Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

Spreading the Joy

Making “Stricterness” More Relevant

Stefan Holdermans
(Joint work with Jurriaan Hage)

Dept. of Information and Computing Sciences, Utrecht University
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands
E-mail: `stefan@cs.uu.nl`
Web pages: `http://people.cs.uu.nl/stefan/`

Software Technology Colloquium
June 18, 2009

The need for strictness analysis

Advantages of lazy evaluation: infinite data structures, custom control structures, avoiding unnecessary computations, program optimisations, ...
(Hughes 1989, ...).

Huge disadvantage: inefficiency.

Strictness analysis: identify as many function applications as possible that can be safely evaluated eagerly rather than lazily.

☞ Safely: without changing the meaning of a program.

Universiteit Utrecht

[Faculty of Science
Information and Computing Sciences]

Limitations of strictness analysis

- Strictness analyses are necessarily conservative: if a function cannot be guaranteed to be strict, it is treated as nonstrict. (“Err on the safe side.”)
- Moreover: many functions are “nearly” strict, but not quite. Strictness analysers have to classify them as nonstrict.

Countermeasure: lazy languages give the programmer a means to selectively make functions stricter.

Strictness annotations

Haskell provides a primitive function

$$seq :: \alpha \to \beta \to \beta$$

that first forces its first argument to weak-head normal form and then returns its second argument.

Making functions stricter

Compare

$$const :: \alpha \to \beta \to \alpha$$
$$const\ x\ y = x$$

with

$$const' :: \alpha \to \beta \to \alpha$$
$$const'\ x\ y = y\ `seq`\ x$$

☞ While $const$ is strict only in its first argument (and lazy in its second), $const'$ is strict in both its arguments.

Propagating stricterness

Of course, stricterness propagates:

$$force :: \alpha \rightarrow ()$$
$$force\ x = const'\ ()\ x$$

☞ $force$ is strict, because $const'$ is.

Strict application

With seq, we can define a custom operator for strict function application:

$$(\$!) :: (\alpha \to \beta) \to \alpha \to \beta$$
$$f\ \$!\ x = x\ `seq`\ f\ x$$

For example:

$$const\ ()\ \quad \bot \quad \Downarrow \quad ()$$
$$const\ ()\ \$!\ \bot \quad \Downarrow \quad \bot$$

Semantic peculiarities

Using seq, we can tell $\bot$ and $(\lambda x \to \bot)$ apart:

$$\begin{array}{lll} \bot \; \text{`}seq\text{`} \; () & \Downarrow & \bot \\ (\lambda x \to \bot) \; \text{`}seq\text{`} \; () & \Downarrow & () \end{array}$$

Dealing with strictness annotations

When reasoning about programs and implementing compiler optimisations, one has to be aware of the semantic implications of having seq in the language:

- Parametricity does not hold.
- Fold-build fusion is invalid.
- ...

- See Danielsson et al. (2006), Van Eekelen and De Mol (2006), Johann and Voigtländer (2006), ...

What about strictness analysis?

Outline

- Relevant Typing
- Naïve Refinements
- Our Approach

Relevant Typing

Relevant typing

- Strictness analysis by means of a nonstandard (annotated) type system.
- Type-based approach to keeping track of neededness (Barendregt et al. 1987).
- Neededness (intensional) used to approximate strictness (extensional).
- Through a Curry-Howard lens: connection with (substructural) relevant logic.

- See Wright (1991), Baker-Finch (1992), Amtoft (1993), Benton (1996), ...

Typing rules

$$\frac{}{\Gamma \vdash n :: \mathsf{Int}}\ [\mathit{const}]$$

$$\frac{}{\Gamma \vdash \bot :: \tau}\ [\mathit{bot}]$$

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x :: \tau}\ [\mathit{var}]$$

$$\frac{\Gamma +\!\!+ [x \mapsto \tau_1] \vdash t_1 :: \tau_2}{\Gamma \vdash \lambda x \to t_1 :: \tau_1 \to \tau_2}\ [\mathit{lam}]$$

$$\frac{\Gamma \vdash t_1 :: \tau_2 \to \tau \quad \Gamma \vdash t_2 :: \tau_2}{\Gamma \vdash t_1\ t_2 :: \tau}\ [\mathit{app}]$$

Careful context management

$$\frac{}{[\,] \vdash n :: \mathsf{Int}}\ [\mathit{const}]$$

$$\frac{}{[\,] \vdash \bot :: \tau}\ [\mathit{bot}]$$

$$\frac{}{[x \mapsto \tau] \vdash x :: \tau}\ [\mathit{var}]$$

Substructural typing rule

Weakening:

$$\frac{\Gamma_1 + \Gamma_2 \vdash t :: \tau}{\Gamma_1 + [x \mapsto \tau_0] + \Gamma_2 \vdash t :: \tau}\ [\textit{weak}]$$

Annotated types

Decorate function types with an annotation $\varphi \in \{S, L\}$:

- $\tau_1 \xrightarrow{S} \tau_2$ for relevant (strict) abstractions.
- $\tau_1 \xrightarrow{L} \tau_2$ for uncommitted (lazy) abstractions.

