DPPL Project Report

March 10, 2019

introduction Unification-based type inference for a ML-like language supporting boolean operations, binary and monary integers operations, Algebraic Data Types with custom-defined type constructors, and Generalized Algebraic Data Types with the help of user-specified type annotations.

language definition

$$\begin{aligned}\text{Terms } t ::=\ & x \mid \text{constant} \mid C\ \bar{t} \mid \bar{t} \mid \lambda x.t \mid t\ t \mid \text{let } x = t \text{ in } t \\ & \mid \text{let } x : \sigma = t \text{ in } t \mid \text{match } t \text{ with } \overline{p \to t} \\ & \mid t \odot t \mid\ \diamond t \mid \text{if } t \text{ then } t \text{ else } t\end{aligned}$$

$$\text{Values } v ::=\ \lambda x.t \mid \text{constant} \mid C\ \overline{v}$$

$$\text{Patterns } p ::= x \mid C\ \overline{x}$$

$$\begin{aligned}\text{Monotypes } \tau ::=\ & X \mid \tau \to \tau \\ & \mid \text{int} \mid \text{bool} \mid \text{unit} \\ & \mid \Pi\ \overline{\tau} \mid T\ \overline{\tau}\end{aligned}$$

$$\text{Polytypes } \sigma ::= \forall \overline{\alpha}.\tau$$

Algebraic Data Types Result of the fusion of iso-recursive types with structural, labeled products and sums. This suppresses the verbosity of explicit folds and unfolds.

An algebraic data types is introduced via a definition

$$T\ \overline{\alpha} \approx \Sigma_i C_i : \forall \overline{\alpha}.\overline{\tau}_i \to T\ \overline{\alpha}$$

And a term $C\ \bar{t}$ where $C : \forall \overline{\alpha}.\overline{\tau} \to T\ \overline{\alpha}$ is equivalent to $fold\ [T]\ t$. For example, take $C = Leaf$, $T = tree$:

$$\alpha\ tree = \mu X.X * \alpha * X$$

$$Leaf = fold\ [\alpha\ tree]$$

typing judgments Type system for core language:

T-Var:

$$\frac{x : \forall\overline{\alpha}.\tau \in \Gamma}{\Gamma \vdash x : \tau' \text{ where } \tau' \text{ is an instance of } \forall\overline{\alpha}.\tau}$$

T-Abs:

$$\frac{\Gamma, x : \tau_1 \vdash t_1 : \tau_2}{\Gamma \vdash \lambda x.t_1 : \tau_1 \to \tau_2}$$

T-App:

$$\frac{\Gamma \vdash t_1 : \tau_1 \to \tau \quad \Gamma \vdash t_2 : \tau_1}{\Gamma \vdash t_1\ t_2 : \tau}$$

T-Let-Mono:

$$\frac{\Gamma, x : \tau_1 \vdash t_1 : \tau_1 \quad \overline{\alpha} = ftv(\tau_1) - ftv(\Gamma) \quad \Gamma, x : \forall\overline{\alpha}.\tau_1 \vdash t_2 : \tau}{\Gamma \vdash \text{let } x = t_1 \text{ in } t_2 : \tau}$$

T-MatchWith:

$$\frac{\Gamma \vdash t_s : \tau_p \quad \Gamma \vdash \overline{p \to t} : \tau_p \to \tau}{\Gamma \vdash \text{match } t_s \text{ with } \overline{p \to t} : \tau}$$

T-Pattern:

$$\frac{C : \forall\overline{\alpha}.\overline{\tau}_1 \to T\ \overline{\alpha} \in \Gamma \quad \theta(\overline{\alpha}) = \overline{\tau}_p \quad \Gamma, \overline{x : \theta(\tau_1)} \vdash t : \tau_t}{\Gamma \vdash C\ \overline{x} \to t : T\ \overline{\tau}_p \to \tau_t}$$

Constraint typing relations:

$$\Gamma \vdash t : T \mid \sigma$$

$$\text{Property: } \Gamma \vdash t : T \mid \sigma \Rightarrow \sigma\Gamma \vdash t : \sigma T$$

Interpretation: if we want to infer t has type T, then we will generate substitution σ

