Type-based Nondeterminism Checking in Functional Logic Programs

Frank Steiner

CAU Kiel

Joint work with

Michael Hanus

CAU Kiel

Problem Definition

Functional logic languages:

- nondeterministic search facilities $\rightsquigarrow$ problems with (non-backtrackable) I/O
- solution: encapsulated search [PLILP/ALP'98]
- problem: detecting nondeterminism (ND)

Program analysis:

- develope analysis for detecting ND
- Type and Effect Systems (provide for powerful analyses)

Our target:

- to detect **every** possible ND

Benefits:

- increasing program stability
- compiler optimization

Curry: A Truly Integrated Functional Logic Language

[Dagstuhl'96, POPL'97]

- multi-paradigm language, combines
  - functional programming
  - logic programming
  - concurrent programming
- based on an optimal evaluation strategy (residuation + narrowing)
- conservative extension of lazy functional and (concurrent) logic programming
- nondeterministic search facilities (narrowing), encapsulated search
- monadic I/O
- equational constraints, external functions and constraint solvers
- polymorphic type system, modules
- communication and distributed programming

Evaluation in Curry

Search for solutions with needed narrowing: [POPL'94]

- lazily reduce expressions to their values
- evaluate subterms as demanded by lhs of rules
- bind free variables in goals if neccessary

$\Rightarrow$ compute answer pairs $\{\sigma\}\ e$

```
sq 1 = 1            sq x = fcase x of 1:1;
sq 2 = 4                              2:4
```

sq 2 $\overset{*}{\leadsto}$ 4

sq x $\overset{*}{\leadsto}$ {x=1} 1 | {x=2} 4

$\Rightarrow$ Nondeterminism causes splitting!

Monadic I/O = transformations on the outside world

$\Rightarrow$ splitting not allowed!

What should be the output behaviour for

`writeFile "dummy" (sq x)`?

Program Analysis

Compile-time technique for predicting safe and computable approximations to the run-time behaviour of a program

Our target: Identify functions/expressions that *might* cause nondeterministic reductions.

Type and Effect Systems

- extend type system with effect φ:
  $f :: \tau_1 \xrightarrow{\varphi} \tau_2 \qquad (\varphi = \text{effect of the application of } f)$
- analysis defines *type judgements*:
  $E \vdash e :: \tau / \varphi \qquad (\varphi = \text{effect of reducing } e)$
- example: analyses of
  - side effects,
  - exception behaviour,
  - communication behaviour

ND Analysis

P = a program, $\mathcal{F}$ = set of function symbols in P,
$\mathcal{C}$ = set of constructor symbols in P

- **Effect:** $\varphi \subseteq \mathcal{F} \cup \{case, or\}$
- **Types:**
  - ND in Curry caused by narrowing $\Rightarrow$ distinguish on the kind of arguments

    $$\begin{array}{lrcll} \leadsto & \tau & = & G & \textbf{G}\text{roundterm} \\ & & \mid & A & \textbf{A}\text{ny term} \\ & & \mid & \tau_1 \ldots \tau_n \to \tau & \end{array}$$

    with $G \leq A$ (subtyping)
  - $$\left.\begin{array}{l} f :: G \stackrel{\emptyset}{\to} G \\ f :: A \stackrel{\{f\}}{\to} G \end{array}\right\} \textit{several types}$$
- **Type environment** E contains
  - $f :: \tau_1 \ldots \tau_n \stackrel{\varphi}{\to} \tau$ with $f \in \mathcal{F} \cup \mathcal{C}$
  - $x :: \tau$ with $\tau = G \mid A$

safe approximations $\Rightarrow$ effect possibly too large, type imprecise

ND Analysis

First step: Typing a program P, i.e., finding a *correct* type environment for P.

Definition:
E is a correct type environment for a Curry program P iff each rule in P is correctly typed w.r.t. all matching type annotations in E.

Second step: Typing an expression w.r.t. the program.

Theorem (Correctness of typing rules):
Let E be a correct type environment for P, e an expression and $E \vdash e :: \tau/\varphi$.

1. If $\tau = G$ then e reduces to a ground term.
2. If $\varphi = \emptyset$ then e reduces in a deterministic way. (Constructive method: If $\varphi \neq \emptyset$ then φ contains all sources of nondeterminism.)

Example 1

Typing rules for program rules and (function/constructor) variables:

$$\text{DECL} \quad \frac{E[x_1 :: \tau_1, \ldots, x_n :: \tau_n] \vdash e :: \tau / \emptyset}{E \vdash f\ x_1 \ldots x_n \ = \ e} \quad \text{if } f :: \tau_1 \ldots \tau_n \xrightarrow{\psi} \tau \in E$$

$$\text{DECL} \quad \frac{E[x_1 :: \tau_1, \ldots, x_n :: \tau_n] \vdash e :: \tau / \varphi}{E \vdash f\ x_1 \ldots x_n \ = \ e} \quad \begin{array}{l} \text{if } \varphi \neq \emptyset \text{ and} \\ f :: \tau_1 \ldots \tau_n \xrightarrow{\psi \cup \varphi \cup \{f\}} \tau \in E \end{array}$$

$$\text{VAR} \quad \frac{}{E \vdash x :: \tau / \emptyset} \quad \text{if } x :: \tau \in E$$

