Functional programming and type systems (MPRI 2-4)
Mid-term exam 2010-2011

Yann Régis Gianas and Didier Rémy

Duration 2h30

Section 1 is independent of other sections ***and*** *must be written on a separate sheet.*

1 GADTs

In our formal presentation of GADTs, we restricted the syntax of patterns to flatten patterns only, that is, data constructors applied exclusively to variables. The purpose of this exercise is the extension of this formalization to nested patterns p whose syntax is defined below together with that of values v:

$$\begin{array}{rcl} p & ::= & x \mid K\ (p_1, \ldots, p_n) \mid (p_1, p_2) \\ v & ::= & \lambda x.\, t \mid K\ (v_1, \ldots, v_n) \mid (v_1, v_2) \mid 0 \mid 1 \mid 2 \mid \ \ldots \end{array}$$

A notion of extended substitution is convenient to define the reduction rules for pattern matching with nested patterns. In the following grammar, the metavariable S denotes an extended substitution, ϕ denotes a simple substitution and E_s denotes an evaluation context for extended substitutions.

$$\begin{array}{rcl} S & ::= & \bot \mid \{p \mapsto v\} \mid S \otimes S \\ E_s & ::= & [] \mid E_s \otimes S \mid \phi \otimes E_s \\ \phi & ::= & \{x \mapsto v\} \mid \phi \otimes \phi \end{array}$$

A reduction relation $\xrightarrow{m}$ is defined by the following reduction rules. This relation computes the simple substitution ϕ that can be applied to a pattern to match a particular value, or returns the undefined substitution $\bot$ if no such simple substitution exists.

$$\begin{array}{rcll} \{(p_1, p_2) \mapsto (v_1, v_2)\} & \xrightarrow{m} & \{p_1 \mapsto v_1\} \otimes \{p_2 \mapsto v_2\} & \\ \{K\ (p_1, \ldots, p_n) \mapsto K\ (v_1, \ldots, v_n)\} & \xrightarrow{m} & \{p_1 \mapsto v_1\} \otimes \ldots \otimes \{p_n \mapsto v_n\} & \\ \{K\ (p_1, \ldots, p_n) \mapsto K'\ (v_1, \ldots, v_m)\} & \xrightarrow{m} & \bot & \text{if } K \neq K' \text{ or } n \neq m \\ \bot \otimes S & \xrightarrow{m} & \bot & \\ \phi \otimes \bot & \xrightarrow{m} & \bot & \\ E_s[S] & \xrightarrow{m} & E_s[S'] & \text{if } S \xrightarrow{m} S' \end{array}$$

The new reduction rules for pattern matching with nested patterns are:

$$\begin{array}{rcll} \mathsf{match}\, v\, \mathsf{with}\, p \Rightarrow t \mid \vec{b} & \longrightarrow & \phi\ t & \text{if } \{p \mapsto v\} \xrightarrow{m}{}^{\star}\ \phi \\ \mathsf{match}\, v\, \mathsf{with}\, p \Rightarrow t \mid \vec{b} & \longrightarrow & \mathsf{match}\, v\, \mathsf{with}\, \vec{b} & \text{if } \{p \mapsto v\} \xrightarrow{m}{}^{\star}\ \bot \end{array}$$

Question 1 *Let IsInt and IsPair be data constructors. Consider the following two terms:*

$$t_1 \stackrel{\mathsf{def}}{=} \mathit{match}\ (IsInt, 42)\ \mathit{with}\ (IsPair(r_1, r_2), (x_1, x_2)) \Rightarrow v_1 \mid (IsInt, x) \Rightarrow v_2$$

$$t_2 \stackrel{\mathsf{def}}{=} \mathit{match}\ (42, IsInt)\ \mathit{with}\ ((x_1, x_2), IsPair(r_1, r_2)) \Rightarrow v_1 \mid (x, IsInt) \Rightarrow v_2$$

Give their sequence of reductions. □

Question 2 *Characterize the normal forms of the reduction $\xrightarrow{m}$. (No justification is needed.)* $\square$

Question 3 *Devise a conversion rule for patterns and extend the existing typing rules for patterns and branches to handle nested patterns.* $\square$

2 Polymorphic records

We consider an extension $\mathsf{F}^{\{\!\{\}\!\}}$ of explicitly typed System F with polymorphic records. We introduce kinds to allow variables to range over regular types or over records of a certain type. The syntactic definition of the language is:

$$
\begin{array}{rcl}
R & ::= & \ell : T, \ldots \ell : T \\
\kappa & ::= & * \mid \{\!\{R\}\!\} \\
T & ::= & \alpha \mid T \to T \mid \forall(\alpha :: \kappa)\, T \mid \{R\} \\
t & ::= & x \mid \lambda(x : T)\, t \mid t\; t \mid \Lambda(\alpha :: \kappa)\, t \mid t\; T \mid t.\ell \mid \{\ell = t, \ldots \ell = t\} \\
\Gamma & ::= & \emptyset \mid \Gamma, \alpha :: \kappa \mid \Gamma, x : T
\end{array}
$$

