Generalized Algebraic Data Types

in OCaml

Marius Weidner – Lehrstuhl für Programmiersprachen – Proseminar '22

Sprachdefinition

$$
\begin{aligned}
\mathit{Expr} ::= \ & \texttt{true} \\
\mid \ & \texttt{false} \\
\mid \ & -^{?}\, 0..9^{+} \\
\mid \ & \textbf{if } \mathit{Expr} \textbf{ then } \mathit{Expr} \textbf{ else } \mathit{Expr}
\end{aligned}
$$

Gültige Programme

```
true
```

```
if true then 42 else 0
```

```
if true then
  if true then 42 else 0
else
  0
```

Ungültige Programme

```
if true then 0
```

```
if 0 then true else false
```

```
if true then
  if true then 42 else 0
else
  false
```

In einer Welt ohne GADTs

```
type expr =
  | Bool of bool
  | Int  of int
  | If   of expr * expr * expr
```

Syntaxbaum ohne GADTs

```
let rec eval : expr → expr = function
  | If (c, t, e) → begin match eval c with
      | Bool true  → eval t
      | Bool false → eval e
      | _ → failwith "need bool!"
    end
  | e → e
```

Evaluationsfunktion ohne GADTs

```
eval (If
      (Bool true,
      (Int 42),
      (Int 0)))
```

```
- : expr = Int 42
```

```
eval (If
      (Int 42,
      (Bool false),
      (Int 0)))
```

```
Exception: Failure "need bool!"
```

Beispiele ohne GADTs

- Ungültige Programmdefinitionen
- Laufzeitfehler im Interpreter
- Zweige mit verschiedenen Typen

In einer Welt mit GADTs

```
type _ expr =
  | Bool : bool → bool expr
  | Int  : int  → int  expr
  | If   : bool expr * 'a expr * 'a expr → 'a expr
```

Syntaxbaum mit GADTs

```
let rec eval : .. = function
  | Bool b          → b
  | Int i           → i
  | If (c, t, e)    → if eval c then eval t else eval e
```

Evaluationsfunktion mit GADTs

```
eval (If
      (Int 42,
      (Bool true),
      (Bool false)))
```

```
eval (If
      (Bool true,
      (Bool false),
      (Int 42)))
```

```
Error: Type int is not compatible with type bool
```

Beispiele mit GADTs

- Nur gültige Programmdefinitionen
- Keine Laufzeitfehler im Interpreter
- Exhaustive Patternmatching

Locally Abstract Types

```
let rec eval (type a) (e : a expr) : a = match e with
  | Bool b → b
  | Int i  → i
  | If (c, t, e)  → if eval c then eval t else eval e
```

```
Error: This expression has type a expr but an
       expression was expected of type bool expr
```

Evaluationsfunktion mit GADTs

Polymorphic Recursion

```
let rec eval : 'a. 'a expr → 'a =
    fun (type a) (e : a expr) : a → match e with
      | Bool b → b
      | Int i  → i
      | If (c, t, e)  → if eval c then eval t else eval e
```

Evaluationsfunktion mit GADTs

Syntactic Sugar

```
let rec eval : type a. a expr → a = function
  | Bool b → b
  | Int i  → i
  | If (c, t, e)  → if eval c then eval t else eval e
```

Evaluationsfunktion mit GADTs

Existential Types

```
type stringable =
  Stringable : {
    item: 'a;
    to_string: 'a → string
  } → stringable
```

```
let print (Stringable s) =
  print_endline (s.to_string s.item)
```

Vertiefendes Beispiel

```
type (_, _) mode =
  | Unsafe : ('a, 'a) mode
  | Option : ('a, 'a option) mode
```

```
let first : type a r. a list → (a, r) mode → r =
  fun lst mde → match lst with
    | hd :: tl → (match mde with
      | Unsafe → hd
      | Option → Some hd)
    | [ ] → (match mde with
      | Unsafe → failwith "list is empty"
      | Option → None)
```

Limitationen

- Typinferenz unentscheidbar
  → Typannotationen notwendig
- `|` -Patterns nicht auflösbar
  → Manuell auflösen und Logik auslagern
- `[@@-deriving ..]` -Annotation nicht möglich
  → Non-GADT Version benötigt

Zusammengefasst

- GADTs erlauben Konstrukturen verschiedene Typparameter einzusetzen
- Stärkere Aussagen auf Typebene möglich
- Patternmatching nutzt die zusätzlichen Informationen
- GATDs erlauben existenziell quantifizierte Typen zu formen
- Typinferenz wird unentscheidbar

Folien & Code

github.com/Mari-W/ocaml-gadts

Literatur

- Real World OCaml: GADTs
  Yaron Minsky, Anil Madhavapeddy `2021`
- Detecting use-cases for GADTs in OCaml
  Mads Hartmann `2015`
- Stanford CS242: Programming Languages
  Will Crichton `2019`