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.

Guard Syntax

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

Constraint Formula Syntax

$$
\begin{array}{lll@{\qquad}l}
\Gamma & ::= & \varnothing \mid \Gamma, x : \tau \mid \Gamma, a & \text{Context} \\
\delta & ::= & \checkmark \mid \times \mid K\ \overline{a}\ \overline{\gamma}\ \overline{y : \tau} \leftarrow x \mid x \not\approx K \mid x \approx \bot \mid x \not\approx \bot \mid x \approx e & \text{Constraint Literals} \\
\Delta & ::= & \delta \mid \Delta \wedge \Delta \mid \Delta \vee \Delta & \text{Formula} \\
\varphi & ::= & \gamma \mid x \approx K\ \overline{a}\ \overline{y} \mid x \not\approx K \mid x \approx \bot \mid x \not\approx \bot \mid x \approx y & \text{Simple constraints without scoping} \\
\Phi & ::= & \varnothing \mid \Phi, \varphi & \text{Set of simple constraints} \\
\nabla & ::= & \Gamma \triangleright \Phi & \text{Inert Set}
\end{array}
$$

Clause Tree Syntax

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

Fig. 1. Syntax

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_f:

$$
\begin{array}{c}
\texttt{Guard}\ (!x)\ \texttt{Guard}\ (\texttt{Nothing} \leftarrow x)\ \texttt{Rhs}\ 1; \\
\texttt{Guard}\ (!x)\ \texttt{Guard}\ (\texttt{Just}\ y \leftarrow x)\ \texttt{Rhs}\ 2
\end{array}
$$

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)

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

(2)

$$
\Delta_2 \quad := \quad \mathcal{U}(t_f) = \Delta_1 \wedge x \not\approx \bot \wedge (x \not\approx \texttt{Just} \vee \times)
$$

Checking Guard Trees

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

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

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

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

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).

Fig. 2. Pattern-match checking

The right operands of $\vee$ are vacuous, but the purely syntactical transformation doesn't see that.

We can see that Δ_2 is in fact uninhabited, because the three constraints $x \not\approx \bot$, $x \not\approx \mathtt{Nothing}$ and $x \not\approx \mathtt{Just}$ cover all possible data constructors of the `Maybe` data type. And indeed $\mathcal{G}(x : \mathtt{Maybe\ Int}, \Delta_2) = \emptyset$, as we'll see later.

1.2.2 Redundancy. In order to compute the annotated clause tree $\mathcal{A}_\Gamma(\checkmark, t_f)$, we need to perform the following four inhabitance checks, 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: $\Delta_5 := \checkmark \wedge \Delta_1 \wedge x \approx \bot$
(4) Upon reaching the second RHS: $\Delta_6 := \checkmark \wedge \Delta_1 \wedge x \not\approx \bot \wedge \mathtt{Just}\ y \leftarrow x$

Except for Δ_5, these are all inhabited, i.e. $\mathcal{G}(x : \mathtt{Maybe\ Int}, \Delta_i) \neq \emptyset$ (as we'll see in the next section).

Thus, we will get the following annotated tree:

$$\mathtt{MayDiverge}\ \mathtt{AccessibleRhs}\ 1; \mathtt{AccessibleRhs}\ 2$$

1.3 Generating inhabitants

The last section left open how $\mathcal{G}(,)$ works, which was used to establish or refute vacuosity of a Δ.

Generate inhabitants of Δ

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

$$\mathcal{G}(\Gamma, \Delta) = \bigcup \{ \mathcal{E}(\nabla, \mathsf{dom}(\Gamma)) \mid \nabla \in \mathcal{C}(\Gamma \triangleright \varnothing, \Delta) \}$$

Construct inhabited ∇s from Δ

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

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

Expand variables to Pat with ∇

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

$$\begin{array}{lcl}
\mathcal{E}(\nabla, \epsilon) & = & \{\epsilon\} \\
\mathcal{E}(\Gamma \triangleright \Phi, x_1 ... x_n) & = & \begin{cases} \{ (K\ q_1 ... q_m)\, p_2 ... p_n \mid (q_1 ... q_m\, p_2 ... p_n) \in \mathcal{E}(\Gamma \triangleright \Phi, y_1 ... y_m x_2 ... x_n) \} & \text{if } \Phi(x) \approx K\ \overline{a}\ \overline{y} \in \\ \{ \_\ p_2 ... p_n \mid (p_2 ... p_n) \in \mathcal{E}(\Gamma \triangleright \Phi, x_2 ... x_n) \} & \text{otherwise} \end{cases}
\end{array}$$

Finding the representative of a variable in Φ

$$\boxed{\Phi(x) = y}$$

$$\Phi(x) \quad = \quad \begin{cases} \Phi(y) & x \approx y \in \Phi \\ x & \text{otherwise} \end{cases}$$

Fig. 3. Bridging between the facade Δ and ∇

$\mathcal{G}(,)$ proceeds in two steps: First it constructs zero, one or many *inert sets* ∇ with $\mathcal{C}(,)$ (each of them representing a set of mutually compatible constraints) and then expands each of the returned inert sets into one or more pattern vectors $\overline{p}$ with $\mathcal{E}(,)$, which is the preferred representation to show to the user.

