2.4 Validité du typage

Le mécanisme que nous avons décrit précédemment pour définir les systèmes de types consiste en la construction d'un « arbre d'inférence » à partir d'un ensemble de règles d'inférence. Si ce système de types est implémenté, nous obtenons un algorithme d'inférence permettant d'inférer un schéma de type pour des expressions. Il serait de bon goût que cet algorithme se termine, quel que soit le programme qui lui est donné en entrée. Pour ce faire, nous allons démontrer qu'un arbre d'inférence est toujours de taille finie.

D'autre part, nous avons jusqu'à maintenant défini la sémantique de notre langage complètement indépendamment de son système de types. Rien ne prouve qu'ils sont « compatibles », au sens où si le système de types accepte un programme, celui-ci s'exécutera correctement. Une telle propriété se nomme « validité du système de types » (sous-entendu vis-à-vis d'une sémantique). Nous allons donc formaliser ici ce théorème, puis donner le schéma général de preuve que nous utiliserons pour le prouver sur les différents systèmes de types que nous étudions dans cette thèse, et enfin le prouver pour le système de types de base que nous avons défini dans la section précédente.

2.4.1 Définitions préliminaires

L'énoncé et la preuve des théorèmes de terminaison et de validité nécessite quelques définitions préliminaires sur les ensembles de contraintes et les schémas de type.

Propriétés des ensembles de contraintes

Nous commençons par définir la notion de « validité d'une contrainte de sous-typage » vis-à-vis d'un système de types :

Définition 1 (Validité d'une contrainte « $\tau^l \leqslant \tau^r$ »)**.**

Une contrainte de type $\tau^l \leqslant \tau^r$ est dite « valide » s'il existe une règle de saturation dont la conclusion est de la forme $\Phi \vdash \tau^l \leq \tau^r \rhd \Phi'$, autrement dit s'il est possible d'effectuer au moins un pas de saturation à partir de $\Phi \vdash \tau^l \leq \tau^r \rhd \Phi'$.

Avec le système de types de base que nous avons défini précédemment, toute relation entre une variable de type et un type est valide. Par contre, certaines relations entre types construits ne sont pas valides comme par exemple « `int` $\leqslant$ `string` » ou « `bool` $\leqslant \alpha_1 \times \alpha_2$ » ou encore « $\alpha_1 \to \alpha_2 \leqslant \{$ `A ‖ B` $\}$ ».
Nous étendons cette définition à un ensemble de contraintes :

Définition 2 (Validité d'un ensemble « Φ » de contraintes)**.**

Un ensemble de contraintes Φ est dit « valide » si toutes ses contraintes sont valides.

La simple « validité » d'un ensemble de contraintes Φ n'est pas très intéressante en soi puisque cette notion n'empêche pas Φ de contenir des contraintes qui pourraient impliquer des contraintes invalides. Pour éviter cela, nous allons maintenant définir la notion de « saturation » d'un ensemble de contraintes :

Définition 3 (Saturation d'un ensemble « Φ » de contraintes)**.**

Un ensemble de contraintes Φ est dit « saturé » s'il satisfait toutes les propriétés suivantes :

- $\forall \alpha, \tau^l, \tau^r \;.\; (\tau^l \leqslant \alpha) \in \Phi \wedge (\alpha \leqslant \tau^r) \in \Phi \Rightarrow$
  $\wedge\; (\tau^l \leqslant \tau^r) \in \Phi$
- $\forall \mathtt{t}, \alpha_1, \ldots, \alpha_n, \alpha'_1, \ldots, \alpha'_n \;.\; ((\alpha_1, \ldots, \alpha_n)\ \mathtt{t} \leqslant (\alpha'_1, \ldots, \alpha'_n)\ \mathtt{t}) \in \Phi \Rightarrow$
  $\wedge\; (\alpha_1 \leqslant \alpha'_1) \in \Phi \wedge \ldots \wedge (\alpha_n \leqslant \alpha'_n) \in \Phi$
  $\wedge\; (\alpha'_1 \leqslant \alpha_1) \in \Phi \wedge \ldots \wedge (\alpha'_n \leqslant \alpha_n) \in \Phi$
- $\forall K, \alpha, \alpha' \;.\; (K\,\alpha \leqslant \{\ \ldots\ \|\ K\,\alpha'\ \|\ \ldots\ \}) \in \Phi \Rightarrow$
  $\wedge\; (\alpha \leqslant \alpha') \in \Phi$
- $\forall K, K_1, \ldots, K_n, \alpha, \alpha_d, \alpha_1, \ldots, \alpha_n \;.$
  $(K\,\alpha \leqslant \{\ K_1\ \alpha_1\ \|\ \ldots\ \|\ K_n\ \alpha_n\ \|\ \alpha_d\ \}) \in \Phi \wedge (\forall i \;.\; K \neq K_i) \Rightarrow$
  $\wedge\; (K\,\alpha \leqslant \alpha_d) \in \Phi$
- $\forall t, \alpha_d, \alpha_1, \ldots, \alpha_n \;.\; ((\alpha_1, \ldots, \alpha_n)\ t \leqslant \{\ \ldots\ \|\ \alpha_d\ \}) \in \Phi \Rightarrow$
  $\wedge\; ((\alpha_1, \ldots, \alpha_n)\ t \leqslant \alpha_d) \in \Phi$

Tout comme la validité, la saturation d'un ensemble de contraintes est directement liée au système de types. Ces deux notions évolueront donc au cours des chapitres de cette thèse.

Nous définissons maintenant une relation, notée $(\overset{\alpha}{=})$ représentant l'égalité modulo α-renommage entre deux ensembles de contraintes :

Définition 4 (Égalité de deux ensembles de contraintes modulo α-renommage)**.**

Deux ensembles de contraintes Φ et Φ' vérifient la relation $\Phi \overset{\alpha}{=} \Phi'$ s'il existe une fonction totale de renommage R des variables de type qui soit telle que $R(\Phi') = \Phi$.

Par ailleurs, nous définissons une relation d'ordre notée $(\leqslant)$ entre deux ensembles de contraintes :

Définition 5 (Relation d'ordre sur les ensembles de contraintes)**.**

Deux ensembles de contraintes Φ et Φ' vérifient $\Phi \leqslant \Phi'$ s'il existe une fonction totale de renommage R des variables de type de telle sorte que $R(\Phi) \subset \Phi'$.

Il est important de remarquer que la fonction de renommage R n'est pas supposée injective. Deux variables de type peuvent donc avoir la même image par R.

La relation ($\leqslant$) est bien une relation d'ordre vis-à-vis de la relation d'égalité ($\overset{\alpha}{=}$). En effet, elle est dotée des propriétés suivantes :

- réflexivité : tout ensemble de contraintes Φ vérifie $\Phi \leqslant \Phi$. Il suffit de prendre la fonction de renommage identité pour le démontrer
- transitivité : si $\Phi_1 \leqslant \Phi_2$ et $\Phi_2 \leqslant \Phi_3$ alors $\Phi_1 \leqslant \Phi_3$. La démonstration est évidente, il suffit de choisir la composée des deux fonctions de renommage
- anti-symétrie : si $\Phi_1 \leqslant \Phi_2$ et $\Phi_2 \leqslant \Phi_1$ alors $\Phi_1 \overset{\alpha}{=} \Phi_2$. Un petit raisonnement sur les cardinaux montre que les fonctions de renommage sont ici des bijections, et donc que les ensembles de contraintes Φ_1 et de Φ_2 ont obligatoirement le même cardinal. Les inclusions entre ensembles de contraintes sont donc des égalités.

Nous noterons $\Phi_1 \leqslant_R \Phi_2$ pour spécifier la fonction de renommage utilisée.

Cette relation d'ordre sera importante pour énoncer les « lemmes de réduction du sujet » spécifiant que l'ensemble de contraintes généré par le typage d'une expression « décroît » lorsque l'on évalue l'expression.

Propriétés des schémas de type

Les définitions de validité et de saturation d'un ensemble de contraintes s'étendent naturellement aux schémas de type :

Définition 6 (Validité d'un schéma de type)**.**

Un schéma de type $[\ \forall\alpha_1 \dots \alpha_n\ .\ \alpha_0 \mid \Phi\ ]$ *est « valide » si* Φ *est valide.*

Définition 7 (Saturation d'un schéma de type)**.**

Un schéma de type $[\ \forall\alpha_1 \dots \alpha_n\ .\ \alpha_0 \mid \Phi\ ]$ *est « saturé » si* Φ *est saturé.*

Nous étendons de la même manière la relation d'égalité modulo α-renommage ($\overset{\alpha}{=}$) aux schémas de type :

Définition 8 (Égalité de deux schémas de type modulo α-renommage)**.**

Deux schémas de type $[\ \forall\alpha_1 \dots \alpha_n\ .\ \alpha_0 \mid \Phi\ ]$ *et* $[\ \forall\alpha'_1 \dots \alpha'_{n'}\ .\ \alpha'_0 \mid \Phi'\ ]$ *vérifient la relation* $[\ \forall\alpha_1 \dots \alpha_n\ .\ \alpha_0 \mid \Phi\ ] \overset{\alpha}{=} [\ \forall\alpha'_1 \dots \alpha'_{n'}\ .\ \alpha'_0 \mid \Phi'\ ]$ *s'il existe une fonction totale de renommage R des variables de type qui soit telle que :*

- $R(\alpha_0) = \alpha'_0$
- $R(\{\ \alpha_1, \dots, \alpha_n\ \}) = \{\ \alpha'_1, \dots, \alpha'_{n'}\ \}$
- $\forall\alpha\ .\ \alpha \notin \{\ \alpha_1, \dots, \alpha_n\ \} \Rightarrow R(\alpha) = \alpha$
- $R(\Phi) = \Phi'$

De même, nous étendons la comparaison ($\leqslant$) aux schémas de type :

Définition 9 (Relation d'ordre sur les schémas de type)**.**

Deux schémas de type $[\ \forall\alpha_1 \dots \alpha_n\ .\ \alpha_0 \mid \Phi\ ]$ *et* $[\ \forall\alpha'_1 \dots \alpha'_{n'}\ .\ \alpha'_0 \mid \Phi'\ ]$ *vérifient la relation* $[\ \forall\alpha_1 \dots \alpha_n\ .\ \alpha_0 \mid \Phi\ ] \leqslant [\ \forall\alpha'_1 \dots \alpha'_{n'}\ .\ \alpha'_0 \mid \Phi'\ ]$ *s'il existe une fonction totale de renommage R des variables qui soit telle que :*

- $R(\alpha_0) = \alpha'_0$
- $R(\{\ \alpha_1, \dots, \alpha_n\ \}) \subset \{\ \alpha'_1, \dots, \alpha'_{n'}\ \}$
- $\forall\alpha\ .\ \alpha \notin \{\ \alpha_1, \dots, \alpha_n\ \} \Rightarrow R(\alpha) = \alpha$
- $R(\Phi) \subset \Phi'$

Tout comme pour les comparaisons entre ensembles de contraintes, la fonction de renommage R n'est pas supposée injective.

Cette relation d'ordre entre schémas de type est construite pour représenter l'inverse de l'inclusion des ensembles de valeurs qu'ils dénotent. En effet, imposer $R(\Phi) \subset \Phi'$ autorise Φ' à avoir « plus de contraintes » que Φ. Puisque Φ et Φ' sont des conjonctions de contraintes, avoir « plus de contraintes » permet effectivement de dénoter un ensemble plus petit. Par exemple, ce que l'on exprime habituellement par « le type $(\alpha \to \alpha)$ est un sous-type de $(\texttt{int} \to \texttt{int})$ » s'écrira dans notre langage :

$$[\ \forall\alpha_0\ \alpha\ .\ \alpha_0 \mid \alpha \to \alpha \leqslant \alpha_0\ ] \ \leqslant\ [\ \forall\alpha_0\ \alpha\ .\ \alpha_0 \mid \alpha \to \alpha \leqslant \alpha_0\ \wedge\ \texttt{int} \leqslant \alpha\ \wedge\ \alpha \leqslant \texttt{int}\ ]$$

De plus, le fait que la fonction de renommage R ne soit pas supposée injective autorise des variables de type distinctes de $[\ \forall\alpha'_1 \dots \alpha'_{n'}\ .\ \alpha'_0 \mid \Phi'\ ]$ à avoir leur variables correspondantes égales dans $[\ \forall\alpha_1 \dots \alpha_n\ .\ \alpha_0 \mid \Phi\ ]$. Par exemple, ce que l'on exprime habituellement par « le type $(\alpha_1 \to \alpha_2)$ est un sous-type de $(\alpha \to \alpha)$ » s'écrira dans notre langage :

$$[\ \forall\alpha_0\ \alpha_1\alpha_2\ .\ \alpha_0 \mid \alpha_1 \to \alpha_2 \leqslant \alpha_0\ ] \ \leqslant\ [\ \forall\alpha_0\ \alpha\ .\ \alpha_0 \mid \alpha \to \alpha \leqslant \alpha_0\ ]$$

La fonction R à choisir pour montrer une telle inégalité est définie par :

- $R(\alpha_0) = \alpha_0$
- $R(\alpha_1) = R(\alpha_2) = \alpha_1$ (non-injectivité de R)
- $\forall\alpha' \notin \{\ \alpha_0, \alpha_1, \alpha_2\ \}\ .\ R(\alpha') = \alpha'$

En réalité, cette relation ($\leqslant$) est strictement plus forte que l'inverse de l'inclusion des ensembles : si deux schémas de type σ_1 et σ_2 vérifient $\sigma_1 \leqslant \sigma_2$, alors les ensembles de valeurs qu'ils dénotent V_1 et V_2 vérifient $V_2 \subset V_1$ mais l'inverse n'est pas toujours vrai. Par exemple, les schémas de type :

- $\sigma_1 = [\ \forall\alpha_0\ \alpha\ .\ \alpha_0 \mid \alpha \to \alpha \leqslant \alpha_0\ ]$
- $\sigma_2 = [\ \forall\alpha_0\ \alpha_1\ \alpha_2\ .\ \alpha_0 \mid \alpha_1 \to \alpha_2 \leqslant \alpha_0\ \wedge\ \alpha_1 \leqslant \alpha_2\ \wedge\ \alpha_2 \leqslant \alpha_1\ ]$

dénotent les mêmes ensembles mais ne vérifient ni $\sigma_1 \leqslant \sigma_2$ ni $\sigma_2 \leqslant \sigma_1$.

