L'inférence de types dans ML^F

Des types graphiques aux contraintes graphiques

Qui ? Boris Yakobowski, Didier Rémy

Où ? INRIA, équipe Gallium

Quand ? 11 Février 2008

Plan

L'inférence de types en ML^F

Inférence dans ML

Utilise un mélange d'unification et de généralisation, suivant une stratégie donnée

L'inférence de types en ML^F

Inférence dans ML

Utilise un mélange d'unification et de généralisation, suivant une stratégie donnée

ML^F est une extension de ML, l'inférence se fait de la même façon.

- L'unification dans les types graphiques est un problème déjà résolu
  - Solutions principales
  - Même complexité qu'en ML (linéaire)

L'inférence de types en ML^F

Inférence dans ML

Utilise un mélange d'unification et de généralisation, suivant une stratégie donnée

ML^F est une extension de ML, l'inférence se fait de la même façon.

- L'unification dans les types graphiques est un problème déjà résolu
- Étendre le concept de généralisation à la structure plus riche des types de ML^F

L'inférence de types en ML^F

Inférence dans ML

Utilise un mélange d'unification et de généralisation, suivant une stratégie donnée

ML^F est une extension de ML, l'inférence se fait de la même façon.

- L'unification dans les types graphiques est un problème déjà résolu
- Étendre le concept de généralisation à la structure plus riche des types de ML^F
- Pas de stratégie prédéfinie, mais une approche à base de contraintes

L'inférence de types en ML^F

Inférence dans ML

Utilise un mélange d'unification et de généralisation, suivant une stratégie donnée

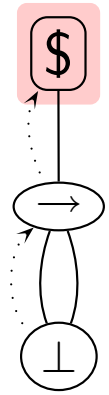
ML^F est une extension de ML, l'inférence se fait de la même façon.

- L'unification dans les types graphiques est un problème déjà résolu
- Étendre le concept de généralisation à la structure plus riche des types de ML^F
- Pas de stratégie prédéfinie, mais une approche à base de contraintes

Une généralisation des types graphiques

Les contraintes graphiques

Les noeuds schéma

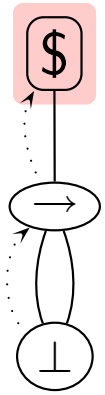


Deux visions

- Introduire un schéma de types, l'unique fils du noeud
- Indiquer que l'on peut généraliser à ce point
- Reflètent la forme de la contrainte.

Dans les contraintes graphiques, chaque noeud du terme source est un noeud schéma

Les noeuds schéma



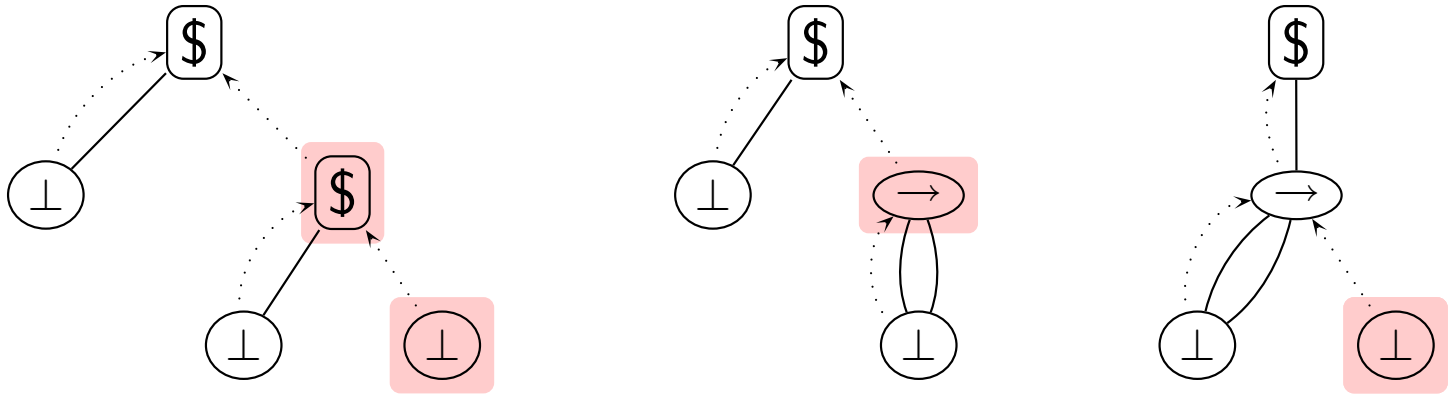
Deux visions

- Introduire un schéma de types, l'unique fils du noeud
- Indiquer que l'on peut généraliser à ce point
- Reflètent la forme de la contrainte.

Dans les contraintes graphiques, chaque noeud du terme source est un noeud schéma

Noeuds "gelés", ne suivant pas la relation d'instance usuelle.
(Par exemple : impossible de fusionner deux noeuds schémas)

Noeuds existentiels

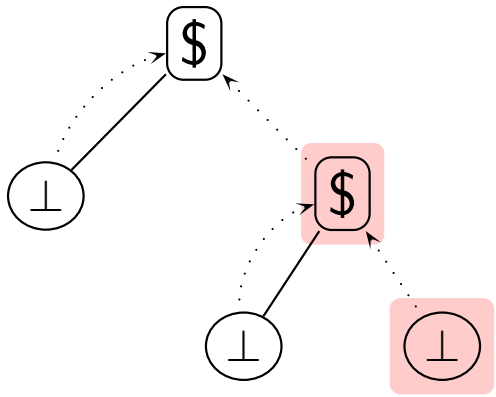


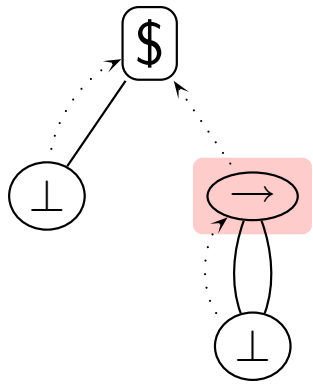
- Des noeuds non accessibles en suivant uniquement la structure

  Un noeud est existentiel s'il relie deux composantes connexes pour les flêches de structure

- Les noeuds schémas autres que la racine doivent être existentiels
- Nécessairement liés sur un noeud schéma

Noeuds existentiels



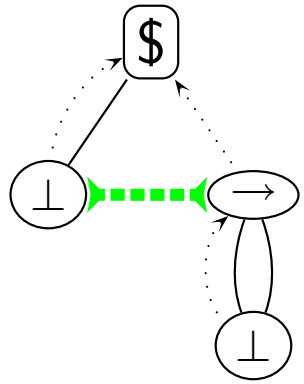


- Des noeuds non accessibles en suivant uniquement la structure

  Un noeud est existentiel s'il relie deux composantes connexes pour les flêches de structure

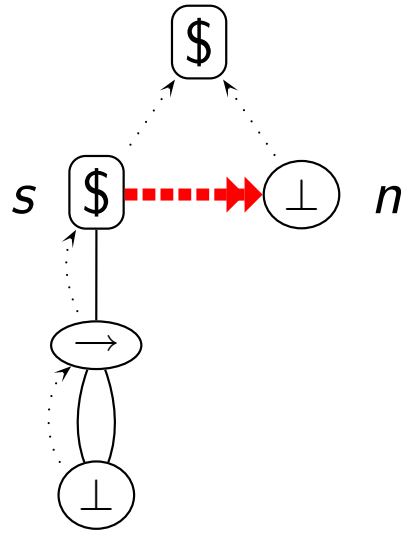
- Les noeuds schémas autres que la racine doivent être existentiels
- Nécessairement liés sur un noeud schéma

Arcs d'unification



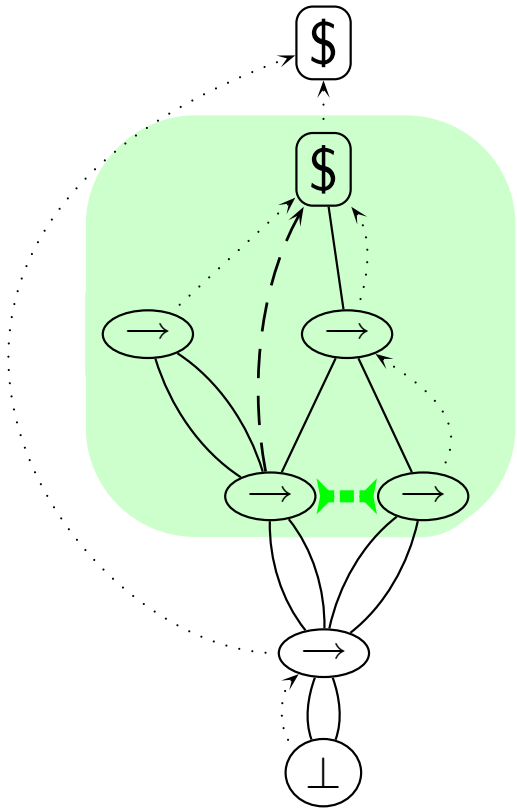
- Relient deux noeuds (non schéma)
- Demandent que les deux noeuds soient unifés (fusionnés)
- Quelques conditions pour que l'arc puisse être résolu de façon principale (non limitantes en pratique)

Arcs d'instance



- Relient un noeud schéma à un noeud non schéma
(lié sur un noeud schéma)
- Le type sous n doit être une instance du schéma représenté par s.

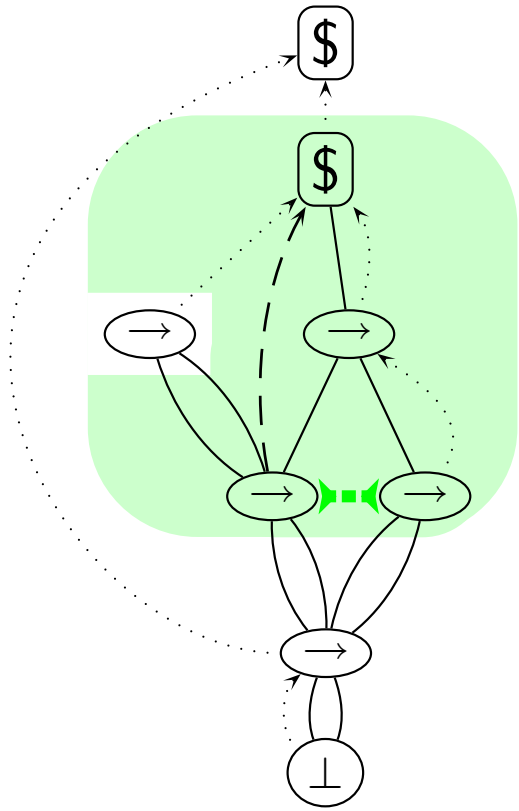
Noeuds à l'intérieur d'un schéma



Intérieur

Ensemble des noeuds transitivement liés sur un noeud

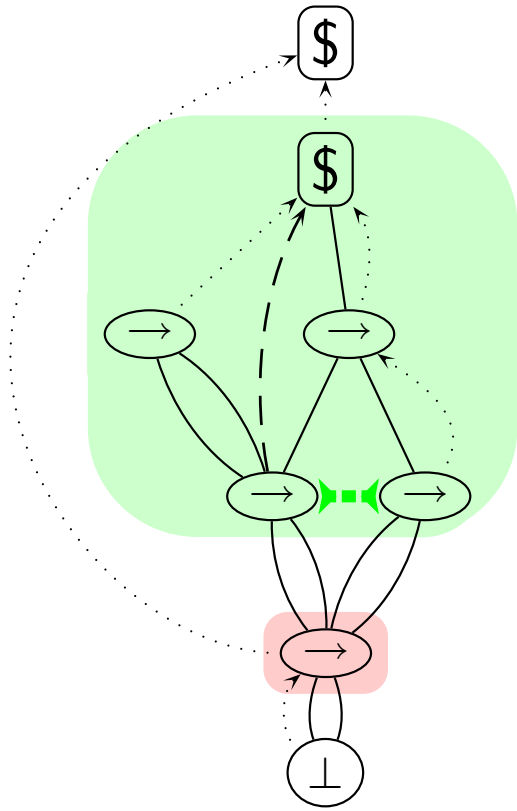
Noeuds à l'intérieur d'un schéma



Intérieur structurel

Restriction de l'intérieur aux noeuds accessibles structurellement

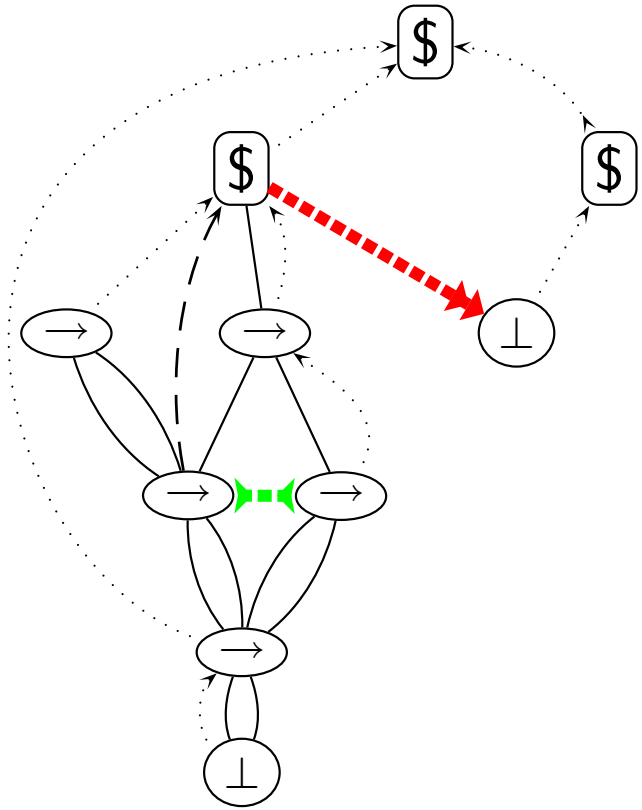
Noeuds à l'intérieur d'un schéma



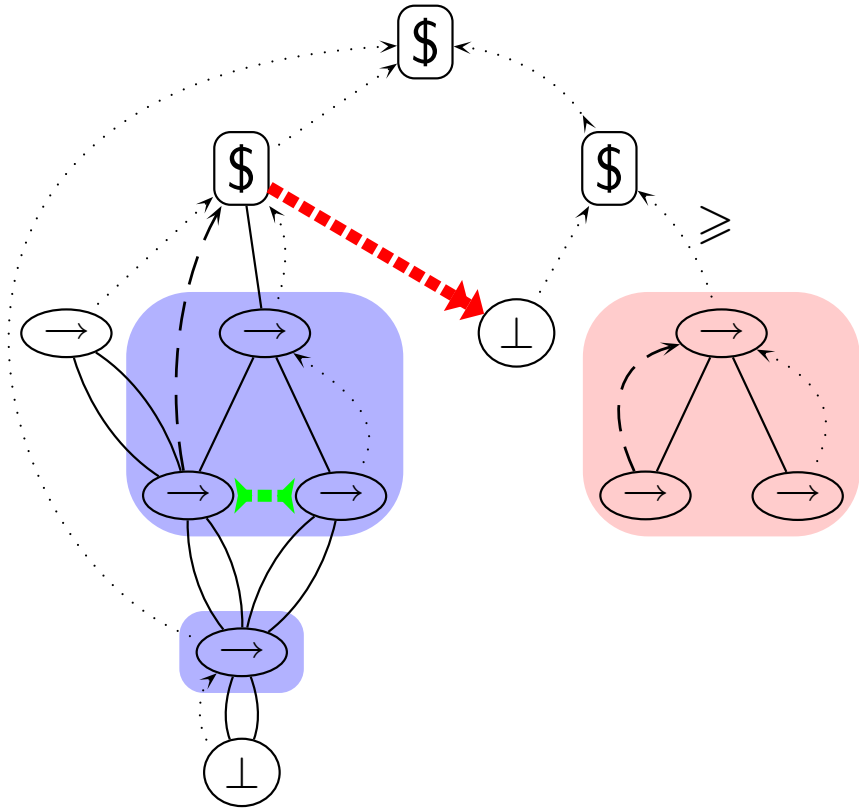
Frontière

Noeuds dans l'extérieur ayant un ancêtre immédiat dans l'intérieur.