Judgments for well-formedness of environments $\vdash \Gamma$, kinds $\Gamma \vdash \kappa$, and types $\Gamma \vdash T :: \kappa$ are:

$$
\textsc{E-Empty}\;\; \vdash \emptyset
\qquad
\textsc{E-Tvar}\;\; \frac{\vdash \Gamma \qquad \Gamma \vdash \kappa \qquad \alpha \notin \mathsf{dom}\,\Gamma}{\vdash \Gamma, \alpha :: \kappa}
\qquad
\textsc{E-Var}\;\; \frac{\vdash \Gamma \qquad \Gamma \vdash T :: * \qquad x \notin \mathsf{dom}\,\Gamma}{\vdash \Gamma, x : T}
\qquad
\textsc{K-Type}\;\; \Gamma \vdash *
$$

$$
\textsc{K-Record}\;\; \frac{\begin{array}{c} i \mapsto \ell_i \text{ injective} \\ (\Gamma \vdash T_i :: *)^{i \in I} \end{array}}{\Gamma \vdash \{\!\{(\ell_i : T_i)^{i \in I}\}\!\}}
\qquad
\textsc{T-Tvar}\;\; \frac{\vdash \Gamma \qquad \alpha :: \kappa \in \Gamma}{\Gamma \vdash \alpha :: \kappa}
\qquad
\textsc{T-Arrow}\;\; \frac{\Gamma \vdash T_1 :: * \qquad \Gamma \vdash T_2 :: *}{\Gamma \vdash T_1 \to T_2 :: *}
\qquad
\textsc{T-All}\;\; \frac{\Gamma, \alpha :: \kappa \vdash T :: *}{\Gamma \vdash \forall(\alpha :: \kappa)\, T :: *}
$$

$$
\textsc{T-Record}\;\; \frac{\Gamma \vdash \{\!\{R\}\!\}}{\Gamma \vdash \{R\} :: \{\!\{R\}\!\}}
\qquad
\textsc{T-Sub-Record}\;\; \frac{\Gamma \vdash T :: \{\!\{(\ell_i : T_i)^{i \in I}\}\!\} \qquad J \subseteq I}{\Gamma \vdash T :: \{\!\{(\ell_j : T_j)^{j \in J}\}\!\}}
\qquad
\textsc{T-Sub-Type}\;\; \frac{\Gamma \vdash T :: \{\!\{R\}\!\}}{\Gamma \vdash T :: *}
$$

The interesting rules are T-Record, T-Sub-Record, and T-Sub-Type for record kinds. The typing rules are:

$$
\textsc{Var}\;\; \frac{\vdash \Gamma \qquad x : T \in \Gamma}{\Gamma \vdash x : T}
\qquad
\textsc{Fun}\;\; \frac{\Gamma, x : T_2 \vdash t : T_1}{\Gamma \vdash \lambda(x : T_2)\, t : T_2 \to T_1}
\qquad
\textsc{App}\;\; \frac{\Gamma \vdash t_1 : T_2 \to T_1 \qquad \Gamma \vdash t_2 : T_2}{\Gamma \vdash t_1\; t_2 : T_1}
$$

$$
\textsc{Gen}\;\; \frac{\Gamma, \alpha :: \kappa \vdash t : T}{\Gamma \vdash \Lambda(\alpha :: \kappa)\, t : \forall(\alpha :: \kappa)\, T}
\qquad
\textsc{Inst}\;\; \frac{\Gamma \vdash t : \forall(\alpha :: \kappa)\, T \qquad \Gamma \vdash T_0 :: \kappa}{\Gamma \vdash t\; T_0 : [\alpha \mapsto T_0]T}
$$

$$
\textsc{Proj}\;\; \frac{\Gamma \vdash t : T \qquad \Gamma \vdash T :: \{\!\{\ell_1 : T_1\}\!\}}{\Gamma \vdash t.\ell_1 : T_1}
\qquad
\textsc{Record}\;\; \frac{(\Gamma \vdash t_i : T_i)^{i \in I} \qquad i \mapsto \ell_i \text{ injective}}{\Gamma \vdash \{(\ell_i = t_i)^{i \in I}\} : \{(\ell_i : T_i)^{i \in I}\}}
$$

Typing rules may be inverted as usual; you may invoke the *inversion* lemma in proofs without stating it formally nor proving it. You may also admit that $\Gamma \vdash t : T$ implies $\Gamma \vdash T : *$. Type erasure is defined as usual. The language is equipped with a call-by-value type-erasing semantics. In examples, we assume an extension of the language with primitive booleans, but you may ignore booleans in formal developments.

