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
  - Einschränkungen der Syntax
  - Bereitstellung von Code
- 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
- Language Levels und Teach Packs

Unterstützte Sprache

Caml$_{-m}$: Caml ohne Moduldeklarationen

Werte Primitive, Tupel, Listen, Arrays, Varianten, Records, Funktionen, Ausnahmen.

Structure items Typ-, Ausnahmen-, Wertdeklarationen und Expressions.

Pattern Geschachtelt auf alle möglichen Werte.

Ausdrücke Konstruktoren, Ausnahmebehandlung, Konditionale, Abstraktionen, Typannotationen.

Einbettung in OCaml

Ablauf von EasyOCaml

- Kommandozeilenparameter: `-easy`, `-easyerrorprinter`, `-easylevel`, `-easyteachpack`
- Modifikation des Parsers, Laden von Teach Packs
- Syntaxanalyse mit Camlp4
- Constraintbasierte Typrekonstruktion
- Zurückführung in den Compiler bzw. die REPL

Typchecker

Haack & Wells, 2004

```
# let f b x y = x + if b then y ;;
* Type error:  Type constructor clash between int and
unit.  Slice:  ..  + if ..  then ..  else () ..
```

- Ziele
  - mehrere Fehler auf einmal berichten
  - Fehler sind minimal und vollständig

Typchecker

Haack & Wells, 2004

```
# let f b x y = x + if b then y ;;
* Type error:  Type constructor clash between int and
unit.  Slice:  ..  + if ..  then ..  else () ..
```

- Ziele
  - mehrere Fehler auf einmal berichten
  - 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

```
# let f b x y = x + if b then y ;;
* Type error:  Type constructor clash between int and
unit.  Slice:  ..  + if ..  then ..  else () ..
```

- Ziele
  - mehrere Fehler auf einmal berichten
  - 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: Unbekannte Variablen erkennen

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

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

Typannotationen

$(lexp : ct)$

- $lexp$ und Kontext werden unabhängig getypt
- Validität der Annotation wird nachträglich geprüft

TYPE-ANNOT

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

Beispiel: Generierung von Typconstraint

```
x + if b then y
```

Fehlerbehandlung

Drei Klassen von Fehlern

- Einfach: Meldung nach Typrekonstruktion
- Schwer: Meldung nach Constraintgenerierung
- Fatal: Sofortige Meldung

Anpassung der Fehlermeldungen

- Internationalisierung der Fehlerbeschreibung
- Formatierung für unterschiedliche Ausgaben (Plugin)

Language Levels und Teach Packs

- Definition und einfache Auslieferung verfügbarer Sprachkonstrukte und vordefiniertem Codes
- Interface

  Sprachspezifikation

  - Deaktivierung von Optionen für die nicht-terminale Deklaration, Ausdruck, Pattern im Parser

  $\Rightarrow$ direkte Manipulation des Parsers

  - für jede Benutzung unabhängig

  Vordefinierter Code Liste von (zu öffnenden) Modulen

Änderungen am Parser

EasyOCaml parst mit Camlp4 Parser

- hart kodierte Beschreibung
- Streamparser erlauben nur Zeichenketten als Information

  $\Rightarrow$ `exception Stream.Error of string`

Lösung (?)

- interne Struktur der Zeichenkette: `"<msg>\000<mshl>"`
- Wiederherstellung des “gemarshalten” 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 Fehlerausgaben
- ✓ Portierung auf OCaml 3.11

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 Fehlerausgaben
- ✓ Portierung auf OCaml 3.11
- Integration in DrOCaml oder Camelia
- Heuristiken/Tips zum Lösen von Fehlern
- Fehlermeldungen mit mehr Informationen versehen
- selbstdokumentierende Language Levels und Teach Packs
- dynamisch getypter Interpreter

Fazit

- Zweigeteilt: Arbeit am Typchecker – Integration in OCaml
- moderne Forschungsergebnisse (Haack & Wells) anwenden
- OCaml verbessern und open source zu arbeiten

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.