Překladač z jazyka Scheme do C/C++

Jan Novák **(novakjan@gmx.com)**

Vedoucí práce: RNDr. Jakub Yaghob, Ph.D., Univerzita Karlova v Praze, Matematicko-fyzikální fakulta, 2015

Motivace

Scheme je běžně používaný dynamicky typovaný jazyk, který obsahuje lambda výrazy, přiřazení do proměnných a kde mají hodnoty proměnných formálně neomezenou životnost. Tyto vlastnosti komplikují překlad do strojového kódu. Aby byl vygenerovaný výstup efektivní, je třeba provést globální analýzu celého programu. Tu provádí jen některé překladače, které bývají velmi komplexní a jejich algoritmy jsou složité a málo dokumentované. Cílem této práce je vytvořit nový překladač jazyka Scheme, který se jim bude snažit přiblížit.

Použité metody

Zjednodušení syntaxe. V prvotní implementaci je vhodné co nejvíce prvků jazyka přesunout do knihovny. Vstupní program transformujeme tak, že v něm zbydou jen lambda výrazy, volání funkcí, volání vestavěných funkcí a přiřazení do proměnných. Příklad převodu:

```
(if (>= a 0) a (- a))
```

```
(%if (>= a 0)
     (λ () a)
     (λ () (- a)))
```

```
(let ((epsilon 0.01))
  (f epsilon))
```

```
((λ (epsilon) (f epsilon))
 0.01)
```

Zjednodušení sémantiky. Stav běhu programu je obyčejně dán hodnotami proměnných a zásobníkem volání. V této práci převádíme program do tvaru zvaného continuation-passing-style. Tento tvar fixuje pořadí vyhodnocování. Programy pak neobsahují vnořená volání funkcí a stav programu je dán jen hodnotami proměnných. Příklad převodu:

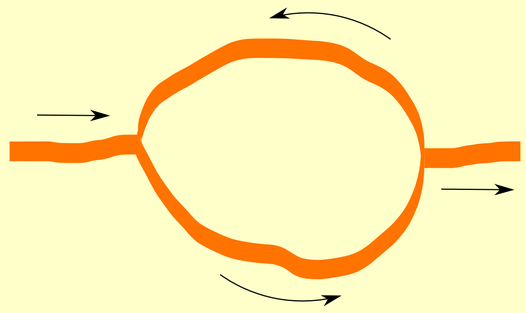
```
(λ (n)
  (if (zero? n)
      1
      (* n (fact (- n 1)))))

(λ (n return)
  (if (zero? n)
      (return 1)
      (fact (- n 1) (λ (m) (return (* m n))))))
```

Analýza toku řízení. V jazyce Scheme nejde graf toku řízení (CFG) přímo vyčíst ze zdrojového kódu. V této práci ho počítáme pomocí abstraktní interpretace.

Základem je varianta abstraktní interpretace zvaná 0-CFA [1]. Hodnota proměnné je zde sdílena mezi všemi instancemi lambda výrazu, kterého je ta proměnná parametrem.

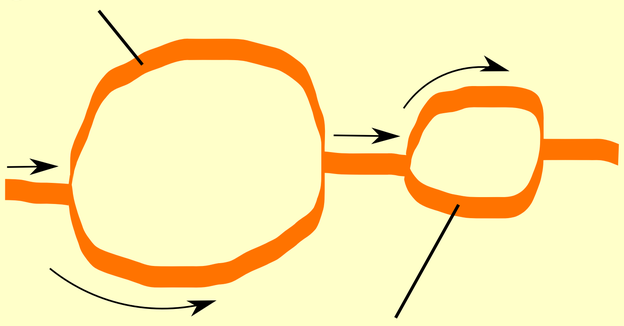
CFG pro výpočet faktoriálu iterativně



Všechny proměnné jsou globalizovatelné, protože v každé iteraci dostávají novou hodnotu a ta je používána pouze v jedné iteraci.