☞ S and L constitute a two-point lattice with $S \sqsubset L$.

Relevance typing: constants and variables

$$\frac{}{[\,] \vdash n :: \mathsf{Int}^{\varphi}}\ [\mathit{const}]$$

$$\frac{}{[\,] \vdash \bot :: \tau^{\varphi}}\ [\mathit{bot}]$$

$$\frac{}{[x \mapsto \tau^{\varphi}] \vdash x :: \tau^{\varphi}}\ [\mathit{var}]$$

Relevance typing: functions

Function bodies are analysed as if functions are always needed:

$$\frac{\varphi \triangleright \Gamma \quad \Gamma +\!\!+ [x \mapsto {\tau_1}^{\varphi_1}] \vdash t_1 :: {\tau_2}^{\mathsf{S}}}{\Gamma \vdash \lambda x \to t_1 :: (\tau_1 \xrightarrow{\varphi_1} \tau_2)^{\varphi}} \; [\mathit{lam}]$$

Containment constraint: $\varphi \triangleright \Gamma$ iff $\forall (x \mapsto {\tau_0}^{\varphi_0}) \in \Gamma .\ \varphi \sqsubseteq \varphi_0$.

☞ Recall: $\mathsf{S} \sqsubset \mathsf{L}$.

Universiteit Utrecht

[Faculty of Science
Information and Computing Sciences]

Relevance typing: applications

In an application, a variable is needed if it is needed in either the function or the argument (or in both):

$$\frac{\Gamma_1 \vdash t_1 :: (\tau_2 \xrightarrow{\varphi_1} \tau)^{\varphi} \quad \Gamma_2 \vdash t_2 :: {\tau_2}^{\varphi \sqcup \varphi_1}}{\Gamma_1 \sqcap \Gamma_2 \vdash t_1\ t_2 :: \tau^{\varphi}}\ [\mathit{app}]$$

Least upper bound: $\varphi \sqcup \varphi_1 = \mathsf{S}$ iff $\varphi = \varphi_1 = \mathsf{S}$.

Context splitting: $\Gamma = \Gamma_1 \sqcap \Gamma_2$ iff Γ is the pointwise meet of Γ_1 and Γ_2.

Relevance typing: substructural rule

Only L-annotated bindings can be discarded:

$$\frac{\Gamma_1 +\!\!+ \Gamma_2 \vdash t :: \tau^{\varphi}}{\Gamma_1 +\!\!+ [x \mapsto \tau_0{}^{\mathsf{L}}] +\!\!+ \Gamma_2 \vdash t :: \tau^{\varphi}} \text{ [\textit{weak}]}$$

Call-by-value transformation

If $t_1 :: \tau_2 \xrightarrow{S} \tau$ and $t_2 :: \tau_2$, then $t_1\ t_2$ is transformed into $t_1\ \$!\ t_2$.

Correctness: if t is transformed into t' and $t \Downarrow v$, then $t' \Downarrow v'$ with $v' \leqslant_{\$!} v$.

☞ In particular: if $v \not\equiv \bot$, then $v' \not\equiv \bot$.

Universiteit Utrecht

[Faculty of Science
Information and Computing Sciences]

What about strictness annotations?

If we want to deal with strictness annotations in source programs, we have to give relevant typing rules for seq.

Or—take $\$!$ as a primitive and derive seq as a library function:

$$seq :: \alpha \to \beta \to \beta$$
$$seq\ x = const\ id\ \$!\ x$$

☞ Objective: sound and effective analysis in the presence of strict application.

Universiteit Utrecht

[Faculty of Science
Information and Computing Sciences]

Naïve Refinements

A simple rule for strict application

It is tempting to define:

$$\frac{\Gamma_1 \vdash t_1 :: (\tau_2 \xrightarrow{\varphi_1} \tau)^{\varphi} \quad \Gamma_2 \vdash t_2 :: \tau_2{}^{\varphi}}{\Gamma_1 \sqcap \Gamma_2 \vdash t_1 \ \$! \ t_2 :: \tau^{\varphi}} \ [\textit{strict-app}]$$

☞ Here, we discard the relevance φ_1 of the function.

Problem

$$f\ x = const\ ()\ \$!\ (\backslash\_ \to x)$$

Note: f is lazy in its argument x, i.e., $f\ \bot \Downarrow ()$.

Still:

- The body of f is analysed as if it is needed.
- The argument $(\backslash\_ \to x)$ of the strict application is analysed as if it is needed (i.e., the laziness of $const\ ()$ is discarded).
- The containment constraint for $(\backslash\_ \to x)$ is satisfied trivially.
- The body of $(\backslash\_ \to x)$ is typed as if it is needed.
- x is needed and, hence, $f :: \alpha \xrightarrow{S} ()$!!