∞

Example 1

Program:

```
f :: A -∅-> G
f x = 0
```

Initial environment: $E = \{\texttt{f} :: A \xrightarrow{\emptyset} G,\ \texttt{0} :: G\}$

Analysis:

$$\dfrac{\dfrac{\texttt{0} :: G \in E[\texttt{x} :: A]}{E[\texttt{x} :: A] \vdash \texttt{0} :: G/\emptyset}\ \text{VAR} \qquad \texttt{f} :: A \xrightarrow{\emptyset} G \in E}{E \vdash \texttt{f}\ \ \texttt{x} = \texttt{0}}\ \text{DECL}$$

Example 2

Typing rules for case expressions (without pattern variables):

$$\text{FCASE}\ \frac{E \vdash x :: G/\emptyset \qquad E \vdash e_1 :: \tau_1/\varphi_1 \ \ \ldots \ \ E \vdash e_n :: \tau_n/\varphi_n}{E \vdash \ \texttt{fcase}\ x\ \texttt{of}\ p_1 : e_1; \ldots p_n : e_n \ :: \ max_i(\tau_i)/\bigcup_i \varphi_i}$$

$$\text{FCASE}\ \frac{E \vdash x :: A/\emptyset \qquad E \vdash e_1 :: \tau_1/\varphi_1 \ \ \ldots \ \ E \vdash e_n :: \tau_n/\varphi_n}{E \vdash \ \texttt{fcase}\ x\ \texttt{of}\ p_1 : e_1; \ldots p_n : e_n \ :: \ max_i(\tau_i)/\bigcup_i \varphi_i \cup \{case\}}$$

Example 2

Program:

$$\texttt{sq} :: A \xrightarrow{\{\texttt{sq}, case_{\texttt{sq}}\}} G$$

```
sq x = fcase x of 1:1; 2:4
```

Initial environment: $E = \{\texttt{sq} :: A \xrightarrow{\{\texttt{sq}, case_{\texttt{sq}}\}} G,\ \texttt{1/2/4} :: G\}$

Analysis:

$$\dfrac{\begin{array}{c}\dfrac{\dfrac{\texttt{x} :: A \in E[\texttt{x} :: A]}{E[\texttt{x} :: A] \vdash \texttt{x} :: A/\emptyset}\,\text{VAR} \qquad \dfrac{\texttt{1} :: G \in E[\texttt{x} :: A]}{E[\texttt{x} :: A] \vdash \texttt{1} :: G/\emptyset}\,\text{VAR}}{E[\texttt{x} :: A] \vdash \texttt{fcase x of 1:1;2:4} :: G/\{case_{\texttt{sq}}\}}\,\text{FCASE} \\ \texttt{sq} :: A \xrightarrow{\{\texttt{sq}, case_{\texttt{sq}}\}} G \in E\end{array}}{E \vdash \texttt{sq x} = \texttt{fcase x of 1:1; 2:4}}\,\text{DECL}$$

Example 3

$$\text{APP}\ \frac{E \vdash e_1 :: \tau_1/\varphi_1 \quad \ldots \quad E \vdash e_n :: \tau_n/\varphi_n \qquad E \vdash f :: \tau_1 \ldots \tau_n \xrightarrow{\varphi} \tau/\emptyset}{E \vdash f\ e_1 \ldots e_n\ ::\ \tau/\bigcup_i \varphi_i \cup \varphi}$$

Expression: `sq (sq x)`

Initial environment: $E = \{\mathtt{x} :: A,\ \mathtt{sq} :: A \xrightarrow{\{\mathtt{sq}, case_{\mathtt{sq}}\}} G,\ \mathtt{sq} :: G \xrightarrow{\emptyset} G\}$

Analysis:

$$\frac{\dfrac{\overline{E \vdash \mathtt{x} :: A/\emptyset} \qquad \overline{E \vdash \mathtt{sq} :: A \xrightarrow{\{\mathtt{sq}, case_{\mathtt{sq}}\}} G/\emptyset}}{E \vdash \mathtt{sq}\ \ \mathtt{x} :: G/\{\mathtt{sq}, case_{\mathtt{sq}}\}} \qquad \overline{E \vdash \mathtt{sq} :: G \xrightarrow{\emptyset} G/\emptyset}}{E \vdash \mathtt{sq}\ \ (\mathtt{sq}\ \ \mathtt{x}) :: G/\{\mathtt{sq}, case_{\mathtt{sq}}\}}$$

Conclusions and Future Work

- nondeterministic search facilities (narrowing) of functional logic languages
  $\leadsto$ conflicts with monadic I/O
- Type and Effect System for detecting ND
- type checker implemented: good results

Future work:

- type inference algorithm
- extension to special language features
- refinement of the type domain

Subtyping and Subeffecting

$\texttt{sq :: } A \xrightarrow{\{\texttt{sq}, fcase_{\texttt{sq}}\}} G$

```
sq x = fcase x of 1:2; 2:4
```

Additionally, correct types for `sq` are:

- $\texttt{sq} :: A \xrightarrow{\{\texttt{sq}, fcase_{\texttt{sq}}\}} A$ (subtyping)
- $\texttt{sq} :: A \xrightarrow{\{\texttt{sq}, fcase_{\texttt{sq}}, \texttt{g}\}} G$ (subeffecting)
- $\texttt{sq} :: G \xrightarrow{\emptyset} G$