COMPILER OF A LANGUAGE WITH USER-DEFINED SYNTAX FOR NEW CONSTRUCTS

Lukáš Kuklínek
Master Degree Programme (3), FIT BUT
E-mail: xkukli01@stud.fit.vutbr.cz

Supervised by: Dušan Kolář
E-mail: kolar@fit.vutbr.cz

Abstract: This project aims to design and implement a compiler of an experimental programming language featuring user-defined extensions to the syntax of the language. The language consists of a statically typed core and a mechanism to introduce new syntax. User-defined syntactic constructs preserve all the safety guarantees that are provided by the type system of the core language.

Keywords: compilers, programming languages, extensible syntax

1 ÚVOD

Vývoj programovacích jazyků míří stále k vyšším úrovním abstrakce. V oblasti syntaxe vznikají nové jazykové konstrukce usnadňující život programátora (*syntactic sugar*). Pro řešení specializovaných úloh jsou dostupné *doménově specifické jazyky*. Cílem tohoto projektu je poskytnout uživateli programovacího jazyka podporu pro definici vlastních syntaktických konstrukcí. Tato vlastnost může značně zúžit sémantickou propast mezi myšlením programátora či experta v dané oblasti a zápisem řešení problému v programovacím jazyce.

Nový deklarativní jazyk nazvaný EEL (Experimental Extensible Language) se skládá z minimalistického jádra, nad kterým je definována sémantika jazyka, a mechanismu pro rozšiřování syntaxe, který převádí uživatelem definované konstrukce na konstrukce jádra.

2 MINIMALISTICKÉ JÁDRO

Jádro jazyka EEL je zásobníkově orientovaný jazyk založený na kompozici funkcí, jejichž vstupem je zásobník a výstupem nový zásobník. Funkce vždy manipulují pouze s několika položkami na vrcholu zásobníku a jsou plně polymorfní vzhledem ke zbytku zásobníku. Výraz `f g` je ekvivalentní matematickému zápisu $g \circ f$ (až na případné vedlejší efekty). Jádro používá striktní strategii vyhodnocení výrazů. Na uvedený zápis lze tedy nahlížet také imperativně jako na sekvenční provedení podprogramů `f` a `g` v tomto pořadí. Vyhodnocení funkce lze odložit pomocí anonymní funkce (*quotation*), zapsané do hranatých závorek. Například funkce `sin` spočítá sinus číselného argumentu na vrcholu zásobníku, ale funkce `[sin]` vloží funkci `sin` na zásobník a ta může poté sloužit jako vstupní hodnota pro funkce vyššího řádu. Shrnutí syntaxe a datových typů jádra je uvedeno v tabulce 1.

Zásobníkový kód je jednou z klasických reprezentací mezikódu v překladačích [1]. Jeho hlavními výhodami jsou jednoduchost navázání gramatických pravidel syntaxe jazyka na vygenerovaný kód a absence nutnosti generovat pomocné proměnné. Jádro založené na tomto paradigmatu se jeví jako vhodný cílový jazyk pro překlad vysokoúrovňových uživatelských konstrukcí.

Jádro je vybaveno statickým typovým systémem ve stylu *Hindley-Milner* [2]. Polymorfismus je podporován skrze typové proměnné, které jsou implicitně univerzálně kvantifikovány. Rekurzivní typy

nejsou podporovány. Seznam je přidán jako vestavěný typ. Typy všech výrazů jsou odvozeny překladačem [3], uživatelské typové anotace ani definice nových typů nejsou podporovány.

P	::=	$D\ P \mid \varepsilon$	program	τ	::=	int | char | float	vestavěný typ
D	::=	*name* { F }	definice funkce		|	unit	jednotkový typ
F	::=	*name*	invokace funkce		|	$\tau + \tau$	součtový typ
	|	$F\ F$	kompozice funkcí		|	$\tau \times \tau$	součinový typ
	|	[F]	anonymní funkce		|	$\tau \to \tau$	typ funkce
	|	\$*name*[F]	lokální proměnná (`let`)		|	$[\tau]$	seznam
	|	ε	prázdná funkce (identita)		|	*var*	typová proměnná

Tabulka 1: Abstraktní syntaxe (vlevo) a přehled dostupných datových typů (vpravo) v jádře EEL.

Jádro taktéž disponuje několika vestavěnými funkcemi. Jedná se především o funkce pro manipulaci se zásobníkem, několik kombinátorů (zvláště operátor pevného bodu) a funkce pro manipulaci s vestavěnými datovými typy (za zmínku stojí zejména konstruktory a de-konstruktory složených datových typů). Tyto základní funkce jsou použity pro implementaci minimální knihovny dalších často používaných funkcí, která je dostupná automaticky při každém spuštění překladače.

3 ROZŠIŘITELNOST SYNTAXE

Rozšiřitelnost syntaxe v jazyce EEL spočívá v možnosti přidávat pravidla bezkontextové gramatiky definující lokální dialekt jazyka EEL. Součástí pravidel je i specifikace, jaké konstrukce jádra se danému pravidlu přiřadí. Syntaktická rozšíření jsou odvozené větné formy [2, s. 119]. Lze na ně nahlížet jako na zkrácený či jinak vhodnější zápis příslušné konstrukce v jazyce jádra.