Prendre une instance d'un schéma : l'expansion

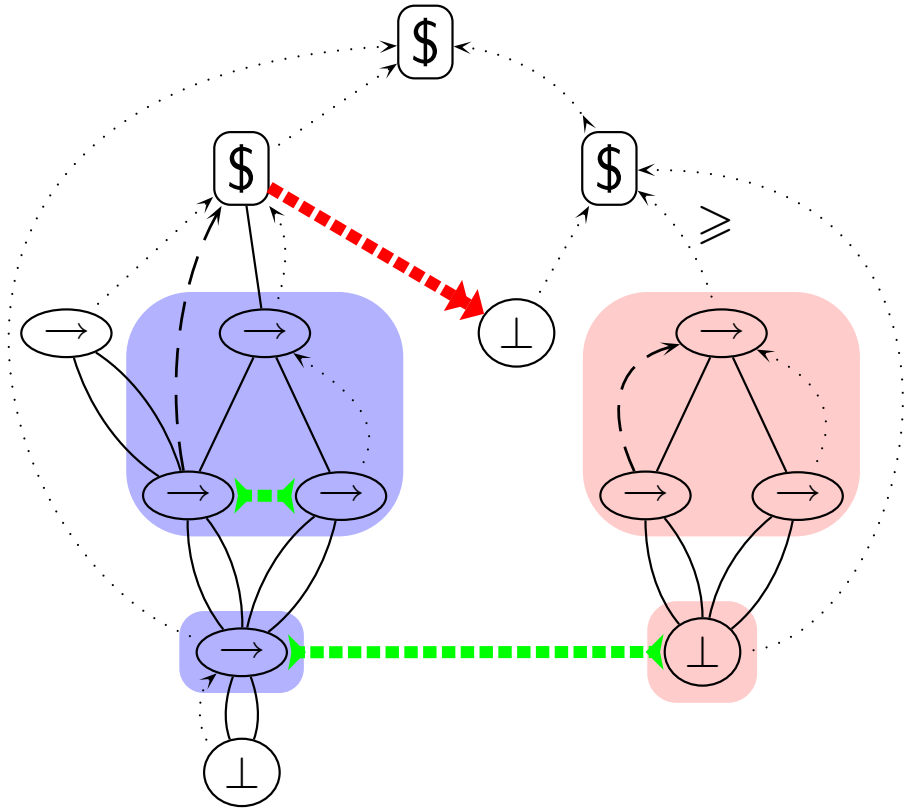


Prendre une instance d'un schéma : l'expansion



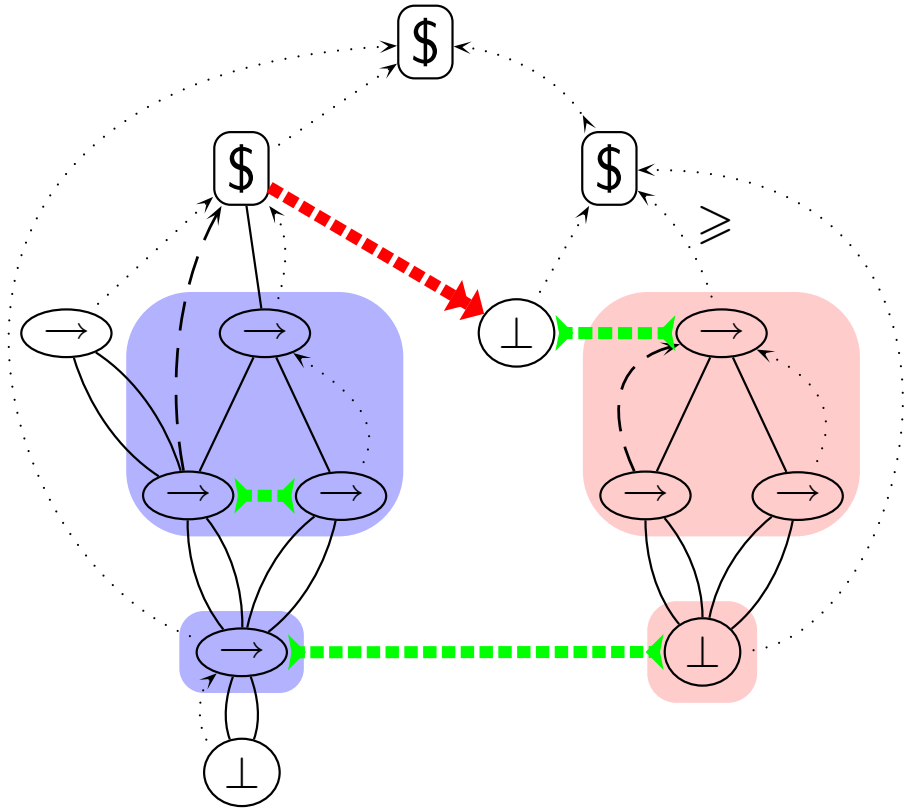
- Copier les noeuds de l'intérieur structurel (sauf le noeud schéma lui-même). La copie est liée flexiblement

Prendre une instance d'un schéma : l'expansion



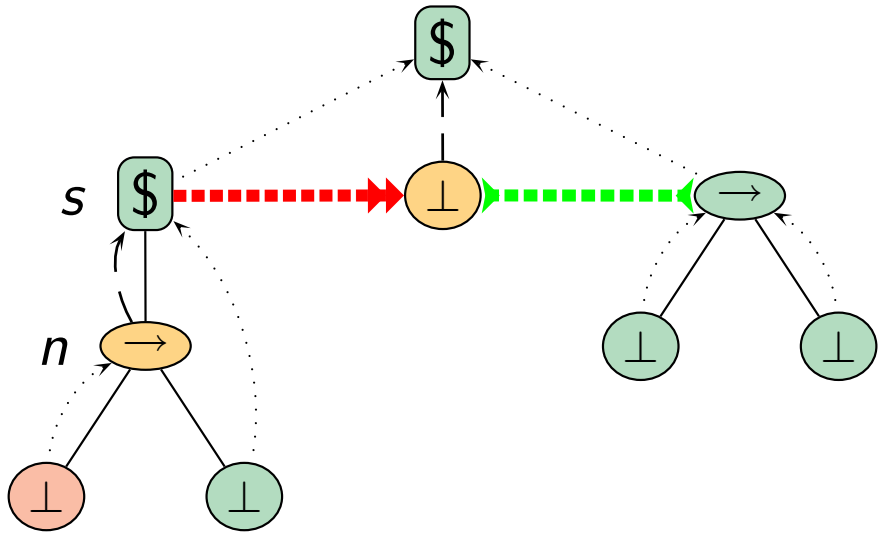
- Copier les noeuds de l'intérieur structurel (sauf le noeud schéma lui-même). La copie est liée flexiblement
- Transformer les noeuds de la frontière structurelle en variables devant être partagées avec le schéma d'origine

Prendre une instance d'un schéma : l'expansion



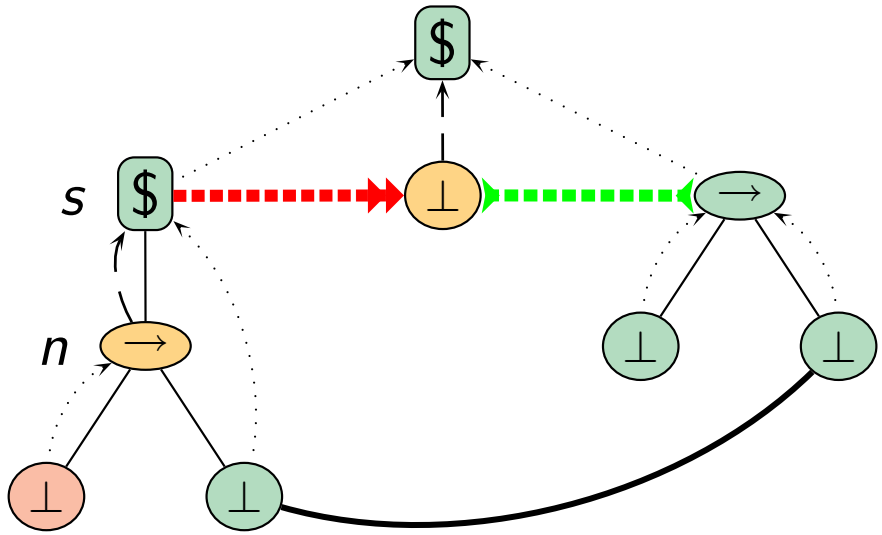
- Copier les noeuds de l'intérieur structurel (sauf le noeud schéma lui-même). La copie est liée flexiblement
- Transformer les noeuds de la frontière structurelle en variables devant être partagées avec le schéma d'origine
- Ajouter un arc d'unification entre la copie et le noeud contraint

La généralisation en ML^F



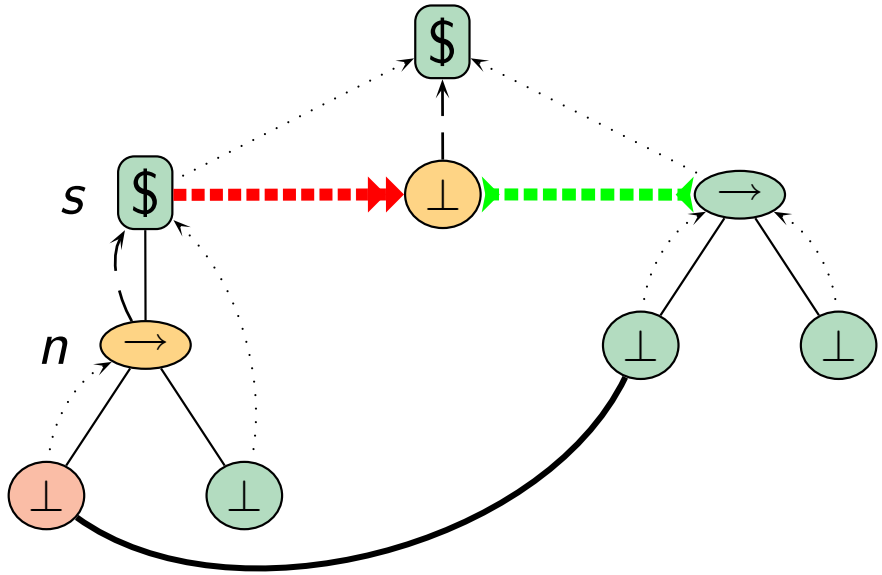
- Le symbole \$ symbolise (plus ou moins) le $\vdash$ d'un jugement de typage.
  - Les noeuds liés sur s sont à gauche du $\vdash$
  - Les noeuds liés sur $n = \langle s \cdot 1 \rangle$ sont à droite du $\vdash$ (quantifiés dans le type)

La généralisation en ML^F



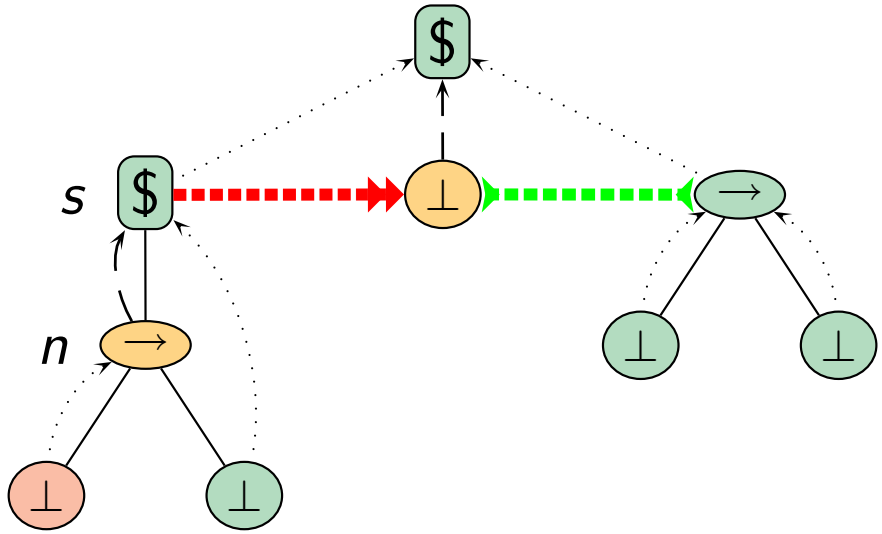
- Le symbole \$ symbolise (plus ou moins) le $\vdash$ d'un jugement de typage.
  - Les noeuds liés sur s sont à gauche du $\vdash$
  - Les noeuds liés sur $n = \langle s \cdot 1 \rangle$ sont à droite du $\vdash$ (quantifiés dans le type)
- La généralisation fait passer une variable de gauche à droite

La généralisation en ML^F



- Le symbole \$ symbolise (plus ou moins) le $\vdash$ d'un jugement de typage.
  - Les noeuds liés sur s sont à gauche du $\vdash$
  - Les noeuds liés sur $n = \langle s \cdot 1 \rangle$ sont à droite du $\vdash$ (quantifiés dans le type)
- La généralisation fait passer une variable de gauche à droite
- Et "libère" le polymorphisme

La généralisation en ML^F



- Le symbole \$ symbolise (plus ou moins) le $\vdash$ d'un jugement de typage.
  - Les noeuds liés sur s sont à gauche du $\vdash$
  - Les noeuds liés sur $n = \langle s \cdot 1 \rangle$ sont à droite du $\vdash$ (quantifiés dans le type)
- La généralisation fait passer une variable de gauche à droite
- Et "libère" le polymorphisme
- On ne peut pas généraliser tant qu'il reste des contraintes sous s

Sémantique des arcs

Arc d'unification résolu

Un arc n_1 ⟼ n_2 est résolu si n_1 et n_2 sont fusionnés.

