Typage en FUN.

1 Définition du système de types.

L'ensemble Typ des types, notés $\tau, \tau_1, \tau', \ldots$, est définit par la grammaire suivante :

$$\tau ::= \texttt{int} \mid \tau_1 \to \tau_2.$$

Note 1. Attention ! Le type $\tau_1 \to \tau_2 \to \tau_3$ n'est pas égal au type $(\tau_1 \to \tau_2) \to \tau_3$. En effet, dans le premier cas, c'est une fonction qui renvoie une fonction ; et, dans le second cas, c'est une fonction qui prend une fonction.

Définition 1. Un *environnent de typage*, noté $\Gamma, \Gamma_1, \Gamma', \ldots$, est un dictionnaire sur $(\mathscr{V}, \mathsf{Typ})$, où Typ est l'ensemble des types.

Une *hypothèse de typage*, notée $x : \tau$, est un couple (x, τ).

On note $\Gamma, x : \tau$ l'extension de Γ avec l'hypothèse de typage $x : \tau$ qui n'est définie que lorsque $x \notin \operatorname{dom} \Gamma$. [1]

Remarque 1. On peut voir/implémenter Γ comme des listes finies de couples (x, τ).

Définition 2. La *relation de typage*, notée $\Gamma \vdash e : \tau$ (« sous les hypothèses Γ, l'expression e a le type τ ») est définie par les règles d'inférences suivantes.

1. La définition de $\Gamma, x : \tau$ est « comme on le pense ».

$$\dfrac{}{\Gamma \vdash k : \texttt{int}}\ \mathscr{T}_\mathrm{k} \qquad \dfrac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau}\ \mathscr{T}_\mathrm{v} \qquad \dfrac{\Gamma, x : \tau_1 \text{[2]} \vdash e_2}{\Gamma \vdash \texttt{fun}\, x \to e : \tau_1 \to \tau_2}\ \mathscr{T}_\mathrm{f}$$

$$\dfrac{\Gamma \vdash e_1 : \texttt{int} \qquad \Gamma \vdash e_2 : \texttt{int}}{\Gamma \vdash e_1 + e_2 : \texttt{int}}\ \mathscr{T}_\mathrm{p} \qquad \dfrac{\Gamma \vdash e : \tau_1 \to \tau_2 \qquad \Gamma \vdash e' : \tau_1}{\Gamma \vdash e\ e' : \tau_2}\ \mathscr{T}_\mathrm{a}.$$

Remarque 2. Pour l'instant, on parle uniquement d'expressions et pas du tout de valeurs ou de sémantique opérationnelle.

Remarque 3.
1. On dit que e est *typable* s'il existe Γ et τ tel que $\Gamma \vdash e : \tau$.
2. Il y a une règle de typage par construction du langage des expressions.

Exemple 1.
1. L'expression $\texttt{fun}\, x \to x$ est particulière : on peut la typer avec $\tau \to \tau$ quel que soit τ. Par exemple,

$$\dfrac{\dfrac{}{x : \texttt{int} \vdash x : \texttt{int}}\ \mathscr{T}_\mathrm{v}}{\emptyset \vdash \texttt{fun}\, x \to x : \texttt{int} \to \texttt{int}}\ \mathscr{T}_\mathrm{f}.$$

   On aurait pu faire de même avec le type $(\texttt{int} \to \texttt{int}) \to (\texttt{int} \to \texttt{int})$.
2. Quel est le type de $\texttt{fun}\, g \to g\ (g\ 7)$?

$$\dfrac{\dfrac{\dfrac{}{g : \texttt{int} \to \texttt{int} \vdash g : \texttt{int} \to \texttt{int}}\ \mathscr{T}_\mathrm{v} \qquad \dfrac{\dfrac{}{\Gamma \vdash g : \texttt{int} \to \texttt{int}}\ \mathscr{T}_\mathrm{v} \qquad \dfrac{}{\Gamma \vdash 7 : \texttt{int}}\ \mathscr{T}_\mathrm{k}}{g : \texttt{int} \to \texttt{int} \vdash g\ 7 : \texttt{int}}\ \mathscr{T}_\mathrm{p}}{g : \texttt{int} \to \texttt{int} \vdash g\ (g\ 7)}\ \mathscr{T}_\mathrm{a}}{\emptyset \vdash \texttt{fun}\, g \to g\ (g\ 7) : (\texttt{int} \to \texttt{int}) \to \texttt{int}}\ \mathscr{T}_\mathrm{f}.$$

2. On peut toujours étendre Γ ainsi, modulo α-conversion.

2 Propriétés du système de types.

Lemme 1.
- ▷ Si $\Gamma \vdash e : \tau$ alors $\mathscr{V}\ell(e) \subseteq \operatorname{dom}(\Gamma)$.
- ▷ *Affaiblissement.* Si $\Gamma \vdash e : \tau$ alors

$$\forall x \notin \operatorname{dom}(\Gamma),\ \forall \tau_0, \quad \Gamma, x : \tau_0 \vdash e : \tau.$$

- ▷ *Renforcement.* Si $\Gamma, x : \tau_0 \vdash e : \tau$, et si $x \notin \mathscr{V}\ell(e)$ alors on a le typage $\Gamma \vdash e : \tau$.

Preuve. Par induction sur la relation de typage (5 cas). □

2.1 Propriété de progrès.

Lemme 2.
1. Si $\emptyset \vdash e : \texttt{int}$ et $e \not\to$ alors, il existe $k \in \mathbb{Z}$ tel que $e = k$.
2. Si $\emptyset \vdash e : \tau_1 \to \tau_2$ et $e \not\to$ alors il existe x et e_0 tels que l'on ait $e = \texttt{fun}\, x \to e_0$.