Question 4 *Is the term* $\lambda(x : \texttt{bool})\,\texttt{if}\ x\ \texttt{then}\ \{\ell = \texttt{true}\}\ \texttt{else}\ \{\ell = \texttt{true}, \ell_0 = \texttt{false}\}$ *well-typed? (Give the full typing derivation of* t *or explain why there is none.)* $\square$

Question 5 *Give a well-typed term* t_5*, without its typing derivation, whose erasure is* $a_5 \overset{\mathsf{def}}{=} (\lambda(f)\, f\ \{\ell = f\ \{\ell = \texttt{true}\}, \ell_0 = \texttt{false}\})\ \ (\lambda(x)\, x.\ell)$. $\square$

Question 6 *What can you say about* T *when* $\Gamma \vdash T :: \{\!\!\{R\}\!\!\}$*? Verify that terms have unique types,* i.e. $\Gamma \vdash t : T_1$ *and* $\Gamma \vdash t : T_2$ *implies* $T_1 = T_2$*? (Justify briefly, without a formal proof.)* $\square$

3 Representation of records

Question 7 *Describe a simple implementation of record operations where record creation is in* $n \log n$ *and record access is in* $\log n$ *where* n *is the size of the domain of the record.* $\square$

We consider the language $\mathsf{F}^{\{\}}$ obtained from $\mathsf{F}^{\{\!\!\{\}\!\!\}}$ by replacing rules T-Record and Proj by R-Record' and Proj' given below and by disallowing the use of record kinds *everywhere.* (As a consequence, rules K-Record, T-Sub-Record, and T-Sub-Type become useless and could also be removed).

$$\text{T-Record'}\quad \frac{(\Gamma \vdash T_i :: *)^{i \in I} \qquad i \mapsto \ell_i \ \mathsf{injective}}{\Gamma \vdash \{(\ell_i : T_i)^{i\in I}\} :: *} \qquad\qquad \text{Proj'}\quad \frac{\Gamma \vdash t : \{R, \ell : T, R'\}}{\Gamma \vdash t.\ell : T}$$

Question 8 *Show that* $\mathsf{F}^{\{\}}$ *is a subset of* $\mathsf{F}^{\{\!\!\{\}\!\!\}}$. $\square$

Question 9 *Is the term* t_5 *found at question 5 (or another elaboration of* a_5*) well-typed in* $\mathsf{F}^{\{\}}$*? (Answer precisely, but do not prove anything.)* $\square$

Question 10 *Describe a more efficient implementation of records in* $\mathsf{F}^{\{\}}$*. (Justify briefly, but no formal proof of correction is needed.)* $\square$

4 Compiling polymorphic access away

In this section, we define a translation of $\mathsf{F}^{\{\!\!\{\}\!\!\}}$ into $\mathsf{F}^{\{\}}$ that eliminates the need for polymorphic record access. We use the following "row folding" notations:

$$\begin{aligned}
(\alpha \to (\ell_1 : T_1, \dots \ell_n : T_n)) \to T \ &= \ (\alpha \to T_1) \to \dots (\alpha \to T_n) \to T \\
x^\alpha : \alpha \to (\ell_1 : T_1, \dots \ell_n : T_n) \ &= \ x^\alpha_{\ell_1} : \alpha \to T_1, \dots x^\alpha_{\ell_n} : \alpha \to T_n \\
\lambda(x : \alpha \to (\ell_1 : T_1, \dots \ell_n : T_n))\, t \ &= \ \lambda(x^\alpha_{\ell_1} : \alpha \to T_1)\ \dots \lambda(x^\alpha_{\ell_n} : \alpha \to T_n)\, t \\
t\ (\alpha.(\ell_1 : T_1, \dots \ell_n : T_n)) \ &= \ t\, x^\alpha_{\ell_1}\ \dots\ x^\alpha_{\ell_n} \\
t\ (\{R\}.(\ell_1 : T_1, \dots \ell_n : T_n)) \ &= \ t\ (\lambda(x : \{R\})\, x.\ell_1)\ \dots (\lambda(x : \{R\})\, x.\ell_n)
\end{aligned}$$