CFG pro výpočet faktoriálu rekurzivně

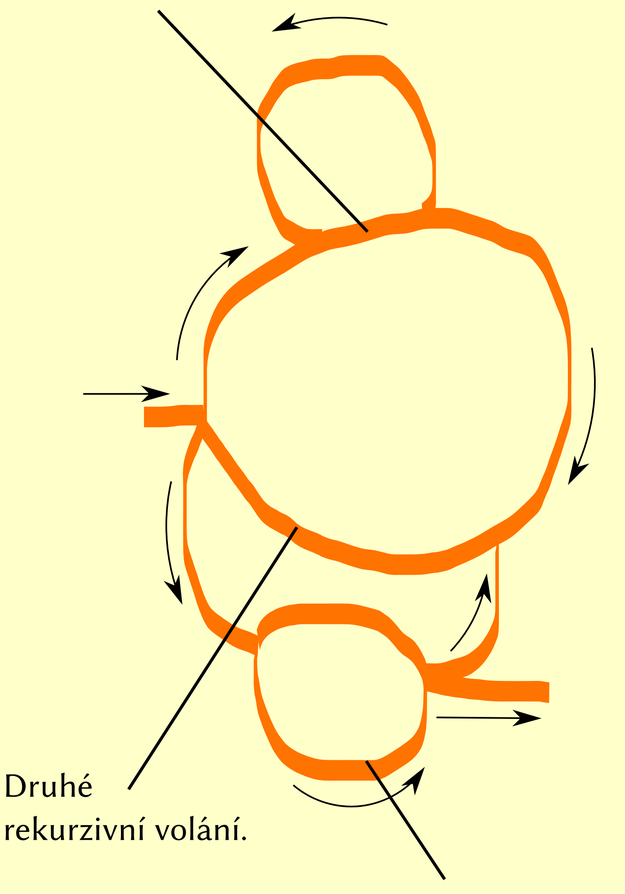
Zanořování do rekurze.
Lokální proměnné jsou zde globalizovatelné.



Vynořování z rekurze.
Argument volání zde není glovalizovatelný, ale jeho hodnoty jsou uloženy na zásobníku. S nimi tam je také jeden bit, zda vynořování skončilo.

CFG pro výpočet Fibonacciho čísel rekurzivně

První rekurzivní volání.



Druhé rekurzivní volání.

Sčítání podvýsledků.

Dále je použito rozšíření zvané Γ-CFA [2]. Při něm se používá garbage collector, který zahazuje abstraktní hodnoty nedosažitelných proměnných. Díky tomu algoritmus automaticky provádí integraci kódu funkcí, které jsou volány na různých místech v programu.

Globalizace proměnných. Procedura přistuje k proměnným přes prostředí, což je struktura, která obsahuje hodnoty volných proměnných lambda výrazu. Proměnná je globalizovatelná, pokud její hodnotu lze předat pomocí globálního místa v paměti. Lokální proměnné funkcí, globální proměnné a řídící proměnné v cyklech jsou většinou globalizovatelné.

Otypování proměnných. Proměnné se díky integraci kódu funkcí mohou vyskytovat na různých místech grafu toku řízení a přitom se navzájem nemusí ovlivňovat. V tomto případě překladač tyto proměnné přejmenuje a přiřadí jim typ, který určí z výsledku abstraktní interpretace.

Generování výstupního kódu. Překladač generuje výstupní kód v jazyce C++. Pro každý vrchol grafu toku řízení vygeneruje kód, který ve všech případech končí skokem na jeho vhodného následníka. Vygenerovaný kód používá vlastní zásobník volání. Veškerý vygenerovaný kód je uvnitř jedné funkce jazyka C++.

Výsledky

Díky použití garbage collectoru při abstraktní interpretaci jde docela dobře už při psaní kódu odhadnout, jak bude graf toku řízení vypadat. Dále je díky němu většina proměnných globalizovatelná, většina prostředí je eliminována a lambda výrazy jsou často jen značky, podle kterých se řídí běh programu. Implementace se zaměřuje na korektnost výstupního kódu. Je zatím pomalejší než výstup překladače Stalin [3] nebo obdobného programu v jazyce C++.

Literatura:

[1] SHIVERS, Olin. *Control-Flow Analysis of Higher-Order Languages*. Pittsburgh, 1991. School of Computer Science Carnegie MellonUniversity Pittsburgh. Ph.D. Thesis.

[2] MIGHT, Matthew. *Environment Analysis of Higher-Order Languages*. 2007. Georgia Institute of Technology. Ph.D. Thesis.

[3] SISKIND, Jeffrey Mark. *Flow-Directed Lightweight Closure Conversion*. Tech. Rep. 99-190R, NEC Research Institute. 1999.