Arc d'instance résolu

Un arc $e = s \dashrightarrow d$ d'une contrainte χ est résolu si l'expansion de e dans χ peut être réinstanciée en χ.

(*i.e.* χ vérifie déjà les contraintes imposées par e)

Sémantique des contraintes

Projection

Type obtenu à partir d'une contrainte en ne suivant pas les arcs existentiels et en enlevant les arcs de contraintes.

Présolutions et solutions

Une présolution d'une contrainte χ est une instance de χ dans laquelle tous les arcs de contraintes sont résolus.
Une solution de χ est la projection d'une présolution de χ.

Équivalence entre contraintes

Deux contraintes sont équivalentes si elles ont les mêmes solutions.

Sémantique des contraintes

Projection

Type obtenu à partir d'une contrainte en ne suivant pas les arcs existentiels et en enlevant les arcs de contraintes.

Présolutions et solutions

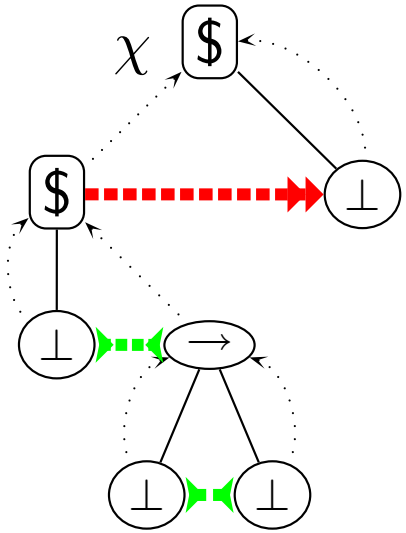
Une présolution d'une contrainte χ est une instance de χ dans laquelle tous les arcs de contraintes sont résolus.
Une solution de χ est la projection d'une présolution de χ.

Équivalence entre contraintes

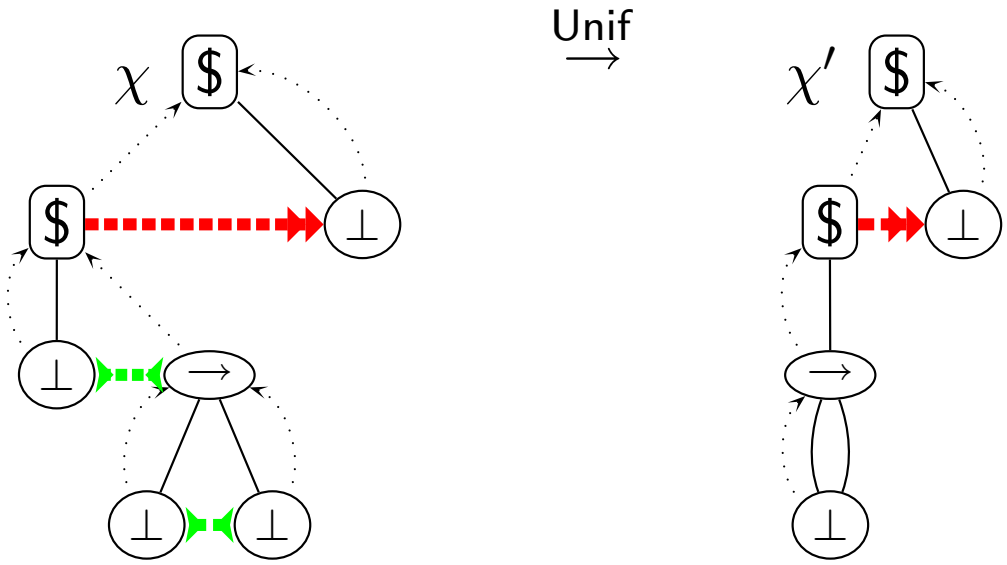
Deux contraintes sont équivalentes si elles ont les mêmes solutions.

- Permet de réutiliser tous les résultats sur la relation d'instance.
- Si $\chi \sqsubseteq \chi'$, toute solution de χ' est solution de χ.
- Deux contraintes avec deux structures de noeuds \$ différentes peuvent être équivalentes, mais n'auront pas les mêmes présolutions.