The translation of types is:

$$\begin{aligned}
[\![\alpha]\!] \ &= \ \alpha \\
[\![T_1 \to T_2]\!] \ &= \ [\![T_1]\!] \to [\![T_2]\!] \\
[\![\{(\ell_i : T_i)^{i\in I}\}]\!] \ &= \ \{(\ell_i : [\![T_i]\!])^{i\in I}\} \\
[\![\forall(\alpha :: *)\, T]\!] \ &= \ \forall(\alpha :: *)\, [\![T]\!] \\
[\![\forall(\alpha :: \{\!\!\{R\}\!\!\})\, T]\!] \ &= \ \forall(\alpha :: *)\, (\alpha \to [\![R]\!]) \to [\![T]\!] \\
\\
[\![(\ell_i : T_i)^{i\in I}]\!] \ &= \ (\ell_i : [\![T_i]\!])^{i\in I}
\end{aligned}$$

The translation of typing contexts is the concatenation of the translation of their bindings, defined as:

$$[\![x : T]\!] = x : [\![T]\!] \qquad [\![\alpha :: *]\!] = \alpha :: * \qquad [\![\alpha :: \{\!\!\{R\}\!\!\}]\!] = \alpha :: *, x^\alpha : \alpha \to [\![R]\!]$$

You may admit that $\vdash \Gamma$ implies $\vdash \llbracket\Gamma\rrbracket$ and $\Gamma \vdash T :: \kappa$ implies $\llbracket\Gamma\rrbracket \vdash \llbracket T\rrbracket :: *$.
Expressions are elaborated as follows:

E-Var
$$\frac{\vdash \Gamma \qquad x : T \in \Gamma}{\Gamma \vdash x : T \rhd x}$$

E-Fun
$$\frac{\Gamma, x : T_2 \vdash t : T_1 \rhd t'}{\Gamma \vdash \lambda(x : T_2)\, t : T_2 \to T_1 \rhd \lambda(x : \llbracket T_2\rrbracket)\, t'}$$

E-App
$$\frac{\Gamma \vdash t_2 : T_2 \rhd {t_2}' \qquad \Gamma \vdash t_1 : T_2 \to T_1 \rhd {t_1}'}{\Gamma \vdash t_1\, t_2 : T_1 \rhd {t_1}'\, {t_2}'}$$

E-Record
$$\frac{(\Gamma \vdash t_i : T_i \rhd t'_i)^{i\in I} \qquad i \mapsto \ell_i \text{ injective}}{\Gamma \vdash \{(\ell_i = t_i)^{i\in I}\} : \{(\ell_i : T_i)^{i\in I}\} \rhd \{(\ell_i = t'_i)^{i\in I}\}}$$

E-Gen-Type
$$\frac{\Gamma, \alpha :: * \vdash t : T \rhd t'}{\Gamma \vdash \Lambda(\alpha :: *)\, t : \forall(\alpha :: *)\, T \rhd \Lambda(\alpha :: *)\, t'}$$

E-Inst-Type
$$\frac{\Gamma \vdash t : \forall(\alpha :: *)\, T \rhd t' \qquad \Gamma \vdash T_0 :: *}{\Gamma \vdash t\, T_0 : [\alpha \mapsto T_0]T \rhd t'\, \llbracket T_0\rrbracket}$$

E-Gen-Record
$$\frac{\Gamma, \alpha :: \{\!\{R\}\!\} \vdash t : T \rhd t'}{\begin{array}{l}\Gamma \vdash \Lambda(\alpha :: \{\!\{R\}\!\})\, t : \forall(\alpha :: \{\!\{R\}\!\})\, T \\ \qquad \rhd \Lambda(\alpha :: *)\, \lambda(x : \alpha \to \llbracket R\rrbracket)\, t'\end{array}}$$

E-Inst-Record
$$\frac{\Gamma \vdash t : \forall(\alpha :: \{\!\{R\}\!\})\, T \rhd t' \qquad \Gamma \vdash T_0 :: \{\!\{R\}\!\}}{\Gamma \vdash t\, \beta : [\alpha \mapsto T_0]T \rhd t\, \llbracket T_0\rrbracket\, (\llbracket T_0\rrbracket.\llbracket R\rrbracket)}$$

E-Proj-Var
$$\frac{\Gamma \vdash t : \alpha \rhd t' \qquad \Gamma \vdash \alpha :: \{\!\{\ell : T\}\!\}}{\Gamma \vdash t.\ell : T \rhd x^\alpha_\ell\, t'}$$

E-Proj-Record
$$\frac{\Gamma \vdash t : \{R\} \rhd t' \qquad \Gamma \vdash \{R\} :: \{\!\{\ell : T\}\!\}}{\Gamma \vdash t.\ell : T \rhd t'.\ell}$$

If $\Gamma \vdash t : T \rhd t'$, we write $\llbracket\Gamma \vdash t : T\rrbracket$ for the judgment $\llbracket\Gamma\rrbracket \vdash t' : \llbracket T\rrbracket$. You may admit that the translation is well-defined, *i.e.* if $\Gamma \vdash t : T$ holds then there exists a unique t' so that $\Gamma \vdash t : T \rhd t'$.

