementaci klasickým způsobem a pomocí frameworku

Zadání a cíle

- prostudovat framework Squander
- popsat význam jednotlivých komponent
- implementace sady algoritmů, které je obtížné implementovat imperativně
- pro vybrané algoritmy porovnat implementaci klasickým způsobem a pomocí frameworku

Zadání a cíle

- prostudovat framework Squander
- popsat význam jednotlivých komponent
- implementace sady algoritmů, které je obtížné implementovat imperativně
- pro vybrané algoritmy porovnat implementaci klasickým způsobem a pomocí frameworku

Zadání a cíle

- prostudovat framework Squander
- popsat význam jednotlivých komponent
- implementace sady algoritmů, které je obtížné implementovat imperativně
- pro vybrané algoritmy porovnat implementaci klasickým způsobem a pomocí frameworku

Základní pojmy

- deklarativní programování
- anotace
- Squander

Základní pojmy

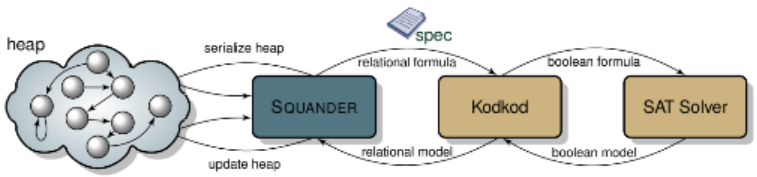
- deklarativní programování
- anotace
- Squander

Základní pojmy

- deklarativní programování
- anotace
- Squander

Architektura

- serializace heap do relací
- překlad relací na heap do Kodkodu
- překlad relací do booleovské logiky
- pokud je nalezeno řešení, tak následuje obnovení relací z ohodnocené booleovské formule a hodnot atributů z relací
- obnova heapu a promítnutí řešení



Kodkod

- solver pro relační logiku
- požaduje ohraničení pro každou relaci a relační formuli
- překládá daný problém do SAT
- používá SAT solver k nalezení vyhovujícího řešení, které vrací, pokud je nalezeno

Kodkod

- solver pro relační logiku
- požaduje ohraničení pro každou relaci a relační formuli
- překládá daný problém do SAT
- používá SAT solver k nalezení vyhovujícího řešení, které vrací, pokud je nalezeno

Kodkod

- solver pro relační logiku
- požaduje ohraničení pro každou relaci a relační formuli
- překládá daný problém do SAT
- používá SAT solver k nalezení vyhovujícího řešení, které vrací, pokud je nalezeno

Kodkod

- solver pro relační logiku
- požaduje ohraničení pro každou relaci a relační formuli
- překládá daný problém do SAT
- používá SAT solver k nalezení vyhovujícího řešení, které vrací, pokud je nalezeno

JFSL

- specifikace pro Javu podporující relační a množinovou algebru
- pomocí této specifikace je možno deklarovat jaká část heapu se změní a jak
- poskytuje algebraické operátory společně s celočíselnými operátory a booleovskými operátory

JFSL

- specifikace pro Javu podporující relační a množinovou algebru
- pomocí této specifikace je možno deklarovat jaká část heapu se změní a jak
- poskytuje algebraické operátory společně s celočíselnými operátory a booleovskými operátory

JFSL

- specifikace pro Javu podporující relační a množinovou algebru
- pomocí této specifikace je možno deklarovat jaká část heapu se změní a jak
- poskytuje algebraické operátory společně s celočíselnými operátory a booleovskými operátory

Implementované algoritmy

Algoritmy naimplementované ve frameworku Squander

- Problém batohu
- Zobecněná bisekční šířka
- L-dominující množina grafu
- Nalezení Hamiltonovské cesty
- Trojúhelníkový soliter
- Problém n dam

Implementované algoritmy

Algoritmy naimplementované ve frameworku Squander

- Problém batohu
- Zobecněná bisekční šířka
- L-dominující množina grafu
- Nalezení Hamiltonovské cesty
- Trojúhelníkový soliter
- Problém n dam

Implementované algoritmy

Algoritmy naimplementované ve frameworku Squander

- Problém batohu
- Zobecněná bisekční šířka
- L-dominující množina grafu
- Nalezení Hamiltonovské cesty
- Trojúhelníkový soliter
- Problém n dam