Propriétés des environnements de typage

Nous étendons ici la relation de comparaison ($\leqslant$) aux environnements de typage :

Définition 10 (Relation d'ordre sur les environnements de typage)**.**

Deux environnements de typage Γ_1 et Γ_2 vérifient la relation $\Gamma_1 \leqslant \Gamma_2$ s'ils ont exactement le même domaine de variables X et si :

$$\forall x \in X \,.\, \Gamma_1[x] \leqslant \Gamma_2[x]$$

Relation entre une expression et un schéma de type

Étant donné un système de types, nous allons maintenant définir une relation notée $(:)$ entre une expression et un schéma de type :

Définition 11 (« $\mathsf{e} \ : \ \sigma$ »)**.**

Une expression e et un schéma de type σ vérifient la relation $\mathsf{e} : \sigma$ si, étant donnée une variable de type fraîche α, les deux propriétés suivantes sont vérifiées :

- $\varnothing, \varnothing \vdash \mathsf{e} : \alpha \rhd \Phi$
- $\textsc{gen}(\alpha, \Phi, \varnothing) \leqslant \sigma$

Nous remarquerons que, pour toute expression e, s'il existe un σ tel que $\mathsf{e} : \sigma$, alors une infinité de σ vérifient $\mathsf{e} : \sigma$. De plus, on dira qu'une expression e est « typable » si et seulement s'il existe au moins un σ tel que $\mathsf{e} : \sigma$.

2.4.2 Terminaison

Nous cherchons ici à démontrer que les algorithmes d'inférence que nous définissons dans cette thèse se terminent. Une telle preuve utilisera très peu de propriétés du système de types et sera donc presque identique pour tous les systèmes étudiés dans cette thèse.

L'algorithme d'inférence consiste à construire l'arbre d'inférence grâce aux règles de typage, d'instanciation et de saturation du système de types. Nous allons simplement prouver qu'un tel arbre est toujours de taille finie.

En premier lieu, il est important de noter que la répartition des noeuds de typage et de saturation n'est pas « quelconque » dans un arbre d'inférence. Ceci est dû à deux propriétés :

- La racine de l'arbre est toujours un noeud de typage.
- Aucune règle de saturation n'a en prémisse la conclusion d'une règle de typage.

Par conséquent, la partie de l'arbre d'inférence constituée de noeuds de typage forme un arbre dont les feuilles sont soit des noeuds d'instanciation, soit les racines de sous-arbres constitués uniquement de noeuds de saturation. Nous appellerons par la suite « sous-arbre de saturation » un sous-arbre maximal composé uniquement d'instances de règles de saturation. Les noeuds d'instanciation ont également en prémisse des conclusion de règles de saturation. Les prémisses des noeuds d'instanciation sont donc également les racines de « sous-arbres de saturation ».

Pour prouver que tout arbre d'inférence est de taille finie, nous allons commencer par prouver que la « partie typage » de l'arbre est de taille finie, puis que le nombre de noeuds d'instanciation est fini, et enfin que tous les « sous-arbres de saturation » sont de taille finie.

Étude de la « partie typage »

Nous démontrons ici que le nombre de noeuds de typage d'un arbre d'inférence est égal au nombre de noeuds de l'expression que l'on cherche à typer.

Il est facile de vérifier que pour chaque règle de typage de nos systèmes de types, chacune de ses prémisses est :

- soit la conclusion d'une règle de saturation
- soit la conclusion d'une règle de typage dont l'expression est une sous-expression de celle présente dans la conclusion de la règle de typage
- soit la conclusion de la règle Inst.

La « partie typage » de l'arbre d'inférence est par conséquent isomorphe à l'arbre de l'expression analysée. Il possède en particulier autant de noeuds. L'expression que l'on cherche à typer étant finie, la partie typage de l'arbre d'inférence est donc finie.

Étude des noeuds d'instanciation

Les seules règles d'inférence ayant en prémisse la conclusion d'une règle d'instanciation sont les règles de typage TConst, TApplyPrim1, TApplyPrim2 et TVar. Le nombre de noeuds d'instanciation dans un arbre de typage est alors inférieur ou égal au nombre de noeuds de typage. Comme prouvé précédemment, les noeuds de typage sont en nombre fini dans l'arbre, et chacun engendre au plus un noeud d'instanciation. Le nombre d'occurrences de noeuds d'instanciation est donc fini.

Étude des sous-arbres de saturation

Nous avons déjà prouvé que les sous-arbres de saturation sont en nombre fini puisqu'ils sont engendrés par des noeuds de typage et d'instanciation qui sont en nombre fini. Nous démontrons ici que leur taille est finie.

Nous remarquons qu'aucune règle de saturation ne génère de nouvelles variables de type. Le nombre de variables de type apparaissant dans un sous-arbre de saturation est donc fini. De

plus, le nombre de constructeurs de types est également fini ainsi que leur arité. En effet, les constructeurs de types sont pour l'instant uniquement prédéfinis (`int`, `bool`, $(\rightarrow)$, $(\times)$, etc.), et ils seront enrichis dans le chapitre 5 sur les **GADT** avec un nombre fini de constructeurs de types déclarés dans le programme par l'utilisateur. Comme les définitions de τ^l et de τ^r ne sont pas récursives, il existe un ensemble fini de τ^l et de τ^r différents dans le sous-arbre de saturation, et donc un ensemble fini de contraintes de type de la forme $\tau^l \leqslant \tau^r$.

Chaque occurrence de la règle SNewConstraint ajoute une contrainte de type dans Φ qui n'y est pas encore. Aucune règle ne supprime de contrainte de Φ. Le nombre de contraintes étant borné dans le sous-arbre de saturation, la taille de Φ est bornée, et le nombre d'occurrences de la règle SNewConstraint est donc fini.

La règle SNewConstraint est la seule règle de saturation à avoir une prémisse de la forme $\Phi \vdash \tau^l \leq \tau^r \rhd \Phi'$. Comme le nombre d'occurrences de SNewConstraint est fini, le nombre d'occurrences de conclusions de la forme $\Phi \vdash \tau^l \leq \tau^r \rhd \Phi'$ est fini. Par conséquent, toutes les règles de saturation ayant une conclusion de la forme $\Phi \vdash \tau^l \leq \tau^r \rhd \Phi'$ (à savoir STypeConstr, SVariantMatch, SVariantDefault, SConstrDefault, SSameVar, STransRight, STransLeft et STransLeftRight) ont un nombre fini d'occurrences dans le sous-arbre de saturation.

Enfin, la seule règle de saturation dont on n'a pas encore prouvé que son nombre d'occurrences dans le sous-arbre de saturation était fini est SAlreadyProved. Il s'agit d'un axiome qui ne peut donc apparaître qu'au niveau d'une feuille. Nous avons donc montré que le sous-arbre de saturation a un intérieur fini, on en conclut donc que le sous-arbre de saturation lui-même est fini.

2.4.3 Théorèmes de validité

Nous souhaitons ici énoncer formellement un « théorème de validité » signifiant que si un programme est « accepté » par le typeur, alors son exécution se passera correctement :

Théorème 1 (Validité faible)**.**

Pour toute expression `e` *et tout schéma de type* σ*, si* `e` *et* σ *vérifient* `e` $: \sigma$ *alors* `eval(e)` $\neq$ `ERROR`*.*

En réalité, nous allons énoncer un second théorème, plus fort que le premier, permettant en plus, lorsque qu'une expression `e` est typable et que son exécution se termine, de relier le résultat de l'exécution au schéma de type inféré pour `e` :

Théorème 2 (Validité forte)**.**

Pour toute expression `e` *et tout schéma de type* σ*, si* `e` *et* σ *vérifient* `e` $: \sigma$ *alors l'une des propriétés suivantes est vérifiée :*

- *L'évaluation de* `e` *boucle indéfiniment.*
- *L'expression* `e` *s'évalue en la valeur* `v` *(c'est-à-dire* `e` $\longmapsto\!\!\!\!\!\rightarrow$ `v`*) et* `v` $: \sigma$*.*

Il n'est pas nécessaire d'imposer, dans ces théorèmes, que σ soit valide et saturé car la définition de $\mathsf{e} : \sigma$ suffit à imposer qu'il existe un σ' (potentiellement différent de σ) valide et saturé tel que $\mathsf{e} : \sigma'$. La propriété 1 que nous allons voir nous montrera qu'il suffit de choisir $\sigma' = \textsc{gen}(\alpha, \Phi, \varnothing)$.

2.4.4 Schéma des preuves

Les preuves des théorèmes de validité que nous allons donner sont inspirées de celles présentées par Wright et Felleisen [WF92]. Nous en donnons ici le schéma général. Une telle démonstration s'obtient en montrant les lemmes suivants.

Préservation des propriétés de Φ

Nous commençons par énoncer deux lemmes mettant en évidence la préservation des propriétés de « validité » et de « saturation » de l'ensemble de contraintes Φ lors du typage d'une expression.

Lemme 1 (Préservation de la validité de Φ)**.**

Soient :

- Γ *un environnement de typage ne contenant que des schémas de type valides*
- Φ *un ensemble de contraintes valides*
- e *une expression*
- α *une variable de type*
- Φ' *l'ensemble de contraintes obtenu par le déroulage réussi de l'arbre d'inférence en partant de la racine* $(\Phi, \Gamma \vdash \mathsf{e} : \alpha \rhd \Phi')$*.*

Alors Φ' *est valide.*

La démonstration de ce lemme découle directement de la définition de « validité d'une contrainte ».

Le lemme de préservation de la saturation de Φ est très similaire :

Lemme 2 (Préservation de la saturation de Φ)**.**

Soient :

- Γ *un environnement de typage ne contenant que des schémas de type saturés*
- Φ *un ensemble de contraintes saturé*
- e *une expression*
- α *une variable de type*
- Φ' *l'ensemble de contraintes obtenu par le déroulage réussi de l'arbre d'inférence en partant de la racine* $(\Phi, \Gamma \vdash \mathsf{e} : \alpha \rhd \Phi')$

Alors Φ' *est saturé.*

De ces deux lemmes se déduit une propriété importante pour le système de types :

Propriété 1 (Cohérence du système de types)**.**

Le typage d'une expression, lorsqu'il réussit, génère un schéma de type valide et saturé. Autrement dit, si Φ est obtenu par construction d'un arbre d'inférence ayant pour racine :

$$\varnothing, \varnothing \vdash \mathsf{e} : \alpha \rhd \Phi$$

alors $\textsc{gen}(\alpha, \Phi, \varnothing)$ *est valide et saturé.*

Les deux lemmes suivants mettent en évidence comment évolue l'ensemble de contraintes généré par le typeur lorsque l'on réduit l'ensemble de contraintes fourni :

Lemme 3 (Monotonie de la saturation)**.**

Soient :

- $(\tau^l \leqslant \tau^r)$ *une contrainte de types*
- Φ_1 *et* Φ_2 *deux ensembles de contraintes vérifiant* $\Phi_2 \leqslant \Phi_1$
- Φ'_1 *l'ensemble de contraintes obtenu par saturation de* $(\tau^l \leqslant \tau^r)$ *dans* Φ_1 *en dérivant l'arbre d'inférence au dessus de* $\Phi_1 \vdash \tau^l \leqslant \tau^r \rhd \Phi'_1$.

Alors le déroulage de l'arbre d'inférence au dessus de :

$$\Phi_2 \vdash \tau^l \leqslant \tau^r \rhd \Phi'_2$$

réussit et génère un ensemble de contraintes Φ'_2 *vérifiant* $\Phi'_2 \leqslant \Phi'_1$.

Lemme 4 (Monotonie du typage)**.**

Soient :

- e *une expression*
- α *une variable de type*
- R *une fonction de renommage des variables de type*
- Φ_1 *et* Φ_2 *deux ensembles de contraintes vérifiant* $\Phi_2 \leqslant_R \Phi_1$
- Γ_1 *et* Γ_2 *deux environnements de typage vérifiant* $\Gamma_2 \leqslant_R \Gamma_1$
- Φ'_1 *l'ensemble de contraintes obtenu par typage de* $\mathsf{e} : \alpha$ *dans* Φ_1, Γ_1 *en dérivant l'arbre d'inférence au dessus de* $\Phi_1, \Gamma_1 \vdash \mathsf{e} : \alpha \rhd \Phi'_1$.

Alors le déroulage de l'arbre d'inférence au dessus de :

$$\Phi_2, \Gamma_2 \vdash \mathsf{e} : \alpha \rhd \Phi'_2$$

réussit et génère un ensemble de contraintes Φ'_2 *vérifiant* $\Phi'_2 \leqslant \Phi'_1$.

Nous remarquerons que pour que ce lemme soit correct, la fonction de renommage liant Φ_1 et Φ_2 doit être la même que celle liant Γ_1 et Γ_2.