CT-Var:

$$\frac{x : \forall\overline{\alpha}.\tau' \in \Gamma}{\Gamma \vdash \tau \mid unify\{\tau, fresh(\forall\overline{\alpha}, \tau')\}}$$

CT-Abs:

$$\frac{\Gamma, x : \tau_1 \vdash t : \tau_2 \mid \sigma}{\Gamma \vdash \lambda x.t : \tau \mid unify\{\sigma\tau, \sigma\tau_1 \to \sigma\tau_2\} \circ \sigma}$$

CT-App:

$$\frac{\Gamma \vdash t_1 : \tau_1 \mid \sigma_1 \quad \sigma_1\Gamma \vdash t_2 : \sigma_1\tau_2 \mid \sigma_2}{\Gamma \vdash t_1\ t_2 : \tau \mid unify\{\sigma_2\sigma_1\tau_1, \sigma_2\sigma_1\tau_2 \to \sigma_2\sigma_1\tau\} \circ \sigma_2 \circ \sigma_1}$$

CT-Let-Mono:

$$\frac{\Gamma, x : \tau_1 \vdash t_1 : \tau_1 \mid \sigma_1 \quad \overline{\alpha} = ftv(\sigma_1\tau_1) - ftv(\sigma_1\Gamma) \quad \sigma_1\Gamma, x : \forall\overline{\alpha}.\sigma_1\tau_1 \vdash t_2 : \sigma_1\tau \mid \sigma_2}{\Gamma \vdash \text{let } x = t_1 \text{ in } t_2 : \tau \mid \sigma_2 \circ \sigma_1}$$

CT-MatchWith:

$$\frac{\Gamma \vdash t_s : \tau_p \quad \sigma_1\Gamma \vdash_p \overline{p \to t} : \sigma_1\tau_p \to \sigma_1\tau \mid \sigma_2}{\Gamma \vdash \text{match } t_s \text{ with } \overline{p \to t} : \tau \mid \sigma_2 \circ \sigma_1}$$

Typing rules for the matches (flat pattern) :

$$\frac{\Gamma, x : \tau_p \vdash t : \tau \mid \sigma}{\Gamma \vdash_p x \to t : \tau_p \to \tau \mid \sigma} \quad \text{(PVar)}$$

$$\frac{C : \forall\overline{\alpha}.\overline{\tau_1} \to T\ \overline{\alpha} \in \Gamma \quad \overline{\alpha_1}\#\text{everything} \quad \theta = \overline{[\alpha \mapsto \alpha_1]} \quad \Gamma, \overline{x : \theta\tau_1} \vdash t : \tau_t \mid \sigma}{\Gamma \vdash_p C\ \overline{x} \to t : \tau_p \to \tau_t \mid unify\{\tau_p, \overline{\theta\alpha}\} \circ \sigma} \quad \text{(PCon)}$$

Additional typing rules Typing rules for polymorphic recursion:

T-Let-Poly:

$$\frac{ftv(\forall\overline{\alpha}.\tau_1) = \emptyset \quad \Gamma, x : \forall\overline{\alpha}.\tau_1 \vdash t_1 : \tau_1 \quad \Gamma, x : \forall\overline{\alpha}.\tau_1 \vdash t_2 : \tau}{\Gamma \vdash \text{let } x : \forall\overline{\alpha}.\tau_1 = t_1 \text{ in } t_2 : \tau}$$

CT-Let-Poly:

$$\frac{\overline{\alpha}\#\Gamma \quad ftv(\forall\overline{\alpha}.\tau_1) = \emptyset \quad \Gamma, x : \forall\overline{\alpha}.\tau_1 \vdash t_1 : \tau_1 \mid \sigma_1 \quad \sigma_1\Gamma, x : \forall\overline{\alpha}.\tau_1 \vdash t_2 : \sigma_1\tau \mid \sigma_2}{\Gamma \vdash \text{let } x : \forall\overline{\alpha}.\tau_1 = t_1 \text{ in } t_2 : \tau \mid \sigma_2 \circ \sigma_1}$$

Generalized Algebraic Data Types Generalization of ADTs, allowing the parameters of the result type of a type constructor to be different from the quantified type variables. The main difficulties are the following:

1. GADTs lack of principal type.
2. Need polymorphic recursion.

Key Ideas: combine type inference with type checking. With the help of user defined types, we know exactly what type the term has, so we need only to check consistency.