Implementované algoritmy

Algoritmy naimplementované ve frameworku Squander

- Problém batohu
- Zobecněná bisekční šířka
- L-dominující množina grafu
- Nalezení Hamiltonovské cesty
- Trojúhelníkový soliter
- Problém n dam

Implementované algoritmy

Algoritmy naimplementované ve frameworku Squander

- Problém batohu
- Zobecněná bisekční šířka
- L-dominující množina grafu
- Nalezení Hamiltonovské cesty
- Trojúhelníkový soliter
- Problém n dam

Implementované algoritmy

Algoritmy naimplementované ve frameworku Squander

- Problém batohu
- Zobecněná bisekční šířka
- L-dominující množina grafu
- Nalezení Hamiltonovské cesty
- Trojúhelníkový soliter
- Problém n dam

Nalezení Hamiltonovské cesty

```
1  @Ensures ({
2      "return[int] in this.edges.elts",
3      "return[int].(src + dest) = this.nodes.elts",
4      "return.length = #this.nodes.elts - 1 ",
5      "all i: int | i >= 0 && i < return.length - 1 =>
6      return[i].dest = return[i+1].src"
7      })
8  @Modifies ({ "return.length", "return.elems" })
9  @FreshObjects ( cls = Edge[].class , num = 1 )
10 public Edge [] solveHamiltonianPath()
11 {
12   return Squander.exe(this);
13 }
```

Listing 1: Implementace nalezení Hamiltonovské cesty ve Squanderu

Porovnání

- k zvolené sadě algoritmů byly naimplementovány jejich „imperativní protějšky“
- imperativní algoritmy byly naimplementovány technikou BB-DFS (L-dominující množina grafu,. . .) nebo pomocí bactrackingu (Nalezení Hamiltonovské cesty)
- k jednotlivým algoritmům byly vygenerovány instance daného problému, které sloužily k porovnání výpočetního času, zatížení procesoru a paměťové náročnosti daného algoritmu v dané implementaci

Porovnání

- k zvolené sadě algoritmů byly naimplementovány jejich „imperativní protějšky“
- imperativní algoritmy byly naimplementovány technikou BB-DFS (L-dominující množina grafu,. . .) nebo pomocí bactrackingu (Nalezení Hamiltonovské cesty)
- k jednotlivým algoritmům byly vygenerovány instance daného problému, které sloužily k porovnání výpočetního času, zatížení procesoru a paměťové náročnosti daného algoritmu v dané implementaci