Lemmes de « réduction du sujet »

Les trois lemmes suivants mettent en évidence la préservation de la typabilité lors de l'évaluation d'une expression par les différentes relations $(\longrightarrow)$, $(\longmapsto)$ et $(\longmapsto\!\!\!\!\!\rightarrow)$.

Lemme 5 (Réduction du sujet par $(\longrightarrow)$)**.**

Soient :

- e_1 *et* e_2 *deux expressions vérifiant* $e_1 \longrightarrow e_2$
- α *une variable de type*
- Φ *un ensemble de contraintes valide et saturé*
- Φ'_1 *l'ensemble de contraintes obtenu par typage de* $e_1 : \alpha$ *dans* $(\Phi, \varnothing)$ *en dérivant l'arbre d'inférence au dessus de* $\Phi, \varnothing \vdash e_1 : \alpha \rhd \Phi'_1$.

Alors le typage de $e_2 : \alpha$ *dans* $(\Phi, \varnothing)$ *réussit et l'ensemble de contraintes* Φ'_2 *obtenu en dérivant l'arbre d'inférence au dessus de* $\Phi, \varnothing \vdash e_2 : \alpha \rhd \Phi'_2$ *vérifie* $\Phi'_2 \leqslant \Phi'_1$.

La démonstration de ce lemme consiste à analyser les différents cas de la définition de $(\longrightarrow)$, et pour chaque réduction de la forme $e_1 \longrightarrow e_2$, construire l'arbre d'inférence de $\Phi, \varnothing \vdash e_2 : \alpha \rhd \Phi'_2$ par transformation de l'arbre d'inférence de $\Phi, \varnothing \vdash e_1 : \alpha \rhd \Phi'_1$ et montrer $\Phi'_2 \leqslant \Phi'_1$.

On étend ensuite ce lemme de réduction du sujet à $(\longmapsto)$ ainsi :

Lemme 6 (Réduction du sujet par $(\longmapsto)$)**.**

Soient :

- e_1 *et* e_2 *deux expressions vérifiant* $e_1 \longmapsto e_2$
- α *une variable de type*
- Φ *un ensemble de contraintes valide et saturé*
- Φ'_1 *l'ensemble de contraintes obtenu par typage de* $e_1 : \alpha$ *dans* $(\Phi, \varnothing)$ *en dérivant l'arbre d'inférence au dessus de* $\Phi, \varnothing \vdash e_1 : \alpha \rhd \Phi'_1$.

Alors le typage de $e_2 : \alpha$ *dans* $(\Phi, \varnothing)$ *réussit et l'ensemble de contraintes* Φ'_2 *obtenu en dérivant l'arbre d'inférence au dessus de* $\Phi, \varnothing \vdash e_2 : \alpha \rhd \Phi'_2$ *vérifie* $\Phi'_2 \leqslant \Phi'_1$.

La démonstration de ce lemme se fait par induction sur la structure du contexte d'évaluation et une analyse de cas sur la définition de la grammaire E.

On étend enfin ce lemme à $(\longmapsto\!\!\!\!\!\rightarrow)$ ainsi :

Lemme 7 (Réduction du sujet par $(\longmapsto\!\!\!\!\!\rightarrow)$)**.**

Soient :

- e_1 *et* e_2 *deux expressions vérifiant* $e_1 \longmapsto\!\!\!\!\!\rightarrow e_2$.
- α *une variable de type*
- Φ *un ensemble de contraintes valide et saturé*
- Φ'_1 *l'ensemble de contraintes obtenu par typage de* $e_1 : \alpha$ *dans* $(\Phi, \varnothing)$ *en dérivant l'arbre d'inférence au dessus de* $\Phi, \varnothing \vdash e_1 : \alpha \rhd \Phi'_1$.

Alors le typage de $e_2 : \alpha$ *dans* $(\Phi, \varnothing)$ *réussit et l'ensemble de contraintes* Φ'_2 *obtenu en dérivant l'arbre d'inférence au dessus de* $\Phi, \varnothing \vdash e_2 : \alpha \rhd \Phi'_2$ *vérifie* $\Phi'_2 \leqslant \Phi'_1$.

La démonstration est presque immédiate par simple analyse de cas sur la définition de $(\longmapsto\!\!\!\!\!\rightarrow)$.

Propriétés des expressions bloquées

Indépendamment des autres lemmes, on démontre le lemme suivant :

Lemme 8 (Les expressions bloquées ne sont pas typables)**.**

Soit e_b *une expression bloquée. Il n'existe aucun schéma de type* σ *tel que* $e_b : \sigma$.

Ce lemme se démontre en analysant les différentes catégories d'expressions bloquées et en démontrant que pour chacune d'entre elles, il est impossible de construire un arbre d'inférence complet.

Encore indépendamment, on montre le lemme suivant classifiant l'évaluation d'une expression en trois catégories exclusives :

Lemme 9 (Évaluation uniforme)**.**

Toute expression e *vérifie une et une seule des propriétés suivantes :*

- *L'évaluation de* e *boucle indéfiniment*
- *Il existe une valeur* v *telle que* $e \longmapsto\!\!\!\!\!\rightarrow v$
- *Il existe une expression bloquée* e_b *telle que* $e \longmapsto\!\!\!\!\!\rightarrow e_b$.

Ce lemme découle directement de la définition d'une « expression bloquée ».

Preuves des théorèmes de validité

La preuve du théorème de validité forte s'obtient alors en démontrant par l'absurde que s'il existe σ tel que $\mathsf{e} : \sigma$, alors le dernier cas du lemme d'évaluation uniforme est impossible.

Le théorème de validité faible est un simple corollaire du théorème de validité forte.

2.4.5 Preuve de validité du système de base

Nous démontrons ici les différents lemmes et théorèmes énoncés dans le chapitre précédent pour le système de types de base.

Lemme 1 (Préservation de la validité de Φ)**.**

Soient :

- Γ *un environnement de typage ne contenant que des schémas de type valides*
- Φ *un ensemble de contraintes valides*
- e *une expression*
- α *une variable de type*
- Φ' *l'ensemble de contraintes obtenu par le déroulage réussi de l'arbre d'inférence en partant de la racine* $(\Phi, \Gamma \vdash \mathsf{e} : \alpha \rhd \Phi')$*.*

Alors Φ' *est valide.*

Démonstration (Lemme 1) :

La seule règle d'inférence ajoutant une contrainte de la forme $(\tau^l \leqslant \tau^r)$ dans Φ est SNewConstraint, et cette règle possède une prémisse de la forme :

$$\Phi \wedge \tau^l \leqslant \tau^r \vdash \tau^l \leq \tau^r \rhd \Phi'$$

La construction de l'arbre d'inférence a réussi, on en déduit qu'il existe une règle ayant pour conclusion $(\Phi \wedge \tau^l \leqslant \tau^r \vdash \tau^l \leq \tau^r \rhd \Phi')$. L'existence d'une telle règle de saturation correspond bien à la définition de la validité de $(\tau^l \leqslant \tau^r)$.

Lemme 2 (Préservation de la saturation de Φ).

Soient :

- Γ *un environnement de typage ne contenant que des schémas de type saturés*
- Φ *un ensemble de contraintes saturé*
- e *une expression*
- α *une variable de type*
- Φ' *l'ensemble de contraintes obtenu par le déroulage réussi de l'arbre d'inférence en partant de la racine* $(\Phi, \Gamma \vdash \mathsf{e} : \alpha \rhd \Phi')$

Alors Φ' *est saturé.*

Démonstration (Lemme 2) :

Pour montrer que Φ' est saturé, nous montrons indépendamment les différents points de la définition de la « saturation d'un ensemble de contraintes » :

- Soient α, τ^l, τ^r tels que $(\tau^l \leqslant \alpha) \in \Phi'$ et $(\alpha \leqslant \tau^r) \in \Phi'$.
  Montrons $(\tau^l \leqslant \tau^r) \in \Phi'$. Les contraintes $(\tau^l \leqslant \alpha)$ et $(\alpha \leqslant \tau^r)$ ont obligatoirement été ajoutées dans l'ensemble de contraintes par des instances de la règle SNewConstraint l'une après l'autre. On distingue trois cas :
  - Soit les contraintes $(\tau^l \leqslant \alpha)$ et $(\alpha \leqslant \tau^r)$ sont déjà présentes dans Φ. Comme Φ est saturé, $(\tau^l \leqslant \tau^r) \in \Phi$. Aucune règle ne supprime de contrainte, nous avons donc $\Phi \subset \Phi'$ et donc $(\tau^l \leqslant \tau^r) \in \Phi'$.
  - Soit la contrainte $(\tau^l \leqslant \alpha)$ a été ajoutée alors que la contrainte $(\tau^l \leqslant \alpha)$ était déjà présente dans l'ensemble de contraintes. L'application de la règle SNewConstraint sur $(\alpha \leqslant \tau^r)$ a donc généré un noeud de la forme $\Phi_0 \vdash \alpha \leq \tau^r \rhd \Phi_1$ avec $(\tau^l \leqslant \alpha) \in \Phi_0$. La règle appliquée sur $\Phi_0 \vdash \alpha \leq \tau^r \rhd \Phi_1$ est STransLeft. Puisque $(\tau^l \leqslant \alpha) \in \Phi_0$, l'application de STransLeft possède une prémisse de la forme $\Phi'_0 \rhd \tau^l \leqslant \tau^r \vdash \Phi'_1$. Dans cette situation, soit $(\tau^l \leqslant \tau^r)$ était déjà présente dans l'ensemble de contraintes Φ'_0 et la règle SAlreadyProved a été appliquée ; soit $(\tau^l \leqslant \tau^r)$ n'y était pas encore et une instance de la règle SNewConstraint l'y a ajouté. Dans tous les cas, la contrainte $(\tau^l \leqslant \tau^r)$ est bien présente dans l'ensemble de contraintes généré Φ'_1. Comme aucune règle ne supprime de contrainte de l'ensemble de contraintes, Φ'_1 est bien inclus dans Φ'. On en conclut $(\tau^l \leqslant \tau^r) \in \Phi'$.
  - Soit la contrainte $(\tau^l \leqslant \alpha)$ a été ajoutée alors que $(\alpha \leqslant \tau^r)$ était déjà présente dans l'ensemble de contraintes. Un raisonnement similaire basé sur l'utilisation de STransRight à la place de STransLeft prouve également $(\tau^l \leqslant \tau^r) \in \Phi'$.
- Soient $t, \alpha_1, \ldots, \alpha_n, \alpha'_1, \ldots, \alpha'_n$ tels que $((\alpha_1, \ldots, \alpha_n)\, t \leqslant (\alpha'_1, \ldots, \alpha'_n)\, t) \in \Phi'$.
  Montrons que $\forall i \in [\,1;\, n\,]\,.\, (\alpha_i \leqslant \alpha'_i) \in \Phi' \wedge (\alpha'_i \leqslant \alpha_i) \in \Phi'$.
  La seule règle qui a pu ajouter la contrainte $((\alpha_1, \ldots, \alpha_n)\ t \leqslant (\alpha'_1, \ldots, \alpha'_n)\ t)$ dans l'ensemble de contraintes est SNewConstraint et l'instance de cette règle a en prémisse un noeud de la forme $\Phi_0 \vdash ((\alpha_1, \ldots, \alpha_n)\ t \leq (\alpha'_1, \ldots, \alpha'_n)\ t) \rhd \Phi_1$. La seule règle ayant pu

s'appliquer sur un tel noeud est STypeConstr qui a imposé la présence de toutes les contraintes $(\alpha_i \leqslant \alpha'_i)$ et $(\alpha'_i \leqslant \alpha_i)$ pour $i \in [\,1;\, n\,]$ dans l'ensemble contraintes résultat. □

- De la même manière, la propriété « $\forall \mathtt{K}, \alpha, \alpha' \,.\, (\mathtt{K}\ \alpha \leqslant \{\ \ldots\ \|\ \mathtt{K}\ \alpha'\ \|\ \ldots\ \}) \in \Phi \Rightarrow (\alpha \leqslant \alpha') \in \Phi$ » découle directement de l'usage de la règle SVariantMatch lors de l'ajout de la contrainte « $\mathtt{K}\ \alpha \leqslant \{\ \ldots\ \|\ \mathtt{K}\ \alpha'\ \|\ \ldots\ \}$ » dans l'ensemble de contraintes.
- La propriété $\forall \mathtt{K}, \mathtt{K}_1, \ldots, \mathtt{K}_n, \alpha, \alpha_d, \alpha_1, \ldots, \alpha_n \,.\, (\mathtt{K}\ \alpha \leqslant \{\, \mathtt{K}_1\ \alpha_1\ \|\ \ldots\ \|\ \mathtt{K}_n\ \alpha_n\ \|\ \alpha_d\,\}) \in \Phi \Rightarrow (\forall i \,.\, \mathtt{K} \neq \mathtt{K}_i) \Rightarrow (\mathtt{K}\ \alpha \leqslant \alpha_d) \in \Phi$ découle de l'usage de la règle SVariantDefault lors de l'ajout de la contrainte « $\mathtt{K}\ \alpha \leqslant \{\, \mathtt{K}_1\ \alpha_1\ \|\ \ldots\ \|\ \mathtt{K}_n\ \alpha_n\ \|\ \alpha_d\,\}$ » dans l'ensemble de contraintes.
- La propriété $\forall t, \alpha_d, \alpha_1, \ldots, \alpha_n \,.\, ((\alpha_1, \ldots, \alpha_n)\ t \leqslant \{\ \ldots\ \|\ \alpha_d\ \}) \in \Phi \Rightarrow ((\alpha_1, \ldots, \alpha_n)\ t \leqslant \alpha_d) \in \Phi$ découle quant à elle de l'usage de la règle SConstrDefault lors de l'ajout de la contrainte « $(\alpha_1, \ldots, \alpha_n)\ t \leqslant \{\ \ldots\ \|\ \alpha_d\ \}$ » dans l'ensemble de contraintes.

