GADTs Meet Their Match:

Pattern-Matching Warnings That Account for GADTs, Guards, and Laziness

SEBASTIAN GRAF, Karlsruhe Institute of Technology, Germany
SIMON PEYTON JONES, Microsoft Research, UK

Authors' addresses: Sebastian Graf, Karlsruhe Institute of Technology, Karlsruhe, Germany, sebastian.graf@kit.edu; Simon Peyton Jones, Microsoft Research, Cambridge, UK, simonpj@microsoft.com.

1 END TO END EXAMPLE

We'll start from the following source Haskell program and see how each of the steps (translation to guard trees, checking guard trees and ultimately generating inhabitants of the occurring Δs) work.

```
f :: Maybe Int -> Int
f Nothing = 0 -- RHS 1
f x | Just y <- x = y -- RHS 2
```

1.1 Translation to guard trees

The program (by a function we probably only give in the appendix?) corresponds to the following guard tree $t_\mathtt{f}$:

$$\mathtt{Guard}\ (!x)\ \mathtt{Guard}\ (\mathtt{Nothing} \leftarrow x)\ \mathtt{Rhs}\ 1;$$
$$\mathtt{Guard}\ (!x)\ \mathtt{Guard}\ (\mathtt{Just}\ y \leftarrow x)\ \mathtt{Rhs}\ 2$$

Data constructor matches are strict, so we add a bang for each match.

1.2 Checking

1.2.1 Uncovered values. First compute the uncovered Δs, after the first and the second clause respectively.

(1)

$$\begin{aligned}\Delta_1 &:= \mathcal{U}(\mathtt{Guard}\ (!x)\ \mathtt{Guard}\ (\mathtt{Nothing} \leftarrow x)\ \mathtt{Rhs}\ 1)\\ &= x \not\approx \bot \wedge (x \not\approx \mathtt{Nothing} \vee \times)\end{aligned}$$

(2)

$$\Delta_2 := \mathcal{U}(t_\mathtt{f}) = \Delta_1 \wedge x \not\approx \bot \wedge (x \not\approx \mathtt{Just} \vee \times)$$

Note how Δ_1 gets duplicated in Δ_2. The right operands of $\vee$ are vacuous, but the purely syntactical transformation doesn't see that. Hence it makes sense for the implementation to do work on Δ_1 prior to duplicating it, so that the same work doesn't have to be performed twice (or exponentially often). In practice, this works by converting to ∇ eagerly. It's quite similar to the situation with call-by-name (where we might need to "evaluate" Δ_1 multiple times) vs. call-by-value (where we evaluate once up front).

1.2.2 Redundancy. We'll just give the four Δs that we need to generate the inhabitants for (as part of computing $\mathcal{A}_\Gamma(\Delta, t)$): One for each bang (for knowing whether we need to wrap a `MayDiverge` and one for each RHS (where we have to decide for `InaccessibleRhs` or `AccessibleRhs`).

(1) The first divergence check: $\Delta_3 := \checkmark \wedge x \approx \bot$
(2) Upon reaching the first RHS: $\Delta_4 := \checkmark \wedge x \not\approx \bot \wedge \mathtt{Nothing} \leftarrow x$
(3) The second divergence check: $\Delta_5 := \Delta_1 \wedge x \approx \bot$
(4) Upon reaching the second RHS: $\Delta_6 := \Delta_1 \wedge x \not\approx \bot \wedge \mathtt{Just}\ y \leftarrow x$

The missing equations and the annotated tree then depend on the inhabitants of these Δs, i.e. on the result of $\mathcal{G}(x : \mathtt{Maybe\ Int}, \Delta_i)$.

1.3 Generating inhabitants

Let's start with $\mathcal{G}(\Gamma, \Delta_3)$, where $\Gamma = x : \mathtt{Maybe\ Int}$.

We immediately have $C(\Gamma \triangleright \varnothing, \Delta_3)$ as a sub-goal. The first constraint $\checkmark$ is added very easily to the initial ∇ by discarding it, the second one ($x \approx \bot$) is not conflicting with any $x \not\approx \bot$ constraint in the incoming $\nabla\ \varnothing$, so we end up with $\Gamma \triangleright x \approx \bot$ as proof that Δ_3 is in fact inhabited. Indeed, $\mathcal{E}(\Gamma \triangleright x \approx \bot, x)$ generate _ as the inhabitant (which is rather unhelpful, but correct).