Preuve. Vu en TD. □

Proposition 1 (Propriété de progrès). Si $\emptyset \vdash e : \tau$ alors on a la disjonction :

1. soit e est une valeur ;
2. soit il existe e' telle que $e \to e'$.

Remarque 4.
- ▷ Si $\emptyset \vdash e_1\ e_2 : \tau$ alors il existe e' tel que $e_1\ e_2 \to e'$.
- ▷ Si $\emptyset \vdash e_1 + e_2 : \tau$ alors il existe e' tel que $e_1 + e_2 \to e'$.

Remarque 5. Par le typage, on a exclu les expressions bloquées car « mal formées » (*e.g.* $3\ 2$ ou $3 + (\texttt{fun}\, x \to x)$).

2.2 Propriété de préservation.

Cette propriété a plusieurs noms : préservation du typage, réduction assujettie, *subject reduction*.

Lemme 3 (typage et substitution). Si l'on a le typage $\emptyset \vdash v : \tau_0$ et $\Gamma, x : \tau_0 \vdash e : \tau$ alors on a $\Gamma \vdash e[{}^v/_x] : \tau$

Preuve. On prouve cette propriété par induction sur e. Il y a 5 cas.

- ▷ *Cas* $e = y$. On a deux sous-cas.
  - *1^er^ sous-cas* $x \neq y$. Dans ce cas, $e[{}^v/_x] = y$. Il faut montrer $\Gamma \vdash y : \tau$ sachant que $\Gamma, x : \tau_0 \vdash y : \tau$. On applique le lemme de renforcement.
  - *2^nd^ sous-cas* $x = y$. Dans ce cas, $e[{}^v/_x] = v$. Il faut montrer que $\Gamma \vdash v : \tau$. Or, on sait que $\Gamma, x : \tau_0 \vdash x : \tau$ (d'où $\tau = \tau_0$) et $\emptyset \vdash v : \tau_0$. On conclut par affaiblissement.
- ▷ Les autres cas sont en exercice.

□

Proposition 2 (Préservation du typage). Si $\emptyset \vdash e : \tau$, et $e \to e'$ alors $\emptyset \vdash e' : \tau$.

Preuve. On montre la propriété par induction sur $\emptyset \vdash e : \tau$. Il y a 5 cas.

- ▷ *Cas* $\mathcal{T}_\mathrm{v}$. C'est absurde ! (On n'a pas $\emptyset \vdash x : \tau$.)
- ▷ *Cas* $\mathcal{T}_\mathrm{f}$. Si $(\texttt{fun}\, x \texttt{ -> } e) \to e'$ alors ... On peut conclure immédiatement car les fonctions sont des valeurs, elles ne se réduisent donc pas.
- ▷ *Cas* $\mathcal{T}_\mathrm{k}$. C'est le même raisonnement.
- ▷ *Cas* $\mathcal{T}_\mathrm{a}$. On a $e = e_1\ e_2$. On sait qu'il existe τ_0 un type tel que $\emptyset \vdash e_1 : \tau_0 \to \tau$ (H_1) et $\emptyset \vdash e_2 : \tau_0$ (H_2). On a également

les hypothèses d'induction :

- (H_1') : si $e_1 \to e_1'$ alors $\emptyset \vdash e_1' : \tau_0 \to \tau$;
- (H_2') : si $e_2 \to e_2'$ alors $\emptyset \vdash e_2' : \tau_0$.

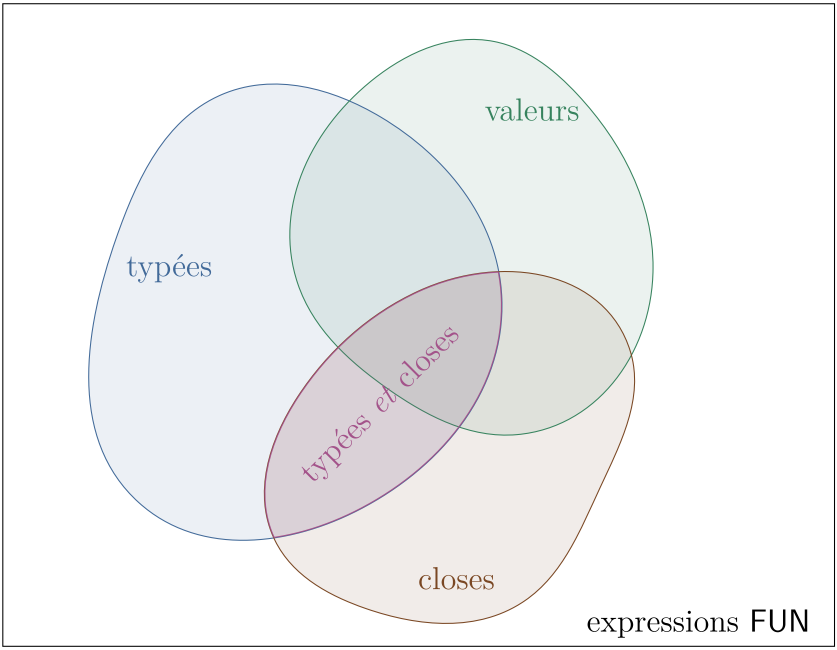
On doit montrer que si $e_1\ e_2 \to e'$ alors $\emptyset \vdash e' : \tau$. Supposons que $e_1\ e_2 \to e'$, il y a 3 sous-cas.