En conclusion, l'ensemble de contraintes Φ' est bien saturé selon tous les critères de la définition de la saturation.

Propriété 1 (Cohérence du système de types)**.**

Le typage d'une expression, lorsqu'il réussit, génère un schéma de type valide et saturé. Autrement dit, si Φ est obtenu par construction d'un arbre d'inférence ayant pour racine :

$$\varnothing, \varnothing \vdash \mathsf{e} : \alpha \rhd \Phi$$

alors $\textsc{gen}(\alpha, \Phi, \varnothing)$ est valide et saturé.

Démonstration (Propriété 1) :

Puisqu'un ensemble de contraintes vide est valide et saturé, cette propriété est une conséquence directe des lemmes 1 et 2.

Lemme 3 (Monotonie de la saturation)**.**

Soient :

- $(\tau^l \leqslant \tau^r)$ *une contrainte de types*
- Φ_1 *et* Φ_2 *deux ensembles de contraintes vérifiant* $\Phi_2 \leqslant \Phi_1$
- Φ'_1 *l'ensemble de contraintes obtenu par saturation de* $(\tau^l \leqslant \tau^r)$ *dans* Φ_1 *en dérivant l'arbre d'inférence au dessus de* $\Phi_1 \vdash \tau^l \leqslant \tau^r \rhd \Phi'_1$.

Alors le déroulage de l'arbre d'inférence au dessus de :

$$\Phi_2 \vdash \tau^l \leqslant \tau^r \rhd \Phi'_2$$

réussit et génère un ensemble de contraintes Φ'_2 *vérifiant* $\Phi'_2 \leqslant \Phi'_1$.

Démonstration (Lemme 3) :

Comme $\Phi_2 \leqslant \Phi_1$, il existe une fonction de renommage R des variables de type telle que $R(\Phi_2) \subset \Phi_1$. Nous construisons alors l'arbre d'inférence de $\Phi_2 \vdash \tau^l \leqslant \tau^r \rhd \Phi'_2$ à partir de l'arbre d'inférence de $\Phi_1 \vdash \tau^l \leqslant \tau^r \rhd \Phi'_1$ en supprimant certains sous-arbres concernant les contraintes de Φ_1 dont leurs correspondantes via R ne sont pas présentes dans Φ_2, et en répliquant certains sous-arbres de Φ_1 concernant les variables de type qui sont fusionnées par la fonction de renommage (non-obligatoirement-injective) R. Le Φ'_2 obtenu après cette transformation vérifie alors $R(\Phi'_2) \subset \Phi'_1$, ce qui montre bien $\Phi'_2 \leqslant \Phi'_1$.

Lemme 4 (Monotonie du typage)**.**

Soient :

- e *une expression*
- α *une variable de type*
- R *une fonction de renommage des variables de type*
- Φ_1 *et* Φ_2 *deux ensembles de contraintes vérifiant* $\Phi_2 \leqslant_R \Phi_1$
- Γ_1 *et* Γ_2 *deux environnements de typage vérifiant* $\Gamma_2 \leqslant_R \Gamma_1$
- Φ'_1 *l'ensemble de contraintes obtenu par typage de* $\mathsf{e} : \alpha$ *dans* Φ_1, Γ_1 *en dérivant l'arbre d'inférence au dessus de* $\Phi_1, \Gamma_1 \vdash \mathsf{e} : \alpha \rhd \Phi'_1$.

Alors le déroulage de l'arbre d'inférence au dessus de :

$$\Phi_2, \Gamma_2 \vdash \mathsf{e} : \alpha \rhd \Phi'_2$$

réussit et génère un ensemble de contraintes Φ'_2 *vérifiant* $\Phi'_2 \leqslant \Phi'_1$.

Démonstration (Lemme 4) :

Nous construisons alors l'arbre d'inférence de $\Phi_2, \Gamma_2 \vdash \mathsf{e} : \alpha \rhd \Phi'_2$ à partir de l'arbre d'inférence de $\Phi_1, \Gamma_1 \vdash \mathsf{e} : \alpha \rhd \Phi'_2$. La partie de cet arbre composée de noeuds de typage reste inchangée car elle est structurellement isomorphe à la structure de l'expression e, qui n'a pas changée. De même, les noeuds d'instanciation restent aux mêmes endroits dans l'arbre puisque leur position est imposée par la partie typage de l'arbre. Comme $\Gamma_2 \leqslant_R \Gamma_1$, les schémas de types instanciés sont plus petits ce qui ne contribue qu'à réduire l'ensemble de contraintes généré. Seuls les sous-arbres de saturation changent de structure. Pour chacun d'entre eux, on applique le lemme 3 montrant qu'on obtient un ensemble de contraintes plus petit (c'est-à-dire ayant moins de contraintes) que l'ensemble de contraintes correspondant dans l'arbre initial. Au final, l'ensemble de contraintes Φ'_2 généré est effectivement plus petit que l'ensemble Φ'_1 initial.

Lemme 5 (Réduction du sujet par $(\longrightarrow)$)**.**

Soient :

- e_1 *et* e_2 *deux expressions vérifiant* $\mathsf{e}_1 \longrightarrow \mathsf{e}_2$
- α *une variable de type*
- Φ *un ensemble de contraintes valide et saturé*
- Φ'_1 *l'ensemble de contraintes obtenu par typage de* $\mathsf{e}_1 : \alpha$ *dans* $(\Phi, \varnothing)$ *en dérivant l'arbre d'inférence au dessus de* $\Phi, \varnothing \vdash \mathsf{e}_1 : \alpha \rhd \Phi'_1$.

Alors le typage de $\mathsf{e}_2 : \alpha$ *dans* $(\Phi, \varnothing)$ *réussit et l'ensemble de contraintes* Φ'_2 *obtenu en dérivant l'arbre d'inférence au dessus de* $\Phi, \varnothing \vdash \mathsf{e}_2 : \alpha \rhd \Phi'_2$ *vérifie* $\Phi'_2 \leqslant \Phi'_1$.

Démonstration (Lemme 5) :

Nous analysons les différents cas de la définition de $(\longrightarrow)$:

- Cas « $\mathsf{p}^1\ \mathsf{v} \longrightarrow \delta_1\ (\mathsf{p}^1, \mathsf{v})$ » :
  La validité du typage repose ici sur la validité de la fonction T vis-à-vis de la fonction d'évaluation des primitives unaires δ_1. En effet, l'arbre d'inférence de $\Phi, \varnothing \vdash \mathsf{p}^1\ \mathsf{v} : \alpha \rhd \Phi'_1$ est obligatoirement de la forme :

$$\textsc{TApplyPrim1}\ \dfrac{\textsc{Inst}\ \dfrac{\bigtriangledown}{\Phi \vdash T(\mathsf{p}^1) \leqslant \alpha_1 \to \alpha_2 \rhd \Phi'} \quad \dfrac{\bigtriangledown}{\Phi' \vdash \alpha_2 \leqslant \alpha \rhd \Phi''} \quad \dfrac{\bigtriangledown}{\Phi'', \varnothing \vdash \mathsf{v} : \alpha_1 \rhd \Phi'_1}}{\Phi, \varnothing \vdash \mathsf{p}^1\ \mathsf{v} : \alpha \rhd \Phi'_1}$$

  Le schéma de type σ retourné par $T(\mathsf{p}^1)$ est instancié et comparé à $\alpha_1 \to \alpha$. La saturation

des contraintes doit alors imposer suffisamment de contraintes sur α_1 pour que si le typage de $\mathtt{v} : \alpha$ réussit, assurer que $\delta_1(\mathtt{p}^1, \mathtt{v})$ soit défini ; et suffisamment de contraintes sur α_2 (lié à α via la contrainte $(\alpha_2 \leqslant \alpha)$) pour que le typage de $\delta_1(\mathtt{p}^1, \mathtt{v}) : \alpha$ réussisse et génère un ensemble de contraintes Φ'_2 plus petit que Φ'_1.

- Cas « $\mathtt{p}^2\ \mathtt{v}_1\ \mathtt{v}_2 \longrightarrow \delta_2\,(\mathtt{p}^2, \mathtt{v}_1, \mathtt{v}_2)$ » :
  L'arbre d'inférence de $\Phi, \varnothing \vdash \mathtt{p}^2\ \mathtt{v}_1\ \mathtt{v}_2\ \mathtt{:}\ \alpha \rhd \Phi'_1$ est obligatoirement de la forme :

$$\textsc{TApplyPrim2}\ \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Phi \vdash \alpha_0 \leqslant \alpha_2 \to \alpha_3 \rhd \Phi_1}\quad \dfrac{}{\Phi_2 \vdash T(\mathtt{p}^2) \leqslant \alpha_1 \to \alpha_0 \rhd \Phi_3}\ \textsc{Inst}\quad \dfrac{}{\Phi_1 \vdash \alpha_3 \leqslant \alpha \rhd \Phi_2}}{\Phi_3, \varnothing \vdash \mathtt{v}_1 : \alpha_1 \rhd \Phi_4}\quad \dfrac{}{\Phi_4, \varnothing \vdash \mathtt{v}_2 : \alpha_2 \rhd \Phi'_1}}{\Phi, \varnothing \vdash \mathtt{p}^2\ \mathtt{v}_1\ \mathtt{v}_2\ \mathtt{:}\ \alpha \rhd \Phi'_1}$$

  De manière similaire au cas de l'application d'un opérateur unaire, le schéma de type généré par $T(\mathtt{p}^2)$ doit imposer suffisamment de contraintes sur α_1 et α_2 pour que si le typage de $\mathtt{v}_1 : \alpha_1$ et $\mathtt{v}_2 : \alpha_2$ réussit, alors $\delta_2(\mathtt{p}^2, \mathtt{v}_1, \mathtt{v}_2)$ soit défini ; et de contraintes sur α de telle sorte que le typage de $\delta_2(\mathtt{p}^2, \mathtt{v}_1, \mathtt{v}_2) : \alpha$ réussisse et génère un ensemble de contraintes Φ'_2 plus petit que Φ'_1.

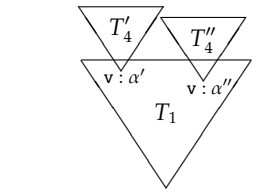
- Cas « $(\lambda\ \mathtt{x}\ .\ \mathtt{e})\ \mathtt{v} \longrightarrow \mathtt{e}[\mathtt{x} \mapsto \mathtt{v}]$ » :
  L'arbre d'inférence de $\Phi, \varnothing \vdash (\lambda\ \mathtt{x}\ .\ \mathtt{e})\ \mathtt{v}\ \mathtt{:}\ \alpha \rhd \Phi'_1$ est obligatoirement de la forme :

$$\textsc{TApp}\ \dfrac{\textsc{STransLeft}\ \dfrac{\textsc{SNewConstraint}\ \dfrac{\Phi_1 \vdash \alpha_1 \leq \alpha_2 \to \alpha_3 \rhd \Phi_1}{\Phi \vdash \alpha_1 \leqslant \alpha_2 \to \alpha_3 \rhd \Phi_1}\quad T_0\ \dfrac{}{\Phi_1 \vdash \alpha_3 \leqslant \alpha \rhd \Phi_2}\quad T_4\ \dfrac{}{\Phi_7, \varnothing \vdash \mathtt{v} : \alpha_2 \rhd \Phi'_1}}{\Phi, \varnothing \vdash (\lambda\ \mathtt{x}\ .\ \mathtt{e})\ \mathtt{v}\ \mathtt{:}\ \alpha \rhd \Phi'_1}$$

$$\textsc{TLambda}\ \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Phi_2, \{\,(\mathtt{x}, \alpha'_1)\,\} \vdash \mathtt{e} : \alpha'_2 \rhd \Phi_3}\quad \dfrac{\dots\quad \dfrac{}{\Phi_4 \vdash \alpha_2 \leqslant \alpha'_1 \rhd \Phi_5}\quad \dfrac{}{\Phi_5 \vdash \alpha'_2 \leqslant \alpha_3 \rhd \Phi_6}\quad \dots}{\Phi_3 \vdash \alpha'_1 \to \alpha'_2 \leq \alpha_2 \to \alpha_3 \rhd \Phi_7}\ \textsc{STypeConstr}}{\Phi_3 \vdash \alpha'_1 \to \alpha'_2 \leqslant \alpha_2 \to \alpha_3 \rhd \Phi_7}\ \textsc{SNewConstraint}}{\Phi_3 \vdash \alpha'_1 \to \alpha'_2 \leq \alpha_1 \rhd \Phi_7}\ \textsc{STransRight}}{\Phi_3 \vdash \alpha'_1 \to \alpha'_2 \leqslant \alpha_1 \rhd \Phi_7}\ \textsc{SNewConstraint}}{\Phi_2, \varnothing \vdash \lambda\ \mathtt{x}\ .\ \mathtt{e} : \alpha_1 \rhd \Phi_7}$$

  Nous commençons par construire un arbre d'inférence de $\Phi_2, \varnothing \vdash \mathtt{e}[\mathtt{x} \mapsto \mathtt{v}] : \alpha'_2 \rhd \Phi_8$ à partir de T_1 en y remplaçant les sous-arbres correspondant au typage des différentes

occurrences de $\mathtt{x}$ dans $\mathtt{e}$ par une adaptation de T_4 correspondant au typage de $\mathtt{v}$:



$$\Phi_2, \varnothing \vdash \mathtt{e}[\mathtt{x} \mapsto \mathtt{v}] : \alpha'_2 \vdash \Phi_8$$

Nous allons maintenant montrer comment sont construits T'_4, T''_4, etc., à partir de T_4.