The result of $\mathcal{G}(\Gamma, \Delta_3)$ is thus {_}, which is not empty. Thus, $\mathcal{A}_\Gamma(\Delta, t)$ will wrap a `MayDiverge` around the first RHS.

Similarly, $\mathcal{G}(\Gamma, \Delta_4)$ needs $C(\Gamma \triangleright \varnothing, \Delta_4)$, which in turn will add $x \not\approx \bot$ to an initially empty ∇. That entails an inhabitance check to see if x might take on any values besides $\bot$.

This is one possible derivation of the $\Gamma \triangleright x \not\approx \bot \vdash x$ predicate:

$$\frac{\begin{array}{c}x : \texttt{Maybe Int} \in \Gamma \quad \texttt{Nothing} \in \mathsf{Cons}(\Gamma \triangleright x \not\approx \bot, \texttt{Maybe Int}) \\ \mathsf{Inst}(\Gamma, x, \texttt{Nothing}) = \texttt{Nothing} \leftarrow x \\ (\Gamma \triangleright x \not\approx \bot \oplus \texttt{Nothing} \leftarrow x) \neq \bot\end{array}}{\Gamma \triangleright x \not\approx \bot \vdash x}$$

The subgoal $\Gamma \triangleright x \not\approx \bot \oplus \texttt{Nothing} \leftarrow x$ is handled by the second case of the match on constructor pattern constraints, because there are no other constructor pattern constraints yet in the incoming ∇. Since there are no type constraints carried by `Nothing`, no fields and no constraints of the form $x \not\approx K$ in ∇, we end up with $\Gamma \triangleright x \not\approx \bot, \texttt{Nothing} \leftarrow x$. Which is not $\bot$, thus we conclude our proof of $\Gamma \triangleright x \not\approx \bot \vdash x$.

Next, we have to add $\texttt{Nothing} \leftarrow x$ to our $\nabla = x \not\approx \bot$, which amounts to computing $\Gamma \triangleright x \not\approx \bot \oplus \texttt{Nothing} \leftarrow x$. Conveniently, we just did that! So the result of $C(\Gamma \triangleright \varnothing, \Delta_4)$ is $\Gamma \triangleright x \not\approx \bot, \texttt{Nothing} \leftarrow x$.

Now, we see that $\mathcal{E}(\Gamma \triangleright (x \not\approx \bot, \texttt{Nothing} \leftarrow x), x) = \{\texttt{Nothing}\}$, which is also the result of $\mathcal{G}(\Gamma, \Delta_4)$.

The checks for Δ_5 and Δ_6 are quite similar, only that we start from $C(\Gamma \triangleright \varnothing, \Delta_1)$ (which occur syntactically in Δ_5 and Δ_6) as the initial ∇. So, we first compute that.

Fast forward to computing $\Gamma \triangleright x \not\approx \bot \oplus x \not\approx \texttt{Nothing}$. Ultimately, this entails a proof of $\Gamma \triangleright x \not\approx \bot, x \not\approx \texttt{Nothing} \vdash x$, for which we need to instantiate the `Just` constructor:

$$\frac{\begin{array}{c}x : \texttt{Maybe Int} \in \Gamma \quad \texttt{Just} \in \mathsf{Cons}(\Gamma \triangleright (x \not\approx \bot, x \not\approx \texttt{Nothing}), \texttt{Maybe Int}) \\ \mathsf{Inst}(\Gamma, x, \texttt{Just}) = \texttt{Just } y \leftarrow x \\ (\Gamma, y : \texttt{Int} \triangleright (x \not\approx \bot, x \not\approx \texttt{Nothing}) \oplus \texttt{Just } y \leftarrow x) \neq \bot\end{array}}{\Gamma \triangleright x \not\approx \bot, x \not\approx \texttt{Nothing} \vdash x}$$

$\Gamma, y : \texttt{Int} \triangleright (x \not\approx \bot, x \not\approx \texttt{Nothing}) \oplus \texttt{Just } y \leftarrow x)$ is in fact not $\bot$, which is enough to conclude $\Gamma \triangleright x \not\approx \bot, x \not\approx \texttt{Nothing} \vdash x$.