Dialekty jazyka EEL poskytují stejné záruky typové bezpečnosti jako samotné jádro, neboť sémantiku ani typový systém jádra nelze uživatelsky modifikovat. Důležitou vlastností je také schopnost lokalizovat typové a jiné sémantické chyby ve zdrojovém kódu s rozšířenou syntaxí, ačkoli veškeré sémantické kontroly se provádí až na úrovni jádra.

D	::=	grammar { G }	syntaktické rozšíření
G	::=	$R\ G \mid \varepsilon$	seznam pravidel
R	::=	*name* $\to T$	pravidlo pro nonterminál *name*
T	::=	*name* T	nonterminál
	|	*'string'* T	terminál vyjádřený textovým řetězcem *string*
	|	[F] T	sémantická akce (F je definováno v tabulce 1)
	|	;	konec pravidla

Tabulka 2: Metajazyk pro rozšiřování syntaxe v jazyce EEL.

EEL poskytuje klauzuli `grammar` sloužící k rozšiřování syntaxe (viz tabulka 2). Tato klauzule obsahuje libovolný počet pravidel k přidání do gramatiky jazyka. Pravá strana každého pravidla se skládá z řetězce terminálů, neterminálů a sémantických akcí. Mezi terminály patří identifikátory funkcí a ad-hoc klíčová slova či speciální symboly zapsané v apostrofech. Jedním z neterminálů je F, reprezentující funkci sestavenou jako kompozici jiných funkcí, a dále neterminály definované uživatelem. Sémantická akce je funkce v jazyku jádra včetně případných uživatelských rozšíření k neterminálu F.

3.1 PŘÍKLAD: INTERPOLACE ŘETĚZCŮ

Interpolací řetězce v tomto kontextu rozumíme vložení výrazu přímo do řetězcového literálu a substituci hodnoty výrazu na dané místo v řetězci. Jedná se o mírné rozšíření přímo nad jádrem EEL.

V příkladu 1 je rozšíření použito. Funkce `dup` duplikuje položku na vrcholu zásobníku, `fmult` je násobení dvou čísel, `read_float` načítá ze std. vstupu, `ftostr` převede hodnotu typu *float* na řetězec (což je pouhý seznam hodnot typu *char*) a konečně `print` řetězce tiskne.

```
square { dup fmult }
main { read_float dup "Ctverec #(ftostr) je #(square ftostr)." print }
```

Příklad 1: Použití interpolace řetězců.

Příklad 2 ukazuje definici interpolace řetězců. Neterminál `str` představuje řetězcový literál bez uvozovek, `char` zpracovává a vrací jeden znak. Funkce `nil` vrací prázdný seznam, `cons` připojuje prvek na začátek seznamu. První tři řádky specifikují pravidla pro textové řetězce obecně, samotnou interpolaci implementuje poslední řádek. Identifikátor `func` představuje neterminál F, kombinátor `i` aplikuje funkci na aktuální zásobník a funkce `concat` spojuje dva seznamy.

```
grammar {
    string  -->  '"' str '"' ;          // textový literál
    str     -->  char str [ cons ] ;    // řetězec = znak : řetězec
    str     -->  [ nil ] ;              // konec řetězce
    str     -->  '#(' func ')' [ i ] str [ concat ] ;  // interpolace
}
```

Příklad 2: Implementace interpolace řetězců pomocí rozšiřitelné syntaxe.

4 IMPLEMENTAČNÍ DETAILY

Prototyp překladače jazyka EEL je napsán v jazyce *Haskell*[1]. Knihovna kombinátorů *Parsec*[2] je použita k dynamickému sestavení parseru na základě uživatelských gramatických pravidel. Jako cílová platforma je použita textová reprezentace mezikódu projektu *LLVM*[3], která je následně přeložena do nativní binární podoby. Automatickou dealokaci objektů obstarává garbage collector pro C/C++ *gc*[4].

5 ZÁVĚR

Samotný jazyk EEL je velmi experimentální a jeho překladač je stále pouhým prototypem. Jako takový má svá zásadní omezení a není příliš vhodný pro praktické programování ve velkém, avšak jako nástroj pro výzkum v oblasti rozšiřitelné syntaxe prokázal svou hodnotu. Ačkoli je zápis nových pravidel volně založen na bezkontextových gramatikách, bylo by vhodné i rozšiřování syntaxe jako takové postavit na formálním základě. Toto je ponecháno jako možné zaměření budoucího výzkumu.

REFERENCE

[1] Aho, Alfred V., Sethi, R., Ullman, Jeffrey D.: *Compilers: principles, techniques, and tools*. Addison-Wesley Longman Publishing Co., Inc., 1986. 1st edition. ISBN 978-0201100884.

[2] Pierce, Benjamin C.: *Types and Programming Languages*. The MIT Press, 2002. 1st edition. ISBN 0-262-16209-1.

[3] Diggins, C.: *Simple Type Inference for Higher-Order Stack-Oriented Languages*. Technická zpráva Cat-TR-2008-001. 2008.

[1] http://www.haskell.org/

[2] http://legacy.cs.uu.nl/daan/parsec.html

[3] http://llvm.org/

[4] http://www.hpl.hp.com/personal/Hans_Boehm/gc/