Les sous-arbre de T_1 correspondant aux différentes occurrences de $\mathtt{x}$ dans $\mathtt{e}$ sont de la forme :

$$\textsc{Inst}\ \frac{\dfrac{\ldots}{\Phi_9 \vdash \alpha'_1 \leqslant \alpha' \rhd \Phi_{10}}}{\Phi_9, \{\ldots, (\mathtt{x} : \alpha'_1), \ldots\} \vdash \mathtt{x} : \alpha' \rhd \Phi_{10}}$$

Ce qui montre que Φ_{10} contient la contrainte $(\alpha'_1 \leqslant \alpha')$. De plus, le sous-arbre T_2 est une preuve de cohérence de la contrainte $(\alpha_2 \leqslant \alpha'_1)$ avec Φ_4. L'ensemble de contraintes Φ_5 contient donc la contrainte $(\alpha_2 \leqslant \alpha'_1)$. Comme aucune règle ne supprime de contrainte, l'ensemble de contraintes Φ_7 contient obligatoirement les contraintes $(\alpha_2 \leqslant \alpha'_1)$ et $(\alpha'_1 \leqslant \alpha')$. D'après le lemme 2, puisque Φ est saturé, l'ensemble de contraintes Φ_7 est saturé. Il contient donc en particulier la contrainte $(\alpha_2 \leqslant \alpha')$. Tous les sous-arbres de T_4 contiennent également cette contrainte, ce qui nous permet de déduire que, quelle que soit la structure de la valeur $\mathtt{v}$, le sous-arbre T_4 possède un sous-arbre T_0 de la forme :

$$\textsc{SNewConstraint}\ \dfrac{\textsc{STransLeft}\ \dfrac{\dfrac{\ldots}{\ldots}\quad\dfrac{\quad T_5\quad}{\Phi_{12} \vdash \tau_0^l \leqslant \alpha' \rhd \Phi_{13}}\quad\dfrac{\ldots}{\ldots}}{\Phi_{11} \vdash \tau_0^l \leq \alpha_2 \rhd \Phi_{14}}}{\Phi_{11} \vdash \tau_0^l \leqslant \alpha_2 \rhd \Phi_{14}}$$

où τ_0^l est un type défini par la structure de $\mathtt{v}$ (une flèche si $\mathtt{v}$ est un λ, une variable de type si $\mathtt{v}$ est une constante, etc.).

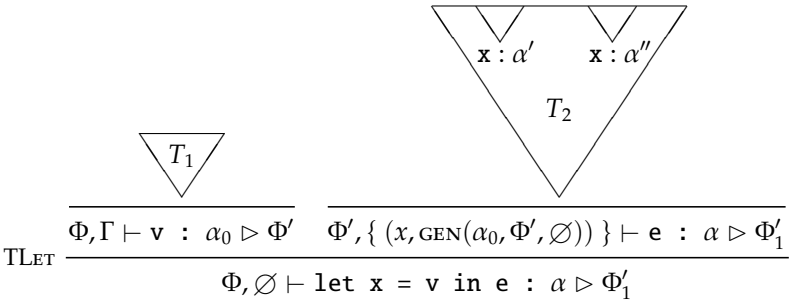
Comme la variable de type α'_1 apparaît libre dans les environnements de typage présents dans T_1, cela nous assure que α'_1 ne sera jamais généralisée dans T_1. Il n'y a donc aucun risque que des contraintes soient imposées sur une version renommée de α'_1 dans T_1 et donc ignorées dans T_4. Toutes les contraintes accumulées sur α'_1 dans T_1 sont donc prouvées compatibles avec τ_0^l dans T_4.

Au final, l'arbre d'inférence de $\Phi_2, \varnothing \vdash e[x \mapsto v] : \alpha'_2 \vdash \Phi_8$ est obtenu en remplaçant tous

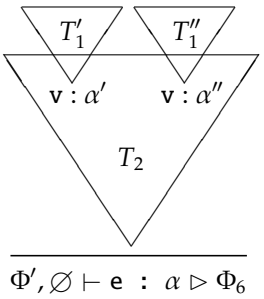
les sous-arbres de T_1 correspondant au typage des différentes occurrences de `x` dans `e` par T_4 dans lequel on a remplacé T_0 par T_5. Comme T_3 a prouvé que $\alpha'_2 \leqslant \alpha$ était compatible avec les autres contraintes, en saturant l'ensemble de contraintes, il a en particulier ajouté sur α toutes les contraintes accumulées sur α'_5 dans T_1. Nous en déduisons qu'il est possible de construire un arbre au dessus de $\Phi_2, \varnothing \vdash e[x \mapsto v] : \alpha \vdash \Phi'_8$ générant un ensemble de contraintes $\Phi'_8 \leqslant \Phi_8$.

Cette transformation d'arbre n'a ajouté aucune contrainte qui n'était pas déjà dans Φ'_1, nous en déduisons que $\Phi'_8 \leqslant \Phi'_1$. De plus $\Phi_2 = \Phi \wedge \alpha_1 \leqslant \alpha_2 \to \alpha \wedge \alpha_3 \leqslant \alpha$, donc $\Phi \leqslant \Phi_2$. D'après le lemme 4, nous en déduisons que $\texttt{e}[\texttt{x} \mapsto \texttt{v}]$ est typable dans Φ et génère un ensemble de contraintes Φ'_2 vérifiant $\Phi'_2 \leqslant \Phi'_8$. Comme $\Phi'_8 \leqslant \Phi'_1$, par transitivité de $(\leqslant)$, les ensembles de contraintes Φ'_1 et Φ'_2 vérifient bien $\Phi'_2 \leqslant \Phi'_1$. □

- Cas « `let x = v in e` $\longrightarrow$ `e[x` $\mapsto$ `v]` » :
  L'arbre d'inférence de $\Phi, \varnothing \vdash \texttt{let x = v in e} : \alpha \rhd \Phi'_1$ est obligatoirement de la forme :



Nous construisons alors l'arbre d'inférence de $\Phi', \varnothing \vdash \texttt{e} : \alpha \rhd \Phi_6$ en remplaçant dans T_2 les sous-arbres correspondant au typage des différentes occurrences de `x` dans `e` par une adaptation de T_1 :



Ces sous-arbres sont tous de la forme :

$$\textsc{TVar}\ \dfrac{\textsc{Inst}\ \dfrac{\dfrac{\overset{T_3}{\triangledown}}{\Phi_3 \vdash \alpha_0[\alpha_i \mapsto \alpha'_i]_{i=1}^n \leqslant \alpha' \rhd \Phi_4} \quad \dfrac{\overset{T_4}{\triangledown}}{\Phi_4 \vdash \Phi'[\alpha_i \mapsto \alpha'_i]_{i=1}^n \rhd \Phi_5}}{\Phi_3 \vdash \textsc{gen}(\alpha_0, \Phi', \varnothing) \leqslant \alpha' \rhd \Phi_5}}{\Phi_3, \{\ \ldots, (\mathtt{x}, \textsc{gen}(\alpha_0, \Phi', \varnothing)), \ldots\} \vdash \mathtt{x} : \alpha' \rhd \Phi_5}$$

Nous les remplaçons par T_1 dans lequel les instances des noeuds de typage et d'instanciation sont identiques (car imposés par la structure de v). De plus, les sous-arbres T_3 et T_4 ont montré que les contraintes accumulées sur α_0 lors de la construction de T_1 sont compatibles avec les contraintes accumulées sur α' dans T_2, ce qui montre que les sous-arbres de saturation sont constructibles et que $\Phi_6 \leqslant \Phi'_1$.

Enfin, nous avons $\Phi \leqslant \Phi'$ et $\Phi', \varnothing \vdash \mathtt{e} : \alpha \rhd \Phi_6$, donc d'après le lemme 4, l'expression e est typable dans $(\Phi, \varnothing)$ et la construction de l'arbre d'inférence au dessus de $\Phi, \varnothing \vdash \mathtt{e} : \alpha \rhd \Phi'_2$ génère un ensemble de contraintes Φ'_2 vérifiant $\Phi'_2 \leqslant \Phi_6$. Comme $\Phi_6 \leqslant \Phi'_1$, par transitivité de $(\leqslant)$, nous obtenons bien $\Phi'_2 \leqslant \Phi'_1$.

- Cas « `if true then` $\mathtt{e}_1$ `else` $\mathtt{e}_2 \longrightarrow \mathtt{e}_1$ » :
  L'arbre d'inférence de $\Phi, \varnothing \vdash$ `if true then` $\mathtt{e}_1$ `else` $\mathtt{e}_2 : \alpha \rhd \Phi'_1$ est de la forme :

$$\textsc{TIf}\ \dfrac{\dfrac{\overset{}{\triangledown}}{\Phi \vdash \alpha' \leqslant \mathtt{bool} \rhd \Phi_1} \quad \dfrac{\overset{}{\triangledown}}{\Phi_1 \vdash \mathtt{true} : \alpha' \rhd \Phi_2} \quad \dfrac{\overset{T}{\triangledown}}{\Phi_2, \varnothing \vdash \mathtt{e}_1 : \alpha \rhd \Phi_3} \quad \dfrac{\overset{}{\triangledown}}{\Phi_3, \varnothing \vdash \mathtt{e}_2 : \alpha \rhd \Phi'_1}}{\Phi, \varnothing \vdash \mathtt{if\ true\ then}\ \mathtt{e}_1\ \mathtt{else}\ \mathtt{e}_2 : \alpha \rhd \Phi'_1}$$

  avec $\Phi \leqslant \Phi_2$ et $\Phi_3 \leqslant \Phi'_1$.

  Le sous-arbre T montre $\Phi_2, \varnothing \vdash \mathtt{e}_1 : \alpha \rhd \Phi_3$. Puisque $\Phi \leqslant \Phi_2$, une simple application du lemme 4 nous montre qu'il existe un Φ'_2 tel que $\Phi, \varnothing \vdash \mathtt{e}_1 : \alpha \rhd \Phi'_2$ avec $\Phi'_2 \leqslant \Phi_3$. Comme $\Phi_3 \leqslant \Phi'_1$, par transitivité de $(\leqslant)$, nous obtenons bien $\Phi'_2 \leqslant \Phi'_1$.

- Cas « `if false then` $\mathtt{e}_1$ `else` $\mathtt{e}_2 \longrightarrow \mathtt{e}_2$ » :
  La démonstration est très similaire à celle du cas précédent.

- Cas « `match K v with ... ‖ K x → e ‖ ... ⟶ e[x ↦ v]` » :
  L'arbre d'inférence de

$$\Phi, \varnothing \vdash \texttt{match K v with ... ‖ K x → e ‖ ... :} \ \alpha \rhd \Phi_1'$$

  est de la forme :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Phi_2 \vdash \alpha_0 \leqslant \alpha_1 \rhd \Phi_3}\,T_3}{\Phi_2 \vdash \mathtt{K}\,\alpha_0 \leq \{ \ldots \parallel \mathtt{K}\,\alpha_1 \parallel \ldots \} \rhd \Phi_3}\,\textsc{SVariantMatch}}{\Phi_2 \vdash \mathtt{K}\,\alpha_0 \leqslant \{ \ldots \parallel \mathtt{K}\,\alpha_1 \parallel \ldots \} \rhd \Phi_3}\,\textsc{SNewConstraint}}{\Phi_2 \vdash \mathtt{K}\,\alpha_0 \leq \alpha_e \rhd \Phi_3}\,\textsc{STransRight}}{\Phi_1, \varnothing \vdash \mathtt{v} : \alpha_0 \rhd \Phi_2 \qquad \Phi_2 \vdash \mathtt{K}\,\alpha_0 \leqslant \alpha_e \rhd \Phi_3}\,\textsc{SNewConstraint}}{\Phi_1, \varnothing \vdash \mathtt{K\ v} : \alpha_e \rhd \Phi_3}\,\textsc{TConstr}$$

$$\dfrac{\dfrac{\dots \qquad \dfrac{\mathtt{x}:\alpha' \quad \mathtt{x}:\alpha''}{\Phi_3, \{ (\mathtt{x}, \alpha_1) \} \vdash \mathtt{e} : \alpha \rhd \Phi_4}\,T_4 \qquad \dots}{\dots \qquad\qquad\qquad\qquad \dots}}{\Phi, \varnothing \vdash \texttt{match K v with ... ‖ K x → e ‖ ... :} \ \alpha \rhd \Phi_1'}\,\textsc{TMatch}$$

  Nous nous retrouvons alors dans une situation très similaire à celle du troisième cas (« `(λ x . e) v ⟶ e[x ↦ v]` »). De la même manière, nous construisons un arbre d'inférence au dessus de $\Phi_3, \varnothing \vdash \mathtt{e[x \mapsto v]} : \alpha \rhd \Phi_5$ en remplaçant les sous-arbres correspondant au typage des différentes occurrences de `x` dans `e` par une adaptation de T_2. Ceci est possible puisque `x` est lié à α_1 dans l'environnement de typage utilisé pour construire T_4, le sous-arbre T_2 représente une preuve de typage de $(\mathtt{v} : \alpha_0)$ et les variables α_0 et α_1 sont reliées par la relation $\alpha_0 \leqslant \alpha_1$ grâce à T_3.