- *Sous-cas* $\mathscr{R}_{\mathrm{ad}}$. Cela veut dire que $e_2 \to e_2'$ et $e' = e_1\ e_2'$. On conclut $\emptyset \vdash e_1\ e_2' : \tau$ par (H_2') et (H_1).
- *Sous-cas* $\mathscr{R}_{\mathrm{ag}}$. Cela veut dire que $e_1 \to e_1'$ et $e' = e_1'\ e_2$. On conclut $\emptyset \vdash e_1'\ e_2 : \tau$ par (H_1') et (H_2).
- *Sous-cas* $\mathscr{R}_\beta$. On a $e_1 = \mathtt{fun}\, x \to e_0$, $e_2 = v$ et finalement $e' = e_0[{}^v/_x]$. On doit montrer $\emptyset \vdash e_0[{}^v/_x] : \tau$. De plus, (H_1) s'énonce par $\emptyset \vdash \mathtt{fun}\, x \to e_0 : \tau_0 \to \tau$. Nécessairement (c'est un « `inversion` » en Rocq), cela provient de $x : \tau_0 \vdash e_0 : \tau$. On en conclut par le lemme de substitution.

▷ *Cas* $\mathscr{T}_{\mathrm{p}}$. Laissé en exercice.

□

Remarque 6. Avec les propriétés de progrès et préservation implique qu'il n'y a pas de « mauvaises surprises » à l'exécution. On a, en un sens, nettoyé le langage FUN.



C'est la considération d'un langage *statiquement typé*. On aime savoir qu'OCaml ou Rust ont, pour la sémantique et le système de types, une propriété de progrès et de préservation.

Exercice 1. Trouver e et e' deux expressions telles que $\emptyset : e' : \tau$ et $e \to e'$ mais que l'on ait pas $\emptyset \vdash e : \tau$.

Solution. Il suffit de trouver une valeur non typable e_1, par exemple $\texttt{fun}\, x \to (x\ x)$ ou $\texttt{fun}\, x \to (19\ 27)$, puis de considérer

$$e = (\texttt{fun}\, x \to 3)\ e_1 \to 3.$$

Or, 3 est typable mais e non.

3 Questions en lien avec la relation de typage.

▷ *Typabilité.* Pour e donné, existe-t-il Γ, τ tels que $\Gamma \vdash e : \tau$?

- *Vérification/Inférence de types.* Pour Γ et e donnés, existe-t-il τ tel que l'on ait $\Gamma \vdash e : \tau$? (▷ OCaml)
- *Habitation.* Pour τ donné, existe-t-il e tel que $\emptyset \vdash e : \tau$? (▷ Rocq[3])

4 Inférence de types.

4.1 Typage et contraintes.

Exemple 2. Dans une version étendue de FUN (on se rapproche plus au OCaml), si l'on considère le programme :

$$\begin{aligned}&\texttt{let rec } f\ x\ g\texttt{=}\\&\quad \dots g\ x\ \dots\\&\quad \dots \texttt{if } g\ f \texttt{ then } \dots \texttt{ else } \dots\\&\quad \dots \texttt{let } h \texttt{ = } x\ 7 \texttt{ in } \dots\end{aligned}$$

On remarque que

- x et f ont le même type ;
- g a un type $? \to \texttt{bool}$;
- x a un type $\texttt{int} \to ?$.

On doit donc lire le programme, et « prendre des notes ». Ces « notes » sont des contraintes que doivent vérifier le programme.

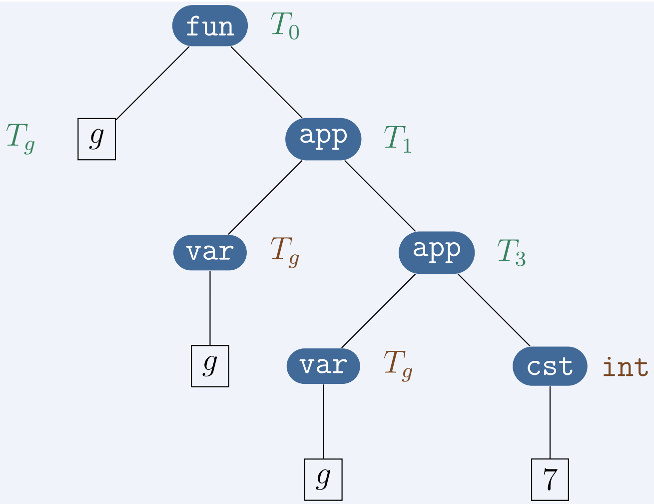
Exemple 3. On souhaite déterminer le type τ tel que

$$\emptyset \vdash \texttt{fun}\, g \texttt{ -> } g\ (g\ 7) : \tau.$$

(On sait que $\tau = (\texttt{int} \to \texttt{int}) \to \texttt{int}$.)

On construit l'arbre de l'expression (l'AST) :

3. On peut voir une preuve d'un théorème en Rocq comme fournir une preuve qu'il existe une expression e avec type τ.



On procède en plusieurs étapes :

1. On ajoute des inconnues de types T_1, T_2 T_3, *etc* (en vert).
2. On écrit des contraintes faisant intervenir les T_i (en orange/marron).

$$T_0 = T_g \to T_1$$
$$T_g = T_2 \to T_1$$
$$T_g = \texttt{int} \to T_1.$$

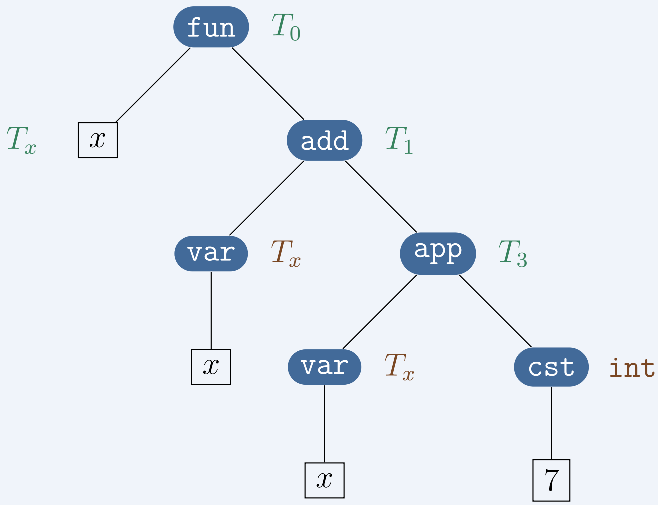
3. On résout les contraintes pour obtenir

$$T_0 = (\texttt{int} \to \texttt{int}) \to \texttt{int}.$$

Exemple 4 (Cas limites). ▷ L'expression $\texttt{fun}\, x \to 7$ admet une infinité de types ($T_x \to \texttt{int}$).
▷ L'expression $(\texttt{fun}\, x \to 7)\ (\texttt{fun}\, z \to z)$ a toujours le type `int` mais admet une infinité de dérivations.