The interesting bit happens in $\mathcal{C}(,)$, where a Δ is basically transformed into disjunctive normal form, represented by a set of independently inhabited ∇. This ultimately happens in the base case of $\mathcal{C}(,)$, by gradually adding individual constraints to the incoming inert set with $\oplus_\varphi$, which starts out empty in $\mathcal{G}(,)$. Conjunction is handled by performing the equivalent of a `concatMap`, whereas disjunction simply translates to set union.

Let's see how that works for Δ_3 above. Recall that $\Gamma = x :$ `Maybe Int` and $\Delta_3 = \checkmark \wedge x \approx \bot$:

Add a constraint to the inert set

$$\boxed{\nabla \oplus_\delta \delta = \nabla}$$

$$
\begin{array}{lcl}
\nabla \oplus_\delta \times & = & \bot \\
\nabla \oplus_\delta \checkmark & = & \nabla \\
\Gamma \triangleright \Phi \oplus_\delta K\ \overline{a}\ \overline{\gamma}\ \overline{y : \tau} \leftarrow x & = & \Gamma, \overline{a}, \overline{y : \tau} \triangleright \Phi \oplus_\varphi \overline{\gamma} \oplus_\varphi x \approx K\ \overline{a}\ \overline{y} \\
\Gamma \triangleright \Phi \oplus_\delta x \approx K\ \overline{\tau'}\ \overline{\tau}\ \overline{\gamma}\ \overline{e} & = & \Gamma, \overline{a}, \overline{y : \sigma} \triangleright \Phi \oplus_\delta K\ \overline{a}\ \overline{\gamma}\ \overline{y} \leftarrow x \oplus_\varphi \overline{a \sim \tau} \oplus_\delta \overline{y \approx e} \text{ where } \overline{a} \# \Gamma, \overline{y} \# \Gamma, \overline{e : \sigma} \\
\nabla \oplus_\delta x \approx e & = & \nabla \\
\Gamma \triangleright \Phi \oplus_\delta \delta & = & \Gamma \triangleright \Phi \oplus_\varphi \delta
\end{array}
$$

Add a simple constraint to the inert set

$$\boxed{\nabla \oplus_\varphi \varphi = \nabla}$$

$$
\begin{array}{lcl}
\Gamma \triangleright \Phi \oplus_\varphi \gamma & = & \begin{cases} \Gamma \triangleright (\Phi, \gamma) & \text{if type checker deems } \gamma \text{ compatible with } \Phi \\ & \text{and } \forall x \in \text{dom}(\Gamma) : \Gamma \triangleright (\Phi, \gamma) \vdash \Phi(x) \\ \bot & \text{otherwise} \end{cases} \\
\Gamma \triangleright \Phi \oplus_\varphi x \approx K\ \overline{a}\ \overline{y} & = & \begin{cases} \Gamma \triangleright \Phi \oplus_\varphi \overline{a \sim b} \oplus_\varphi \overline{y \approx z} & \text{if } \Phi(x) \approx K\ \overline{b}\ \overline{z} \in \Phi \\ \Gamma' \triangleright (\Phi', \Phi(x) \approx K\ \overline{a}\ \overline{y}) & \text{where } \Gamma' \triangleright \Phi' = \Gamma \triangleright \Phi \oplus_\varphi \overline{\gamma} \\ & \text{and } \Phi'(x) \not\approx K \notin \Phi' \\ & \text{and } \overline{\Gamma' \triangleright \Phi' \vdash y} \\ \bot & \text{otherwise} \end{cases} \\
\Gamma \triangleright \Phi \oplus_\varphi x \not\approx K & = & \begin{cases} \bot & \text{if } \Phi(x) \approx K\ \overline{a}\ \overline{y} \in \Phi \\ \bot & \text{if not } \Gamma \triangleright (\Phi, \Phi(x) \not\approx K) \vdash \Phi(x) \\ \Gamma \triangleright (\Phi, \Phi(x) \not\approx K) & \text{otherwise} \end{cases} \\
\Gamma \triangleright \Phi \oplus_\varphi x \approx \bot & = & \begin{cases} \bot & \text{if } \Phi(x) \not\approx \bot \in \Phi \\ \Gamma \triangleright (\Phi, \Phi(x) \approx \bot) & \text{otherwise} \end{cases} \\
\Gamma \triangleright \Phi \oplus_\varphi x \not\approx \bot & = & \begin{cases} \bot & \text{if } \Phi(x) \approx \bot \in \Phi \\ \bot & \text{if not } \Gamma \triangleright (\Phi, \Phi(x) \not\approx \bot) \vdash \Phi(x) \\ \Gamma \triangleright (\Phi, \Phi(x) \not\approx \bot) & \text{otherwise} \end{cases} \\
\Gamma \triangleright \Phi \oplus_\varphi x \approx y & = & \begin{cases} \Gamma \triangleright \Phi & \text{if } \Phi(x) = \Phi(y) \\ \Gamma \triangleright (\Phi, \Phi(x) \approx \Phi(y)) \oplus_\varphi ((\Phi \cap \Phi(x))[\Phi(y)/\Phi(x)]) & \text{otherwise} \end{cases}
\end{array}
$$