The second operand of $\vee$ in Δ_1 is similar, but ultimately ends in $\times$, so will never produce a ∇, so $C(\Gamma \triangleright \varnothing, \Delta_1) = \Gamma \triangleright x \not\approx \bot, x \not\approx \texttt{Nothing}$.

$C(\Gamma \triangleright \varnothing, \Delta_5)$ will then just add $x \approx \bot$ to that ∇, which immediately refutes with $x \not\approx \bot$. So no `MayDiverge` around the second RHS.

$C(\Gamma \triangleright \varnothing, \Delta_6)$ is very similar to the situation with Δ_4, just with more (non-conflicting) constraints in the incoming ∇ and with $\texttt{Just } y \leftarrow x$ instead of $\texttt{Nothing} \leftarrow x$. Thus, $\mathcal{G}(\Gamma, \Delta_6) = \{\texttt{Just \_}\}$.

The last bit concerns $\mathcal{G}(\Gamma, \Delta_2)$, which is empty because we ultimately would add $x \not\approx \texttt{Just}$ to the inert set $x \not\approx \bot, x \not\approx \texttt{Nothing}$, which refutes by the second case of $\_ \oplus \_$. (The $\vee$ operand with $\times$ in it is empty, as usual).

So we have $\mathcal{G}(\Gamma, \Delta_2) = \emptyset$ and the pattern-match is exhaustive.

The result of $\mathcal{A}_\Gamma(\Gamma, t)$ is thus `MayDiverge AccessibleRhs 1; AccessibleRhs 2`.

Guard Syntax

$$
\begin{array}{rlll}
K \in & \text{Con} & & \\
x, y, a, b \in & \text{Var} & & \\
\tau, \sigma \in & \text{Type} & & \\
e \in & \text{Expr} & ::= & x : \tau \\
& & \mid & K\ \overline{\tau}\ \overline{\gamma}\ \overline{e : \tau} \\
& & \mid & \ldots
\end{array}
\qquad
\begin{array}{rlll}
n \in & \mathbb{N} & & \\
\gamma \in & \text{TyCt} & ::= & \tau_1 \sim \tau_2 \mid \ldots \\
p \in & \text{Pat} & ::= & \_ \\
& & \mid & K\ \overline{p} \\
& & \mid & \ldots \\
g \in & \text{Grd} & ::= & \mathsf{let}\ x : \tau = e \\
& & \mid & K\ \overline{a}\ \overline{\gamma}\ \overline{y : \tau} \leftarrow x \\
& & \mid & !x
\end{array}
$$

Constraint Formula Syntax

$$
\begin{array}{rcll}
\Gamma & ::= & \varnothing \mid \Gamma, x : \tau \mid \Gamma, a & \text{Context} \\
\delta & ::= & \checkmark \mid \times \mid K\ \overline{a}\ \overline{\gamma}\ \overline{y : \tau} \leftarrow x \mid x \not\approx K \mid x \approx \bot \mid x \not\approx \bot \mid x \approx e & \text{Constraint Literals} \\
\Delta & ::= & \delta \mid \Delta \wedge \Delta \mid \Delta \vee \Delta & \text{Formula} \\
\nabla & ::= & \varnothing \mid \nabla, \delta & \text{Inert Set}
\end{array}
$$

Clause Tree Syntax

$$
\begin{array}{rcl}
t_G, u_G \in \text{Gdt} & ::= & \mathtt{Rhs}\ n \mid t_G; u_G \mid \mathtt{Guard}\ g\ t_G \\
t_A, u_A \in \text{Ant} & ::= & \mathtt{AccessibleRhs}\ n \mid \mathtt{InaccessibleRhs}\ n \mid t_A; u_A \mid \mathtt{MayDiverge}\ t_A
\end{array}
$$