Porovnání

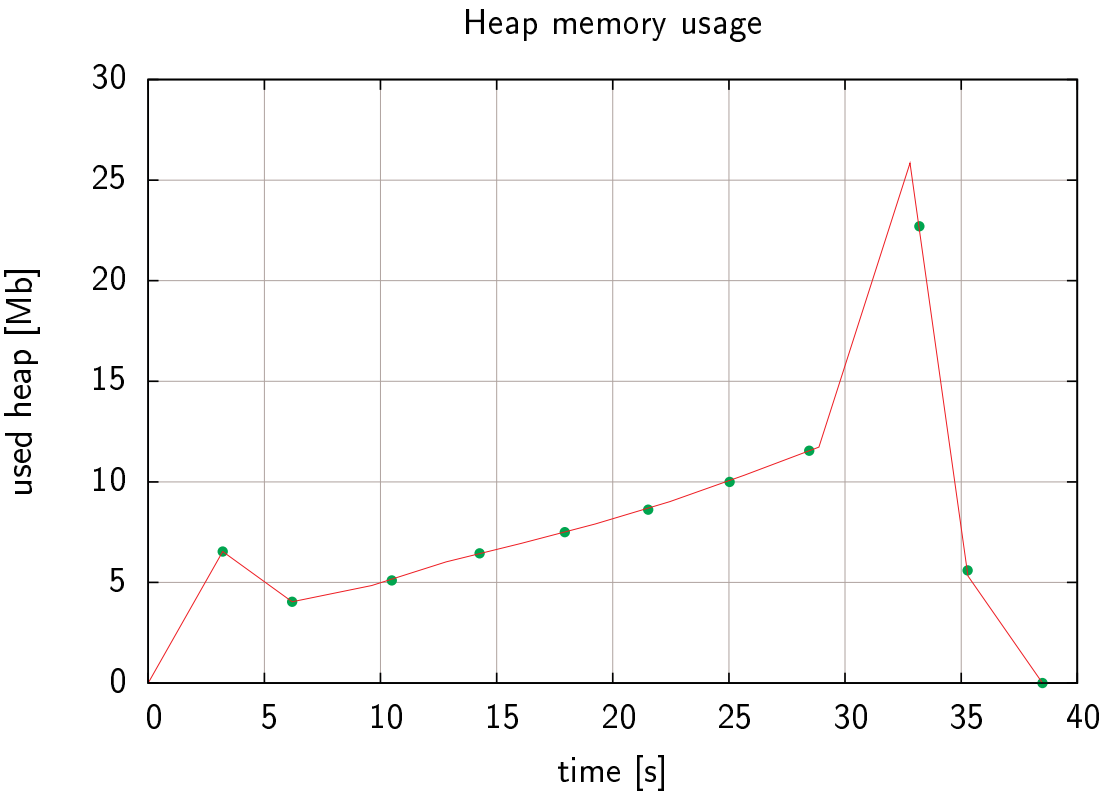
- k zvolené sadě algoritmů byly naimplementovány jejich „imperativní protějšky“
- imperativní algoritmy byly naimplementovány technikou BB-DFS (L-dominující množina grafu,. . .) nebo pomocí bactrackingu (Nalezení Hamiltonovské cesty)
- k jednotlivým algoritmům byly vygenerovány instance daného problému, které sloužily k porovnání výpočetního času, zatížení procesoru a paměťové náročnosti daného algoritmu v dané implementaci

Porovnání výpočetních časů

	Number of nodes in the graph								
	70			80		90		100	
	max grade			max grade		max grade		max grade	
	2	3	4	2	3	2	3	2	3
Imperatively	30	3 940	**t/o**	20	14 920	70	173 090	40	**t/o**
Squander	3 940	5 100	**t/o**	4 060	70 860	5 080	**t/o**	4 700	**t/o**

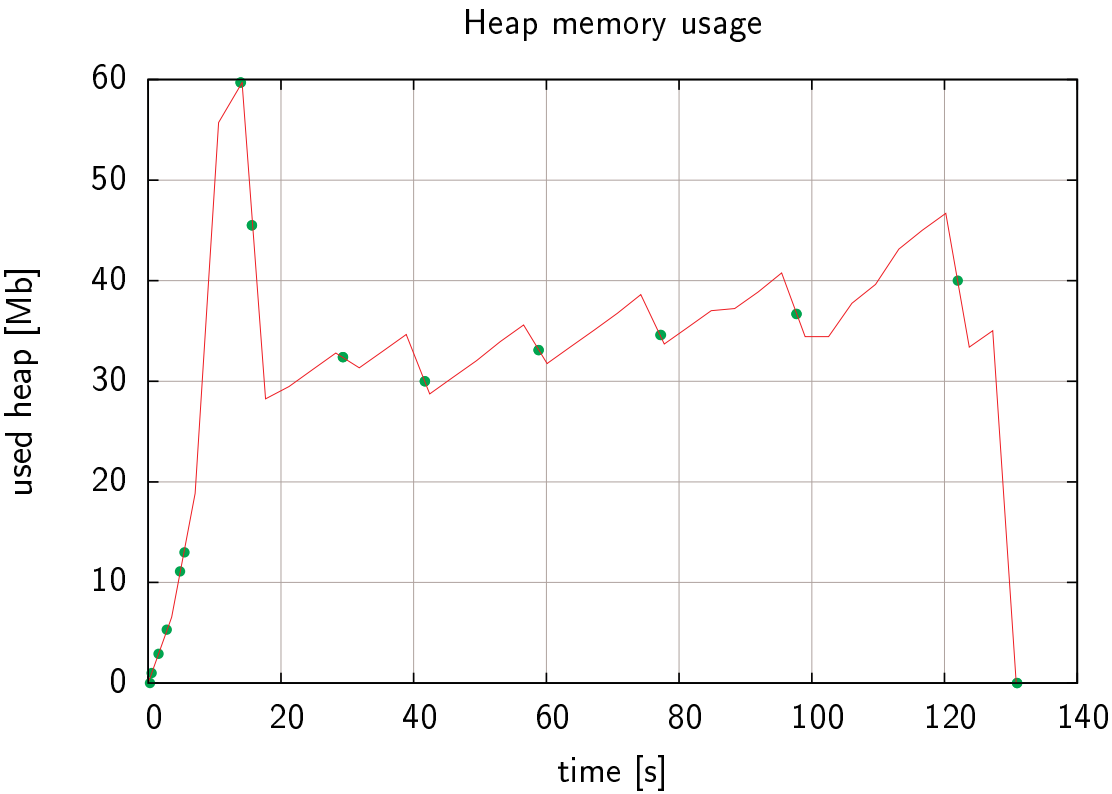
Obrázek: Výpočetní časy jednotlivých instancí obou implementací Nalezení Hamiltonovské cesty