Exemple 5 (Et quand ça ne marche pas ?). On essaie d'inférer le type de l'expression

$$\texttt{fun}\, x \to x + (x\ 2).$$



Les contraintes sont :

$$\begin{gathered} T_0 = T_x \to T_1 \\ T_1 = T_x = T_2 = \texttt{int} \\ T_x = \texttt{int} \to T_2. \end{gathered}$$

Catastrophe ! On ne peut pas résoudre ce système de contraintes (on ne peut pas avoir $T_x = \texttt{int}$ et $T_x = \texttt{int} \to T_2$ en même temps). L'expression n'est donc pas typable.

Définition 3. ▷ On se donne un ensemble infini IType d'*inconnues de type*, notées T, T_1, T', *etc.*

▷ On définit les *types étendus*, notés $\hat{\tau}$, par la grammaire :

$$\hat{\tau} ::= \texttt{int} \mid \hat{\tau}_1 \to \hat{\tau}_2 \mid T.$$

- L'ensemble des types (*resp.* étendus) est noté Typ (*resp.* $\widehat{\mathsf{Typ}}$).
- Les environnement de types étendus sont notés $\widehat{\Gamma}$.
- Ainsi défini, tout τ est un $\hat{\tau}$, tout Γ est un $\widehat{\Gamma}$.
- Un $\hat{\tau}$ est dit *constant* s'il ne contient pas d'inconnue de type (*i.e.* si c'est un τ).

Définition 4. Une *contrainte de typage* est une paire de types étendus[4], notée $\hat{\tau}_1 \overset{?}{=} \hat{\tau}_2$, ou parfois $\hat{\tau}_1 = \hat{\tau}_2$.

On se donne $e \in \mathsf{FUN}$. On suppose que toutes les variables liées de e sont :

- distinctes deux à deux ;
- différentes de toutes les variables libres de e.

On se donne $\widehat{\Gamma}$ tel que $\mathscr{V}\ell(e) \subseteq \mathrm{dom}(\widehat{\Gamma})$. On choisit $T \in \mathrm{IType}$.

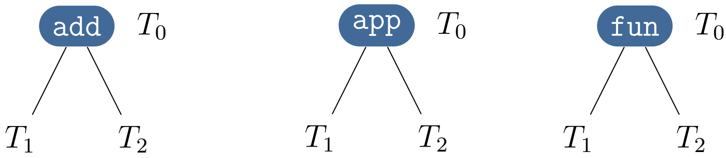
On définit un ensemble de contraintes, notée $\mathsf{CT}(e, \widehat{\Gamma}, T)$ par induction sur e, il y a 5 cas :

- $$\mathsf{CT}(e_1 + e_2, \widehat{\Gamma}, T) = \begin{aligned}[t] &\mathsf{CT}(e_1, \widehat{\Gamma}, T_1) \cup \mathsf{CT}(e_2, \widehat{\Gamma}, T_2) \\ &\cup \{T_1 \overset{?}{=} \mathtt{int}, T_2 \overset{?}{=} \mathtt{int}, T \overset{?}{=} \mathtt{int}\}\end{aligned}$$
- $$\mathsf{CT}(e_1\ e_2, \widehat{\Gamma}, T) = \begin{aligned}[t] &\mathsf{CT}(e_1, \widehat{\Gamma}, T_1) \cup \mathsf{CT}(e_2, \widehat{\Gamma}, T_2) \\ &\cup \{T_1 \overset{?}{=} T_2 \to T\}\end{aligned}$$
- $\mathsf{CT}(x, \widehat{\Gamma}, T) = \{T \overset{?}{=} \widehat{\Gamma}(x)\}$
- $\mathsf{CT}(k, \widehat{\Gamma}, T) = \{T \overset{?}{=} \mathtt{int}\}$
- $$\mathsf{CT}(\mathtt{fun}\, x \to e, \widehat{\Gamma}, T) = \begin{aligned}[t] &\mathsf{CT}(e, (\widehat{\Gamma}, x : T_x), T_2) \\ &\cup \{T \overset{?}{=} T_1 \to T_2\}\end{aligned}$$

où les variables T_1, T_2, T_x sont *fraîches* (on notera par la suite И T_1, T_2, T_x).

4. ***Attention*** c'est une paire, pas un couple.

Remarque 7. On peut résumer les cas « plus », « application » et « abstraction ».



$$T_0 = T_1 = T_2 = \texttt{int} \qquad T_1 = T_2 \to T_0 \qquad T_0 = T_1 \to T_2$$

Définition 5. Soit C un ensemble de contraintes de typage. On note $\mathrm{Supp}(C)$, le *support* de C, l'ensemble des inconnues de type mentionnées dans C.

Une *solution* σ de C est un dictionnaire sur $(\mathrm{ITyp}, \widehat{\mathsf{Typ}})$ tel que $\mathrm{dom}(\sigma) \supseteq \mathrm{Supp}(C)$ et que σ égalise toutes les contraintes de C.

Pour $(\hat{\tau}_1 \stackrel{?}{=} \hat{\tau}_2) \in C$, on dit que σ égalise $\hat{\tau}_1 \stackrel{?}{=} \hat{\tau}_2$ signifie que $\sigma(\hat{\tau}_1)$ et $\sigma(\hat{\tau}_2)$ sont le même type étendu.