  Enfin, tout comme dans le troisième cas, nous utilisons le lemme 4 pour montrer qu'il existe Φ_2' tel que $\Phi, \varnothing \vdash \mathtt{e[x \mapsto v]} : \alpha \rhd \Phi_2'$ avec $\Phi_2' \leqslant \Phi_5 \leqslant \Phi_1'$.

- Cas « `match K v with ... ‖ K x → e ‖ ... ‖ x_d → e_d ⟶ e[x ↦ v]` » :

La démonstration est très similaire à celle du cas précédent.

- Cas « $\texttt{match v with K}_1\ \texttt{x}_1 \to \texttt{e}_1 \parallel \ldots \parallel \texttt{K}_n\ \texttt{x}_n \to \texttt{e}_n \parallel \texttt{x}_d \to \texttt{e}_d \longrightarrow \texttt{e}_d[\texttt{x}_d \mapsto \texttt{v}]$ »
  lorsque v n'est pas de la forme $(\texttt{K}_i\ \_)$ pour $i \in [\,1;\ n\,]$:
  L'arbre d'inférence de

$$\Phi, \varnothing \vdash \texttt{match v with K}_1\ \texttt{x}_1 \to \texttt{e}_1 \parallel \ \ldots \ \parallel \texttt{K}_n\ \texttt{x}_n \to \texttt{e}_n \parallel \ \texttt{x}_d \to \texttt{e}_d : \alpha \rhd \Phi'_1$$

  est de la forme :

$$\textsc{TMatchDefault}\ \dfrac{\dfrac{T_2}{\Phi_1, \varnothing \vdash \texttt{v} : \alpha_e \rhd \Phi_2} \quad \dfrac{\ldots}{\ldots} \quad \dfrac{\overset{\texttt{x}_d : \alpha' \qquad \texttt{x}_d : \alpha''}{T_3}}{\Phi_2, \{\ (\texttt{x}_d, \alpha_d)\ \} \vdash \texttt{e}_d : \alpha \rhd \Phi'_1}}{\Phi, \varnothing \vdash \texttt{match v with K}_1\ \texttt{x}_1 \to \texttt{e}_1 \parallel \ \ldots \ \parallel \texttt{K}_n\ \texttt{x}_n \to \texttt{e}_n \parallel \ \texttt{x}_d \to \texttt{e}_d : \alpha \rhd \Phi'_1}$$

  Comme v n'est pas de la forme $(\texttt{K}_i\ \_)$, le sous-arbre T_2 génère une contrainte entre un type construit τ_0^l de la forme $(\tau_0^l \leqslant \alpha_e)$ avec τ_0^l qui n'est pas de la forme $\texttt{K}_i\ \_$ pour $i \in [\,1;\ n]$. Comme Φ_1 contient la contrainte $(\alpha_e \leqslant \{\ \texttt{K}_1\ \alpha_1 \parallel \ \ldots \ \parallel \texttt{K}_n\ \alpha_n \parallel \ \alpha_d\ \})$, un sous-arbre de T_2 prouve $(\tau_0^l \leqslant \{\ \texttt{K}_1\ \alpha_1 \parallel \ \ldots \ \parallel \texttt{K}_n\ \alpha_n \parallel \ \alpha_d\ \})$ et comme τ_0^l n'est pas de la forme $\texttt{K}_{\ i}$ pour $i \in [\,1;\ n\,]$, une instance de la règle SVariantDefault ou SConstrDefault impose la présence de $\tau_0^l \leqslant \alpha_d$ dans Φ_2.
  On se retrouve alors dans un cas très similaire au précédent. On montre qu'il est possible de construire l'arbre d'inférence de $\Phi_2, \varnothing \vdash \texttt{e}_d[\texttt{x}_d \mapsto \texttt{v}] : \alpha \rhd \Phi_3$ en remplaçant dans T_3 les sous-arbres correspondant au typage des occurrences libres de x dans e par une adaptation de T_2. Enfin, le lemme 4 nous montre qu'il existe Φ'_2 tel que $\Phi, \varnothing \vdash \texttt{e}_d[\texttt{x}_d \mapsto \texttt{v}] : \alpha \rhd \Phi'_2$ avec $\Phi'_2 \leqslant \Phi'_1$.

Lemme 6 (Réduction du sujet par ($\longmapsto$))**.**

Soient :

- e_1 *et* e_2 *deux expressions vérifiant* $\mathsf{e}_1 \longmapsto \mathsf{e}_2$
- α *une variable de type*
- Φ *un ensemble de contraintes valide et saturé*
- Φ'_1 *l'ensemble de contraintes obtenu par typage de* $\mathsf{e}_1 : \alpha$ *dans* $(\Phi, \varnothing)$ *en dérivant l'arbre d'inférence au dessus de* $\Phi, \varnothing \vdash \mathsf{e}_1 : \alpha \rhd \Phi'_1$.

Alors le typage de $\mathsf{e}_2 : \alpha$ *dans* $(\Phi, \varnothing)$ *réussit et l'ensemble de contraintes* Φ'_2 *obtenu en dérivant l'arbre d'inférence au dessus de* $\Phi, \varnothing \vdash \mathsf{e}_2 : \alpha \rhd \Phi'_2$ *vérifie* $\Phi'_2 \leqslant \Phi'_1$.

Démonstration (Lemme 6) :

. . . par induction et analyse de cas sur la structure du contexte d'évaluation `E` :

- Cas « `[]` » :
  Conséquence directe du lemme 5.

- Cas « `E e` » :
  Nous souhaitons montrer qu'il est possible de construire l'arbre d'inférence de $\Phi, \varnothing \vdash \mathsf{e}_2\ \mathsf{e} : \alpha \rhd \Phi'_2$ grâce à l'arbre d'inférence de $\Phi, \varnothing \vdash \mathsf{e}_1\ \mathsf{e} : \alpha \rhd \Phi'_1$ sachant que $\mathsf{e}_1 \longmapsto \mathsf{e}_2$. D'après la structure de l'expression (une application), la règle de typage appliquée à la racine de ces arbres est obligatoirement TApp :

$$\textsc{TApp}\ \dfrac{\dfrac{\overset{T_1}{\triangledown}}{\Phi \vdash \alpha_1 \leqslant \alpha_2 \to \alpha_3 \rhd \Phi_1} \quad \dfrac{\overset{T_2}{\triangledown}}{\Phi_1 \vdash \alpha_3 \leqslant \alpha \rhd \Phi_2} \qquad \dfrac{\dfrac{\overset{T_3}{\triangledown}}{\Phi_2, \varnothing \vdash \mathsf{e}_1 : \alpha_1 \rhd \Phi_3} \quad \dfrac{\overset{T_4}{\triangledown}}{\Phi_3, \varnothing \vdash \mathsf{e} : \alpha_2 \rhd \Phi'_1}}{\Phi, \varnothing \vdash \mathsf{e}_1\ \mathsf{e} : \alpha \rhd \Phi'_1}}$$

  Les sous-arbres T_1 et T_2 sont inchangés. L'hypothèse d'induction nous montre que le sous-arbre remplaçant T_3 peut être construit en générant un ensemble de contraintes plus petit. Et enfin, le lemme 4 nous montre que le sous-arbre remplaçant T_4 peut être construit en générant un ensemble de contraintes Φ'_2 plus petit que Φ'_1.

- Cas « `v E` » :
  La structure est similaire à celle du cas précédents. Les sous-arbres T_1, T_2 et T_3 restent inchangés. On utilise l'hypothèse d'induction pour montrer qu'il est possible de

remplacer T_4 en générant un ensemble de contraintes plus petit.

- Cas « $\mathtt{p}^1$ `E` » :
  La situation est similaire mis à part que la règle TAppPrim1 est utilisée à la place de TApp. La structure de l'arbre initial est :

$$\textsc{TApplyPrim1}\ \frac{\dfrac{\overset{T_1}{\triangledown}}{\Phi \vdash T(\mathtt{p}^1) \leqslant \alpha_1 \to \alpha_2 \rhd \Phi_1} \qquad \dfrac{\dfrac{\overset{T_2}{\triangledown}}{\Phi_1 \vdash \alpha_2 \leqslant \alpha \rhd \Phi_2} \qquad \dfrac{\overset{T_3}{\triangledown}}{\Phi_2, \varnothing \vdash \mathtt{e}_1 : \alpha_1 \rhd \Phi_1'}}{}}{\Phi, \varnothing \vdash \mathtt{p}^1\ \mathtt{e}_1 : \alpha \rhd \Phi_1'}$$

  Les sous-arbres T_1 et T_2 sont inchangés, on utilise l'hypothèse d'induction pour montrer qu'il est possible de remplacer T_3 en générant un ensemble de contraintes Φ_2' plus petit que Φ_1'.

- Cas « $\mathtt{p}^2$ `E e` » :
  Similaire au cas précédent.

- Cas « $\mathtt{P}^2$ `v E` » :
  Similaire au cas précédent.

- Cas « `(E, e)` » :
  Nous souhaitons montrer qu'il est possible de construire l'arbre d'inférence de $\Phi, \varnothing \vdash (\mathtt{e}_2,\ \mathtt{e}) : \alpha \rhd \Phi_2'$ grâce à l'arbre d'inférence de $\Phi, \varnothing \vdash (\mathtt{e}_1,\ \mathtt{e}) : \alpha \rhd \Phi_1'$ sachant que $\mathtt{e}_1 \longmapsto \mathtt{e}_2$. Ce dernier arbre est de la forme :

$$\textsc{TPair}\ \frac{\overset{T_1}{\triangledown} \qquad \overset{T_2}{\triangledown} \qquad \overset{T_3}{\triangledown}}{\Phi, \varnothing \vdash \mathtt{e}_1 : \alpha_1 \rhd \Phi_1 \qquad \Phi_1, \varnothing \vdash \mathtt{e} : \alpha_2 \rhd \Phi_2 \qquad \Phi_2 \vdash \alpha_1 \times \alpha_2 \leqslant \alpha \rhd \Phi_1'}{\Phi, \varnothing \vdash (\mathtt{e}_1,\ \mathtt{e}) : \alpha \rhd \Phi_1'}$$

  Comme dans les cas précédent, on utilise l'hypothèse d'induction pour montrer qu'il est possible de remplacer T_1 par un sous-arbre générant un ensemble de contraintes plus petit, le lemme 4 pour le remplacement du sous-arbre T_2 et le lemme 3 pour le remplacement du sous-arbre T_3.

- Cas « `(v, E)` » :
  Similaire au cas précédent.

- Cas « `K E` » :
  Similaire au cas précédent.

- Cas « `let x = E in e` » :
  Nous souhaitons montrer qu'il est possible de construire l'arbre d'inférence de $\Phi, \varnothing \vdash \texttt{let x = } \mathtt{e}_2 \texttt{ in e} : \alpha \rhd \Phi'_2$ grâce à l'arbre d'inférence de $\Phi, \varnothing \vdash \texttt{let x = } \mathtt{e}_1 \texttt{ in e} : \alpha \rhd \Phi'_1$ sachant que $\mathtt{e}_1 \longmapsto \mathtt{e}_2$. Ce dernier arbre est de la forme :

$$\textsc{TLet}\ \frac{\dfrac{\overset{T_1}{\triangledown}}{\Phi, \varnothing \vdash \mathtt{e}_1 : \alpha' \rhd \Phi_1} \quad \dfrac{\overset{T_2}{\triangledown}}{\Phi_1, \{\ (\mathtt{x}, \textsc{gen}(\alpha', \Phi_1, \varnothing))\ \} \vdash \mathtt{e} : \alpha \rhd \Phi'_1}}{\Phi, \varnothing \vdash \texttt{let x} = \mathtt{e}_1 \texttt{ in e} : \alpha \rhd \Phi'_1}$$

  La structure de l'arbre d'inférence de $\Phi, \varnothing \vdash \texttt{let x = } \mathtt{e}_2 \texttt{ in e} : \alpha \rhd \Phi'_2$ que nous allons construire est similaire :

$$\textsc{TLet}\ \frac{\dfrac{\overset{T'_1}{\triangledown}}{\Phi, \varnothing \vdash \mathtt{e}_2 : \alpha' \rhd \Phi_2} \quad \dfrac{\overset{T'_2}{\triangledown}}{\Phi_2, \{\ (\mathtt{x}, \textsc{gen}(\alpha', \Phi_1, \varnothing))\ \} \vdash \mathtt{e} : \alpha \rhd \Phi'_1}}{\Phi, \varnothing \vdash \texttt{let x} = \mathtt{e}_2 \texttt{ in e} : \alpha \rhd \Phi'_1}$$

  L'hypothèse d'induction nous montre qu'il est effectivement possible de construire T'_1, et que Φ_2 vérifie $\Phi_2 \leqslant \Phi_1$. Nous en déduisons la relation suivante entre schémas de type :

$$\textsc{gen}(\alpha', \Phi_2, \varnothing) \leqslant \textsc{gen}(\alpha', \Phi_1, \varnothing)$$

  et donc la relation suivante entre environnements de typage :

$$\{\ (x, \textsc{gen}(\alpha', \Phi_2, \varnothing))\ \} \leqslant \{\ (x, \textsc{gen}(\alpha', \Phi_1, \varnothing))\ \}$$

  Grâce au lemme 4, nous concluons que T'_2 est effectivement constructible, et que l'ensemble de contraintes Φ'_2 généré vérifie bien $\Phi'_2 \leqslant \Phi'_1$.

