EasyOCaml

Benus Becker

Universität Freiburg

28. Januar 2009

Motivation

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

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
```

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

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

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**

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

Ziele

- Fehlermeldungen verbessern
  - Parser
  - Typchecker
- didaktische Hilfsmittel (Sprachlevels und Teachpacks)
  - Einschränkungen der Syntax
  - Bereitstellung von Code
  - Anpassbarkeit der Fehlerausgabe
- Integration in das existierende OCaml System
  - Nutzen der existierenden Codegenerierung
  - Zugriff auf vorhandene Programmbibliotheken

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.

Ablauf von EasyOCaml

- Kommandozeilenparameter: `-easy`, `-easyerrorprinter`, `-easylevel`, `-easyteachpack`
- Modifikation des Parsers, Laden von Sprachlevels und Teachpacks
- Syntaxanalyse mit Camlp4
- Constraint-basierte Typinferenz
- Zurückführung in den Compiler bzw. die REPL

Typchecker

Haack & Wells, 2004

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

EasyOCaml `function x -> .. if .. then x .. (+) x ..`

- Ziele
  - möglichst viele Fehler anzeigen
  - Fehler sind minimal und vollständig

Typchecker

Haack & Wells, 2004

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

EasyOCaml `function x -> .. if .. then x .. (+) x ..`

- Ziele
  - möglichst viele Fehler anzeigen
  - Fehler sind minimal und vollständig
- Haack & Wells' Typchecker für MiniML
  1. Generierung von Constraints
  2. Lösen von Contraints
  3. Aufzählen der Fehler
  4. Minimierung der Fehler

Typchecker

Haack & Wells, 2004

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

EasyOCaml `function x -> .. if .. then x .. (+) x ..`

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

Typchecker

Erweiterungen für EasyOCaml

- Syntaktische Kategorien

  Deklarationen $\Delta;\ strit \Downarrow_s \langle \Delta',\ C,\ u \rangle$

  Ausdrücke $\Delta;\ lexp \Downarrow_e \langle ty,\ C,\ u \rangle$

  Pattern $\Delta;\ pat \Downarrow_p \langle ty,\ C,\ b \rangle$

- Typen: Varianten, Records

  Beispiel

$$\textsc{Record Access}\quad \frac{\Delta;\ lexp \Downarrow_e \langle ty,\ C,\ u \rangle \qquad \Delta|_{rec}(f) = \langle ty_f,\ ty_r,\ \cdot \rangle \qquad C_0 = \{a \overset{l}{=} ty_f,\ ty \overset{l}{=} ty_r\} \qquad a \text{ fresh}}{\Delta;\ (lexp.f)^l \Downarrow_e \langle a,\ C_0 \cup C,\ u \rangle}$$

Typannotationen

$(lexp : ct)$

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

TYPE-ANNOT

$$\frac{\Delta;\ lexp \Downarrow_e \langle ty,\ C_0,\ u\rangle \qquad C_1 = \{a \overset{l}{=} ty',\ ty \succcurlyeq^l ct\} \qquad a \text{ fresh} \qquad ty' \text{ is a fresh instance of } ct}{\Delta;\ (lexp : ct)^l \Downarrow_e \langle a,\ C_0 \cup C_1,\ u\rangle}$$

Fehlerbehandlung

Drei Klassen von Fehlern

Einfach Meldung nach Typrekonstruktion:
Typfehler, unbekannte Variablen

Schwer Meldung nach Constraintgenerierung:
ungültige Varianten-, Recordkonstruktion,
Variablenbindungen in Patterns

Fatal Sofortige Meldung: Syntaxfehler, unbekannte Module

Anpassung der Fehlermeldungen

- Lokalisierung der Fehlerbeschreibung (Umgebungsvariable)
- Formatierung für unterschiedliche Ausgaben (Plugin)
  - textbasiert • HTML • XML

Sprachlevel und Teachpacks

- Definition und einfache Auslieferung verfügbarer Sprachkonstrukte und vordefiniertem Codes
- Interface
  - (De-)Aktivierung von Optionen für die nicht-terminale Deklaration, Ausdruck, Pattern im Parser
  - Liste von (zu öffnenden) Modulen

Beispiel Sprachlevel

lang-fun/mypervasives.ml

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

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

lang-fun/mylist.ml

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

lang-fun/LANG_META.ml

```
open EzyLangLevel
let _ =
  let pr_f = {
    pr_expr_feats = {
      (all_expr_feats true) with
        e_reference_update = false; (* forbid [x := e]
        e_record_field_update = false;  (* forbid [x.f
        e_if_then = false;              (* forbid [if e
        e_sequence = false;             (* forbid [e;f]
    };
    pr_struct_feats = {
      (all_struct_feats true) with
          (* make annotations for global variables mand
          s_annot_optional = false;
          s_type = Some {
            (all_type_feats true) with
                (* forbid declaration of mutable record
                t_record = Some false } } } in
  configure pr_f [
    "Mylist", true;
    "Mypervasives", true;
  ] ["mylist.cmo"; "mypervasives.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 • Enttäuchte Erwartung • sprachspezifischer/künstlicher Fehler
- interne Struktur der Zeichenkette: `"<msg>\000<mshl>"`
- Wiederherstellung des “gemarshalten” Fehlers in der Interfacefunktion

Bisherige und weitere Entwicklung

- ✓ Typchecker für Teilsprache von OCaml
- ✓ Hilfsmittel für die Lehre mit OCaml
- ✓ internationalisierte und anpassbare Fehlermeldungen
- ✓ erweitertes Fehlersystem für Camlp4
- ✓ HTML/XML/sexp Fehlerausgaben
- ✓ Integration in original Compiler und REPL
- ✓ Portierung auf OCaml 3.11

Bisherige und weitere Entwicklung

- ✓ Typchecker für Teilsprache von OCaml
- ✓ Hilfsmittel für die Lehre mit OCaml
- ✓ internationalisierte und anpassbare Fehlermeldungen
- ✓ erweitertes Fehlersystem für Camlp4
- ✓ HTML/XML/sexp Fehlerausgaben
- ✓ Integration in original Compiler und REPL
- ✓ Portierung auf OCaml 3.11
- Integration in DrOCaml oder Camelia
- Heuristiken/Tipps 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.