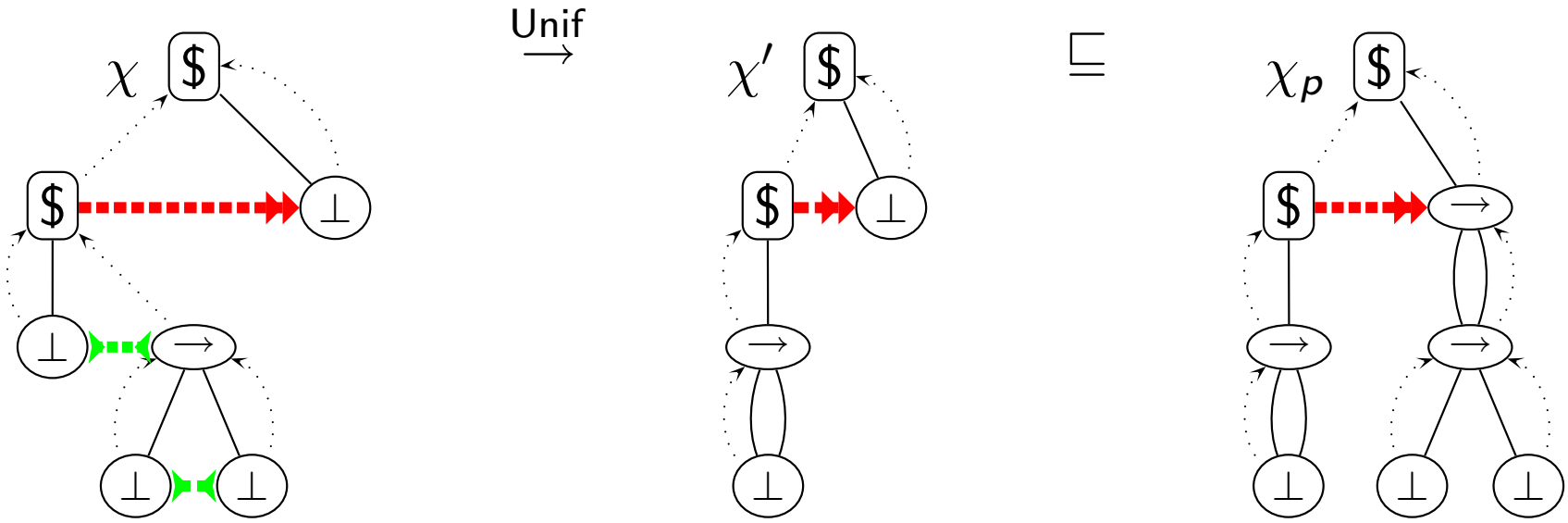
Un exemple



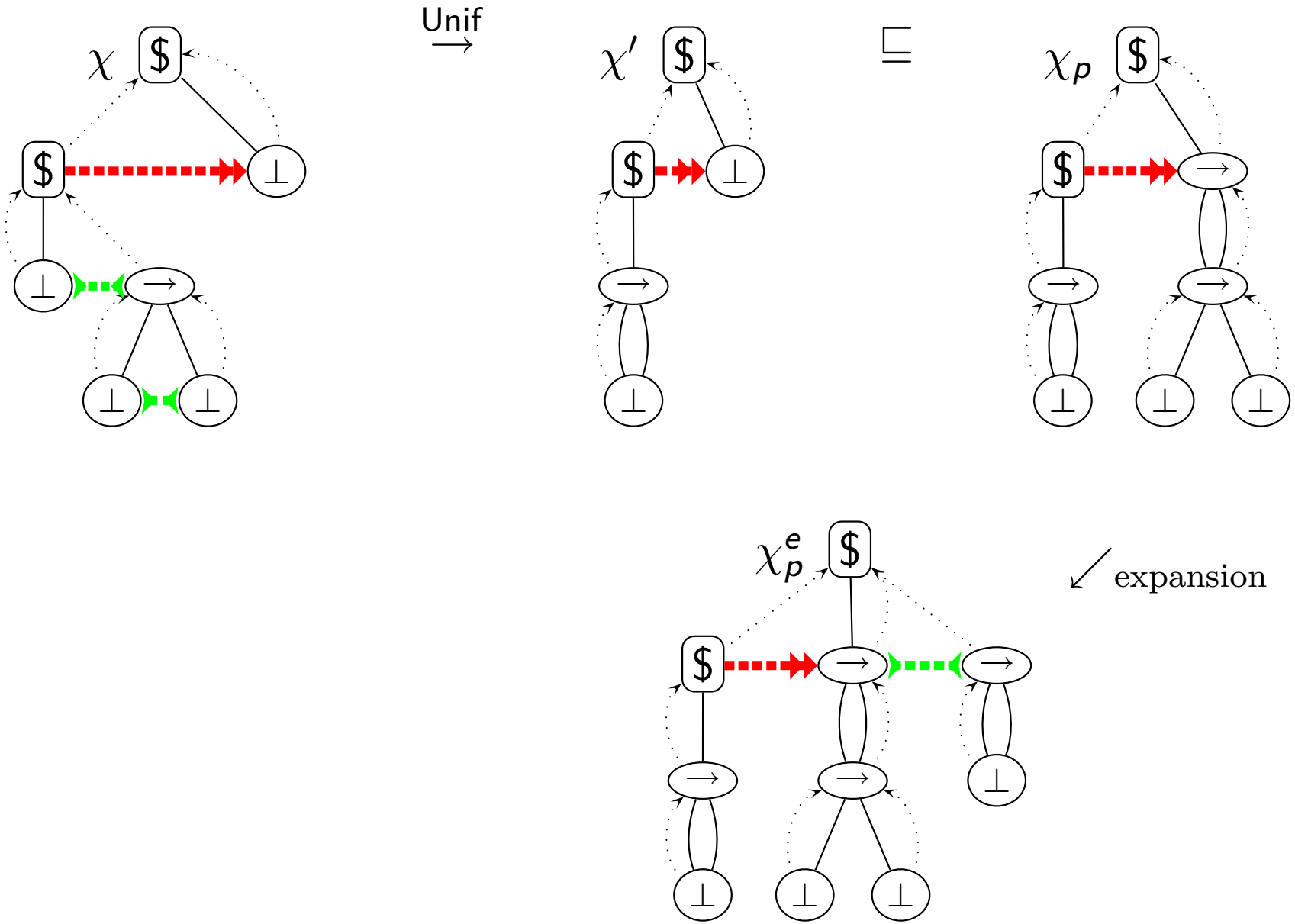
Un exemple



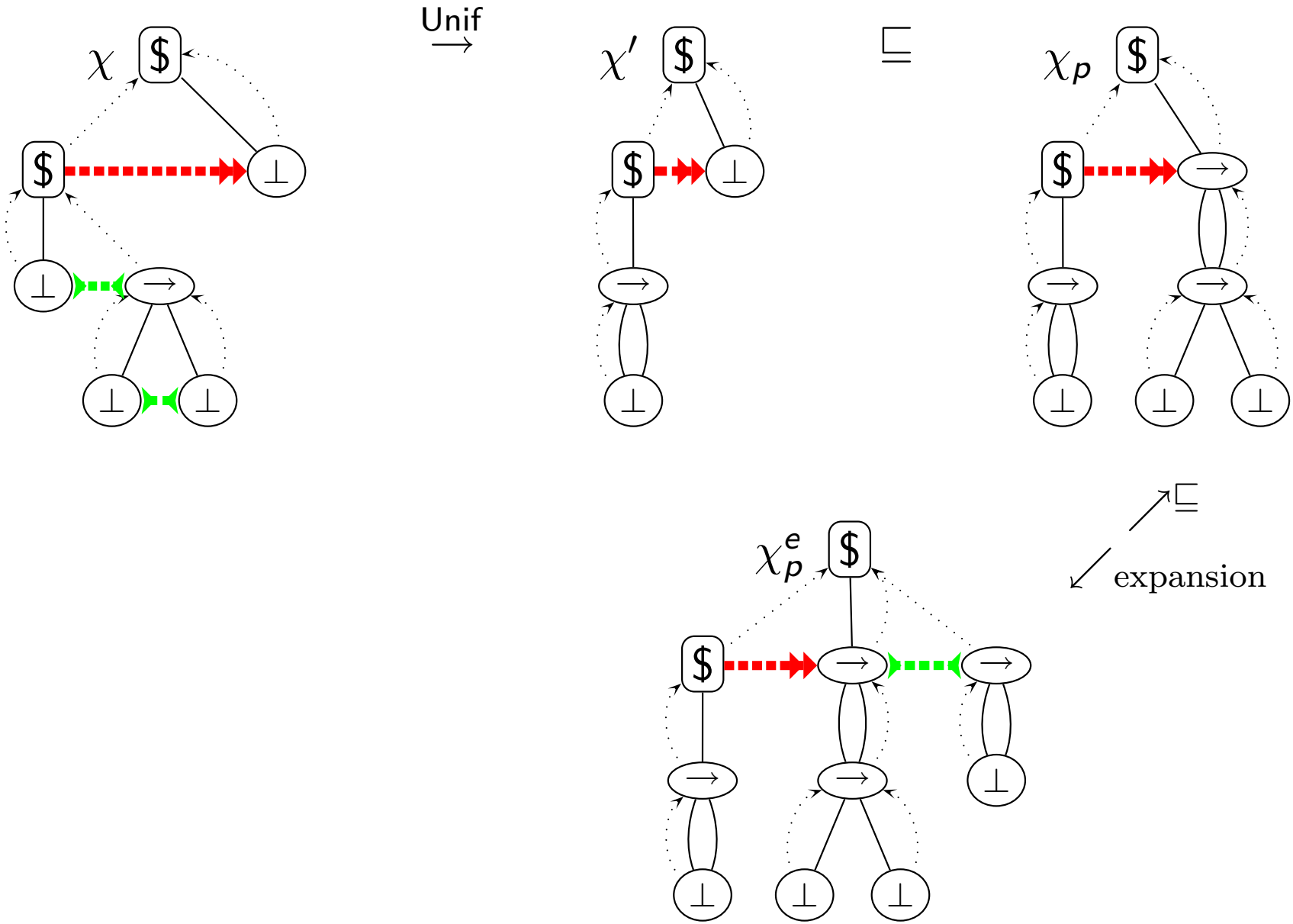
Un exemple



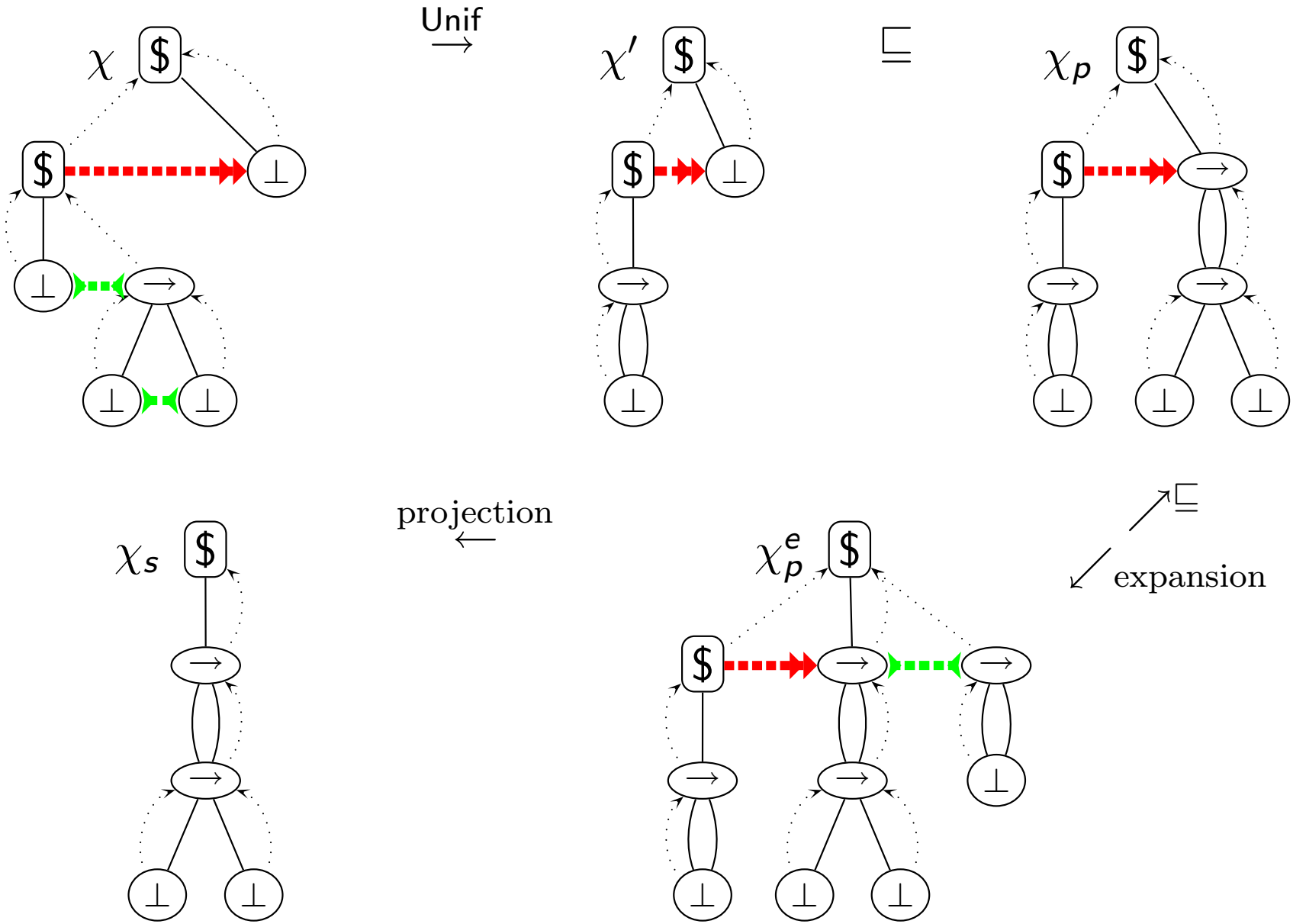
Un exemple


χ
$\xrightarrow{\text{Unif}}$
χ'
$\sqsubseteq$
χ_p
χ_p^e
expansion

Un exemple


χ
$\xrightarrow{\text{Unif}}$
χ'
$\sqsubseteq$
χ_p
χ_p^e
$\sqsubseteq$
expansion

Un exemple


χ
Unif
χ'
$\sqsubseteq$
χ_p
projection
χ_s
χ_p^e
$\sqsubseteq$
expansion

Élimination existentielle

- Les noeuds existentiels servent à introduire des contraintes
- Une fois ces contraintes résolues, on peut les éliminer

Élimination existentielle

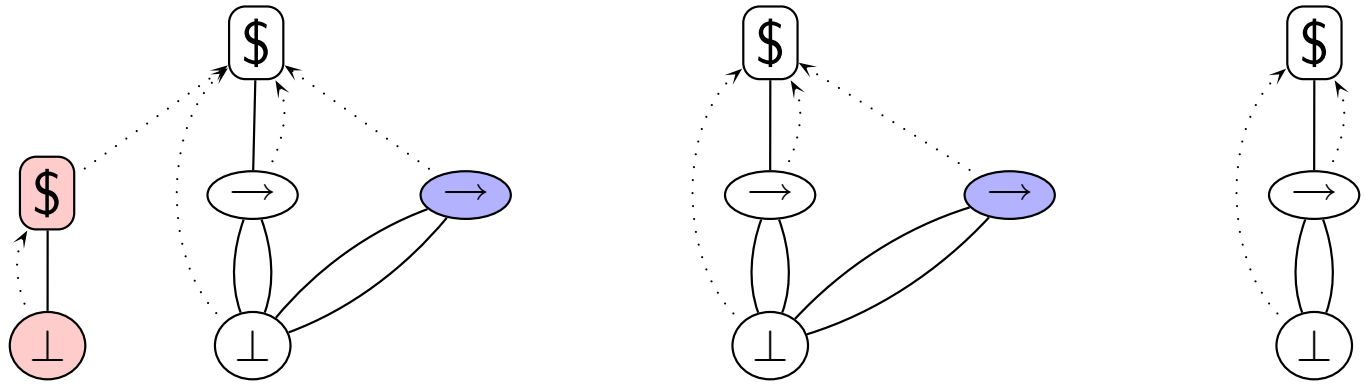
EXIST-ELIM : supprimer un noeud et son intérieur, à condition qu'il n'y ait pas d'arcs de contraintes ciblant l'intérieur.

Élimination existentielle

- Les noeuds existentiels servent à introduire des contraintes
- Une fois ces contraintes résolues, on peut les éliminer

Élimination existentielle

EXIST-ELIM : supprimer un noeud et son intérieur, à condition qu'il n'y ait pas d'arcs de contraintes ciblant l'intérieur.

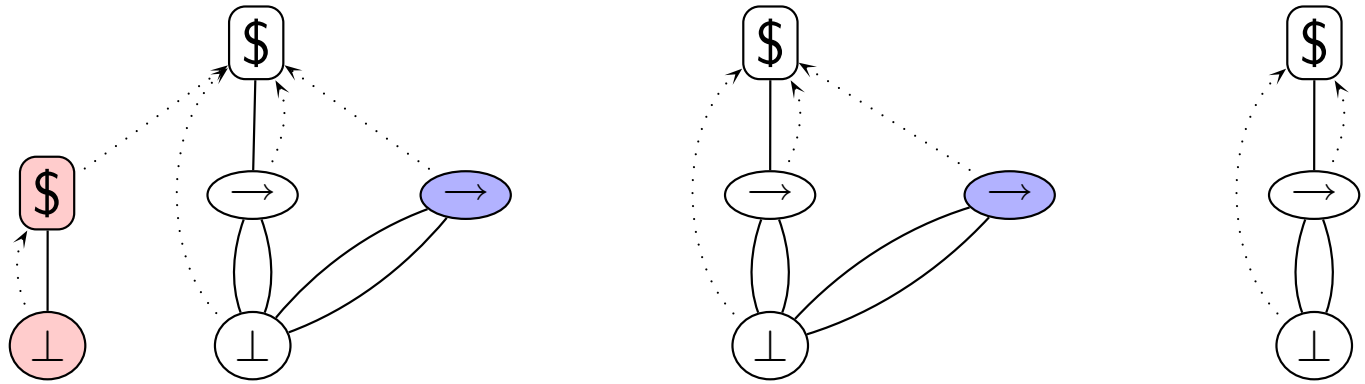


Élimination existentielle

- Les noeuds existentiels servent à introduire des contraintes
- Une fois ces contraintes résolues, on peut les éliminer

Élimination existentielle

EXIST-ELIM : supprimer un noeud et son intérieur, à condition qu'il n'y ait pas d'arcs de contraintes ciblant l'intérieur.



Préserve les solutions

(En l'absence d'existentiels ailleurs que sous un noeud schéma, et parce que les noeuds $ ne peuvent pas être extrudés)

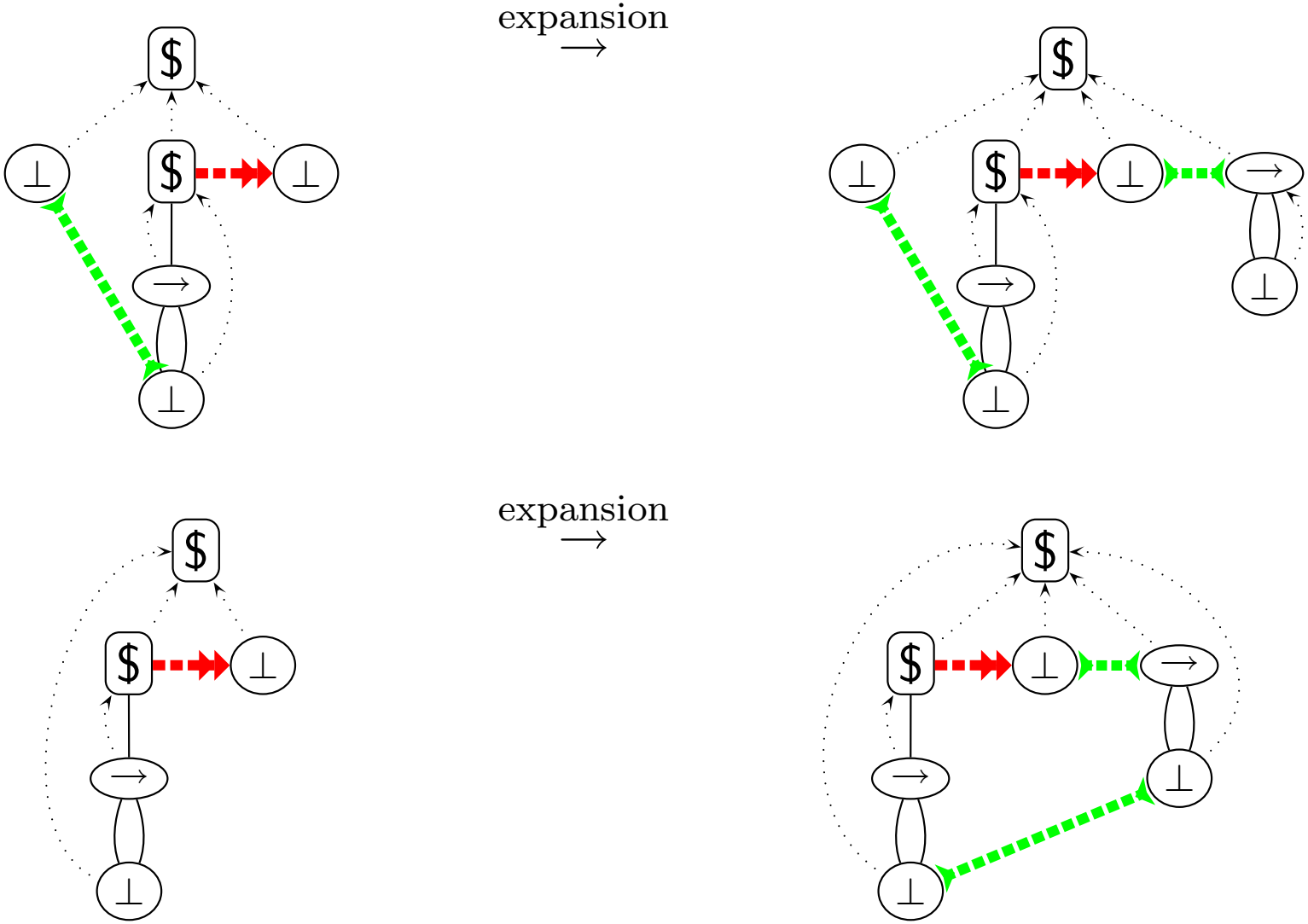
Unification

- L'algorithme d'unification reste totalement inchangé
  en l'absence d'existentiels ailleurs que sous un noeud schéma
- L'unification sur les types graphiques est principale
- Toujours principale en présence de noeuds existentiels

Résoudre un arc d'unification préserve les *pré*solutions
(et donc les solutions)

Expansion

On peut instancier le résultat d'une expansion de façon synchrone avec le schéma



Expansion

On peut instancier le résultat d'une expansion de façon synchrone avec le schéma

Expanser un arc d'instantiation préserve les présolutions.

Schémas dégénérés

- Un schéma dont l'intérieur est vide n'est plus généralisable
- Un arc d'instance partant d'un tel schéma est dégénéré

Expansion monomorphe

INST-ELIM-MONO est une équivalence.

Arcs d'instance résolus

Hypothèses :

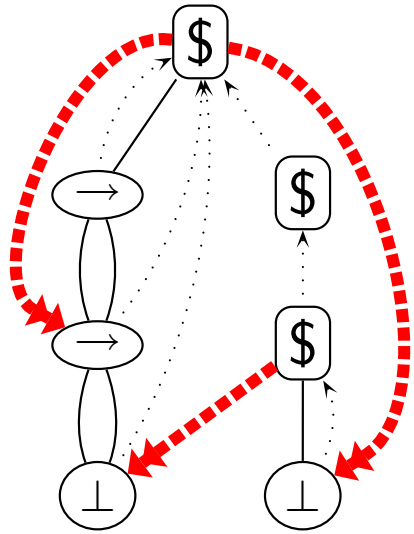
- Un arc d'instance $s \dashrightarrow d$ résolu
- Une instance χ' de χ and laquelle l'intérieur de s est inchangé

L'arc est encore résolu dans χ'

- Résultat intuitivement simple :

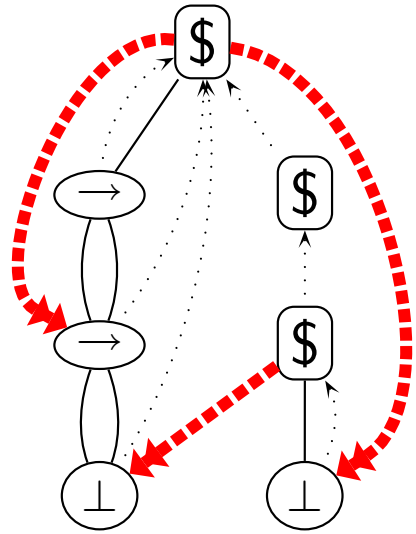
τ' est une instance de τ (*i.e.* le schéma), et toute instance de τ' (*i.e.* χ') est également une instance de τ.