Il reste à définir $\sigma(\hat{\tau})$, le résultat de l'application de σ à $\hat{\tau}$, par induction sur $\hat{\tau}$, il y a trois cas :

- ▷ $\sigma(\hat{\tau}_1 \to \hat{\tau}_2) = \sigma(\hat{\tau}_1) \to \sigma(\hat{\tau}_2)$;
- ▷ $\sigma(\texttt{int}) = \texttt{int}$;
- ▷ $\sigma(T)$ est le type étendu associé à T dans σ.

Exemple 6. Avec $\sigma = [T_1 \mapsto \texttt{int}, T_2 \mapsto (\texttt{int} \to T_3)]$, on a donc

$$\sigma(T_1 \to T_2) = \texttt{int} \to (\texttt{int} \to T_3).$$

Exemple 7. La contrainte $T_1 \stackrel{?}{=} T_2 \to T_3$ est égalisée par la solution $\sigma = [T_1 \mapsto T_2 \to \texttt{int}, T_3 \mapsto \texttt{int}]$.

Définition 6. Une *solution constante* de C est un dictionnaire sur $(\mathrm{ITyp}, \mathsf{Typ})$ (et pas $(\mathrm{ITyp}, \widehat{\mathsf{Typ}})$) qui est une solution de C.

Proposition 3. Soit $e \in \mathsf{FUN}$ et soit Γ tel que $\mathscr{V}\ell(e) \subseteq \mathrm{dom}(\Gamma)$. Soit $T \in \mathrm{ITyp}$. Si σ est une solution constante de $\mathsf{CT}(e, \Gamma, T)$, alors $\Gamma \vdash e : \tau$ où $\tau = \sigma(T)$.

Preuve. On procède par induction sur e ; il y a 5 cas.

- ▷ Dans le cas $e = e_1\ e_2$, on écrit

  $$\mathsf{CT}(e, \Gamma, T) = \mathsf{CT}(e_1, \Gamma, T_1) \cup \mathsf{CT}(e_2, \Gamma, T_2) \cup \{T_1 \stackrel{?}{=} T_2 \to T\},$$

  où И T_1, T_2. Soit σ une solution constante de $\mathsf{CT}(e, \Gamma, T)$. Alors,

  – σ est une solution constante de $\mathsf{CT}(e_1, \Gamma, T_1)$;
  – σ est une solution constante de $\mathsf{CT}(e_2, \Gamma, T_1)$.

  Et, par induction, on sait que

  – $\Gamma \vdash e_1 : \sigma(T_1)$;
  – $\Gamma \vdash e_2 : \sigma(T_2)$.

  Par ailleurs, $\sigma(T_1) = \sigma(T_2) \to \sigma(T)$. On en conclut en appliquant $\mathscr{T}_{\mathrm{a}}$.
- ▷ Les autres cas se traitent similairement.

□

Proposition 4. Supposons $\Gamma \vdash e : \tau$. Alors, pour tout $T \in \mathrm{ITyp}$, il existe σ une solution constante de $\mathsf{CT}(e, \Gamma, T)$ telle que l'on ait l'égalité $\sigma(T) = \tau$.

Preuve. On procède par induction sur e. Il y a 5 cas.

- ▷ Dans le cas $e = e_1\ e_2$, supposons $\Gamma \vdash e_1\ e_2 : \tau$. Nécessairement, cette dérivation provient de $\Gamma \vdash e_1 : \tau_2 \to \tau$ et aussi $\Gamma \vdash e_2 : \tau_2$.

Soit $T_0 \in \text{ITyp}$, on a

$$\mathsf{CT}(e, \Gamma, T_0) = \mathsf{CT}(e_1, \Gamma, T_1) \cup \mathsf{CT}(e_2, \Gamma, T_2) \cup \{T_1 \stackrel{?}{=} T_2 \to T_0\}.$$

Et, par induction, on a σ_1 et σ_2 des solutions constantes de $\mathsf{CT}(e_1, \Gamma, T_1)$ et $\mathsf{CT}(e_2, \Gamma, T_2)$ avec $\sigma_1(T_1) = \tau_2 \to \tau$ et $\sigma_2(T_2) = \tau_2$.

On définit σ en posant :

- $\sigma(T) = \sigma_1(T)$ si $T \in \text{Supp}(\mathsf{CT}(e_1, \Gamma, T_1))$;
- $\sigma(T) = \sigma_2(T)$ si $T \in \text{Supp}(\mathsf{CT}(e_2, \Gamma, T_2))$;
- $\sigma(T_0) = \tau$.

On vérifie bien que σ est solution constante de $\mathsf{CT}(e, \Gamma, T_0)$.

▷ Les autres cas se traitent similairement.

□

Théorème 1. On a $\Gamma \vdash e : \tau$ si, et seulement si $\forall T \in \text{ITyp}$, l'ensemble de contraintes $\mathsf{CT}(e, \Gamma, T)$ admet une solution constante σ tel que $\sigma(T) = \tau$. □

Remarque 8. On a caractérisé l'ensemble des dérivations de $\Gamma \vdash e : \tau$ avec l'ensemble des solutions constantes de $\mathsf{CT}(e, \Gamma, T)$.

4.2 Termes et unification.

On va momentanément oublier FUN, pour généraliser à tout ensemble d'expressions. Ceci permet d'appliquer cet algorithme à une grande variété de « langages ».

Définition 7. On se donne

▷ un ensemble fini Σ de *constantes*, notées f, g, a, b où chaque constante $f \in \Sigma$ a un entier naturel nommé *arité* ;

▷ un ensemble infini V d'*inconnues*/de *variables*/de *variables* *d'unification* ; notées X, Y, Z (mais parfois x, y, z).