Propagate type information To propagate user-defined type information, we need a new typing judgment $\Gamma \vdash t :^m \tau$. Here m is a modifier, indicating whether we've already known τ or not.

T-Var:

$$\frac{x :^n \forall\overline{\alpha}.\tau \in \Gamma}{\Gamma \vdash x :^m \tau' \text{ where } \tau' \text{ is an instance of } \forall\overline{\alpha}.\tau}$$

T-Abs:

$$\frac{\Gamma, x :^m \tau_1 \vdash t_1 :^m \tau_2}{\Gamma \vdash \lambda x.t_1 :^m \tau_1 \to \tau_2}$$

T-App:

$$\frac{\Gamma \vdash t_1 :^w \tau_1 \to \tau \quad \Gamma \vdash t_2 :^w \tau_1}{\Gamma \vdash t_1\ t_2 :^m \tau}$$

T-Let-Mono:

$$\frac{\Gamma, x :^w \tau_1 \vdash t_1 :^w \tau_1 \quad \overline{\alpha} = ftv(\tau_1) - ftv(\Gamma) \quad \Gamma, x :^w \forall\overline{\alpha}.\tau_1 \vdash t_2 :^m \tau}{\Gamma \vdash \text{let } x = t_1 \text{ in } t_2 :^m \tau}$$

T-Let-Poly:

$$\frac{ftv(\forall\overline{\alpha}.\tau_1) = \emptyset \quad \Gamma, x :^r \forall\overline{\alpha}.\tau_1 \vdash t_1 :^r \tau_1 \quad \Gamma, x :^r \forall\overline{\alpha}.\tau_1 \vdash t_2 :^m \tau}{\Gamma \vdash \text{let } x : \forall\overline{\alpha}.\tau_1 = t_1 \text{ in } t_2 :^m \tau}$$

When typing pattern matchings, if the type of the scrutinee of the match is already known, then we only need to the check the types of each branch rather than infer them. Hence two additional judgments : $\Gamma \vdash t : \tau \uparrow^m$ for infering whether the scrutinee's type is already known, $\Gamma \vdash p \to t :^{<m_s,m_t>} \tau_p \to \tau$ for checking/infering the types of the match branches when the type of the scrutinee is known/unknown.

T-Scr-R:

$$\frac{x :^m \tau \in \Gamma}{\Gamma \vdash x : \tau \uparrow^m}$$

T-Scr-Other

$$\frac{\Gamma \vdash t :^w \tau}{\Gamma \vdash t : \tau \uparrow^w}$$

T-Pattern-W:

$$\frac{C :^r \forall\overline{\alpha}.\overline{\tau}_1 \to T\ \overline{\tau}_2 \in \Gamma \quad \theta(\overline{\tau}_2) = \overline{\tau}_p \quad \Gamma, \overline{x :^w \theta(\tau_1)} \vdash t :^m \tau}{\Gamma \vdash C\ \overline{x} \to t :^{<w,m>} T\ \overline{\tau}_p \to \tau}$$

T-Pattern-R:

$$\frac{C :^r \forall\overline{\alpha}.\overline{\tau}_1 \to T\ \overline{\tau}_2 \in \Gamma \quad \theta = mgu(\overline{\tau}_2 = \overline{\tau}_p) \quad \theta\Gamma, \overline{x :^r \theta(\tau_1)} \vdash t :^m \theta^m\tau}{\Gamma \vdash C\ \overline{x} \to t :^{<r,m>} T\ \overline{\tau}_p \to \tau}$$

T-MatchWith:

$$\frac{\Gamma \vdash t_s : \tau_p \uparrow^{m_p} \qquad \Gamma \vdash \overline{p \to t} :^{<m_p,m>} \tau_p \to \tau}{\Gamma \vdash \text{match } t_s \text{ with } \overline{p \to t} :^m \tau}$$