Sémantique des contraintes et (in)décidabilité



- Les contraintes permettent d'encoder des problèmes avec récursion polymorphe
- La sémantique est probablement indécidable dans le cas général

Sémantique des contraintes et (in)décidabilité



- Les contraintes permettent d'encoder des problèmes avec récursion polymorphe
- La sémantique est probablement indécidable dans le cas général

Contraintes acycliques

La relation "dépend de" est un ordre (où s dépend de s' s'il existe un arc $s' \dashrightarrow n$, avec n dans l'intérieur de s).

Un dernier résultat intermédiaire

Expansion-Unification

- Un arc d'instance e dans une contrainte ayclique χ
- χ^e la contrainte résultant de l'expansion de e dans χ
- χ' l'unificateur principal des arcs d'unification créés lors de l'expansion

- χ' et χ ont les mêmes présolutions
(vrai même si χ n'est pas acyclique)

Un dernier résultat intermédiaire

Expansion-Unification

- Un arc d'instance e dans une contrainte ayclique χ
- χ^e la contrainte résultant de l'expansion de e dans χ
- χ' l'unificateur principal des arcs d'unification créés lors de l'expansion

- χ' et χ ont les mêmes présolutions
(vrai même si χ n'est pas acyclique)
- e est résolu dans χ'

Présolutions principales

Calcul des présolutions

1. Choisir un arc d'instance non résolu le plus externe possible (au sens de la relation de dépendance)
2. Expanser cet arc
3. Unifier les nouveaux arcs d'unification
4. Itérer jusqu'à ce que tous les arcs soient résolus

Présolutions principales

Calcul des présolutions

1. Choisir un arc d'instance non résolu le plus externe possible (au sens de la relation de dépendance)
2. Expanser cet arc
3. Unifier les nouveaux arcs d'unification
4. Itérer jusqu'à ce que tous les arcs soient résolus

Calcule une présolution principale

(et donc une solution principale)

Remarque : quelle que soit la stratégie utilisée pour choisir l'arc à expanser, on obtient la même présolution

Enlever les flêches d'instance

Le théorème de principalité montre que, si un schéma est résolu, il ne sera pas instancié dans la présolution principale.

Inst-Expand

- Une contrainte acyclique χ ;
- un schéma s dont l'intérieur n'est contraint par aucun arc d'instance ou d'unification ;
- un arc $e = s \dashrightarrow n$

La contrainte $\chi^e \setminus e$ et χ ont les mêmes solutions.

Intuitivement, étant donné que le schéma ne sera plus instancié, il suffit d'imposer ses contraintes une unique fois.

Solutions principales

- Une présolution correspond à la contrainte d'origine intégralement décorée ; le type de tous les sous-termes est connu
- Peuvent être beaucoup plus grosses que les solutions elles-mêmes

Solutions principales

- Une présolution correspond à la contrainte d'origine intégralement décorée ; le type de tous les sous-termes est connu
- Peuvent être beaucoup plus grosses que les solutions elles-mêmes

Calcul optimisé des solutions

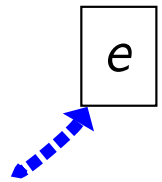
1. Choisir un arc d'instance non résolu le plus externe possible (au sens de la relation de dépendance)
2. Utiliser Inst-Expand sur cet arc
3. Unifier les nouveaux arcs d'unification
4. Appliquer Exist-Elim sur les parties existentielles non contraintes
5. Itérer sur les arcs restants

Calcule la solution principale.

Traduction compositionnelle

$$e \ ::= \ x \ \mid \ \lambda(x)\ e \ \mid \ e\ e \ \mid \ \text{let}\ x = e\ \text{in}\ e$$

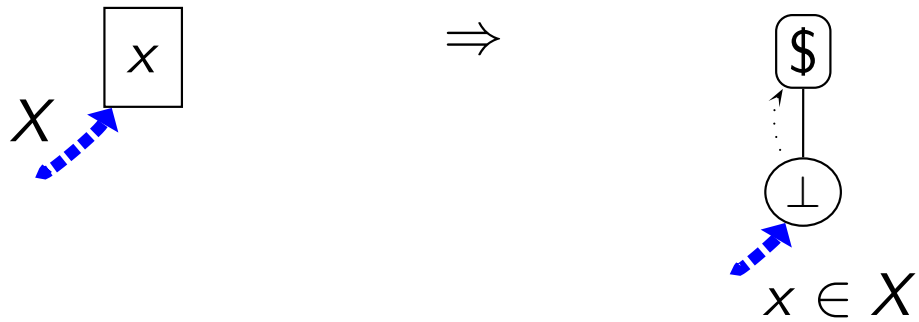
À chaque expression on associe une contrainte “boîte”, représentant son type dans le contexte



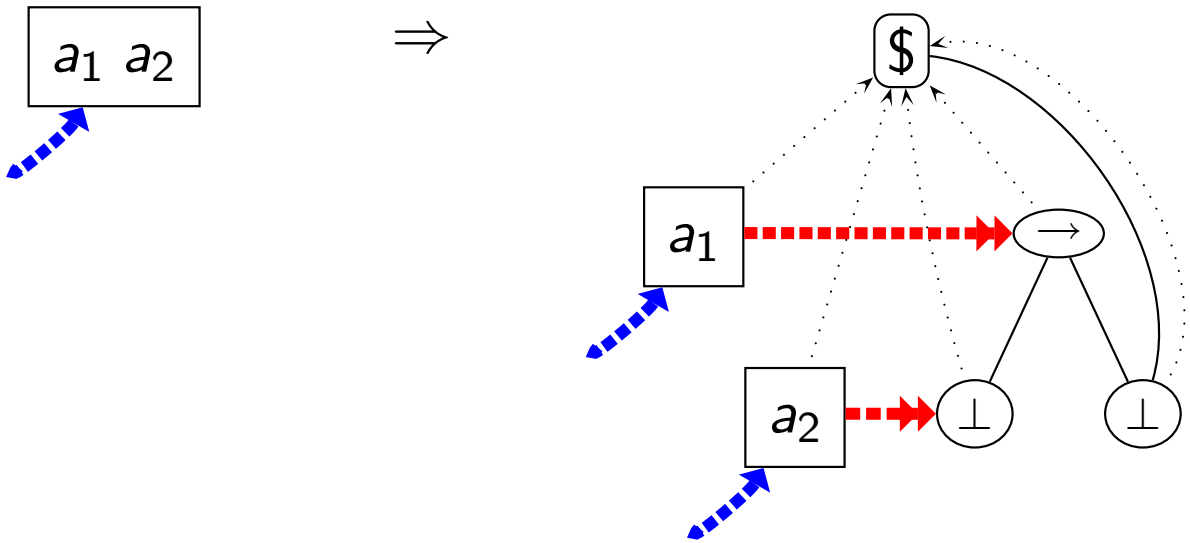
Les flèches bleues sont une méta-notation pour une fonction partielle des variables vers un arc ou représentant la contrainte ciblant cette variable

(unification pour une variable introduite par une abstraction, instance pour une variable introduite par un let)

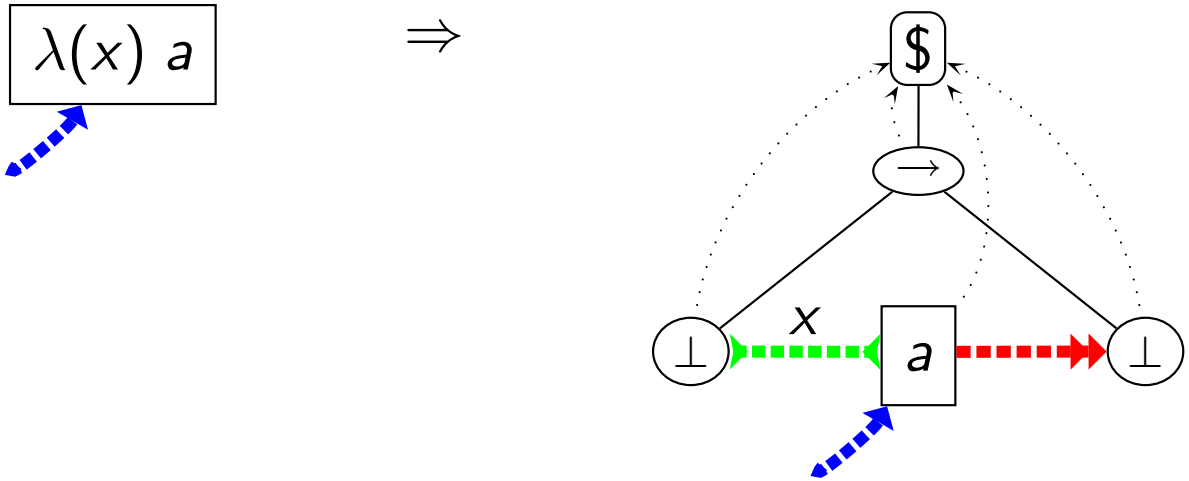
Contraintes pour les variables



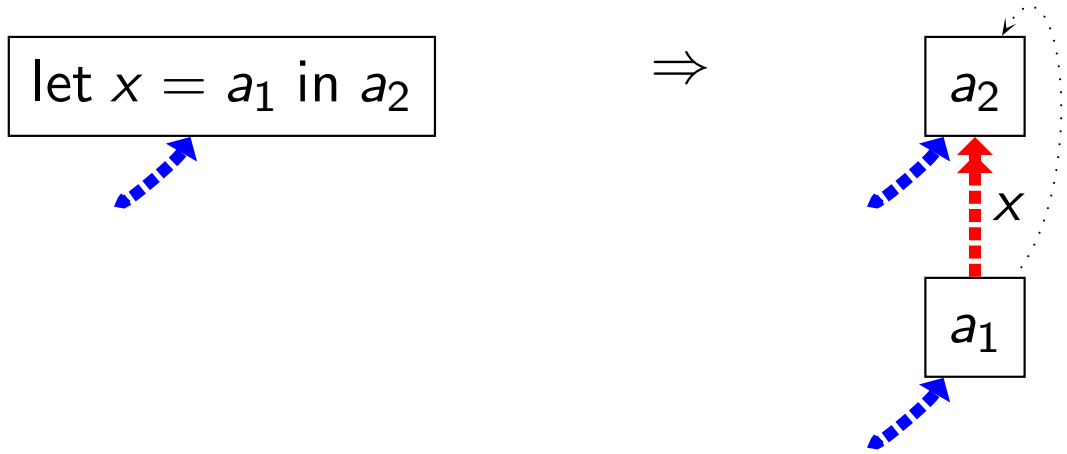
Contraintes pour l'application



Contraintes pour l'abstraction



Contraintes pour les let



Remarques

- Les contraintes sont acycliques : les arcs d'instance suivent la portée des variables

Remarques

- Les contraintes sont acycliques : les arcs d'instance suivent la portée des variables
- Un schéma par noeud de l'arbre de syntaxe

  Il est toutefois possible de simplifier le cas des variables

Remarques

- Les contraintes sont acycliques : les arcs d'instance suivent la portée des variables
- Un schéma par noeud de l'arbre de syntaxe

  Il est toutefois possible de simplifier le cas des variables
- La traduction introduit uniquement des flèches flexibles

  ML^F sans annotations n'est pas plus expressif que ML

Remarques

- Les contraintes sont acycliques : les arcs d'instance suivent la portée des variables
- Un schéma par noeud de l'arbre de syntaxe

  Il est toutefois possible de simplifier le cas des variables
- La traduction introduit uniquement des flèches flexibles

  ML^F sans annotations n'est pas plus expressif que ML
- En introduisant un schéma uniquement pour les let, on obtient presque un système de contraintes pour ML.

Annotations de types

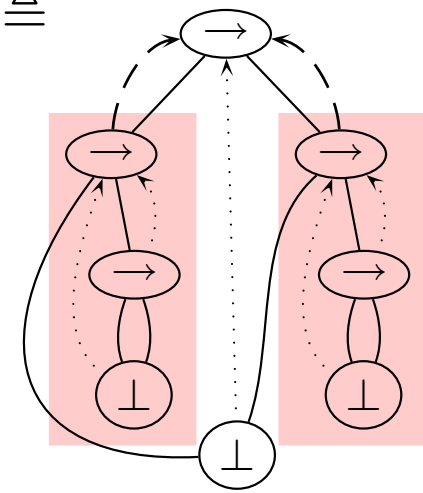
- Pas de constructions primitives, mais des constantes de retypage, sans contenu calculatoire, ajoutées à l'environnement de typage
  - $(e : \kappa) \quad \Rightarrow \quad c_\kappa\ e$
  - $\lambda(x : \kappa)\ e \quad \Rightarrow \quad \lambda(x)\ \text{let}\ x = c_\kappa\ x\ \text{in}\ e$

Annotations de types

- Pas de constructions primitives, mais des constantes de retypage, sans contenu calculatoire, ajoutées à l'environnement de typage
  - $(e : \kappa) \quad \Rightarrow \quad c_\kappa\ e$
  - $\lambda(x : \kappa)\ e \quad \Rightarrow \quad \lambda(x)\ \text{let}\ x = c_\kappa\ x\ \text{in}\ e$
- Exemple :

$$\kappa \triangleq \exists\beta\, \forall\,(\alpha)\ \beta \to (\alpha \to \alpha)$$