L'ensemble $\mathsf{T}(\Sigma, \mathrm{V})$ des *termes* sur (Σ, V), notés t, u, *etc*, est défini de manière inductive, ce qui est décrit par la grammaire :

$$t ::= f^k(t_1, \ldots, t_k) \mid X,$$

où f est une constante d'arité k.

Remarque 9. L'intuition est que l'on étend, comme lors du passage de Typ à $\widehat{\mathsf{Typ}}$, un langage de départ pour ajouter des inconnues. La définition inductive a $|\Sigma| + 1$ constructeurs.

Intuitivement, les $X \in \mathrm{V}$ ne fait pas partie du langage de départ. Il n'y a pas de liens pour X.

Exemple 8. Avec $\Sigma = \{f^2, g^1, a^0, b^0\}$,

$$t_0 := f(g(a), f(X, f(Y, g(X)))) \in \mathsf{T}(\Sigma, V)$$

est un terme.

Définition 8. On définit $\mathsf{Vars}(t)$ l'ensemble des inconnues/variables de t par induction sur t. Il y a deux familles de cas :

- ▷ $\mathsf{Vars}(f(t_1, \ldots, t_k)) = \mathsf{Vars}(t_1) \cup \cdots \cup \mathsf{Vars}(t_k)$;
- ▷ $\mathsf{Vars}(X) = \{X\}$.

Exemple 9. Avec l'expression t_0 précédente, on a

$$\mathsf{Vars}(t_0) = \{X, Y\}.$$

Définition 9. Une *substitution*, notée $\sigma, \sigma_1, \sigma'$, *etc*, est un dictionnaire sur $(\mathrm{V}, \mathsf{T}(\Sigma, \mathrm{V}))$.

Si $X \in \mathrm{dom}(\sigma)$, on dit que σ est *définie* en X.

Soit σ une substitution et $t \in \mathsf{T}(\Sigma, \mathrm{V})$. Le résultat de l'application de σ à t, noté $\sigma(t)$, est défini par induction sur t, il y a deux familles de cas :

- ▷ $\sigma(f(t_1, \dots, t_k)) = f(\sigma(t_1), \dots, \sigma(t_k))$;
- ▷ $\sigma(X) = X$ si $X \notin \mathrm{dom}(\sigma)$;
- ▷ $\sigma(X)$ est le terme associé à X dans σ si $X \in \mathrm{dom}(\sigma)$.

Exemple 10. Avec $\sigma = [X \mapsto g(Y), Y \mapsto b]$, on a

$$\sigma(t_0) = f(g(a), f(\underbrace{g(Y)}, f(b, g(\underbrace{g(Y)})))).$$

Attention ! On n'a pas de terme en $g(b)$: c'est une substitution *simultanée.*

Note 2. On rappelle qu'un dictionnaire peut être vu comme un ensemble fini de couples (X, t) avec $X \in \mathrm{V}$ et $t \in \mathsf{T}(\Sigma, \mathrm{V})$ tel que, pour toute variable $X \in \mathrm{V}$, il y a au plus un couple de la forme (X, t) dans la liste.

On utilise la notation $[{}^t\!/\!{}_X]$ pour représenter la notation $[X \mapsto t]$. Ceci est utiliser que lorsqu'on ne change qu'*une* variable.

Définition 10. Un *problème d'unification* est la donnée d'un ensemble fini de paires de termes (les contraintes) dans $\mathsf{T}(\Sigma, \mathrm{V})$. On note un tel problème $\mathscr{P} = \{t_1 \stackrel{?}{=} u_1, \dots, t_k \stackrel{?}{=} u_k\}$.

Une *solution*, un *unificateur*, d'un tel $\mathscr{P}$ est une substitution σ telle que, pour toute contrainte $t \stackrel{?}{=} u$ dans $\mathscr{P}$, $\sigma(t)$ et $\sigma(u)$ sont le même terme, ce que l'on note $\sigma(t) = \sigma(u)$.

On note $\mathrm{U}(\mathscr{P})$ l'ensemble des unificateurs de P.

Exemple 11. Avec le problème d'unification

$$\mathcal{P}_1 = \{f(a, g(X)) \stackrel{?}{=} f(Z, Y), g(T) \stackrel{?}{=} g(Z)\},$$

les substitutions

- ▷ $\sigma_1 = [Z \mapsto a, Y \mapsto g(X), T \mapsto a]$;
- ▷ $\sigma_2 = [Z \mapsto a, Y \mapsto g(b), T \mapsto a, X \mapsto b]$;

sont des solutions de $\mathcal{P}_1$. Mais,

$$\sigma_3 = [Z \mapsto f(b, b), T \mapsto f(b, b), Y \mapsto g(b), X \mapsto b]$$

n'est pas une solution.

Laquelle des solutions σ_1 et σ_2 est meilleure ? On remarque que $\sigma_2 = [{}^b/_X] \circ \sigma_1$ (où la composition est définie « comme on le pense »[5]). Ainsi, σ_1 est « plus général » que σ_2 ; σ_2 est un « cas particulier » de σ_1.

Exemple 12 (Aucune solution). Les problèmes

- ▷ $\mathcal{P}_2 = \{f(X, Y) \stackrel{?}{=} g(Z)\}$;
- ▷ $\mathcal{P}_3 = \{f(X, Y) \stackrel{?}{=} X\}$

n'ont aucune solution : $\mathrm{U}(\mathcal{P}_2) = \mathrm{U}(\mathcal{P}_3) = \emptyset$.

4.3 Algorithme d'unification (du premier ordre).

Définition 11. Un *état* est soit un couple $(\mathcal{P}, \sigma)$, soit $\perp$ (*l'état d'échec*).