Examples

$$\cfrac{\cfrac{\cfrac{\begin{array}{c} Leaf :^r \forall b.unit \to (b,z)\ gtree \\ \theta = [a \mapsto b, n \mapsto z] \qquad \theta(\Gamma), t :^r (b,z)\ gtree, \_ : Unit\ \vdash\ Z\ () :^r z \mid \sigma \end{array}}{\Gamma, t :^r (a,b)\ gtree \vdash Leaf\ \_\ \to\ Z\ () :^{<r,r>} (a,n)\ gtree \to n \mid \epsilon} \quad \cfrac{t :^r (a,b)\ gtree \in \Gamma, t :^r (a,b)}{\Gamma, t :^r (a,b)\ gtree \vdash t : (a,b)\ gtree \uparrow^r\ \mid \delta}}{\Gamma, t :^r (a,b)\ gtree \vdash \text{match } t \text{ with } Leaf\ \_\ \to Z\ () :^r n \mid \delta.}}{\Gamma \vdash \lambda t.\text{match } t \text{ with } Leaf\ \_\ \to Z\ () :^r (a,n)\ gtree \to\ n \mid \delta}$$

$$\Gamma \vdash \lambda x.\text{match } x \text{ with } Pair(p,q) \to p + 1 :^r a * b \to int$$

We give the proof of properties of type system without GADTs and polymorphic recursion. And we denote $\sim$ to be the equivalence relation of monotypes such that $\tau_1 \sim \tau_2$ if $\tau_1 = \tau_2$ modulo free variables (minus free variables in the context).

Lemma Substitution. If $\Gamma, x : \forall\overline{\alpha}.\tau_1 \vdash t : \tau_2$ where $\overline{\alpha}\#\Gamma$ and $\Gamma \vdash s : \tau_1$, then $\Gamma \vdash [x \mapsto s]t : \tau_2$

Proof. By induction on the derivation of $\Gamma, x : \forall\overline{\alpha}.\tau_1 \vdash t : \tau_2$.

1. $t = y, y : \forall\overline{\beta}.\tau_2 \in \Gamma$. If $y \neq x$, then $[x \mapsto s]y = y$, thus trivial. If $y = x$, then τ_2 is an instance of $\forall\overline{\alpha}.\tau_1$, therefore $\tau_1 \sim \tau_2$ and $\Gamma \vdash x : \tau_1$ is derivable.

2. $t = \lambda y.t_1, \Gamma, x : \forall\overline{\alpha}.\tau_1 \vdash \lambda y.t_1 : \tau_2 \to \tau$, then $\Gamma, x : \forall\overline{\alpha}.\tau_1, y : \tau_2 \vdash t_1 : \tau$, by inductive hypothesis $\Gamma, y : \tau_2 \vdash [x \mapsto s]t_1 : \tau$. Since by convention, $y\#\Gamma, x$, we get $\Gamma \vdash [x \mapsto s]\lambda y.t_1 : \tau_2 \to \tau$.

3. $t = t_1\ t_2$, $\Gamma, x : \forall\overline{\alpha}.\tau_1 \vdash t_1 : \tau_2 \to \tau$, $\Gamma, x : \forall\overline{\alpha}.\tau_1 \vdash t_2 : \tau_2$, directly from inductive hypothesis.

4. $t =$ let $y = t_2$ in t_3, $\Gamma, x : \forall\overline{\alpha}.\tau_1, y : \tau_2 \vdash t_2 : \tau_2, \Gamma, x : \forall\overline{\alpha}.\tau_1, y : \forall\overline{\beta}.\tau_2 \vdash t_3 : \tau$. Then by inductive hypothesis, $\Gamma, y : \forall\overline{\beta}.\tau_2 \vdash [x \mapsto s]t_3 : \tau$, whence $\Gamma \vdash [x \mapsto s]$let $y = t_2$ in $t_3 : \tau$

5. $t =$ match t_s with $C\ \overline{y} \to t$,$\Gamma, x : \forall\overline{\alpha}.\tau_1 \vdash t_2 : T\ \overline{\tau}_p$ and $\Gamma, x : \forall\overline{\alpha}.\tau_1 \vdash C\ \overline{y} \to t : T\ \overline{\tau}_p \to \tau$, and further $\Gamma, x : \forall\overline{\alpha}.\tau_1, \overline{y : \tau_p} \vdash t : \tau$, therefore $\Gamma, \overline{y : \tau_p} \vdash [x \mapsto s]t : \tau$, and finally $\Gamma \vdash [x \mapsto s]$match t_s with $C\ \overline{y} \to t : \tau$.