Paměťová náročnost - imperativní implementace



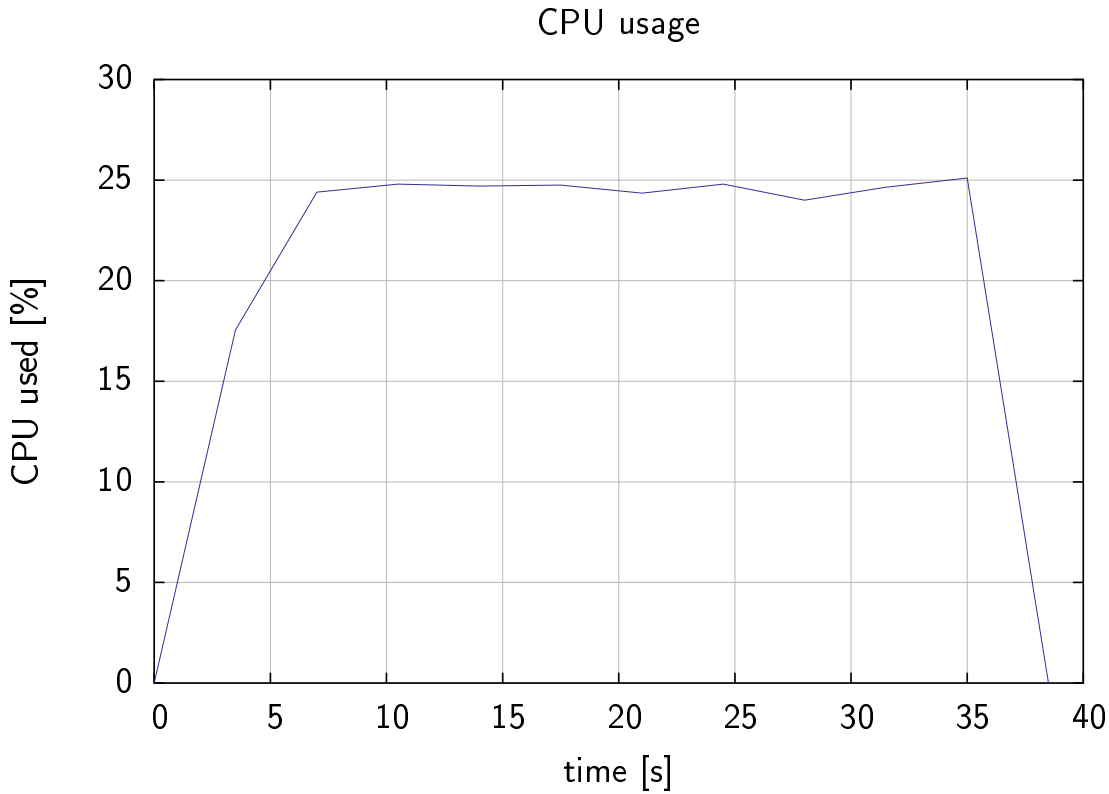
Obrázek: Využití heap v čase - Nalezení Hamiltonovské cesty v imperativní implementataci pro graf se 40 uzly, maximálním stupněm 5