Un état de la forme $(\emptyset, \sigma)$ est appelé *état de succès.*

Un état qui n'est, ni échec, ni succès, peut s'écrire sous la forme $(\{t \stackrel{?}{=} t'\} \sqcup \mathcal{P}, \sigma)$, la contrainte $t \stackrel{?}{=} t'$ étant choisie de manière non-déterministe.

5. Elle sera définie formellement ci-après.

On définit une relation binaire $\to$ entre états par :

- $\bot \not\to$;
- $(\emptyset, \sigma) \not\to$;
- Il ne reste que les cas ni succès, ni échec, que l'on traite par la disjonction de cas :

1. $$(\{f(t_1, \ldots, t_k) \stackrel{?}{=} f(u_1, \ldots, u_n) \sqcup \mathscr{P}, \sigma\}) \to (\{t_1 \stackrel{?}{=} u_1, \ldots, t_k \stackrel{?}{=} u_k\} \cup \mathscr{P}, \sigma) \quad ;$$
2. $(\{f(t_1, \ldots, t_k) \stackrel{?}{=} g(u_1, \ldots, u_n) \sqcup \mathscr{P}, \sigma\}) \to \bot$ si $f \neq g$;
3. $(\{X \stackrel{?}{=} t\} \sqcup \mathscr{P}, \sigma) \to (\mathscr{P}[^t/_X], [^t/_X] \circ \sigma)$ où
   - $X \notin \mathsf{Vars}(t)$,
   - $\mathscr{P}[^t/_X] = \{u[^t/_X] \stackrel{?}{=} u'[^t/_X] \mid (u \stackrel{?}{=} u') \in \mathscr{P}\}$,
   - et $[^t/_X] \circ \sigma$ est la substitution telle que, quel que soit $Y \in \mathrm{V}$, $([^t/_X] \circ \sigma)(Y) = (\sigma(Y))[^t/_X]$;
4. $(\{X \stackrel{?}{=} t\} \sqcup \mathscr{P}, \sigma) \to \bot$ si $X \in \mathsf{Vars}(t)$ et $t \neq X$;
5. $(\{X \stackrel{?}{=} X\} \sqcup \mathscr{P}, \sigma) \to (\mathscr{P}, \sigma)$.

L'*état initial* de l'algorithme correspond à $(\mathscr{P}, \emptyset)$: le problème $\mathscr{P}$ muni de la substitution vide $\emptyset$.

Exemple 13. On applique l'algorithme d'unification comme montré ci-dessous :

$$\begin{aligned}
& \{\underbrace{f(a, X) \stackrel{?}{=} f(Y, a)}_{\text{choix}}, g(X) \stackrel{?}{=} g(Y)\}, \emptyset \\
\to\quad & \{\underbrace{a \stackrel{?}{=} Y}_{\text{choix}}, X \stackrel{?}{=} a, g(X) \stackrel{?}{=} g(Y)\}, \emptyset \\
\to\quad & \{\underbrace{X \stackrel{?}{=} a}_{\text{choix}}, g(X) \stackrel{?}{=} g(a)\}, [Y \mapsto a] \\
\to\quad & \{\underbrace{g(a) \stackrel{?}{=} g(a)}_{\text{choix}}\}, [Y \mapsto a, X \mapsto a] \\
\to\quad & \{\underbrace{a \stackrel{?}{=} a}_{\text{choix}}\}, [Y \mapsto a, X \mapsto a] \\
\to\quad & \emptyset, [Y \mapsto a, X \mapsto a]
\end{aligned}$$

.

On peut remarquer que l'ensemble des clés de σ n'apparaît pas dans le problème ni dans les autres termes de la substitution : lorsqu'on ajoute une clé, elle disparaît du problème.

Définition 12. Un état $(\mathscr{P}, \sigma)$ est en *forme résolue* si, pour toute clé $X \in \operatorname{dom}(\sigma)$, alors X n'apparaît pas dans $\mathscr{P}$ et, quel que soit la clé $Y \in \operatorname{dom}(\sigma)$ alors $X \notin \mathsf{Vars}(\sigma(Y))$.

Remarque 10 (Notation). Une substitution σ peut être vue comme un problème d'unification, que l'on note $\stackrel{?}{\sigma}$. (On passe d'un ensemble de couples à un ensemble de paires.)

Proposition 5. Si $(\mathscr{P}_0, \sigma_0)$ est en forme résolue et $(\mathscr{P}_0, \sigma_0) \to (\mathscr{P}_1, \sigma_1)$ alors $(\mathscr{P}_1, \sigma_1)$ est en forme résolue et

$$\mathrm{U}(\mathscr{P}_0 \cup \stackrel{?}{\sigma}_0) = \mathrm{U}(\mathscr{P}_1 \cup \stackrel{?}{\sigma}_1).$$

Preuve. La vraie difficulté se trouve dans le 3ème cas (les cas 1 et 5 sont immédiats). Pour cela, on utilise le lemme « technique » ci-dessous.

Lemme 4. Si $X \notin \mathrm{dom}(\sigma)$ alors

$$[{}^t/_X] \circ \sigma = \Big[X \mapsto t, Y_1 \mapsto (\sigma(Y_1))[{}^t/_X], \ldots, Y_l \mapsto (\sigma(Y_k))[{}^t/_X]\Big],$$

où $\mathrm{dom}(\sigma) = \{Y_1, \ldots, Y_k\}$. □

□

Proposition 6. On note $\to^\star$ la clôture réflexive et transitive de la relation $\to$.

1. Un *unificateur le plus général* (*mgu*[6] dans la littérature anglaise) est une solution $\sigma \in \mathrm{U}(\mathscr{P})$ telle que, quelle que soit $\sigma' \in \mathrm{U}(\mathscr{P})$, il existe σ'' telle que $\sigma' = \sigma'' \circ \sigma$.