Question 11 *Give the translation t'_5 of the elaboration of a_5 of question 5* ***and*** *the type erasure a'_5 of t'_5.* □

Question 12 *Show that $\Gamma \vdash \alpha :: \{\!\{\ell : T\}\!\}$ implies $x^\alpha_\ell : \alpha \to \llbracket T\rrbracket$ in $\llbracket\Gamma\rrbracket$.* □

Question 13 (long) *Prove that the translation is type preserving,* i.e. *if $\Gamma \vdash t : T$ holds then $\llbracket\Gamma \vdash t : T\rrbracket$ also holds. (The proof structure should be very clear. You only need to treat the cases related to rules* E-Gen-Record*,* E-Inst-Record*, and* E-Proj-**.)* □

Question 14 *Does $\Gamma \vdash t : T \rhd t'$ imply t' in $F^{\{\}}$? (Justify briefly.)* □

Question 15 *Can you give one simple optimization to the compilation schema?* □

Question 16 *Access functions $\lambda(x : \{R\} \to T)\, x.\ell$ introduced during the translation by rule* E-Inst-Record *may be passed around (via rules* E-Gen-Record *and* E-Inst-Record*) and used by rule* E-Proj-Var*. Since their introduction and elimination sites are statically known, there is an opportunity for an optimization in their compilation. Which one?* □

Question 17 *If we extend the language with fix points or references, the translation would not always preserve the semantics. Give an example. Could the source language be restricted to rule out such programs?* □

Solutions

Question 1, page 1

$$\begin{array}{lll}
t_1 & \longrightarrow & \textsf{match } (IsInt, 42) \textsf{ with } (IsInt, x) \Rightarrow v_2 \\
& & \text{Since:} \quad \begin{array}{l} \{(IsPair(r_1, r_2), (x_1, x_2)) \mapsto (IsInt, 42)\} \\ \xrightarrow{m} \{IsPair(r_1, r_2) \mapsto IsInt\} \otimes \{(x_1, x_2) \mapsto 42\} \\ \xrightarrow{m} \bot \otimes \{(x_1, x_2) \mapsto 42\} \\ \xrightarrow{m} \bot \end{array} \\
& \longrightarrow & \{x \mapsto 42\}\ v_2 \\
& & \text{Since:} \quad \begin{array}{l} \{(IsInt, x) \mapsto (IsInt, 42)\} \\ \xrightarrow{m} \{IsInt \mapsto IsInt\} \otimes \{x \mapsto 42\} \\ \xrightarrow{m} \{x \mapsto 42\} \end{array} \\
t_2 & \not\longrightarrow & \\
& & \text{Since:} \quad \begin{array}{l} \{((x_1, x_2), IsPair(r_1, r_2)) \mapsto (42, IsInt)\} \\ \xrightarrow{m} \{(x_1, x_2) \mapsto 42\} \otimes \{IsPair(r_1, r_2) \mapsto IsInt\} \\ \not\xrightarrow{m} \end{array}
\end{array}$$

So, t_1 evaluates into $\{x \mapsto 42\}v_2$ whereas t_2 is stuck.

Question 2, page 2

If $S \not\xrightarrow{m}$ then one of the following assertion holds: (i) $S \equiv \bot$, (ii) $S \equiv \phi$, or (iii) there exists E_s such that $S \equiv E_s[\{K\ (p_1, \ldots, p_n) \mapsto v\}]$ with $v \equiv 0, 1, \ldots$ or $v \equiv \lambda x.\, t$.

Question 3, page 2

Pat-Conv

$$\frac{\Delta \vdash p : \tau' \leadsto (\vec{\beta}, \Delta', \Gamma) \qquad \Delta \models \tau = \tau'}{\Delta \vdash p : \tau \leadsto (\vec{\beta}, \Delta', \Gamma)}$$

Pat

$$\frac{\begin{array}{c} K \preceq \forall \vec{\beta}.\tau_1 \times \ldots \times \tau_n \rightarrow \varepsilon\ \vec{\tau}_1\ \vec{\tau} \\ \forall i \in 1..n, \Delta \wedge \vec{\tau} = \vec{\tau}_2 \wedge \Delta_1 \wedge \ldots \Delta_{i-1} \vdash p_i : \tau_i \leadsto (\vec{\beta}_i, \Delta_i, \Gamma_i) \end{array}}{\Delta \vdash K\ (p_1 \ldots p_n) : \varepsilon\ \vec{\tau}_1\ \vec{\tau}_2 \leadsto (\vec{\beta}\vec{\beta}_1 \ldots \vec{\beta}_n, \vec{\tau} = \vec{\tau}_2 \wedge \Delta_1 \wedge \ldots \wedge \Delta_n, \Gamma_1 \ldots \Gamma_n)}$$