□

Theorem Preservation. If $\Gamma \vdash t : \tau, t \to t'$, then $\Gamma \vdash t' : \tau$

Proof. By induction on typing derivation.

1. If t is an abstraction, then it's trivial.

2. If $t = t_1\ t_2$, and $t \to t'$. We have $\Gamma \vdash t_1 : \tau_1 \to \tau$, $\Gamma \vdash t_2 : \tau_1$. If t_1 can be evaluated, or t_1 is already a value but t_2 can be evaluated, then it's totally routinized. If $t_1 = \lambda x.t_1'$ and t_2 is a value, we have $\Gamma, x : \tau_1 \vdash t_1' : \tau$, and $t \to [x \mapsto t_2]t_1'$. By substitution lemma we have $\Gamma \vdash [x \mapsto t_2]t_1' : \tau$

3. If $t =$ let $x = t_1$ in t_2, then $\Gamma, x : \tau_1 \vdash t_1 : \tau_1$, $\Gamma, x : \forall\overline{\alpha}.\tau_1 \vdash t_2 : \tau$. If t_1 can be further evaluated, then the conclusion is trivially satisfied. If t_1 turns out to be a value, then $t \to [x \mapsto$ fix $(\lambda x.t_1)]t_2$. From the given type judgment we have $\Gamma \vdash$ fix $(\lambda x.t_1) : \tau_1$, thus by applying substitution lemma we attain $\Gamma \vdash [x \mapsto$ fix $(\lambda x.t_1)]t_2 : \tau$

4. If $t =$ match t_s with $C\ \overline{x} \to t_1$, with $\Gamma \vdash t_s : T\ \overline{\tau_p}$ and $\Gamma \vdash C\ \overline{x} \to t_1 : T\ \overline{\tau_p} \to \tau$. If t_s can be further evaluated then it's trivial. If t_s is already a value, then it must be of the form $C\ \overline{v}$, and $t \to \overline{[x \mapsto v]}t_1$. By applying substitution lemma repeatedly we have $\Gamma \vdash \overline{[x \mapsto v]}t_1 : \tau$

$\square$

Theorem Progress. If $\Gamma \vdash t : \tau$, and any pattern matching in t has exhaustive matches, then either t is a value or $t \to t'$.

Proof. By induction on typing derivation.

1. If t is an abstraction then trivial.

2. If $t = t_1\ t_2$, then $\Gamma \vdash t_1 : \tau_1 \to \tau$, $\Gamma \vdash t_2 : \tau_1$. By inductive hypothesis, t_1 and t_2 have the progress property. If one of the two can be further evaluated then it's trivial. If both are values, since the sub-derivation tells us that t_1 has a function type, then t_1 must be of the form $\lambda x.t_1'$, whence $t \to [x \mapsto t_2]t_1'$

3. If $t = \text{ let } x = t_1 \text{ in } t_2$, similar with application.

4. If $t = \text{ match } t_s \text{ with } C\ \overline{x} \to t_1$, with $\Gamma \vdash t_s : T\ \overline{\tau_p}$ and $\Gamma \vdash C\ \overline{x} \to t_1 : T\ \overline{\tau_p} \to \tau$. (Here we suppose the scrutinee is always successfully matched). Then since t_s has the progress property, we suppose it is already a value of the form $C\ \overline{v}$, and therefore $t \to \overline{[x \mapsto v]}t_1$.

$\square$

Lemma Global Principality. If $\Gamma \vdash t : \tau$, then for any substitution σ such that $dom(\sigma)$ disjoint from the scoped type variables in Γ, $\sigma\Gamma \vdash t : \sigma\tau$

Proof. By induction on the typing derivation:

1. If $t = x$ with $x : \forall\overline{\alpha}.\tau_1 \in \Gamma$, then $\tau_1 \sim \tau$. Let θ be substitution such that $\theta(\tau_1) = \tau$, then $dom(\theta) = \overline{\alpha}$. For any such substitution σ, $x : \forall\overline{\alpha}.\sigma\tau_1 \in \sigma\Gamma$, hence $\sigma\Gamma \vdash x : \theta\sigma\tau_1$. By our assumption, it follows that $\sigma \circ \theta = \theta \circ \sigma$, therefore $\sigma\Gamma \vdash t : \sigma\theta\tau_1$ whence $\sigma\Gamma \vdash t : \sigma\tau$.