- Cas « `if E then` $\mathtt{e}_1$ `else` $\mathtt{e}_2$ » :
  Nous souhaitons montrer qu'il est possible de construire l'arbre d'inférence de $\Phi, \varnothing \vdash \texttt{if } \mathtt{e}_2 \texttt{ then } \mathtt{e}_3 \texttt{ else } \mathtt{e}_4 : \alpha \rhd \Phi'_1$ grâce à l'arbre d'inférence de $\Phi, \varnothing \vdash \texttt{if } \mathtt{e}_1 \texttt{ then } \mathtt{e}_3 \texttt{ else } \mathtt{e}_4 : \alpha \rhd \Phi'_1$ sachant que $\mathtt{e}_1 \longmapsto \mathtt{e}_2$. Ce dernier arbre est de la

forme :

$$\textsc{TIf}\ \dfrac{\dfrac{\overset{T_1}{\triangledown}}{\Phi \vdash \alpha' \leqslant \texttt{bool} \rhd \Phi_1} \quad \dfrac{\overset{T_2}{\triangledown}}{\Phi_1, \varnothing \vdash \texttt{e}_1 : \alpha' \rhd \Phi_2} \quad \dfrac{\overset{T_3}{\triangledown}}{\Phi_2, \varnothing \vdash \texttt{e}_3 : \alpha \rhd \Phi_3} \quad \dfrac{\overset{T_4}{\triangledown}}{\Phi_3, \varnothing \vdash \texttt{e}_4 : \alpha \rhd \Phi_4}}{\Phi, \varnothing \vdash \texttt{if}\ \texttt{e}_1\ \texttt{then}\ \texttt{e}_3\ \texttt{else}\ \texttt{e}_4 : \alpha \rhd \Phi_1'}$$

Comme précédemment, le sous-arbre T_1 ne change pas, il suffit d'appliquer l'hypothèse d'induction pour montrer qu'il est possible de remplacer T_2 par un sous-arbre générant un ensemble de contraintes plus petit, et le lemme 4 deux fois pour T_3 et T_4.

- Cas « `match E with` $\texttt{K}_1\ \texttt{x}_1 \to \texttt{e}_1 \parallel \ldots \parallel \texttt{K}_n\ \texttt{x}_n \to \texttt{e}_n$ » :
  Nous souhaitons montrer qu'il est possible de construire l'arbre d'inférence de `match` $\texttt{e}_1$ `with` $\texttt{K}_1\ \texttt{x}_1 \to \texttt{e}_1' \parallel \ldots \parallel \texttt{K}_n\ \texttt{x}_n \to \texttt{e}_n'$ grâce à l'arbre d'inférence de `match` $\texttt{e}_2$ `with` $\texttt{K}_1\ \texttt{x}_1 \to \texttt{e}_1' \parallel \ldots \parallel \texttt{K}_n\ \texttt{x}_n \to \texttt{e}_n'$ sachant que $\texttt{e}_1 \longmapsto \texttt{e}_2$. Ce dernier arbre est de la forme :

$$\textsc{TMatch}\ \dfrac{\dfrac{\overset{T_1}{\triangledown}}{\Phi \vdash \alpha_e \leqslant \{\, \texttt{K}_1\ \alpha_1 \parallel \ldots \parallel \texttt{K}_n\ \alpha_n \,\} \rhd \Phi_0} \quad \dfrac{\overset{T_2}{\triangledown}}{\Phi_0, \varnothing \vdash \texttt{e}_1 : \alpha_e \rhd \Phi_1} \quad \dfrac{\triangledown}{\Phi_1, \varnothing \oplus (\texttt{x}_1,\ \alpha_1) \vdash \texttt{e}_1' : \alpha \rhd \Phi_2} \quad \ldots \quad \dfrac{\triangledown}{\Phi_n, \varnothing \oplus (\texttt{x}_n,\ \alpha_n) \vdash \texttt{e}_n' : \alpha \rhd \Phi_1'}}{\Phi, \varnothing \vdash \texttt{match}\ \texttt{e}_1\ \texttt{with}\ \texttt{K}_1\ \texttt{x}_1 \to \texttt{e}_1 \parallel \ldots \parallel \texttt{K}_n\ \texttt{x}_n \to \texttt{e}_n : \alpha \rhd \Phi_1'}$$

  Comme précédemment, le sous-arbre T_1 est inchangé. On utilise l'hypothèse d'induction pour montrer qu'il est possible de remplacer T_2 par un sous-arbre générant un ensemble de contraintes plus petit, et le lemme 4 pour les autres sous-arbres.

- Cas « `match E with` $\texttt{K}_1\ \texttt{x}_1 \to \texttt{e}_1 \parallel \ldots \parallel \texttt{K}_n\ \texttt{x}_n \to \texttt{e}_n \parallel \texttt{x}_d \to \texttt{e}_d$ » :
  Similaire au cas précédent.

Lemme 7 (Réduction du sujet par $(\longmapsto\!\!\!\!\!\rightarrow\!\!\!\!\!\rightarrow)$)**.**

Soient :

- e_1 *et* e_2 *deux expressions vérifiant* $e_1 \longmapsto\!\!\!\!\!\rightarrow\!\!\!\!\!\rightarrow e_2$.
- α *une variable de type*
- Φ *un ensemble de contraintes valide et saturé*
- Φ'_1 *l'ensemble de contraintes obtenu par typage de* $e_1 : \alpha$ *dans* $(\Phi, \varnothing)$ *en dérivant l'arbre d'inférence au dessus de* $\Phi, \varnothing \vdash e_1 : \alpha \rhd \Phi'_1$.

Alors le typage de $e_2 : \alpha$ *dans* $(\Phi, \varnothing)$ *réussit et l'ensemble de contraintes* Φ'_2 *obtenu en dérivant l'arbre d'inférence au dessus de* $\Phi, \varnothing \vdash e_2 : \alpha \rhd \Phi'_2$ *vérifie* $\Phi'_2 \leqslant \Phi'_1$.

Démonstration (Lemme 7) :

... par induction et analyse de cas sur la définition de $(\longmapsto\!\!\!\!\!\rightarrow\!\!\!\!\!\rightarrow)$:

- Si $e_1 \longmapsto\!\!\!\!\!\rightarrow\!\!\!\!\!\rightarrow e_2$ parce que $e_1 = e_2$:
  Trivial.

- Si $e_1 \longmapsto\!\!\!\!\!\rightarrow\!\!\!\!\!\rightarrow e_2$ parce que $e_1 \longmapsto e_2$:
  Simple conséquence du lemme 6.

- Si $e_1 \longmapsto\!\!\!\!\!\rightarrow\!\!\!\!\!\rightarrow e_2$ parce qu'il existe une expression e telle que $e_1 \longmapsto\!\!\!\!\!\rightarrow\!\!\!\!\!\rightarrow e$ et $e \longmapsto e_2$:
  En appliquant l'hypothèse d'induction, nous montrons que le typage de $e : \alpha$ réussit dans $(\Phi, \varnothing)$ et produit un ensemble de contraintes Φ' vérifiant $\Phi' \leqslant \Phi'_1$:

  $$\Phi, \varnothing \vdash e : \alpha \rhd \Phi'$$

  En appliquant le lemme 6, nous montrons que le typage de $e_2 : \alpha$ dans $(\Phi, \varnothing)$ réussit et produit un ensemble de contraintes que nous nommons Φ'_2 vérifiant $\Phi'_2 \leqslant \Phi'$:

  $$\Phi, \varnothing \vdash e_2 : \alpha \rhd \Phi'_2$$

  Comme la relation $(\leqslant)$ est transitive, nous obtenons bien $\Phi'_2 \leqslant \Phi'_1$.

Lemme 8 (Les expressions bloquées ne sont pas typables)**.**

Soit e_b *une expression bloquée. Il n'existe aucun schéma de type* σ *tel que* $e_b : \sigma$.

Démonstration (Lemme 8) :

... par induction sur la structure des expressions et analyse de cas sur les différentes formes d'expressions bloquées. En réalité, l'analyse de chacun des cas est trivial. Le principal intérêt de cette démonstration est de répertorier toutes les formes d'expressions bloquées.

L'expression $\mathtt{e}_b$ étant bloquée, elle est obligatoirement de l'une des formes suivantes :

- Cas « $\mathtt{e}_b = \mathtt{x}$ » :
  L'environnement de typage étant vide, la règle TVar ne peut pas être appliquée et la variable $\mathtt{x}$ n'est pas typable et on obtient le message classique « unbound variable $\mathtt{x}$ ».
- Cas « $\mathtt{e}_b = \mathtt{c}\ \ \mathtt{v}$ » :
  S'il était constructible complètement, de par la structure de l'expression, l'arbre d'inférence serait de la forme :

$$\dfrac{\dfrac{\dfrac{}{\varnothing \vdash \alpha_1 \leqslant \alpha_2 \to \alpha_3 \rhd \Phi'}\ T_1 \qquad \dfrac{}{\Phi' \vdash \alpha_3 \leqslant \alpha \rhd \Phi''}\ T_2}{\Phi'', \varnothing \vdash \mathtt{c} : \alpha_1 \rhd \Phi'''}\ \textsc{TConst} \qquad \dfrac{}{\Phi''' \vdash T(c) \leqslant \alpha_1 \rhd \Phi'''}\ T_3 \qquad \dfrac{}{\Phi''', \varnothing \vdash \mathtt{v} : \alpha_2 \rhd \Phi''''}\ T_4}{\varnothing, \varnothing \vdash \mathtt{c}\ \mathtt{v} : \alpha \rhd \Phi''''}\ \textsc{TApp}$$

  On fait ici l'hypothèse que la fonction T est correctement construite, et que puisqu'aucune constante n'est fonctionnelle, l'instanciation de $T(\mathtt{c})$ génère obligatoirement un type construit τ^l différent de $(\to)$ et vérifiant $(\tau^l \leqslant \alpha_1)$. Après application de la transitivité de $(\leqslant)$ via la règle STransRight dans T_2, on aboutirait à la contrainte $(\tau^l \leq \alpha_2 \to \alpha_3)$. Comme τ^l est un type construit différent de $(\to)$, aucune règle de typage ne s'appliquerait pour saturer la contrainte $(\tau^l \leq \alpha_2 \to \alpha_3)$ et la construction de l'arbre d'inférence s'arrêterait bien sur une erreur.
- Cas « $\mathtt{e}_b = (\mathtt{K}\ \ \mathtt{v}_1)\ \ \mathtt{v}_2$ » :
  L'arbre d'inférence, s'il était constructible complètement, serait de la forme :

$$\dfrac{\dfrac{\dfrac{}{\varnothing \vdash \alpha_1 \leqslant \alpha_2 \to \alpha_3 \rhd \Phi'}\ T_1 \qquad \dfrac{}{\Phi' \vdash \alpha_3 \leqslant \alpha \rhd \Phi''}\ T_2}{\Phi'', \varnothing \vdash \mathtt{K}\ \mathtt{v}_1 : \alpha_1 \rhd \Phi'''}\ \textsc{TConstr} \qquad \dfrac{}{\Phi''' \vdash \mathtt{v}_2 : \alpha_2 \rhd \Phi''''}\ T_4}{\varnothing, \varnothing \vdash (\mathtt{K}\ \mathtt{v}_1)\ \mathtt{v}_2 : \alpha \rhd \Phi''''}\ \textsc{TApp} \qquad \dfrac{}{\Phi''', \varnothing \vdash \mathtt{v}_2 : \alpha_2 \rhd \Phi''''}\ T_3$$

  L'application de la règle TConstr dans T_3 engendrerait alors une contrainte de la forme $(\mathtt{K}\ \alpha \leqslant \alpha_1)$. Après application de la transitivité, on aboutirait à la contrainte $(\mathtt{K}\ \alpha \leqslant \alpha_2 \to \alpha_3)$ sur laquelle aucune règle ne s'applique.

- Cas « $\mathtt{e}_b = (\mathtt{v}_1, \mathtt{v}_2)\ \mathtt{v}_3$ » : Comme précédemment mais avec la règle TPair, on aboutirait à une contrainte de la forme $(\alpha_3 \times \alpha_4 \leqslant \alpha_2 \to \alpha)$ sur laquelle aucune règle de saturation ne s'applique.

- Cas « $\mathtt{e}_b = \mathtt{p}^1\ \mathtt{v}$ avec $\delta_1(\mathtt{p}^1, \mathtt{v})$ non défini » :
  S'il était constructible, l'arbre d'inférence serait de la forme :

$$\textsc{TApplyPrim1}\ \frac{\dfrac{\overset{T_1}{\triangledown}}{\varnothing \vdash T(\mathtt{p}^1) \leqslant \alpha_1 \to \alpha_2 \rhd \Phi'} \qquad \dfrac{\dfrac{\overset{T_2}{\triangledown}}{\Phi' \vdash \alpha_2 \leqslant \alpha \rhd \Phi''} \qquad \dfrac{\overset{T_3}{\triangledown}}{\Phi'', \varnothing \vdash \mathtt{v} : \alpha_1 \rhd \Phi'''}}{\varnothing, \varnothing \vdash \mathtt{p}^1\ \mathtt{v} : \alpha \rhd \Phi''}}$$

  Il s'agit ici d'une hypothèse sur la compatibilité de T et de δ_1. En l'occurrence, la fonction T doit être définie de telle sorte que les contraintes engendrées par $(T(\mathtt{p}^1) \leqslant \alpha_1 \to \alpha_2)$ doivent contraindre suffisamment α_1 pour que si le typage de $(\mathtt{v} : \alpha_1)$ réussit, alors $\delta_1(\mathtt{p}^1, \mathtt{v})$ est défini.