Branch

$$\frac{\Delta \vdash p : \tau \leadsto (\vec{\beta}', \Delta', \Gamma') \qquad \Delta \wedge \Delta', \Gamma\Gamma' \vdash t : \tau'}{\Delta, \Gamma \vdash p \Rightarrow t : \tau \rightarrow \tau'}$$

Question 4, page 3

t is ill-typed because the two branches of the conditional cannot be given a common type: the records $\{\ell = \texttt{true}\}$ and $\{\ell = \texttt{true}, \ell_0 = \texttt{false}\}$ have types $\{\!\{\ell : \texttt{bool}\}\!\}$ and $\{\!\{\ell : \texttt{bool}, \ell_0 ; \texttt{bool}\}\!\}$.

Question 5, page 3

The term t_5 is the erasure of the application $t_1\ t_2$ where:

$$\begin{array}{rcl}
t_1 & \overset{\text{def}}{=} & (\lambda(f : T_2)\, f\ T_0\ \{\ell = f\ T\ \{\ell = \texttt{true}\}, \ell_0 = \texttt{false}\}) \\
t_2 & \overset{\text{def}}{=} & \Lambda(\alpha :: \{\!\{\ell : \texttt{bool}\}\!\})\, \lambda(x : \alpha)\, x.\ell \\
T_2 & \overset{\text{def}}{=} & \forall(\alpha :: \{\!\{\ell : \texttt{bool}\}\!\})\, \alpha \to \texttt{bool} \\
T & \overset{\text{def}}{=} & \{\ell : \texttt{bool}\} \\
T_0 & \overset{\text{def}}{=} & \{\ell : \texttt{bool}, \ell_0 : \texttt{bool}\}
\end{array}$$

Question 6, page 3

A derivation of $\Gamma \vdash T :: \{\!\{R\}\!\}$ must start with either T-Var or T-Record and may continue with a sequence of rule R-Sub-Record, which may always be replaced by a single one. Hence, either T is a variable α with $\alpha :: \{\!\{R'\}\!\}$ in Γ or T is a record $\{R'\}$ and, in both cases, $R \subseteq R'$.

Terms have unique types. The proof is by induction on the term t and inversion of typing. Only case Proj could be problematic, as several types T_1 could perhaps be used. However, the previous remark implies that T_1 is uniquely dertermined by the judgment $\Gamma \vdash t : T$ which by induction is unique.

Question 7, page 3

We may represent records as heterogeneous maps, implemented as balanced trees, with labels as keys.

Question 8, page 3

Rule T-Record' is *derivable* in $\mathsf{F}^{\{\!\{\}\!\}}$ by K-Record, T-Record, and T-Sub-Type.

Rule T-Proj' is *admissible* (but not derivable) in $\mathsf{F}^{\{\!\{\}\!\}}$: Assume $\Gamma \vdash t : \{R, \ell : T, R'\}$. Since $\Gamma \vdash \{R, \ell : T, R'\} : *$ (admitted), it must come from a use of T-Record concluding $\Gamma \vdash \{R, \ell : T, R'\} :: \{\!\{R, \ell : T : R'\}\!\}$ followed by a sequence of Rule T-Sub-Record and ending with T-Sub-Type. By T-Sub-Record, we have $\Gamma \vdash \{R, \ell : T, R'\} :: \{\!\{\ell : T\}\!\}$. Hence, by Proj we may conclude $\Gamma \vdash t.\ell : T$.

Therefore, adding these rules to $\mathsf{F}^{\{\!\{\}\!\}}$ increases derivations, but does change its judgments. Then, the removal of record kinds may only restrict the set of derivations and of valid judgments in $\mathsf{F}^{\{\}}$.

Question 9, page 3

t_5 is ill-typed and there is no other well-typed term in $\mathsf{F}^{\{\}}$ whose erasure is a_5: the two occurrences of f have arguments of different domains, but a well-typed term whose erasure is $\lambda(x)\, x.\ell$ only accepts records of a fix domain.

Question 10, page 3

Since domain of records are ordered and known from their types (records with different domains or with the same domain but in different order have different types), we may drop labels, see records as tuples, and record projection at a label ℓ as the i'th projection of a tuple where i is the position of ℓ in the record domain. Schematically, $[\![\{\ell_1 = t_1, \ldots \ell_n = t_n\}]\!]$ is $([\![t_1]\!], \ldots [\![t_n]\!])$ and $[\![t.\ell_i]\!]$ is $[\![t]\!].i$ where $\{\ell_1 : T_1, \ldots \ell_i : T_i, \ldots\}$ is the type of t.

