

EasyOCaml

Benus Becker

Universität Freiburg

28. Januar 2009

Ziele

- ▶ Fehlermeldungen verbessern
 - ▶ Parser
 - ▶ Typchecker
- ▶ Didaktische Hilfsmittel
 - ▶ Einschränkungen der Syntax
 - ▶ Bereitstellung von Code
 - ▶ Anpassbarkeit der Fehlerausgabe
- ▶ Integration in das existierende OCaml System
 - ▶ Nutzen der existierenden Codegenerierung
 - ▶ Zugriff auf vorhandene Programmbibliotheken

Motivation

```
let f b x = let y = if b then x in x + y ;;
```

Objective Caml

```
# let f b x = let y = if b then x in x + y  
;;  
Error: This expression has type unit but  
is here used with type int
```

EasyOCaml

- Type constructor clash between `int` and `unit`
- Type constructor clash between `int` and `unit`

- Type constructor clash between `int` and `unit`
- Type constructor clash between `int` and `unit`

```
let f b x = let y = if b then x in x + y let f b x = let y = if b then x in x + y
```

Verwandte Projekte

Haack & Wells (2004)

- ▶ Constraint-basiertes Typchecken von MiniML
- ▶ minimale, ausreichende Begründung der Fehler

Helium

- ▶ Haskell Implementation “für Anfänger”
- ▶ Hinweise zum Lösen der Typfehler

DrScheme

- ▶ Standard IDE für Programmierkurse in Scheme
- ▶ einfacher Debugger
- ▶ Sprachlevel und Teachpacks

Unterstützte Sprache

Caml_m: Caml ohne Moduldeklarationen

- ▶ Primitive: float, int, bool, string
- ▶ Tupel ▶ Listen ▶ Arrays
- ▶ Varianten, Records
- ▶ Deklarationen von ▶ (polymorphen) Typen ▶ Ausnahmen
▶ Werten
- ▶ Schachtelbare, auf alle möglichen Werte anwendbare Patterns
- ▶ Konstruktoren, Ausnahmebehandlung, Konditionale, Abstraktionen, Typannotationen.

Einschränkbar durch Sprachlevels

Alle Teile der stdlib ohne format typbar

Einbettung in OCaml

Ablauf von EasyOCaml

- ▶ Kommandozeilenparameter: -easy, -easyerrorprinter, -easylevel, -easyteachpack
- ▶ Modifikation des Parsers, Laden von Sprachlevels und Teachpacks
- ▶ Syntaxanalyse mit Camlp4
- ▶ Constraintbasierte Typrekonstruktion
- ▶ Zurückführung in den Compiler bzw. die REPL
bisher typst OCaml selbst erneut

Typchecker

Haack & Wells, 2004

OCaml `let f b x y = x + if b then y`
EasyOCaml `function x -> .. if .. then x .. (+) x`
fake!

W erzeugt und löst Constraints gleichzeitig: Keine Nachprüfung möglich
Trennung von Generierung und Lösung

- ▶ Ziele
 - ▶ möglichst viele Fehler anzeigen
 - ▶ Fehler sind minimal und vollständig
definieren
- ▶ Haack & Wells' Typchecker für MiniML
 1. Generierung von Constraints
 2. Lösen von Constraints
 3. Aufzählen der Fehler
 4. Minimierung der Fehler
- ▶ gleichzeitig mit Constraintgenerierung: Erkennung unbekannte Variablen

Typchecker

Erweiterungen für EasyOCaml

- ▶ Syntaktische Kategorien
 - Deklarationen Δ ; *strit* $\Downarrow_s \langle \Delta', C, u \rangle$
 - Ausdrücke Δ ; *lexp* $\Downarrow_e \langle ty, C, u \rangle$
 - Pattern Δ ; *pat* $\Downarrow_p \langle ty, C, b \rangle$
- ▶ Typen: Varianten, Records
Beispiel

$$\begin{array}{c} \text{RECORD ACCESS} \\ \Delta; \text{lexp} \Downarrow_e \langle ty, C, u \rangle \quad \Delta|_{\text{rec}}(f) = \langle ty_f, ty, \cdot \rangle \\ \hline C_0 = \{a \stackrel{!}{=} ty_f, ty \stackrel{!}{=} ty_r\} \quad a \text{ fresh} \\ \hline \Delta; (\text{lexp}.f)^I \Downarrow_e \langle a, C_0 \cup C, u \rangle \end{array}$$

mehr mögliche Fehler

Typannotationen

$(lexp : ct)$

- Typinferenz für $lexp$ und Kontext unabhängig
- nachträgliche Prüfung der Validität der Annotation