Paměťová náročnost - Squander



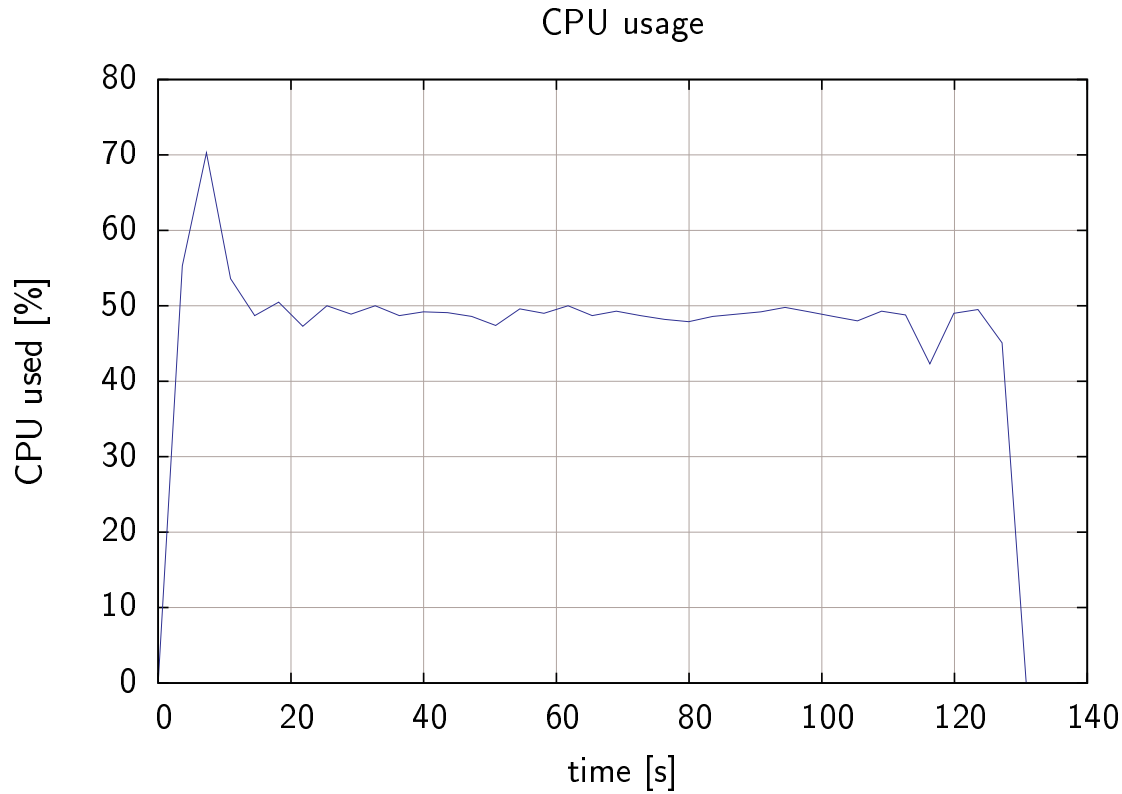
Obrázek: Využití heap v čase - Nalezení Hamiltonovské cesty v implementataci frameworkem Squander pro graf se 40 uzly, maximálním stupněm 5

Zatížení procesoru - imperativní implementace



Obrázek: Zatížení procesoru v čase - Nalezení Hamiltonovské cesty v imperativní implementataci pro graf se 40 uzly, maximálním stupněm 5

Zatížení procesoru - Squander



Obrázek: Zatížení procesoru v čase - Nalezení Hamiltonovské cesty v implementataci frameworkem Squander pro graf se 40 uzly, maximálním stupněm 5

Činnost garbage collectoru

	Number of nodes in the graph		
	40	50	80
	max grade	max grade	max grade
	4	5	3
Imperatively	0.01% (0.005126 secs)	0.01% (0.006266 secs)	0.008% (0.00125 secs)
Squander	0.8% (1.054949 secs)	0.3% (1.908237 secs)	1.2% (0.851782 secs)

Obrázek: Poměr časové činnosti garbage collectoru ku celkovému výpočetnímu času u implementací Nalezení Hamiltonovské cesty