   Si $(\mathscr{P}, \emptyset) \to^\star (\emptyset, \sigma)$ alors σ est un unificateur le plus général de $\mathscr{P}$.
2. Si $(\mathscr{P}, \emptyset) \to^\star \perp$ alors $\mathrm{U}(\mathscr{P}) = \emptyset$.

Preuve.

1. On montre par induction sur $(\mathscr{P}, \emptyset) \to^\star (\emptyset, \sigma)$ l'égalité $\mathrm{U}(\mathscr{P}) = \mathrm{U}(\overset{?}{\sigma})$ à l'aide de la proposition précédente. Puis, on conclut avec le lemme suivant.

   Lemme 5. Pour toute substitution σ, alors σ est un unificateur le plus général de $\overset{?}{\sigma}$.

6. Pour *Most Général Unifier*

Preuve. Soit $\sigma' \in \mathrm{U}(\overset{?}{\sigma})$ et soit $X \in \mathrm{V}$. On montre que $\sigma' \circ \sigma = \sigma'$.

- Si $X \in \mathrm{dom}(\sigma)$, alors $\sigma'(\sigma(X)) = \sigma'(X)$ car σ' satisfait la contrainte $X \overset{?}{=} \sigma(X)$.
- Si $X \notin \mathrm{dom}(\sigma)$ alors $\sigma'(\sigma(X)) = \sigma'(X)$.

Ainsi $\sigma' \circ \sigma = \sigma'$. □

2. On montre que si $(\mathscr{P}, \emptyset) \to \bot$ alors $\mathrm{U}(\mathscr{P} \cup \overset{?}{\sigma})$. Pour le 2nd cas, c'est immédiat. Pour le 4ème cas, on procède par l'absurde. Soit σ_0 qui satisfait $X \overset{?}{=} t$ avec $X \in \mathsf{Vars}(t)$ et $X \neq t$. Alors $\sigma_0(X) = \sigma_0(t)$, qui contient $\sigma_0(X)$ et c'est un sous-ensemble strict. Absurde.

   On raisonne ensuite par induction sur $\to^\star$ pour conclure que $(\mathscr{P}, \emptyset) \to^\star (\mathscr{P}_0, \sigma_0) \to \bot$.

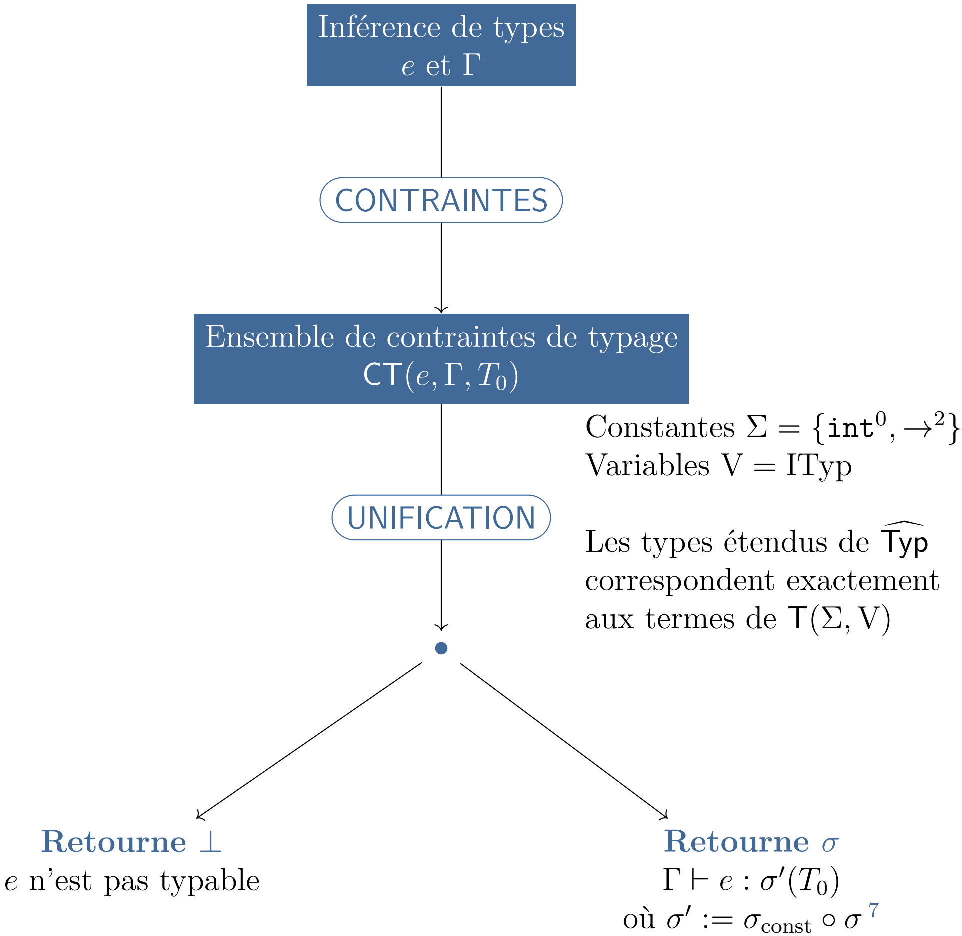
□

Lemme 6. La relation $\to$ est terminante (il n'y a pas de chaîne infinie avec cette relation).

Preuve. Vue plus tard. □

Théorème 2. L'algorithme d'unification calcule un unificateur le plus général si, et seulement si le problème initial a une solution. □

4.4 Retour sur l'inférence de types pour FUN.



Ceci conclut notre étude du petit langage fonctionnel FUN.

7. L'unificateur le plus général peut contenir des variables dans ses valeurs qui ne sont pas des clés (par exemple lors du typage de $\texttt{fun}\, x \to x$). Il faut donc composer σ avec une substitution « constante » pour effacer ces variables inutilisée.