2. If t is an abstraction or application, then the conclusion is true by directly using inductive hypothesis.

3. If $t = \text{ let } x = t_1 \text{ in } t_2$, then $\Gamma, x : \tau_1 \vdash t_1 : \tau_1$, $\Gamma, x : \forall\overline{\alpha}.\tau_1 \vdash t_2 : \tau$. For any such substitution σ, since $\overline{\alpha} = ftv(\tau) - ftv(\Gamma)$, $dom(\sigma)$ disjoint from the scoped type variables in $\Gamma, x : \forall\overline{\alpha}.\tau_1$. By induction hypothesis $\sigma\Gamma, x : \sigma\tau_1 \vdash t_1 : \sigma\tau_1, \sigma\Gamma, x : \forall\overline{\alpha}.\sigma\tau_1 \vdash t_2 : \sigma\tau$, and thus the desired result.

4. If $t = \text{ match } t_s \text{ with } C\ \overline{x} \to t_1$ with $\Gamma \vdash t_s : T\ \overline{\tau_p}$ and $\Gamma \vdash C\ \overline{x} \to t_1 : T\ \overline{\tau_p} \to \tau$, then by directly applying the inductive hypothesis, we reach the desired result. Note that when typing the branches, the type variable bindings of the type constructor don't occur in $dom(\sigma)$ too, so a similar analysis as let-in expression is sufficient.

$\square$

Lemma Local Principality. If $\Gamma \vdash t : \tau$, then for any substitution σ such that $dom(\sigma) \subset ftv(\tau) - ftv(\Gamma)$, then $\Gamma \vdash t : \sigma\tau$. (That is, if $\tau \sim \tau_1$, then $\Gamma \vdash t : \tau_1$)

Proof. By induction on typing derivation.

1. If $t = x$ with $x : \forall\overline{\alpha}.\tau_1 \in \Gamma$, for any such substitution σ, by our assumption $dom(\sigma) \cap ftv(\forall\overline{\alpha}.\tau_1) = \emptyset$, therefore $\sigma\tau_1$ is an instance of $\forall\overline{\alpha}.\tau_1$, whence $\Gamma \vdash x : \sigma\tau_1$ is derivable.
2. If t is an abstraction or application, then it's a straight-forward use of inductive hypothesis.
3. If $t =$ let $x = t_1$ in t_2, then $\Gamma, x : \tau_1 \vdash t_1 : \tau_1$, $\Gamma, x : \forall\overline{\alpha}.\tau_1 \vdash t_2 : \tau$. By our assumption $dom(\sigma) \subset \overline{\alpha}$. Then by inductive hypothesis $\Gamma, x : \forall\overline{\alpha}.\tau_1 \vdash t_2 : \sigma\tau$. And thus $\Gamma \vdash t : \sigma\tau$.
4. If t is a pattern matching, then similar.

□

Theorem Inference Soundness. If $\Gamma \vdash t : \tau \mid \sigma$, then $\sigma\Gamma \vdash t : \sigma\tau$.

Proof. By induction on constraint typing derivation. Without loss of generality, we can always assume that the scoped variables in the context are chosen fresh.

