Derivation Rules in Fialyzer

fialyzer developers

This file shows derivation rules used in fialyzer.

Our derivation rules are almost same as the original success typings paper's one, but extended by remote call, local call, list, etc.

1. Derivation Rules

Here are the BNFs used in the derivation rules:

$$
\begin{array}{lcl}
e & ::= & v \mid x \mid fn \mid \{e,\cdots,e\} \mid \text{let } x = e \text{ in } e \mid \text{letrec } x = fn,\cdots,x = fn \text{ in } e \\
 & \mid & e(e,\cdots,e) \mid \text{case } e \text{ of } pg \to e;\cdots;pg \to e \text{ end} \mid \text{fun } f/a \mid \text{fun } m:f/a \quad \text{(term)} \\
v & ::= & 0 \mid \text{'ok'} \mid \cdots \quad \text{(constant)} \\
x & ::= & \text{(snip)} \quad \text{(variable)} \\
fn & ::= & \text{fun}(x,\cdots,x) \to e \quad \text{(function)} \\
pg & ::= & p \text{ when } g;\cdots;g \quad \text{(pattern with guard sequence)} \\
p & ::= & v \mid x \mid \{p,\cdots,p\} \quad \text{(pattern)} \\
g & ::= & v \mid x \mid \{e,\cdots,e\} \mid e(e,\cdots,e) \quad \text{(guard)} \\
m & ::= & e \quad \text{(module name. a term to be an atom)} \\
f & ::= & e \quad \text{(function name. a term to be an atom)} \\
a & ::= & e \quad \text{(arity. a term to be a non\_neg\_integer)} \\
\tau & ::= & \text{none()} \mid \text{any()} \mid \alpha \mid \{\tau,\cdots,\tau\} \mid (\tau,\cdots,\tau) \to \tau \mid \tau \cup \tau \\
 & \mid & \text{integer()} \mid \text{atom()} \mid 42 \mid \text{'ok'} \mid \cdots \quad \text{(type)} \\
\alpha,\beta & ::= & \text{(snip)} \quad \text{(type variable)} \\
C & ::= & (\tau \subseteq \tau) \mid (C \wedge \cdots \wedge C) \mid (C \vee \cdots \vee C) \quad \text{(constraint)} \\
A & ::= & A \cup A \mid \{x \mapsto \tau,\cdots,x \mapsto \tau\} \quad \text{(context. mapping of variable to type)}
\end{array}
$$

Here are the derivation rules:

$$\frac{}{A \cup \{x \mapsto \tau\} \vdash x : \tau, \emptyset}\text{(VAR)}$$

$$\frac{A \vdash e_1 : \tau_1, C_1 \qquad \cdots \qquad A \vdash e_n : \tau_n, C_n}{A \vdash \{e_1,\cdots,e_n\} : \{\tau_1,\cdots,\tau_n\}, C_1 \wedge \cdots \wedge C_n}\text{(STRUCT)}$$

$$\frac{A \vdash e_1 : \tau_1, C_1 \qquad A \cup \{x \mapsto \tau_1\} \vdash e_2 : \tau_2, C_2}{A \vdash \text{let } x = e_1 \text{ in } e_2 : \tau_2, C_1 \wedge C_2}\text{(LET)}$$

$$\frac{A' \vdash fn_1 : \tau_1, C_1 \cdots A' \vdash fn_n : \tau_n, C_n \qquad A' \vdash e : \tau, C \qquad \text{where } A' = A \cup \{x_i \mapsto \alpha_i\}}{A \vdash \text{letrec } x_1 = f_1, \cdots, x_n = f_n \text{ in } e : \tau, C_1 \wedge \cdots \wedge C_n \wedge C \wedge (\tau_1' = \tau_1) \wedge \cdots \wedge (\tau_n' = \tau_n)}(\text{LETREC})$$

$$\frac{A \cup \{x_1 \mapsto \alpha_1, \cdots, x_n \mapsto \alpha_n\} \vdash e : \tau, C}{A \vdash \text{fun}(x_1, \cdots, x_n) \to e : (\alpha_1, \cdots, \alpha_n) \to \tau, C}(\text{ABS})$$

$$\frac{A \vdash e : \tau, C \qquad A \vdash e_1 : \tau_1, C_1 \cdots A \vdash e_n : \tau_n, C_n}{A \vdash e(e_1, \cdots, e_n) : \beta, (\tau = (\alpha_1, \cdots, \alpha_n) \to \alpha) \wedge (\beta \subseteq \alpha) \wedge (\tau_1 \subseteq \alpha_1) \wedge \cdots \wedge (\tau_n \subseteq \alpha_n) \wedge C \wedge C_1 \wedge \cdots \wedge C_n}(\text{APP})$$

$$\frac{A \vdash p : \tau, C_p \qquad A \vdash g : \tau_g, C_g}{A \vdash p \text{ when } g : \tau, (\tau \subseteq \text{boolean}()) \wedge C_p \wedge C_g}(\text{PAT})$$

$$\frac{A \vdash e : \tau, C_e \qquad A_i \vdash pg_i : \tau_{pg_i}, C_{pg_i} \qquad A_i \vdash b_i : \tau_{b_i}, C_{b_i} \qquad \text{where } A_i = A \cup \{v \mapsto \alpha_v \mid v \in Var(pg_i)\}}{A \vdash \text{case } e \text{ of } pg_1 \to b_1; \cdots pg_n \to b_n \text{end} : \beta, C_e \wedge (C_1 \vee \cdots \vee C_n) \text{where } C_i = ((\beta = \tau_{b_i}) \wedge (\tau = \tau_{pg_i}) \wedge C_{pg_i} \wedge C_{b_i})}(\text{CASE})$$

$$\frac{}{A \cup \{\text{fun } f/a \mapsto \tau\} \vdash \text{fun } f/a : \tau, \emptyset}(\text{LOCALFUN})$$

$$\frac{}{A \cup \{\text{fun } m : f/a \mapsto \tau\} \vdash \text{fun } m : f/a : \tau, \emptyset}\text{if } m \text{ and } f \text{ is atom literal, } a \text{ is non\_neg\_integer literal } (\text{MFA})$$

$$\frac{A \vdash m : \tau_m, C_m \qquad A \vdash f : \tau_f, C_f \qquad A \vdash a : \tau_a, C_a}{A \vdash \text{fun } m : f/a : \beta, (\tau_m \subseteq \text{atom}()) \wedge (\tau_f \subseteq \text{atom}()) \wedge (\tau_a \subseteq \text{number}()) \wedge C_m \wedge C_f \wedge C_a}\text{if } \diamond \ (\text{MFAEXPR})$$

$\diamond$: neither m, f is atom literal nor a is non_neg_integer literal

1.1. Differences from the original paper

The differences from the derivation rules on the original paper are as follows.

- α, β, and τ are clearly distinguished. τ is a type, and α, β are type variables.
- LET is fixed: e_2, not e.
- ABS is modified: τ and constrained function are omitted.
- PAT is modified: type of g is boolean(), not true.
- CASE is fixed: τ, not τ_i. replaced $p_1 \cdots p_n$ with $pg_1 \cdots pg_n$ because these are patterns with guards.
- LOCALFUN is added.
- MFA is added.
- MFAEXPR is added.
- ...and some variables are α-converted for understandability.

1.2. Notes

- In $A \vdash p : \tau, C_p$ of PAT rule, p is not an expression but a pattern. Therefore, we have to convert p to an expression which is the same form of p.
    - This is not described in the original paper.