Checking Guard Trees

$$\boxed{\mathcal{U}(t_G) = \Delta}$$

$$
\begin{array}{lcl}
\mathcal{U}(\mathtt{Rhs}\ n) & = & \times \\
\mathcal{U}(t; u) & = & \mathcal{U}(t) \wedge \mathcal{U}(u) \\
\mathcal{U}(\mathtt{Guard}\ (!x)\ t) & = & (x \not\approx \bot) \wedge \mathcal{U}(t) \\
\mathcal{U}(\mathtt{Guard}\ (\mathsf{let}\ x = e)\ t) & = & (x \approx e) \wedge \mathcal{U}(t) \\
\mathcal{U}(\mathtt{Guard}\ (K\ \overline{a}\ \overline{\gamma}\ \overline{y : \tau} \leftarrow x)\ t) & = & (x \not\approx K) \vee ((K\ \overline{a}\ \overline{\gamma}\ \overline{y : \tau} \leftarrow x) \wedge \mathcal{U}(gs))
\end{array}
$$

$$\boxed{\mathcal{A}_\Gamma(\Delta, t_G) = t_A}$$

$$
\begin{array}{lcll}
\mathcal{A}_\Gamma(\Delta, \mathtt{Rhs}\ n) & = & \begin{cases} \mathtt{InaccessibleRhs}\ n, & \mathcal{G}(\Gamma, \Delta) = \emptyset \\ \mathtt{AccessibleRhs}\ n, & \text{otherwise} \end{cases} & \\
\mathcal{A}_\Gamma(\Delta, (t; u)) & = & \mathcal{A}_\Gamma(\Delta, t); \mathcal{A}_\Gamma(\Delta \wedge \mathcal{U}(t), u) & \\
\mathcal{A}_\Gamma(\Delta, \mathtt{Guard}\ (!x)\ t) & = & \begin{cases} \mathcal{A}_\Gamma(\Delta \wedge (x \not\approx \bot), t), & \mathcal{G}(\Gamma, \Delta \wedge (x \approx \bot)) = \emptyset \\ \mathtt{MayDiverge}\ \mathcal{A}_\Gamma(\Delta \wedge (x \not\approx \bot), t) & \text{otherwise} \end{cases} & \\
\mathcal{A}_\Gamma(\Delta, \mathtt{Guard}\ (\mathsf{let}\ x = e)\ t) & = & \mathcal{A}_\Gamma(\Delta \wedge (x \approx e), t) & \\
\mathcal{A}_\Gamma(\Delta, \mathtt{Guard}\ (K\ \overline{a}\ \overline{\gamma}\ \overline{y : \tau} \leftarrow x)\ t) & = & \mathcal{A}_\Gamma(\Delta \wedge (K\ \overline{a}\ \overline{\gamma}\ \overline{y : \tau} \leftarrow x), t) &
\end{array}
$$

Putting it all together

(0) Input: Context with match vars Γ and desugared Gdt t
(1) Report n pattern vectors of $\mathcal{G}(\Gamma, \mathcal{U}(t))$ as uncovered
(2) Report the collected redundant and not-redundant-but-inaccessible clauses in $\mathcal{A}_\Gamma(\checkmark, t)$ (TODO: Write a function that collects the RHSs).

Generate inhabitants of Δ

$$\boxed{\mathcal{G}(\Gamma, \Delta) = \mathcal{P}(\overline{p})}$$

$$\mathcal{G}(\Gamma, \Delta) = \bigcup \{\mathcal{E}(\Gamma' \triangleright \nabla', \mathrm{fvs}(\Gamma)) \mid \forall(\Gamma' \triangleright \nabla') \in \mathcal{C}(\Gamma \triangleright \varnothing, \Delta)\}$$

Construct inhabited ∇s from Δ

$$\boxed{\mathcal{C}(\Gamma \triangleright \nabla, \Delta) = \mathcal{P}(\Gamma \triangleright \nabla)}$$

$$\begin{array}{lcl}
\mathcal{C}(\Gamma \triangleright \nabla, \delta) & = & \begin{cases} \{\Gamma' \triangleright \nabla'\} & \text{where } \Gamma' \triangleright \nabla' = \Gamma \triangleright \nabla \oplus \delta \\ \emptyset & \text{otherwise} \end{cases} \\
\mathcal{C}(\Gamma \triangleright \nabla, \Delta_1 \wedge \Delta_2) & = & \bigcup \{\mathcal{C}(\Gamma' \triangleright \nabla', \Delta_2) \mid \forall(\Gamma' \triangleright \nabla') \in \mathcal{C}(\Gamma \triangleright \nabla, \Delta_1)\} \\
\mathcal{C}(\Gamma \triangleright \nabla, \Delta_1 \vee \Delta_2) & = & \mathcal{C}(\Gamma \triangleright \nabla, \Delta_1) \cup \mathcal{C}(\Gamma \triangleright \nabla, \Delta_2)
\end{array}$$