1. If $t = x$ with $x : \forall\overline{\alpha}.\tau_1 \in \Gamma$. Let $\tau_2 = fresh(\forall\overline{\alpha}.\tau_1)$, $\delta = unify(\tau, \tau_2)$, and θ such that $\theta(\tau_1) = \tau_2$. Obviously we can always treat $\overline{\alpha}$ comletely fresh, therefore $\delta \backslash \overline{\alpha} = \delta$ Then $x : \forall\overline{\alpha}.\delta\tau_1 \in \delta\Gamma$, therefore $\delta\Gamma \vdash x : \theta\delta\tau_1$ is derivable. Hence $\delta\Gamma \vdash x : \delta\tau_2$ by $\theta\delta = \delta\theta$ and finally $\delta\Gamma \vdash x : \delta\tau$ by applying $\delta(\tau) = \delta(\tau_2)$.
2. If $t = \lambda x.t$ with $\Gamma, x : \tau_1 \vdash t : \tau_2 \mid \sigma$. By inductive hypothsis $\sigma\Gamma, x : \sigma\tau_1 \vdash t : \sigma\tau_2$ and it follows that $\sigma\Gamma \vdash \lambda x.t : \sigma\tau_1 \to \sigma\tau_2$. Then let $\delta = unify(\sigma\tau, \sigma\tau_1 \to \sigma\tau_2)$. By lemma global principality, we have $\delta\sigma\Gamma \vdash t : \delta(\sigma\tau_1 \to \sigma\tau_2)$, and it follows that $\delta\sigma\Gamma \vdash \delta\sigma\tau$
3. If t is an application then it's a straight-forward use of inducitve hypothesis.
4. If $t =$ let $x = t_1$ in t_2 with $\Gamma, x : \tau_1 \vdash t_1 : \tau_1 \mid \sigma_1$, $\quad \overline{\alpha} = ftv(\sigma_1\tau_1) - ftv(\sigma_1\Gamma)$, $\quad \sigma_1\Gamma, x : \forall\overline{\alpha}.\sigma_1\tau_1 \vdash t_2 : \sigma_1\tau \mid \sigma_2$. Then by inductive hypothesis and the global principality lemma, $\sigma_2\sigma_1\Gamma, x : \tau_1 \vdash t_1 : \sigma_2\sigma_1\tau_1$, and $\sigma_2\sigma_1\Gamma, x : \forall\overline{\alpha}.\sigma_2\sigma_1\tau_1 \vdash t_2 : \sigma_2\sigma_1\tau$, therefore the desired result.
5. If t is a pattern matching, and $\Gamma \vdash t_s : \tau_p \mid \sigma_1 \quad \sigma_1\Gamma \vdash_p \overline{p \to t} : \sigma_1\tau_p \to \sigma_1\tau \mid \sigma_2$, then we can directly apply the inductive hypothesis.
6. Consider typing the branches. Then $C : \forall\overline{\alpha}.\overline{\tau_1} \to T\ \overline{\alpha} \in \Gamma \quad \overline{\alpha_1}\#\text{everything} \quad \overline{\theta = [\alpha \mapsto \alpha_1]} \quad \Gamma, \overline{x : \theta(\tau_1)} \vdash t : \overline{\tau_t \mid \sigma}$. Then by inductive hypothesis, $\sigma\Gamma, \overline{x : \sigma\theta(\tau_1)} \vdash t : \sigma\tau_t$. Let $\delta = unify\{\sigma\tau_p, \overline{\sigma\theta(\alpha)}\}$, then further $\delta\sigma\Gamma, \overline{x : \delta\sigma\theta(\tau_1)} \vdash t : \delta\sigma\tau_t$. By our assumption, $(dom(\delta) \cup dom(\sigma)) \cap \overline{\alpha} = \emptyset$, thus $\delta\sigma\theta(\tau_1) = \theta\delta\sigma(\tau_1)$. On the other hand, $\theta(\delta\sigma(\overline{\alpha})) = \delta(\overline{\sigma\theta(\alpha)}) = \delta\sigma\tau_p$, hence we get the desired result.

□

Lemma Weakening. If $\Gamma, \overline{y : \tau_1} \vdash t : \tau \mid \delta$, then for a set of fresh type variables, $\Gamma, \overline{y : \nu} \vdash t : \alpha \mid \delta'$, and $\exists\phi$, $\phi\delta'\alpha = \delta\tau$, $\phi\delta'\overline{\nu} = \delta\overline{\tau}_1$

Proof. By induction on the constraint typing derivation.

1. If $t = z$ is a variable. When $t \not\in \overline{y}$, it's trivial. If $t \in \overline{y}$, without loss of generality let $\delta' = [\alpha \mapsto \nu]$, let $\phi = [\nu \mapsto \delta\tau]$, then the conclusion is trivially satisfied.