Question 11, page 4

The term t'_5 is equal to $t'_1\ t'_2$ where:

$$\begin{array}{rcl} t'_1 & = & (\lambda(f : T'_2)\, f\ T_0\ (\lambda(z : T_0)\, z.\ell)\ \{\ell = f\ T\ (\lambda(z : T)\, z.\ell)\ \{\ell = \texttt{true}\}, \ell_0 = \texttt{false}\}) \\ t'_2 & = & \Lambda(\alpha :: *)\, \lambda(x^\alpha_\ell : \alpha \to \texttt{bool})\, \lambda(x : \alpha)\, x^\alpha_\ell\ x \\ T'_2 & = & \forall(\alpha :: *)\, (\alpha \to \texttt{bool}) \to \alpha \to \texttt{bool} \\ T & = & \{\ell : \texttt{bool}\} \\ T_0 & = & \{\ell : \texttt{bool}, \ell_0 : \texttt{bool}\} \end{array}$$

The erasure a'_5 of t'_5 is:

$$(\lambda(f)\, f\ (\lambda(z)\, z.\ell)\ \{\ell = f\ (\lambda(z)\, z.\ell)\ \{\ell = \texttt{true}\}, \ell_0 = \texttt{false}\})\ \ (\lambda(x_\ell)\, \lambda(x)\, x_\ell\ x)$$

Question 12, page 4

Assume $\Gamma \vdash \alpha :: \{\!\{\ell : T\}\!\}$. From question 6, we know that there is $\alpha :: \{\!\{R\}\!\}$ in Γ with $\ell : T$ in R. By definition, $[\![\Gamma]\!]$ contains $[\![\alpha :: \{\!\{R\}\!\}]\!]$, which contains $x^\alpha : \alpha \to [\![R]\!]$ and in turn, expansing the notation, contains $x^\alpha_\ell : \alpha \to [\![T]\!]$, as expected.

Question 13, page 4

Since the translation is well-defined, it suffices to show that if $\Gamma \vdash t : T \triangleright t'$ (H) holds then $[\![\Gamma \vdash t : T]\!]$ also holds, which we do by induction on the derivation (H):

Case E-Gen-Record. The conclusion is $\Gamma \vdash \Lambda(\alpha :: \{\!\{R\}\!\})\, t : \forall(\alpha :: \{\!\{R\}\!\})\, T \triangleright \Lambda(\alpha :: *)\, \lambda(x^\alpha : \alpha \to [\![R]\!])\, t'$. The premisse is $\Gamma, \alpha :: \{\!\{R\}\!\} \vdash t : T \triangleright t'$. By induction hypothesis, we have $[\![\Gamma]\!], [\![\alpha :: \{\!\{R\}\!\}]\!] \vdash t' : [\![T]\!]$, *i.e.*, $[\![\Gamma]\!], \alpha :: *, x^\alpha : \alpha \to [\![R]\!] \vdash t' : [\![T]\!]$. A sequence of rules Fun, followed by rule Gen gives $[\![\Gamma]\!] \vdash \Lambda(\alpha :: *)\, \lambda(x^\alpha : \alpha \to [\![R]\!])\, t' : \forall(\alpha :: *)\, (\alpha \to [\![R]\!]) \to [\![T]\!]$, as expected.

Case E-Inst-Record. The conclusion is $\Gamma \vdash t\ T : [\alpha \mapsto T_0]T \triangleright t'\ [\![T_0]\!]\ ([\![T_0]\!].[\![R]\!])$. The premisses are $\Gamma \vdash t : \forall(\alpha :: \{\!\{R\}\!\})\, T \triangleright t'$ and $\Gamma \vdash T_0 :: \{\!\{R\}\!\}$. By induction on the former, we have $[\![\Gamma]\!] \vdash t' : [\![\forall(\alpha :: \{\!\{R\}\!\})\, T]\!]$, *i.e.* $[\![\Gamma]\!] \vdash t' : \forall(\alpha :: *)\, (\alpha \to [\![R]\!]) \to [\![T]\!]$. The later implies that α cannot occur in R. It also implies $[\![\Gamma]\!] \vdash [\![T_0]\!] : *$ (admitted). Hence, by rule Inst, we have $[\![\Gamma]\!] \vdash t'\ T_0 : T'$ where T' is $[\alpha \mapsto [\![T_0]\!]](\alpha \to \{\!\{R\}\!\}) \to [\![T]\!]$, *i.e.* $([\![T_0]\!] \to [\![T_1]\!]) \to \ldots ([\![T_0]\!] \to [\![T_n]\!]) \to [\alpha \mapsto [\![T_0]\!]][\![T]\!]$ where R is $(\ell_1 : T_1, \ldots \ell_n : T_n)$.