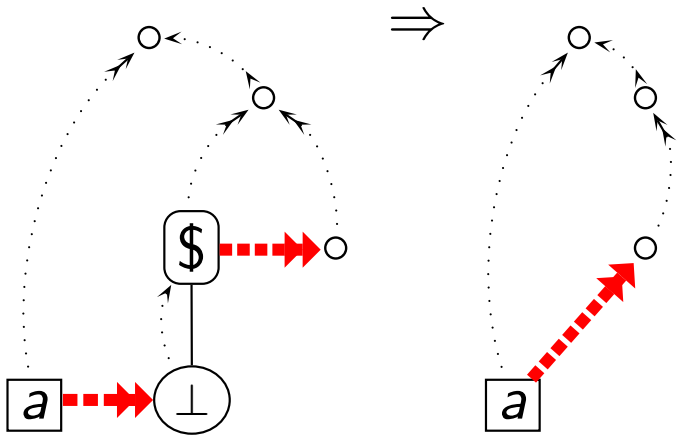
$c_\kappa \triangleq$



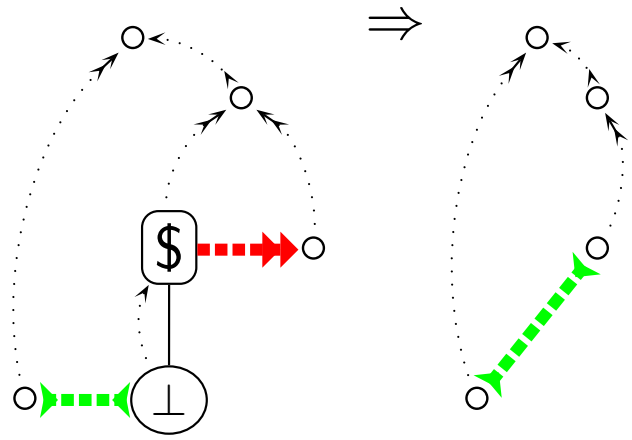
- La partie universellement quantifiée est dupliquée, et liée rigidement des deux cotés de la flèche
- La partie existentiellement quantifiée est partagée

Simplifier les contraintes pour les variables

VAR-LET



VAR-ABS



Préservent les solutions

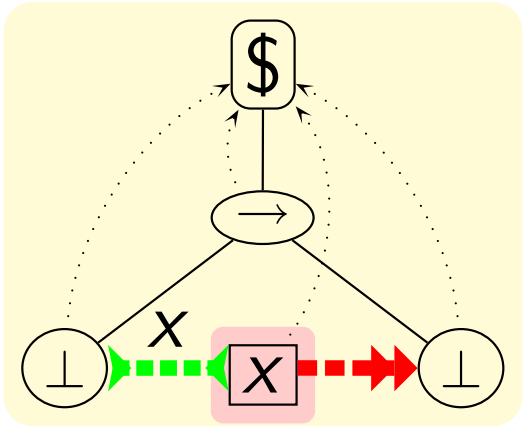
On peut utiliser ces deux règles au vol, lors de la traduction des expressions en contraintes.