Porovnání závislostí

- pro implementaci ve Squanderu podle vypočítaných korelací závisí výpočetní čas více na maximálním stupni uzlu v grafu - hodnota korelace 0.91 - než na počtu uzlů v grafu - hodnota korelace 0.48
- pro imperativní implementaci podle vypočítaných korelací závisí trochu více výpočetní čas na počtu uzlů v grafu - hodnota korelace 0.48 - než na maximálním stupni uzlu v grafu - hodnota korelace 0.44

Porovnání závislostí

- pro implementaci ve Squanderu podle vypočítaných korelací závisí výpočetní čas více na maximálním stupni uzlu v grafu - hodnota korelace 0.91 - než na počtu uzlů v grafu - hodnota korelace 0.48
- pro imperativní implementaci podle vypočítaných korelací závisí trochu více výpočetní čas na počtu uzlů v grafu - hodnota korelace 0.48 - než na maximálním stupni uzlu v grafu - hodnota korelace 0.44

Zhodnocení (1)

- Squander je zajímavý nástroj pro programování problémů obsahujících mnoho závislostí
- JFSL poskytuje dostatečné množství výrazů pro implementaci algoritmů
- díky frameworku lze současně snadno využít výhod jak imperativního, tak deklarativního programování
- při programování občas nastává problém se scopem celočíselných proměnných

Zhodnocení (1)

- Squander je zajímavý nástroj pro programování problémů obsahujících mnoho závislostí
- JFSL poskytuje dostatečné množství výrazů pro implementaci algoritmů
- díky frameworku lze současně snadno využít výhod jak imperativního, tak deklarativního programování
- při programování občas nastává problém se scopem celočíselných proměnných

Zhodnocení (1)

- Squander je zajímavý nástroj pro programování problémů obsahujících mnoho závislostí
- JFSL poskytuje dostatečné množství výrazů pro implementaci algoritmů
- díky frameworku lze současně snadno využít výhod jak imperativního, tak deklarativního programování
- při programování občas nastává problém se scopem celočíselných proměnných

Zhodnocení (1)

- Squander je zajímavý nástroj pro programování problémů obsahujících mnoho závislostí
- JFSL poskytuje dostatečné množství výrazů pro implementaci algoritmů
- díky frameworku lze současně snadno využít výhod jak imperativního, tak deklarativního programování
- při programování občas nastává problém se scopem celočíselných proměnných

Zhodnocení (2)

- framework podporuje pouze first order logic
- při použití frameworku dochází samozřejmě k větší paměťové zátěži než při imperativním programování, nicméně zátěž na procesor je podobná
- výpočetní čas implementace ve frameworku je závislý na jiných parametrech instance než je tomu u imperativní implementace
- programování ve frameworku lze v praxi využít např. pro programování rozvrhů, kontroly studijního plánu a jiných problémů obsahujících velké množství závislostí

Zhodnocení (2)

- framework podporuje pouze first order logic
- při použití frameworku dochází samozřejmě k větší paměťové zátěži než při imperativním programování, nicméně zátěž na procesor je podobná
- výpočetní čas implementace ve frameworku je závislý na jiných parametrech instance než je tomu u imperativní implementace
- programování ve frameworku lze v praxi využít např. pro programování rozvrhů, kontroly studijního plánu a jiných problémů obsahujících velké množství závislostí

Zhodnocení (2)

- framework podporuje pouze first order logic
- při použití frameworku dochází samozřejmě k větší paměťové zátěži než při imperativním programování, nicméně zátěž na procesor je podobná
- výpočetní čas implementace ve frameworku je závislý na jiných parametrech instance než je tomu u imperativní implementace
- programování ve frameworku lze v praxi využít např. pro programování rozvrhů, kontroly studijního plánu a jiných problémů obsahujících velké množství závislostí

Zhodnocení (2)

- framework podporuje pouze first order logic
- při použití frameworku dochází samozřejmě k větší paměťové zátěži než při imperativním programování, nicméně zátěž na procesor je podobná
- výpočetní čas implementace ve frameworku je závislý na jiných parametrech instance než je tomu u imperativní implementace
- programování ve frameworku lze v praxi využít např. pro programování rozvrhů, kontroly studijního plánu a jiných problémů obsahujících velké množství závislostí