Expand variables to Pat with ∇

$$\boxed{\mathcal{E}(\Gamma \triangleright \nabla, \overline{x}) = \mathcal{P}(\overline{p})}$$

$$\begin{array}{lcl}
\mathcal{E}(\Gamma \triangleright \nabla, \epsilon) & = & \{\epsilon\} \\
\mathcal{E}(\Gamma \triangleright \nabla, x_1 \ldots x_n) & = & \begin{cases} \{(K\ q_1 \ldots q_m)\, p_2 \ldots p_n \mid \forall(q_1 \ldots q_m\, p_2 \ldots p_n) \in \mathcal{E}(\Gamma \triangleright \nabla, y_1 \ldots y_m x_2 \ldots x_n)\} & \text{if } K\ \overline{a}\ \overline{\gamma}\ \overline{y : \tau} \leftarrow \\ \{_\ p_2 \ldots p_n \mid \forall(p_2 \ldots p_n) \in \mathcal{E}(\Gamma \triangleright \nabla, x_2 \ldots x_n)\} & \text{otherwise} \end{cases}
\end{array}$$

Add a constraint to the inert set

$$\boxed{\Gamma \triangleright \nabla \oplus \delta = \Gamma \triangleright \nabla}$$

$$\Gamma \triangleright \nabla \oplus \times = \bot$$

$$\Gamma \triangleright \nabla \oplus \checkmark = \Gamma \triangleright \nabla$$

$$\Gamma \triangleright \nabla \oplus \gamma = \begin{cases} \Gamma \triangleright (\nabla, \gamma) & \text{if type checker deems } \gamma \text{ compatible with } \nabla \\ & \text{and } \forall x \in \text{fvs}(\Gamma) : \Gamma \triangleright (\nabla, \gamma) \vdash x \\ \bot & \text{otherwise} \end{cases}$$

$$\Gamma \triangleright \nabla \oplus K\ \overline{a}\ \overline{\gamma}\ \overline{y : \tau} \leftarrow x = \begin{cases} \Gamma, \overline{a}, \overline{y : \tau} \triangleright \nabla \oplus \overline{a \sim b} \oplus \overline{\gamma} \oplus \overline{y \approx z} & \text{if } K\ \overline{b}\ \overline{\gamma}\ \overline{z : \tau} \leftarrow x \in \nabla \\ \Gamma' \triangleright (\nabla', K\ \overline{a}\ \overline{\gamma}\ \overline{y : \tau} \leftarrow x) & \text{where } \Gamma' \triangleright \nabla' = \Gamma, \overline{a}, \overline{y : \tau} \triangleright \nabla \oplus \overline{\gamma} \\ & \text{and } x \not\approx K \notin \nabla \\ & \text{and } \overline{\Gamma' \triangleright \nabla' \vdash y} \\ \bot & \text{otherwise} \end{cases}$$

$$\Gamma \triangleright \nabla \oplus x \not\approx K = \begin{cases} \bot & \text{if } K\ \overline{a}\ \overline{\gamma}\ \overline{y : \tau} \leftarrow x \in \nabla \\ \bot & \text{if } x : \tau \in \Gamma \\ & \text{and } \forall K' \in \text{Cons}(\Gamma \triangleright \nabla, \tau) : x \not\approx K' \in (\nabla, x \not\approx K) \\ \bot & \text{if not } \Gamma \triangleright (\nabla, x \not\approx K) \vdash x \\ \Gamma \triangleright (\nabla, x \not\approx K) & \text{otherwise} \end{cases}$$

$$\Gamma \triangleright \nabla \oplus x \approx \bot = \begin{cases} \bot & \text{if } x \not\approx \bot \in \nabla \\ \Gamma \triangleright (\nabla, x \approx \bot) & \text{otherwise} \end{cases}$$

$$\Gamma \triangleright \nabla \oplus x \not\approx \bot = \begin{cases} \bot & \text{if } x \approx \bot \in \nabla \\ \bot & \text{if not } \Gamma \triangleright (\nabla, x \not\approx \bot) \vdash x \\ \Gamma \triangleright (\nabla, x \not\approx \bot) & \text{otherwise} \end{cases}$$