Typage de $\lambda(x)\ x$

$$\boxed{\lambda(x)\ x}$$

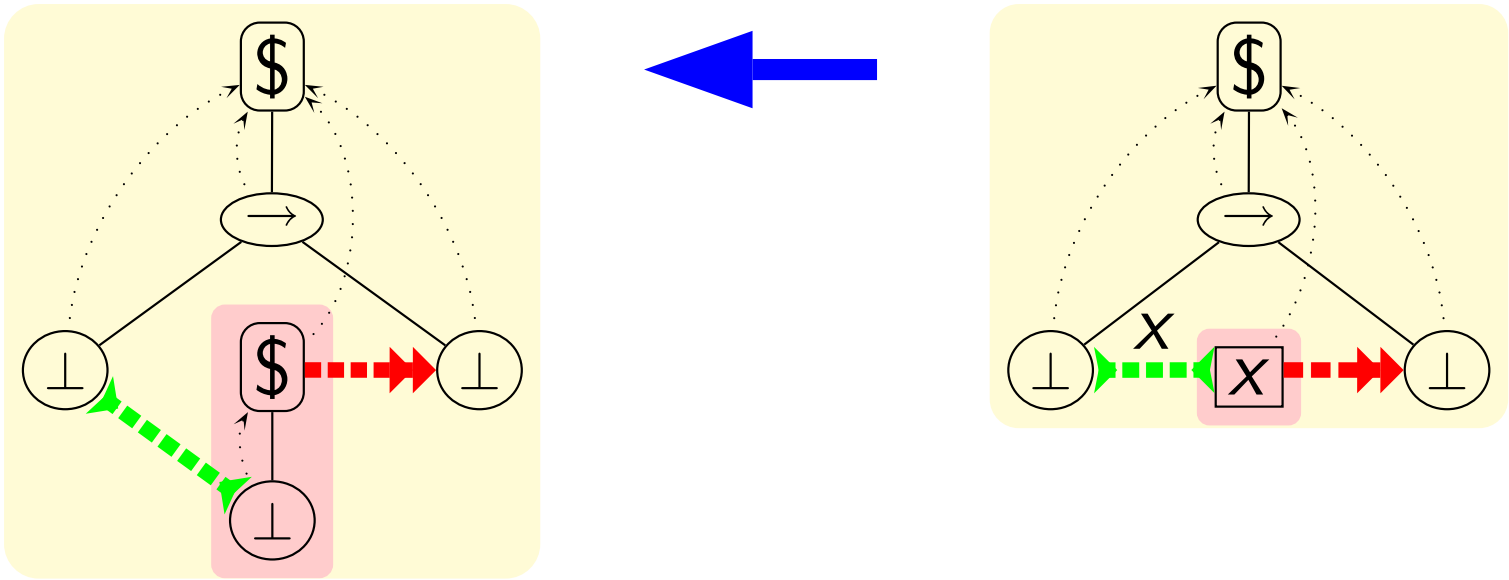
Typage de $\lambda(x)\ x$

$\boxed{\lambda(x)\ x}$



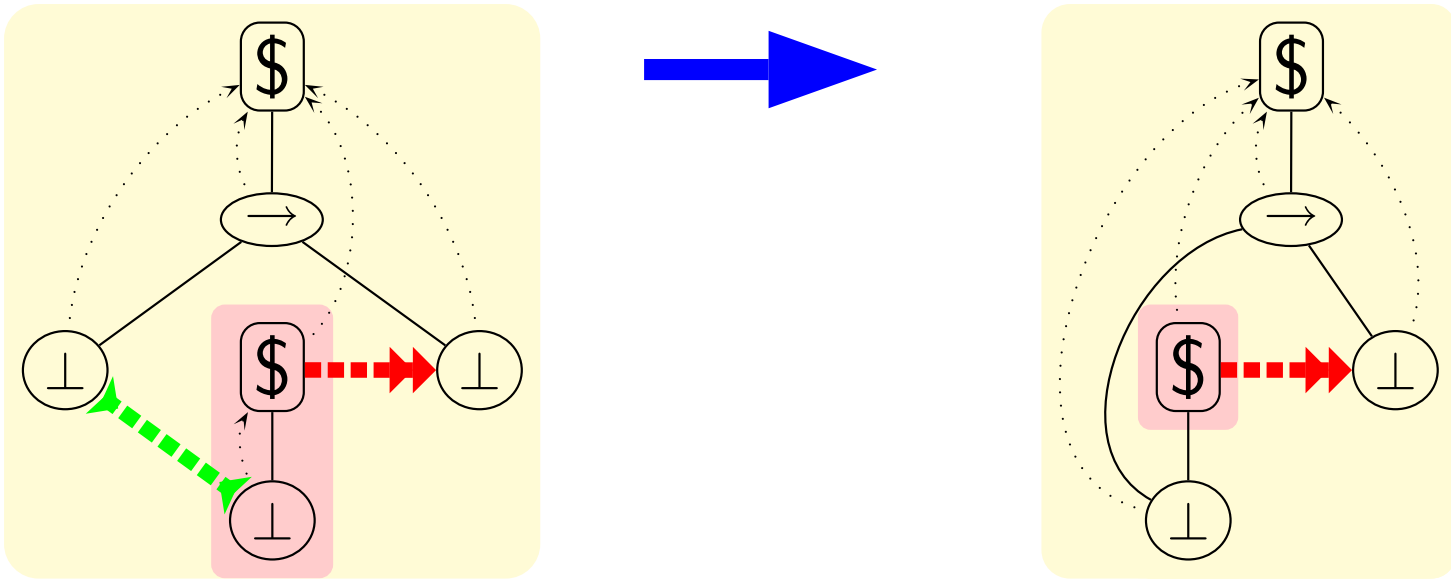
Par définition

Typage de $\lambda(x)\ x$



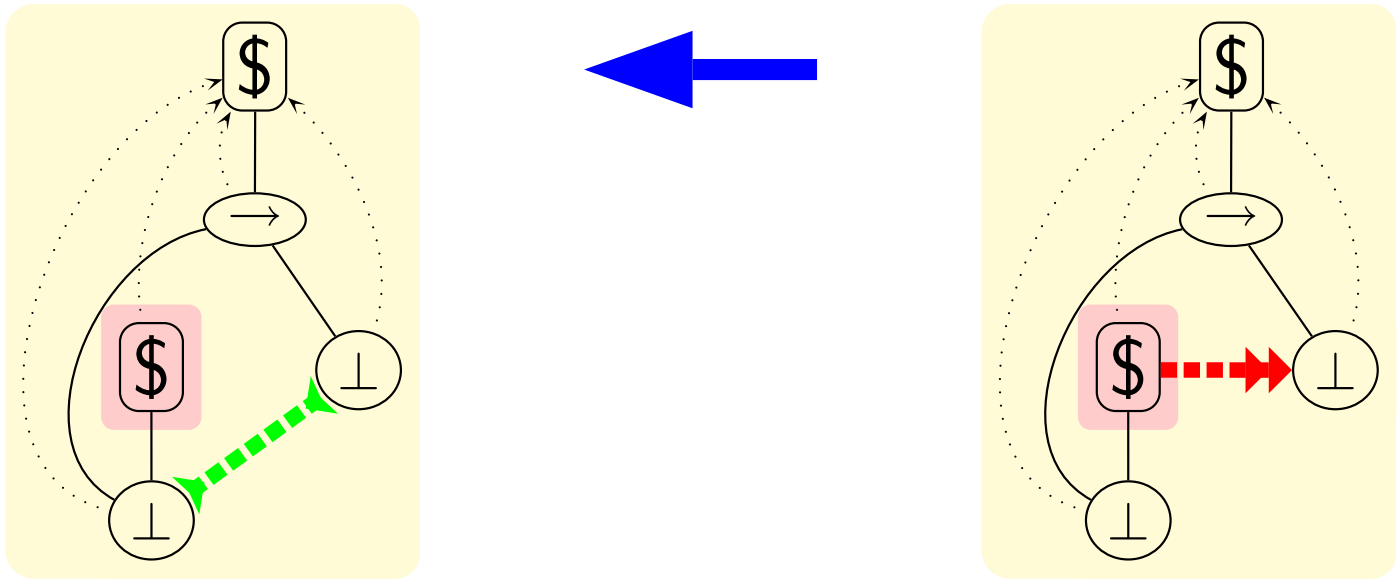
Par définition

Typage de $\lambda(x)\ x$



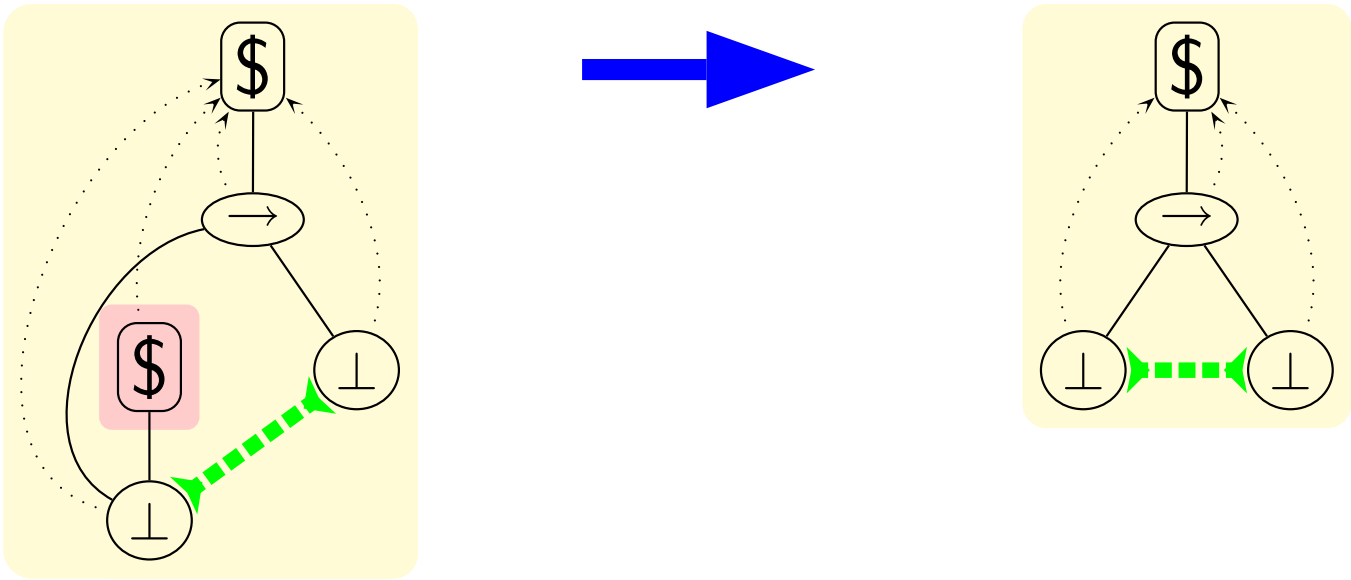
Unif

Typage de $\lambda(x)\ x$



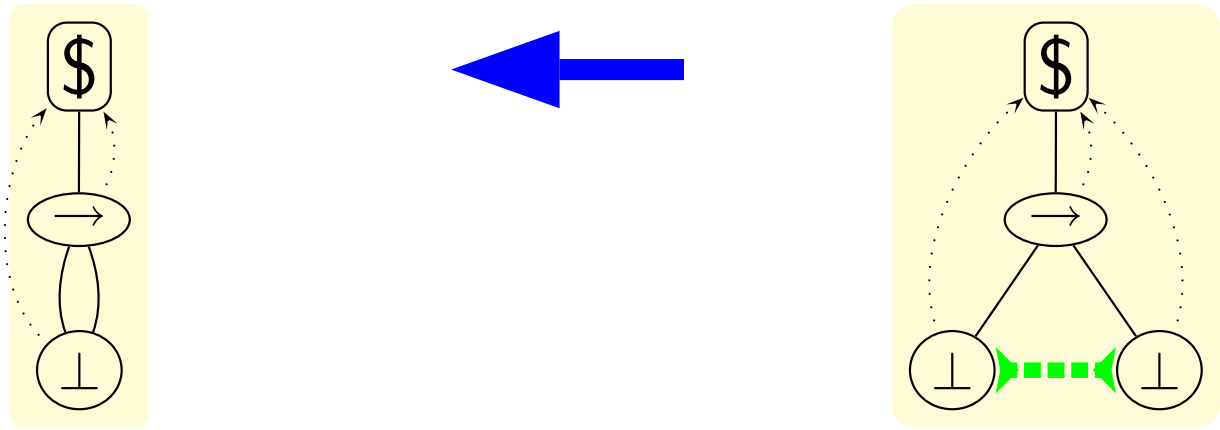
Inst-Elim-Mono

Typage de $\lambda(x)\ x$

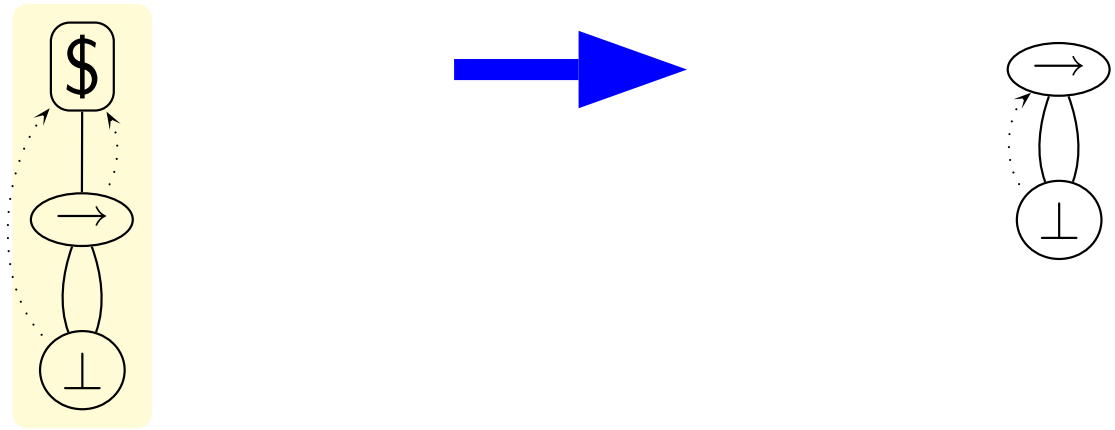


Exist-Elim

Typage de $\lambda(x)\ x$



Typage de $\lambda(x)\ x$

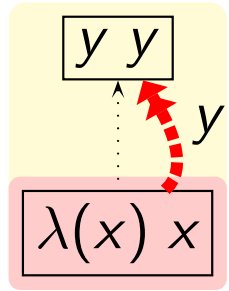


Typage de let $y = \lambda(x)\ x$ in $y\ y$

$$\boxed{\text{let}\ \ y = \lambda(x)\ x\ \text{in}\ y\ y}$$

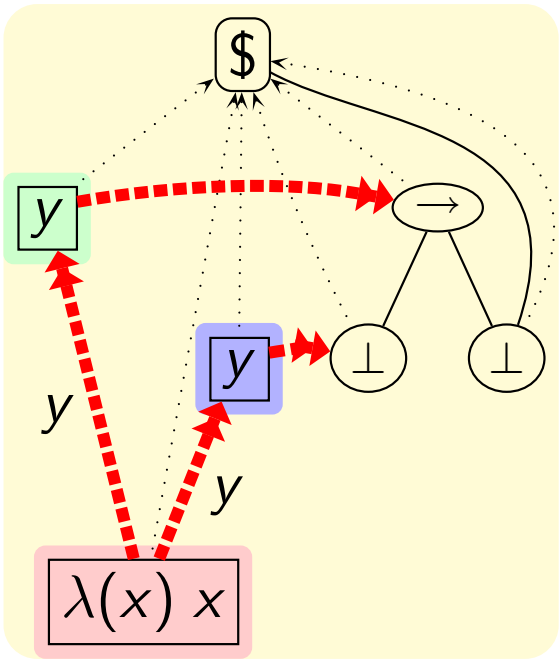
Typage de let $y = \lambda(x)\ x$ in $y\ y$

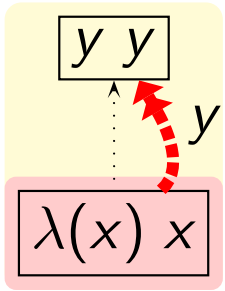
$$\boxed{\text{let}\ \ y = \lambda(x)\ x\ \text{in}\ y\ y}$$



Par définition

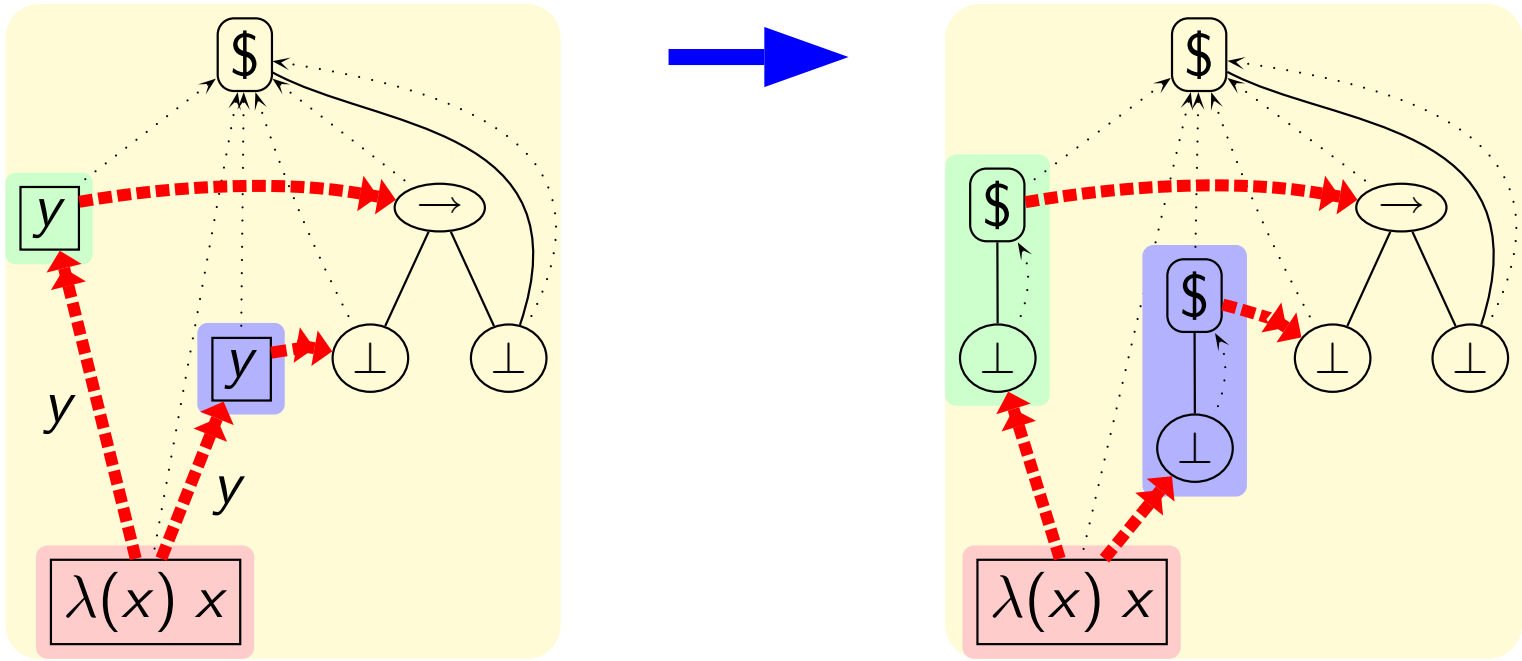
Typage de let $y = \lambda(x)\ x$ in $y\ y$





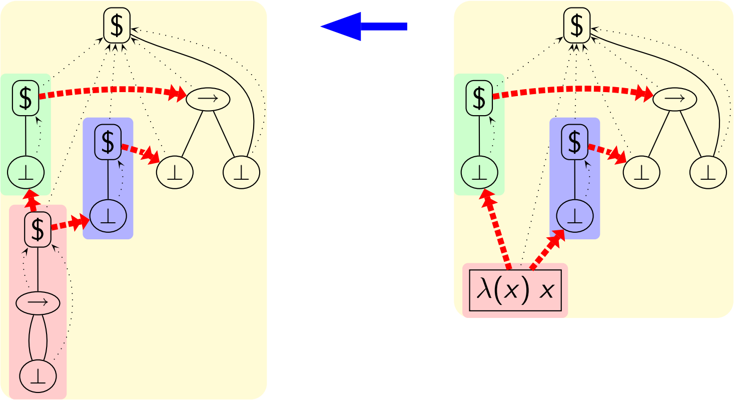
Par définition

Typage de let $y = \lambda(x)\ x$ in $y\ y$



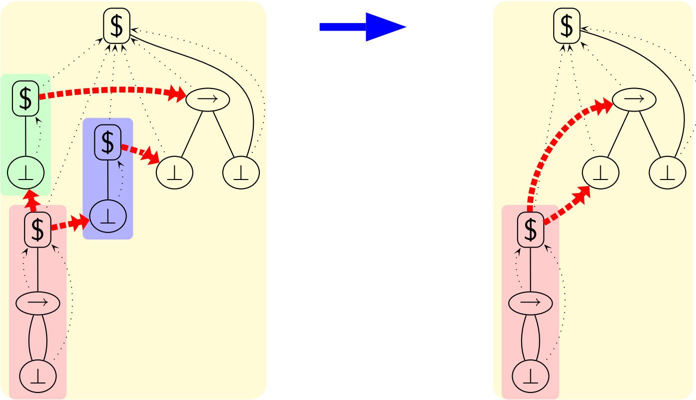
Par définition

Typage de let $y = \lambda(x)\ x$ in $y\ y$



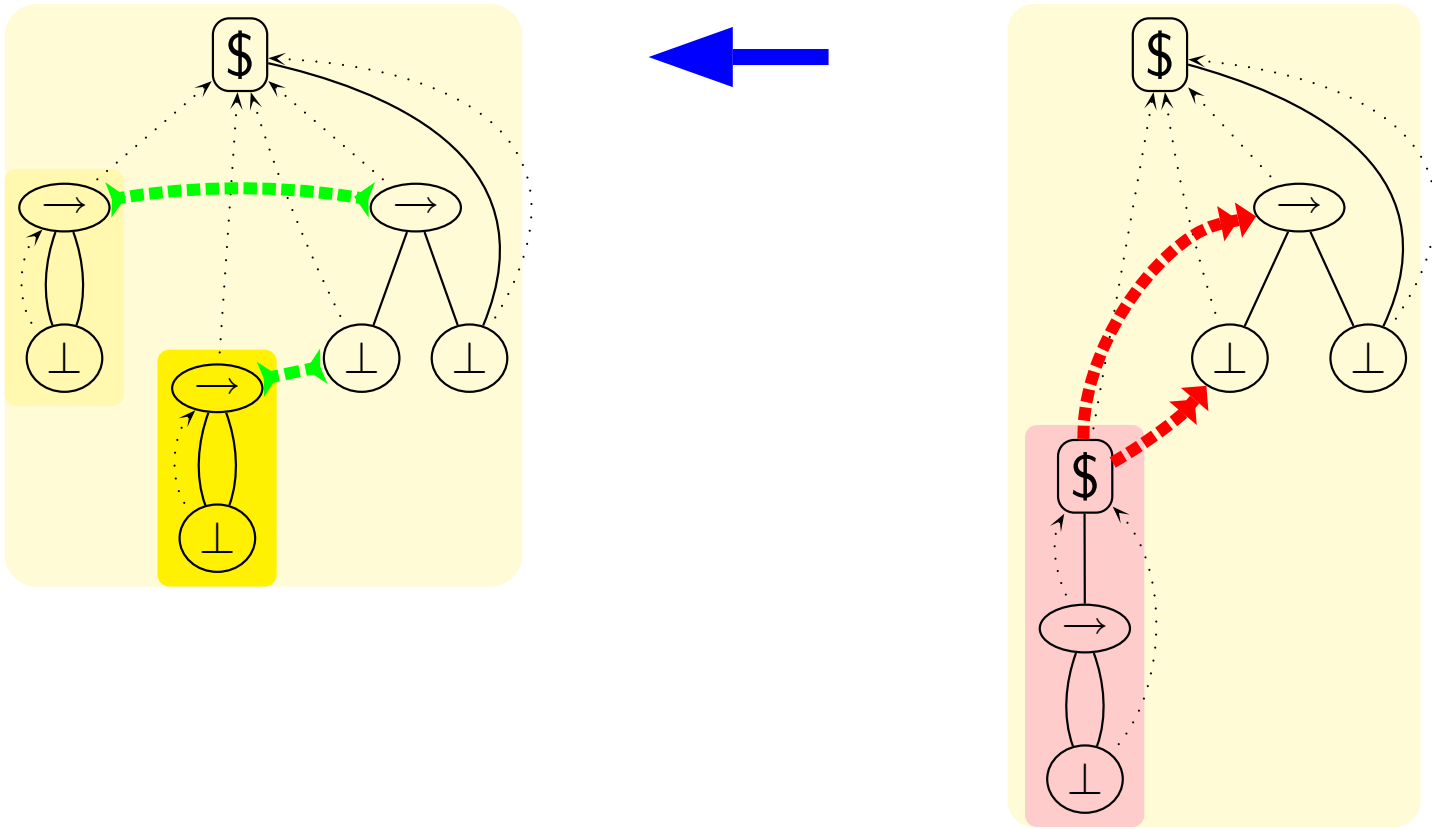
(exemple précédent)