- Cas « $\mathtt{e}_b = \mathtt{p}^2\ \mathtt{v}_1\ \mathtt{v}_2$ avec $\delta_2(\mathtt{p}^2, \mathtt{v}_1, \mathtt{v}_2)$ non défini » :
  Similaire au cas précédent, mais avec δ_2.

- Cas « $\mathtt{e}_b = \mathtt{if}\ \lambda\,\mathtt{x}\,.\,\mathtt{e}\ \mathtt{then}\ \mathtt{e}_1\ \mathtt{else}\ \mathtt{e}_2$ » :
  Si l'arbre d'inférence était constructible, il serait de la forme :

$$\textsc{TIf}\ \frac{\dfrac{\overset{T_1}{\triangledown}}{\varnothing \vdash \alpha' \leqslant \mathtt{bool} \rhd \Phi'} \qquad \textsc{TLambda}\ \dfrac{\dfrac{\dots}{\dots} \quad \dfrac{\overset{T_2}{\triangledown}}{\Phi''' \vdash \alpha_1 \to \alpha_2 \leqslant \alpha' \rhd \Phi''''}}{\Phi', \varnothing \vdash \lambda \mathtt{x.e} : \alpha' \rhd \Phi''} \quad \dfrac{\dots}{\dots} \quad \dfrac{\dots}{\dots}}{\varnothing, \varnothing \vdash \mathtt{if}\ \lambda \mathtt{x.e}\ \mathtt{then}\ \mathtt{e}_1\ \mathtt{else}\ \mathtt{e}_2 \rhd \Phi'''''}$$

  Après application de la transitivité de $(\leqslant)$ dans T_2 par la règle STransRight, on aboutirait obligatoirement à la contrainte $(\alpha_1 \to \alpha_2 \leqslant \mathtt{bool})$ sur laquelle aucune règle de saturation ne s'applique.

- Cas « $\mathtt{e}_b = \mathtt{if}\ (\mathtt{v}_1, \mathtt{v}_2)\ \mathtt{then}\ \mathtt{e}_1\ \mathtt{else}\ \mathtt{e}_2$ » :
  Ce cas est similaire au précédent. L'application de la règle TPair engendrerait une contrainte de la forme $(\alpha_1 \times \alpha_2 \leqslant \mathtt{bool})$ sur laquelle aucune règle ne s'applique.

- Cas « $\mathtt{e}_b = \mathtt{if}\ \mathtt{K}\ \mathtt{v}\ \mathtt{then}\ \mathtt{e}_1\ \mathtt{else}\ \mathtt{e}_2$ » :
  Ce cas est similaire au précédent. L'application de la règle TConstr engendrerait une contrainte de la forme $(\mathtt{K}\,\alpha_1 \leqslant \mathtt{bool})$ sur laquelle aucune règle ne s'applique.

- Cas « $\mathtt{e}_b = \mathtt{if\ c\ then\ e}_1\ \mathtt{else\ e}_2$ avec $\mathtt{c} \notin \{\ \mathtt{true},\ \mathtt{false}\ \}$ » :
  Il s'agit une nouvelle fois d'une hypothèse sur la validité de la fonction T, à savoir que seuls `true` et `false` sont typés comme des booléens. Autrement dit, le schéma de type renvoyé par $T(\mathtt{c})$ doit être incompatible avec `bool` pour toutes les constantes autres que `true` et `false`.

  Les trois cas qui viennent, concernant l'évaluation d'un `match` fermé (c'est-à-dire sans cas par défaut), sont très similaires aux cas précédents concernant le `if` :
- Cas « $\mathtt{e}_b = \mathtt{match}\ \ \lambda\,\mathtt{x}\,.\,\mathtt{e}\ \mathtt{with}\ \ \mathtt{K}_1\ \mathtt{x}_1 \to \mathtt{e}_1 \parallel\ \ldots\ \parallel\ \mathtt{K}_n\ \mathtt{x}_n \to \mathtt{e}_n$ » :
  Si l'arbre d'inférence était constructible, il serait de la forme :

$$\textsc{TMatch}\ \dfrac{\dfrac{}{\varnothing \vdash \alpha_e \leqslant \{\ \mathtt{K}_1\ \alpha_1 \parallel\ \ldots\ \parallel\ \mathtt{K}_n\ \alpha_n\ \} \rhd \Phi'}\,T_1 \qquad \dfrac{\dfrac{}{\ldots} \quad \dfrac{}{\Phi'' \vdash \alpha'_1 \to \alpha'_2 \leqslant \alpha_e \rhd \Phi'''}\,T_2}{\Phi', \varnothing \vdash \lambda\mathtt{x.e} : \alpha_e \rhd \Phi'''} \quad \dfrac{}{\ldots} \qquad \dfrac{}{\ldots}}{\varnothing, \varnothing \vdash \mathtt{match}\ \lambda\mathtt{x.e}\ \mathtt{with}\ \mathtt{K}_1\ \mathtt{x}_1 \to \mathtt{e}_1 \parallel\ \ldots\ \parallel\ \mathtt{K}_n\ \mathtt{x}_n \to \mathtt{e}_n : \alpha \rhd \Phi'''}$$

  L'application de la transitivité de $(\leqslant)$ via la règle STransRight engendre la contrainte $(\alpha'_1 \to \alpha'_2 \leqslant \{\ \mathtt{K}_1\ \alpha_1 \parallel\ \ldots\ \parallel\ \mathtt{K}_n\ \alpha_n\ \})$ sur laquelle aucune règle d'inférence ne s'applique.

- Cas « $\mathtt{e}_b = \mathtt{match}\ (\mathtt{v}_1, \mathtt{v}_2)\ \mathtt{with}\ \ \mathtt{K}_1\ \mathtt{x}_1 \to \mathtt{e}_1 \parallel\ \ldots\ \parallel\ \mathtt{K}_n\ \mathtt{x}_n \to \mathtt{e}_n$ » :
  Ce cas est similaire au précédent. La tentative d'inférence de types pour ce programme engendre une contrainte de la forme $(\alpha'_1 \times \alpha'_2 \leqslant \{\ \mathtt{K}_1\ \alpha_1 \parallel\ \ldots\ \parallel\ \mathtt{K}_n\ \alpha_n\ \})$ sur laquelle aucune règle de saturation ne s'applique.

- Cas « $\mathtt{e}_b = \mathtt{match\ c\ with}\ \ \mathtt{K}_1\ \mathtt{x}_1 \to \mathtt{e}_1 \parallel\ \ldots\ \parallel\ \mathtt{K}_n\ \mathtt{x}_n \to \mathtt{e}_n$ » :
  Pour ce cas, nous avons encore une fois besoin d'une propriété de validité de T associée au fait qu'aucune constante n'est un constructeur de données : quelle que soit la constante `c`, la fonction T appliquée à `c` doit générer un schéma de type incompatible avec un ensemble clos de variants (c'est-à-dire de la forme $\{\ \mathtt{K}_1\ \alpha_1 \parallel\ \ldots\ \parallel\ \mathtt{K}_n\ \alpha_n\ \}$).

  Tous les cas restants concernent une expression qui est bloquée parce qu'une de ses sous-expressions est bloquée. Pour rappel, l'existence d'une sous-expression bloquée n'entraîne pas toujours le blocage de l'expression englobante. Le contexte d'évaluation `E` définit **la** prochaine sous-expression à évaluer et si l'expression englobante est bloquée, ce ne peut être qu'à cause d'elle.

- Cas « $\mathtt{e}_b = \mathtt{e}_1\ \mathtt{e}_2$ avec $\mathtt{e}_1$ bloquée » :
  La seule règle de typage applicable sur une telle expression est TApp, qui impose de typer $\mathtt{e}_1$. Or, $\mathtt{e}_1$ est bloquée, donc non-typable par hypothèse d'induction.

- Cas « $\mathtt{e}_b = \mathtt{v}\ \mathtt{e}$ avec $\mathtt{e}$ bloquée » :
  Similaire au cas précédent.

- Cas « $\mathtt{e}_b = \mathtt{p}^1\ \mathtt{e}$ avec $\mathtt{e}$ bloquée » :
  La seule règle de typage applicable sur une telle expression est TAppPrim1 qui impose de typer $\mathtt{e}$. L'expression $\mathtt{e}$ est bloquée donc non-typable par hypothèse d'induction.

- Cas « $\mathtt{e}_b = \mathtt{p}^2\ \mathtt{e}_1\ \mathtt{e}_2$ avec $\mathtt{e}_1$ bloquée » :
  Similaire au cas précédent avec la règle TAppPrim2.

- Cas « $\mathtt{e}_b = \mathtt{p}^2\ \mathtt{v}\ \mathtt{e}$ avec $\mathtt{e}$ bloquée » :
  Similaire au cas précédent.

- Cas « $\mathtt{e}_b = (\mathtt{e}_1, \mathtt{e}_2)$ avec $\mathtt{e}_1$ bloquée » :
  Similaire au cas précédent avec la règle TPair.

- Cas « $\mathtt{e}_b = (\mathtt{v}, \mathtt{e})$ avec $\mathtt{e}$ bloquée » :
  Similaire au cas précédent.

- Cas « $\mathtt{e}_b = \mathtt{K}\ \mathtt{e}$ avec $\mathtt{e}$ bloquée » :
  Similaire au cas précédent avec la règle TConstr.

- Cas « $\mathtt{e}_b = \mathtt{let}\ \mathtt{x} = \mathtt{e}_1\ \mathtt{in}\ \mathtt{e}_2$ avec $\mathtt{e}_1$ bloquée » :
  Similaire au cas précédent avec la règle TLet.

- Cas « $\mathtt{e}_b = \mathtt{if}\ \mathtt{e}_1\ \mathtt{then}\ \mathtt{e}_2\ \mathtt{else}\ \mathtt{e}_3$ avec $\mathtt{e}_1$ bloquée » :
  Similaire au cas précédent avec la règle TIf.

- Cas « $\mathtt{e}_b = \mathtt{match}\ \mathtt{e}\ \mathtt{with}\ \ldots$ avec $\mathtt{e}$ bloquée » :
  Similaire au cas précédent avec la règle TMatch ou TMatchDefault en fonction de la présence d'un cas par défaut ou non dans le filtrage.

Lemme 9 (Évaluation uniforme)**.**

Toute expression $\mathtt{e}$ *vérifie une et une seule des propriétés suivantes :*

- *L'évaluation de* $\mathtt{e}$ *boucle indéfiniment*
- *Il existe une valeur* $\mathtt{v}$ *telle que* $\mathtt{e} \longmapsto\!\!\!\!\!\rightarrow\ \mathtt{v}$
- *Il existe une expression bloquée* $\mathtt{e}_b$ *telle que* $\mathtt{e} \longmapsto\!\!\!\!\!\rightarrow\ \mathtt{e}_b$.

Démonstration (Lemme 9) :

Si nous ne sommes pas dans le premier cas du lemme, et que donc l'évaluation ne boucle pas, cela signifie qu'il existe une expression e' telle que $\mathsf{e} \longmapsto\!\!\!\!\!\rightarrow \mathsf{e}'$ et aucune expression e'' telle que $\mathsf{e}' \longmapsto \mathsf{e}''$. Nous distinguons alors deux cas :

- Soit e' est une valeur, nous sommes alors dans le second cas du lemme.
- Soit e' n'est pas une valeur, c'est donc une expression bloquée par définition, et nous sommes dans le troisième cas du lemme.

Théorème 2 (Validité forte).

Pour toute expression e et tout schéma de type σ, si e et σ vérifient $\mathsf{e} : \sigma$ alors l'une des propriétés suivantes est vérifiée :

- *L'évaluation de e boucle indéfiniment.*
- *L'expression e s'évalue en la valeur v (c'est-à-dire $\mathsf{e} \longmapsto\!\!\!\!\!\rightarrow \mathsf{v}$) et $\mathsf{v} : \sigma$.*

Démonstration (Théorème 2) :

Nous démontrons tout d'abord que le dernier cas du lemme 9 (évaluation uniforme) est impossible lorsqu'il existe σ tel que $(\mathsf{e} : \sigma)$.
Supposons qu'il existe :

- e et σ vérifiant $(\mathsf{e} : \sigma)$
- α une variable de type
- e_b une expression bloquée telle que $\mathsf{e} \longmapsto\!\!\!\!\!\rightarrow \mathsf{e}_b$

Et déduisons-en une absurdité.
Par définition de la relation $(:)$, l'expression e est typable avec α dans $\varnothing, \varnothing$ en produisant un ensemble de contraintes Φ vérifiant $\textsc{gen}(\alpha, \Phi, \varnothing) \leqslant \sigma$:

$$\varnothing, \varnothing \vdash \mathsf{e} : \alpha \rhd \Phi$$

D'après le lemme 7, puisque $\mathsf{e} \longmapsto\!\!\!\!\!\rightarrow \mathsf{e}_b$, et e est typable, l'expression e_b est typable. Or, e_b est bloquée et donc non-typable d'après le lemme 8. Contradiction.

Montrons maintenant que lorsque $\mathsf{e} \longmapsto\!\!\!\!\!\rightarrow \mathsf{v}$ avec v une valeur, alors v vérifie la relation $(\mathsf{v} : \sigma)$.
D'après le lemme 7, la valeur v est typable dans $\varnothing, \varnothing$ en générant un ensemble de contraintes Φ' vérifiant $\Phi' \leqslant \Phi$:

$$\varnothing, \varnothing \vdash v : \alpha \rhd \Phi'$$

Comme $\Phi' \leqslant \Phi$, nous avons bien $\textsc{gen}(\alpha, \Phi', \varnothing) \leqslant \sigma$. Par définition de la relation $(:)$, nous pouvons conclure directement $(\mathsf{v} : \sigma)$