TYPE-ANNOT

$$\frac{\Delta; lexp \Downarrow_e \langle ty, C_0, u \rangle \quad C_1 = \{a \stackrel{!}{=} ty', ty \not\approx^! ct\} \quad a \text{ fresh} \quad ty' \text{ is a fresh instance of } ct}{\Delta; (lexp : ct)^! \Downarrow_e \langle a, C_0 \cup C_1, u \rangle}$$

Annotation wird als korrekt angenommen

Fehlerbehandlung

Drei Klassen von Fehlern

Einfach Meldung nach Typrekonstruktion:
Typfehler, unbekannte Variablen

Schwer Meldung nach Constraintgenerierung:
ungültige Varianten-, Recordkonstruktion,
Variablenbindungen in Patterns
Analyse der Validität von Varianten, Records, Typkonstruktionen

Fatal Sofortige Meldung: Syntaxfehler, unbekannte Module

Anpassung der Fehlermeldungen

- Lokalisierung der Fehlerbeschreibung (Umgebungsvariable)
- Formatierung für unterschiedliche Ausgaben (Plugin)

Funktionen werden registriert in dynamisch geladenem Code

► textbasiert ► HTML ► XML

Sprachlevel und Teachpacks

- Definition und einfache Auslieferung verfügbarer Sprachkonstrukte und vordefiniertem Codes

um die Sprache an Übungen und Inhalt von Kursen anzupassen

- Interface
 - (De-)Aktivierung von Optionen für die nicht-terminale Deklaration, Ausdruck, Pattern im Parser
 - Liste von (zu öffnenden) Modulen

Beispiel Sprachlevel

lang-minimal/mylist.ml

```
let (+) = Pervasives.(+)
let (-) = Pervasives.(-)
let ( * ) = Pervasives.( * )
let (/) = Pervasives.(/)

let succ = Pervasives.succ
let pred = Pervasives.pred
```

lang-minimal/mylist.ml

```
let cons h t = h :: t
let iter = List.iter
let map = List.map
let fold = List.fold_left
```

lang-minimal/LANG.META.ml

```
let _ =
  let match_pf =
    let full_pf = all_pattern_feats true in
    { full_pf with p_nested = false } in
  let prf =
    { pr_expr_feats =
      { (all_expr_feats true) with
        e_try = None;
        e_match = Some match_pf;
        e_raise = false } ;
      pr_struct_feats = all_struct_feats true } in
  EzyLangLevel.configure prf
  ["Mypervasives", true; "Mylist", true]
  ["mypervasives.cmo"; "mylist.cmo"]
```

Änderungen am Parser

EasyOCaml parst mit Camlp4 Parser

- + veränderbar zur Laufzeit
- hart kodierte Fehlermeldung
 - ▶ Streamparser erlauben nur Zeichenketten als Information





Lösung

- ▶ Variantentyp beschreibt Fehler:
 - ▶ Ungültiger Anfang ▶ Erwartung ... ▶ Sprachspezifisch
- ▶ interne Struktur der Zeichenkette: "<msg>\000<msh1>"
- ▶ Wiederherstellung des "gemarkshalten" Fehlers in der Interfacefunktion

Bisherige und weitere Entwicklung

- ✓ Typchecker für Teilsprache von OCaml
- ✓ internationalisierte und anpassbare Fehlermeldungen
- ✓ Hilfsmittel für die Lehre mit OCaml
- ✓ Integration in original Compiler und REPL
- ✓ anpassbare Fehlermeldungen für den Parser
- ✓ HTML/XML/sexp Fehlerausgaben
- ✓ Portierung auf OCaml 3.11
 - ▶ Integration in DrOCaml oder Camelia
 - ▶ Heuristiken/Tips zum Lösen von Fehlern
 - ▶ Fehlermeldungen mit mehr Informationen versehen
 - ▶ selbstdokumentierende Sprachlevel und Teachpacks
 - ▶ dynamisch getypter Interpreter

Quellenangaben

-  Felleisen, M., R. B. Findler, M. Flatt, and S. Krishnamurthi (1998).
The DrScheme Project: An Overview.
SIGPLAN Notices 33, 17–23.
-  Haack, C. and J. B. Wells (2003).
Type Error Slicing in Implicitly Typed, Higher-order Languages.
In *Sci. Comput. Programming*, pp. 284–301. Springer-Verlag.
-  Heeren, B., D. Leijen, and A. van IJzendoorn (2003).
Helium, for Learning Haskell.
In *ACM Sigplan 2003 Haskell Workshop*, New York, pp. 62 – 71. ACM Press.
-  Leroy, X., D. Doligez, J. Garrigue, D. Rémy, and J. Vouillon (2008).
The Objective Caml System Release 3.11.
<http://caml.inria.fr/pub/docs/manual-ocaml/index.html>.