2. If $t = \lambda z.t_1$, and $\Gamma, \overline{y : \tau_1}, z : \tau_2 \vdash t_1 : \tau_3 \mid \theta$, $\delta = unify\{\theta\tau, \theta\tau_2 \to \theta\tau_3\} \circ \theta$. Let $\eta = unify\{\theta\tau, \theta\tau_2 \to \theta\tau_3\}$. Then by inductive hypothesis, let $\overline{\nu}, \gamma, \beta$ be a set of fresh variables, $\Gamma, \overline{y : \nu}, z : \gamma \vdash t_1 : \beta \mid \theta'$, $\delta' = unify\{\theta'\alpha, \theta'\gamma \to \theta'\beta\} \circ \theta$and there exists κ such that $\kappa\theta'\beta = \theta\tau_3$, $\kappa\theta'\gamma = \theta\tau_2$ and $\kappa\theta'\nu = \theta\tau_1$. Consider $unify\{\theta'\alpha, \theta'\gamma \to \theta'\beta\}$, since α is completely fresh, we can assert that this unifier $= [\alpha \mapsto \theta'\gamma \to \theta'\beta]$. Therefore $\delta'\alpha = \theta'\gamma \to \theta'\beta$. On the other hand, $\delta\tau = \eta\theta\tau = \eta(\theta\tau_2 \to \theta\tau_3) = \eta\kappa(\theta'\gamma \to \theta'\beta) = \eta\kappa\delta'\alpha$. Finally we need to check other fresh variables. $\delta'\nu = \theta'\nu$ is trivially satisfied, on the other hand, $\delta\tau_1 = \eta\theta\tau_1 = \eta\kappa\theta'\nu$, therefore $\delta\tau_1 = \eta\kappa\delta'\nu$. The checking for the variable γ is similar.

3. If $t = t_1\ t_2$, then $\Gamma \vdash t_1 : \tau_1 \mid \sigma_1 \quad \sigma_1\Gamma \vdash t_2 : \sigma_1\tau_2 \mid \sigma_2$ (include the binding of y's into the context). Let $\theta = unify\{\sigma_2\sigma_1\tau_1, \sigma_2\sigma_1\tau_2 \to \sigma_2\sigma_1\tau\}$ and $\theta' = unify\{\sigma_2\sigma_1\tau_1, \sigma_2\sigma_1\tau_2 \to \alpha\}$. Since θ is a unifier, $\sigma_2\sigma_1\tau_1$ must be of the form $\tau_3 \to \tau_4$. Let $\phi = unify\{\sigma_2\sigma_1\tau_2, \tau_3\}$, then $\theta = unify\{\phi\tau_4, \phi\sigma_2\sigma_1\tau\} \circ \phi$ and $\theta' = [\alpha \mapsto \phi\tau_4] \circ \phi$. Then directly $\delta'\alpha = \theta'\alpha = \phi\tau_4$, and $\delta\tau = \theta\sigma_2\sigma_1\tau = unify\{\phi\tau_4, \phi\sigma_2\sigma_1\tau\}(\phi\sigma_2\sigma_1\tau) = unify\{\phi\tau_4, \phi\sigma_2\sigma_1\tau\}(\phi\tau_4)$

4. If t is a let-in expression or pattern matching, then it's similar to abstraction.

□

Theorem Inference Comleteness. If $\Gamma \vdash t : \tau$, then $\Gamma \vdash t : \tau' \mid \sigma$ for some fresh τ', where $\exists\phi$, $\tau = \phi\sigma\tau'$.

Proof. By induction on the typing derivation.

1. If $t = x$, and $x : \forall\overline{\alpha}.\tau_1 \in \Gamma$, then the constraint typing always succeeds with the unifier δ such that it works only on the scoped type variables $\overline{\alpha}$.

2. If $t = \lambda y.t_1, \Gamma \vdash \lambda y.t_1 : \tau_2 \to \tau$ with $\Gamma, y : \tau_2 \vdash t_1 : \tau$. By inductive hypothesis $\Gamma, y : \tau_2 \vdash t_1 : \tau' \mid \sigma$, and $\tau = \phi\sigma\tau'$. Thus $\Gamma \vdash \lambda y.t_1 : \tau_2 \to \tau \mid unify\{\sigma(\tau_2 \to \tau), \sigma(\tau_2 \to \tau'\} \circ \sigma$. By the property of unification algorithm, we have $\delta = unify\{\sigma(\tau_2 \to \tau), \sigma(\tau_2 \to \tau')\} = unify\{\sigma\tau, \sigma\tau'\}$.

□