$$\Gamma \triangleright \nabla \oplus x \approx y = \begin{cases} \Gamma \triangleright \nabla & \text{if } \nabla(x) = z = \nabla(y) \\ \Gamma \triangleright \nabla, x \approx y \oplus (\nabla \cap x)[y/x] & \text{if } \nabla(x) \neq z \text{ or } \nabla(y) \neq z \end{cases}$$

$$\Gamma \triangleright \nabla \oplus x \approx K\ \overline{\tau}\ \overline{\gamma}\ \overline{e} = \Gamma, \overline{a}, \overline{y : \sigma} \triangleright \nabla \oplus K\ \overline{a}\ \overline{\gamma}\ \overline{y} \leftarrow x \oplus \overline{a \sim \tau} \oplus \overline{y \approx e} \text{ where } \overline{a} \# \Gamma, \overline{y} \# \Gamma, \overline{e : \sigma}$$

$$\Gamma \triangleright \nabla \oplus x \approx e = \Gamma \triangleright \nabla$$

$$\boxed{\nabla \cap x = \nabla}$$

$$\begin{aligned} \varnothing \cap x &= \varnothing \\ (\nabla, K\ \overline{a}\ \overline{\gamma}\ \overline{y} \leftarrow x) \cap x &= (\nabla \cap x), K\ \overline{a}\ \overline{\gamma}\ \overline{y} \leftarrow x \\ (\nabla, x \not\approx K) \cap x &= (\nabla \cap x), x \not\approx K \\ (\nabla, x \approx \bot) \cap x &= (\nabla \cap x), x \approx \bot \\ (\nabla, x \not\approx \bot) \cap x &= (\nabla \cap x), x \not\approx \bot \\ (\nabla, x \approx e) \cap x &= (\nabla \cap x), x \approx e \\ (\nabla, \delta) \cap x &= \nabla \cap x \end{aligned}$$

Test if x is inhabited considering ∇

$$\boxed{\Gamma \triangleright \nabla \vdash x}$$

$$\frac{(\Gamma \triangleright \nabla \oplus x \approx \bot) \neq \bot}{\Gamma \triangleright \nabla \vdash x} \qquad \frac{\begin{array}{c} x : \tau \in \Gamma \quad K \in \mathsf{Cons}(\Gamma \triangleright \nabla, \tau) \\ \mathsf{Inst}(\Gamma, x, K) = \overline{\delta} \\ (\Gamma, \overline{y : \tau'} \triangleright \nabla \oplus \overline{\delta}) \neq \bot \end{array}}{\Gamma \triangleright \nabla \vdash x}$$

$$\frac{x : \tau \in \Gamma \quad \mathsf{Cons}(\Gamma \triangleright \nabla, \tau) = \bot}{\Gamma \triangleright \nabla \vdash x} \qquad \frac{\begin{array}{c} x : \tau \in \Gamma \quad K \in \mathsf{Cons}(\Gamma \triangleright \nabla, \tau) \\ \mathsf{Inst}(\Gamma, x, K) = \bot \end{array}}{\Gamma \triangleright \nabla \vdash x}$$

Find data constructors of τ

$$\boxed{\mathsf{Cons}(\Gamma \triangleright \nabla, \tau) = \overline{K}}$$

$$\mathsf{Cons}(\Gamma \triangleright \nabla, \tau) = \begin{cases} \overline{K} & \tau = T\ \overline{\sigma} \text{ and } T \text{ data type with constructors } \overline{K} \\ & \text{(after normalisation according to the type constraints in } \nabla) \\ \bot & \text{otherwise} \end{cases}$$

Instantiate x to data constructor K

$$\boxed{\mathsf{Inst}(\Gamma, x, K) = \overline{\gamma}}$$

$$\mathsf{Inst}(\Gamma, x, K) = \begin{cases} \tau_x \sim \tau, K\ \overline{a}\ \overline{\gamma}\ \overline{y} \leftarrow x, \overline{y' \not\approx \bot} & K : \forall \overline{a}.\overline{\gamma} \Rightarrow \overline{\sigma} \to \tau,\ \overline{y} \# \Gamma,\ \overline{a} \# \Gamma,\ x : \tau_x \in \Gamma,\ \overline{y'} \text{ bind strict fields} \\ \bot & \text{otherwise} \end{cases}$$