The later premisse also implies that T_0 is one of two forms: it may be a record type $\{R'\}$ with $R' \subseteq R$. Thus, $[\![R']\!] \subseteq [\![R]\!]$ and, for all $\ell_i : T_i$ in R, rules Proj' and Fun give $[\![\Gamma]\!] \vdash \lambda(x : [\![T_0]\!])\, x.\ell_i : [\![T_0]\!] \to [\![T_i]\!]$ (notice that $[\![T_0]\!]$ is $\{[\![R']\!]\}$). Otherwise, T_0 must be a variable β with $\beta :: \{\!\{R'\}\!\}$ in Γ and $R \subseteq R'$. Thus, for all $\ell_i : T_i$ in R, we have $\Gamma \vdash \beta :: \{\!\{\ell_i : T_i\}\!\}$, which by question 12 implies $[\![\Gamma]\!] \vdash x^\beta_{\ell_i} : \beta \to [\![T_i]\!]$. Notice that β is also equal to $[\![T_0]\!]$.

In both cases, a sequence of rule App gives $[\![\Gamma]\!] \vdash t\ [\![T_0]\!]\ ([\![T_0]\!].[\![R]\!]) : [\alpha \mapsto [\![T_0]\!]][\![T]\!]$, as expected.

Case E-Proj-Var. The conlusion is $\Gamma \vdash t.\ell : T \triangleright x^\alpha_l\ t'$. The premisses are $\Gamma \vdash t : \alpha \triangleright t'$ and $\Gamma \vdash \alpha :: \{\!\{\ell : T\}\!\}$. By induction, the former implies $[\![\Gamma]\!] \vdash t' : \alpha$. By question 12, the latter implies $[\![\Gamma]\!] \vdash x^\alpha_\ell : \alpha \to [\![T]\!]$. Hence, by Rule App, we get $[\![\Gamma]\!] \vdash x^\alpha_\ell\ t' : [\![T]\!]$, as expected.

Case E-Proj-Record. The conclusion is $\Gamma \vdash t.\ell : T \triangleright t'.\ell$. The premisses are $\Gamma \vdash t : \{R\} \triangleright t'$ and $\Gamma \vdash \{R\} : \{\!\{\ell : T\}\!\}$. By induction, the former implies $[\![\Gamma]\!] \vdash t' : [\![\{R\}]\!]$. By question 6 The later implies that $\ell : T$ is in R. Hence, $\ell : [\![T]\!]$ is in $[\![\{R\}]\!]$. By Proj', which is admissible in $\mathsf{F}^{\{\}}$, we conclude that $[\![\Gamma]\!] \vdash t'.\ell : [\![T]\!]$, as expected.

Other cases are omitted.

Question 14, page 4

Yes, it does. From the proof of question 13, observe that in a derivation of $[\![\Gamma]\!] \vdash t' : [\![T]\!]$, we have never needed Rule Proj but only its restricted form Proj' in cases Proj-Record and Inst-Record, since record access is always performed at a known domain. Moreover, we have never used record kinds.

Question 15, page 4

The translation inserts *extractors* for every field of record kinds, but perhaps only some of these extractors are never applied. The translation could insert extractors only if they are actually needed (applied or passed to other functions). This could be done by finding minimal kinds for type variables.

Question 16, page 4

There is some liberty in the representation of extractors, provided the creation and the elimination sites agree. Currentlty, in $\mathsf{F}^{\{\}}$, the acccess function is compiled into the function that access the tuple field and its elimination applies the function. Instead, the introduction could be just return the position of the field to be accessed as an integer and the elimination would project the argument at that position received as argument—this would save the creation and application of the access function.

Question 17, page 4

The translation may insert abstraction in front of expressions that are not values. It may thus change the order of evaluation and turn a diverging expression into a converging one. Take for example $\Lambda(\alpha :: \{\!\{\ell_1 : T_1\}\!\})\, t_0$ where t_0 diverges. Its translation is $\Lambda(\alpha :: \{\!\{\ell_1 : T_1\}\!\})\, \lambda(x^\alpha_\ell : [\![)]\!]\, t_0$ which is a value and converges.

Since the translation may only insert extra abstractions next to polymorphism (over record kinds), it would suffice to restrict (record kind) polymorphism to value-forms, as in ML with references, to reject such obviously erroneous translations.