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

$$
\begin{array}{lcl}
\varnothing \cap x & = & \varnothing \\
(\Phi, x \approx K\ \overline{a}\ \overline{y}) \cap x & = & (\Phi \cap x), x \approx K\ \overline{a}\ \overline{y} \\
(\Phi, x \not\approx K) \cap x & = & (\Phi \cap x), x \not\approx K \\
(\Phi, x \approx \bot) \cap x & = & (\Phi \cap x), x \approx \bot \\
(\Phi, x \not\approx \bot) \cap x & = & (\Phi \cap x), x \not\approx \bot \\
(\Phi, \varphi) \cap x & = & \Phi \cap x
\end{array}
$$

Fig. 4. Adding a constraint to the inert set ∇

$$
\begin{array}{ll}
 & \mathcal{C}(\Gamma, \checkmark \wedge x \approx \bot) \\
= & \quad \{ \text{ Conjunction } \} \\
 & \bigcup \{ \mathcal{C}(\Gamma' \triangleright \nabla', x \approx \bot) \mid \Gamma' \triangleright \nabla' \in \mathcal{C}(\Gamma \triangleright \varnothing, \checkmark) \} \\
= & \quad \{ \text{ Single constraint } \} \\
 & \begin{cases} \mathcal{C}(\Gamma' \triangleright \nabla', x \approx \bot) & \text{where } \Gamma' \triangleright \nabla' = \Gamma \triangleright \varnothing \oplus_\varphi \checkmark \\ \emptyset & \text{otherwise} \end{cases} \\
= & \quad \{ \checkmark \text{ case of } \oplus_\varphi \ \}
\end{array}
$$

Test if x is inhabited considering ∇

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

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

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

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

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

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

Fig. 5. Inhabitance test

Let's start with $\mathcal{G}(\Gamma, \Delta_3)$, where $\Gamma = x : \texttt{Maybe Int}$ and recall that $\Delta_3 = \checkmark \wedge x \approx \bot$. The first constraint $\checkmark$ is added very easily to the initial nabla by discarding it, the second one ($x \approx \bot$) is not conflicting with any $x \not\approx \bot$ constraint in the incoming, still empty ($\varnothing$) nabla, 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:

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

The subgoal $\Gamma \triangleright x \not\approx \bot \oplus_\varphi \texttt{Nothing} \leftarrow x$ 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 $\Gamma \triangleright x \not\approx \bot \oplus_\varphi \texttt{Nothing} \leftarrow x$. Conveniently, we just did that! So the result of $\mathcal{C}(\Gamma \triangleright \varnothing, \Delta_4)$ 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 $\mathcal{C}(\Gamma \triangleright \varnothing, \Delta_1)$ (which occur syntactically in Δ_5 and Δ_6) as the initial ∇. So, we first compute that.

Fast forward to computing $\Gamma \triangleright x \not\approx \bot \oplus_\varphi x \not\approx \texttt{Nothing}$. 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 $\texttt{Just}$ constructor:

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

$\Gamma, y : \texttt{Int} \triangleright (x \not\approx \bot, x \not\approx \texttt{Nothing}) \oplus_\varphi \texttt{Just}\ y \leftarrow x)$ 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 $\mathcal{C}(\Gamma \triangleright \varnothing, \Delta_1) = \Gamma \triangleright x \not\approx \bot, x \not\approx \texttt{Nothing}$.

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

$\mathcal{C}(\Gamma \triangleright \varnothing, \Delta_6)$ 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 $\_ \oplus_\varphi \_$. (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.