☞ **But then the resulting transformation is unsound:** $f\ \$!\ \bot \Downarrow \bot$.

A less ambitious rule

$$\frac{\Gamma_1 \vdash t_1 :: (\tau_2 \xrightarrow{\varphi_1} \tau)^{\varphi} \quad \Gamma_2 \vdash t_2 :: \tau_2{}^{\varphi_1 \sqcup \varphi}}{\Gamma_1 \sqcap \Gamma_2 \vdash t_1 \,\$!\, t_2 :: \tau^{\varphi}} \; [\textit{strict-app}]$$

☞ Here, we type strict application as lazy application.

But then stricterness does not propagate and both

$$const'\ x\ y = const\ x\ \$!\ y$$

and

$$force\ x = const'\ ()\ x$$

are typed as if they were lazy.

Our Approach

Relevant typing: hidden assumption

Without seq (or $\$!$), the only way to force a function to weak-head normal form is by applying it to an argument.

Hence, there is no essential difference between $\bot$ and $\lambda x \rightarrow \bot$.

This shows in the containment constraint: if a function is needed, the variables that are needed in its body are needed as well.

But with seq, a function can be forced without being applied!

Keeping track of applicativeness

Main idea:

In addition to neededness, we also keep track of which terms are guaranteed to be used as functions, i.e., applied to arguments.

We reuse the lattice $\{\mathsf{S}, \mathsf{L}\}$ with $\mathsf{S} \sqsubset \mathsf{L}$:

- S for applicative terms.
- L for remaining terms.

Metavariable convention: φ for neededness and ψ for applicativeness.

Typing judgements now read: $\Gamma \vdash t : \tau^{(\varphi,\psi)}$.

Refined relevance typing: constants and variables

$$\frac{}{[\,] \vdash n :: \mathsf{Int}^{(\varphi,\mathsf{L})}}\ [const]$$

$$\frac{}{[\,] \vdash \bot :: \tau^{(\varphi,\psi)}}\ [bot]$$

$$\frac{}{[x \mapsto \tau^{(\varphi,\psi)}] \vdash x :: \tau^{(\varphi,\psi)}}\ [var]$$

Refined relevance typing: functions

$$\frac{\psi \triangleright \Gamma \quad \Gamma +\!\!\!+ [x \mapsto {\tau_1}^{(\varphi_1,\psi_1)}] \vdash t_1 :: {\tau_2}^{(\mathsf{S},\psi_2)}}{\Gamma \vdash \lambda x \to t_1 :: ({\tau_1}^{\psi_1} \xrightarrow{\varphi_1} {\tau_2}^{\psi_2})^{(\varphi,\psi)}}\ [\mathit{lam}]$$

The containment constraint is now dominated by the applicativeness of the function rather than its neededness.

☞ Applicativeness implies neededness: $\varphi \sqsubseteq \psi$.

Refined relevance typing: applications

$$\frac{\Gamma_1 \vdash t_1 :: ({\tau_2}^{\psi_2} \xrightarrow{\varphi_1} \tau^{\psi})^{(\varphi,\varphi)} \qquad \Gamma_2 \vdash t_2 :: {\tau_2}^{(\varphi \sqcup \varphi_1, \varphi \sqcup \psi_2)}}{\Gamma_1 \sqcap \Gamma_2 \vdash t_1\ t_2 :: \tau^{(\varphi,\psi)}}\ [\mathit{app}]$$

Applicativeness now participates in the pointwise meet $\Gamma_1 \sqcap \Gamma_2$ as well.

$$\frac{\Gamma_1 \vdash t_1 :: ({\tau_2}^{\psi_2} \xrightarrow{\varphi_1} \tau^{\psi})^{(\varphi,\varphi)} \qquad \Gamma_2 \vdash t_2 :: {\tau_2}^{(\varphi, \varphi \sqcup \psi_2)}}{\Gamma_1 \sqcap \Gamma_2 \vdash t_1\ \$!\ t_2 :: \tau^{(\varphi,\psi)}}\ [\mathit{strict\text{-}app}]$$

☞ The relevenance φ_1 of the function is completely discarded.

Refined relevance typing: substructural rule

Only (L, L)-annotated bindings can be discarded:

$$\frac{\Gamma_1 \mathbin{+\!\!+} \Gamma_2 \vdash t :: \tau^{(\varphi,\psi)}}{\Gamma_1 \mathbin{+\!\!+} [x \mapsto {\tau_0}^{(\mathsf{L},\mathsf{L})}] \mathbin{+\!\!+} \Gamma_2 \vdash t :: \tau^{(\varphi,\psi)}}\ [\textit{weak}]$$

Conclusions

- Adapting a relevant type system to have it take into account strictness annotations is a tricky business.
- Naïve approaches are easily unsound or ineffective.
- Incorporating a notion of applicativeness yields a solution that is both sound and effective.

- Future work: combine neededness and applicativeness into a single three-point lattice.