Typage de let $y = \lambda(x)\ x$ in $y\ y$



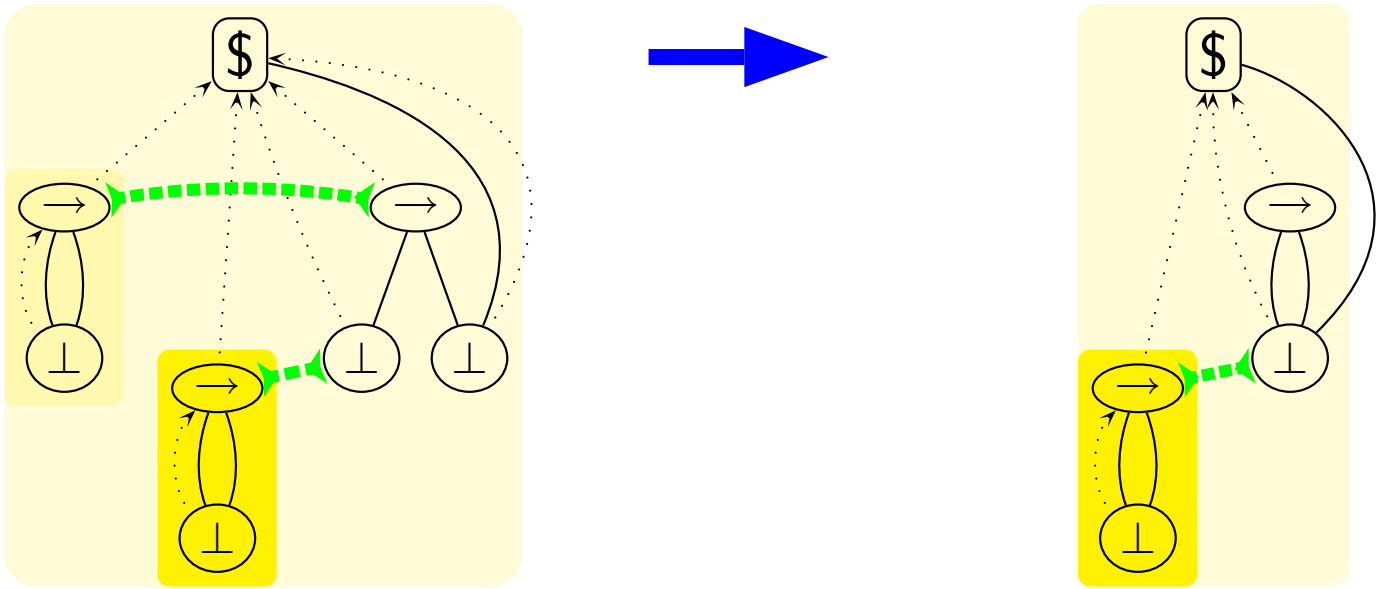
VAR-LET × 2

Typage de let $y = \lambda(x)\ x$ in $y\ y$



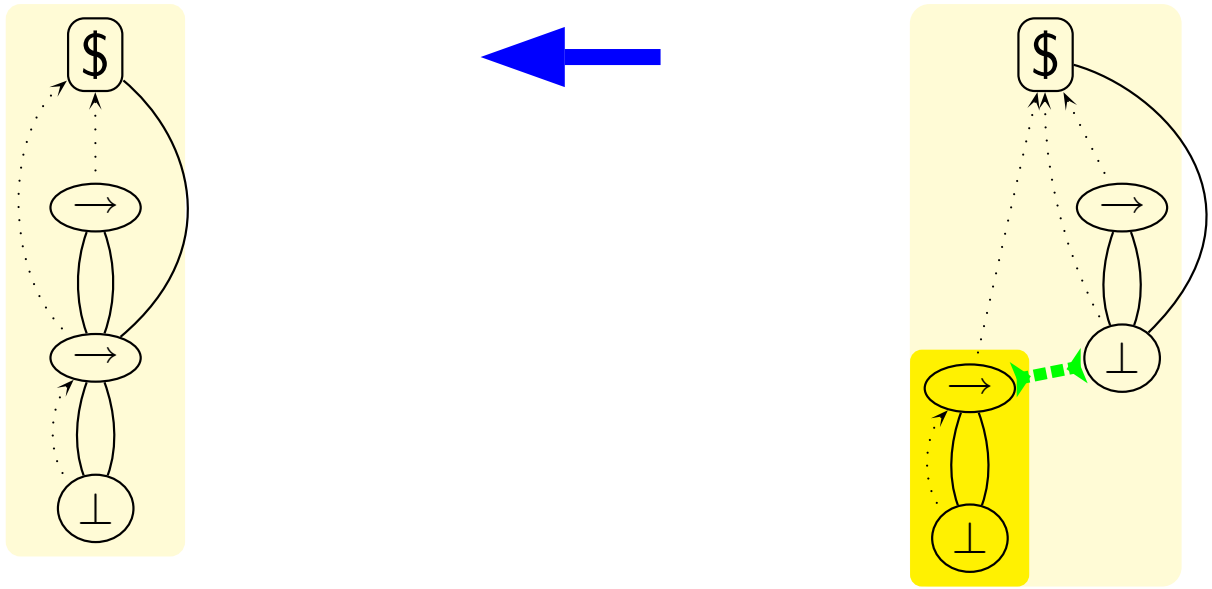
Inst-Expand $\times$ 2, Exist-Elim

Typage de let $y = \lambda(x)\ x$ in $y\ y$



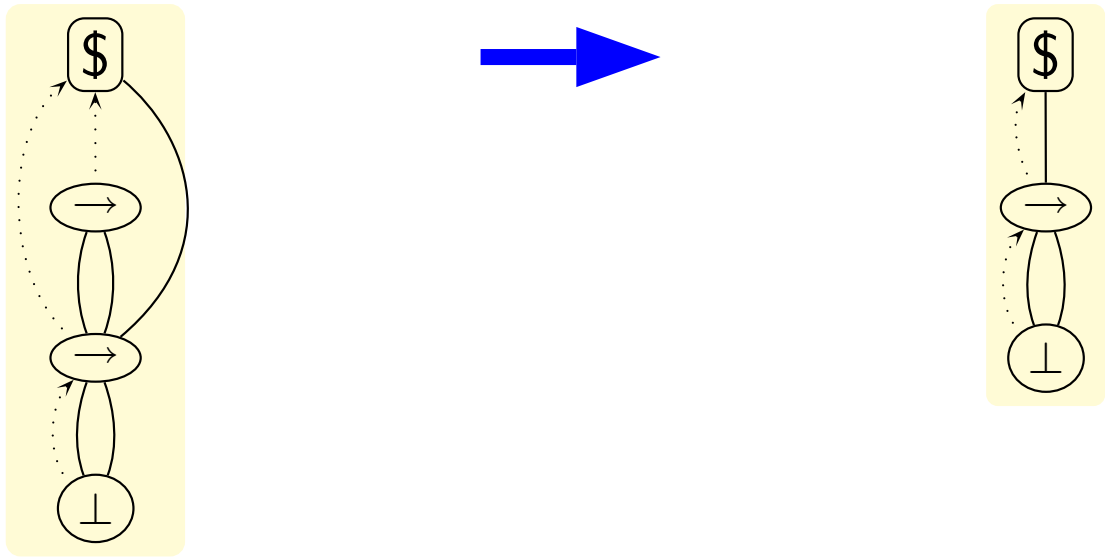
Unif

Typage de let $y = \lambda(x)\ x$ in $y\ y$



Unif

Typage de let $y = \lambda(x)\ x$ in $y\ y$



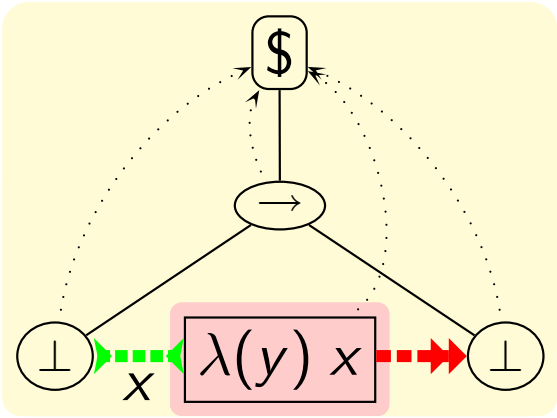
Exist-Elim

Typage de $\lambda(x)\ \lambda(y)\ x$

$$\boxed{\lambda(x)\ \lambda(y)\ x}$$

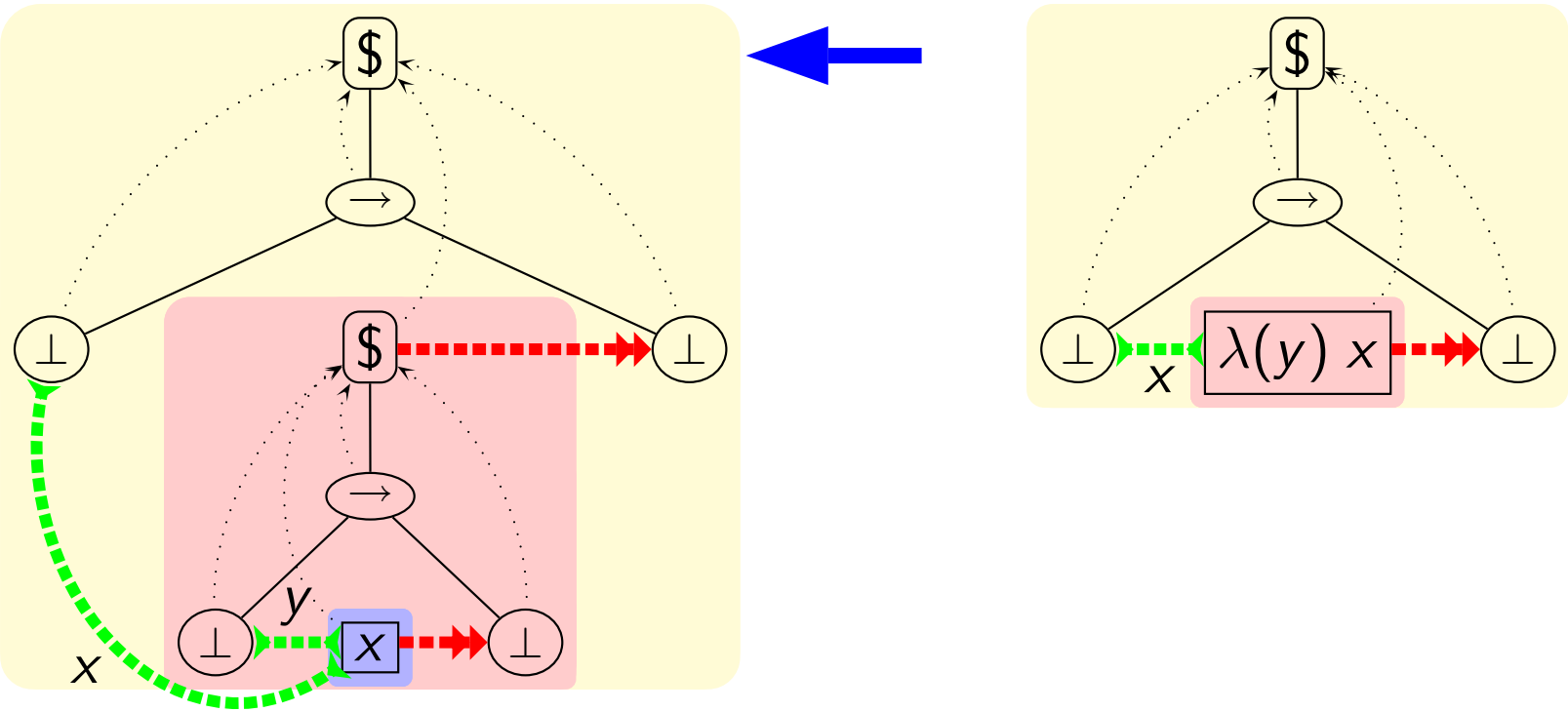
Typage de $\lambda(x)\ \lambda(y)\ x$

$\lambda(x)\ \lambda(y)\ x$



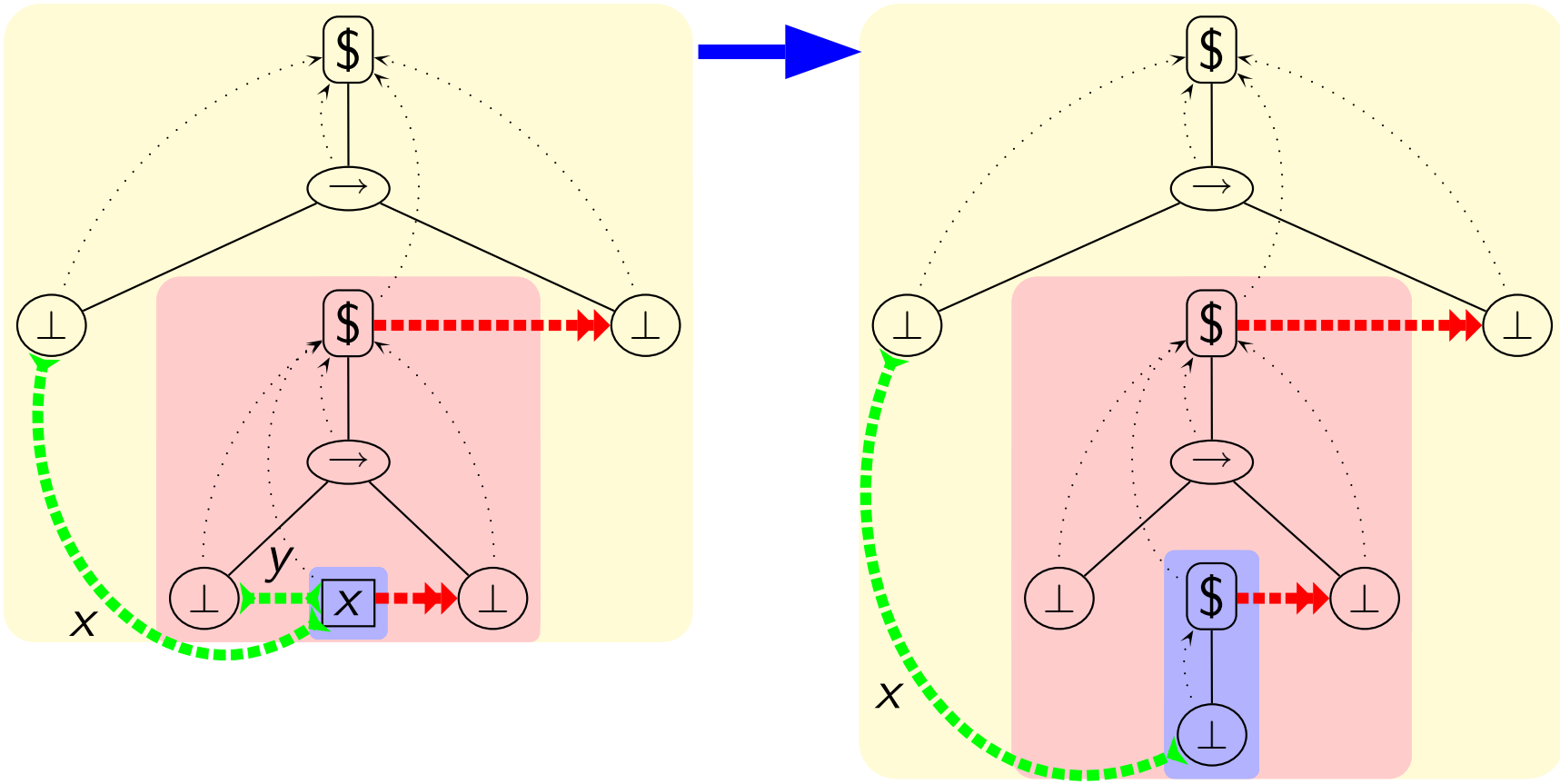
Par définition

Typage de $\lambda(x)\ \lambda(y)\ x$



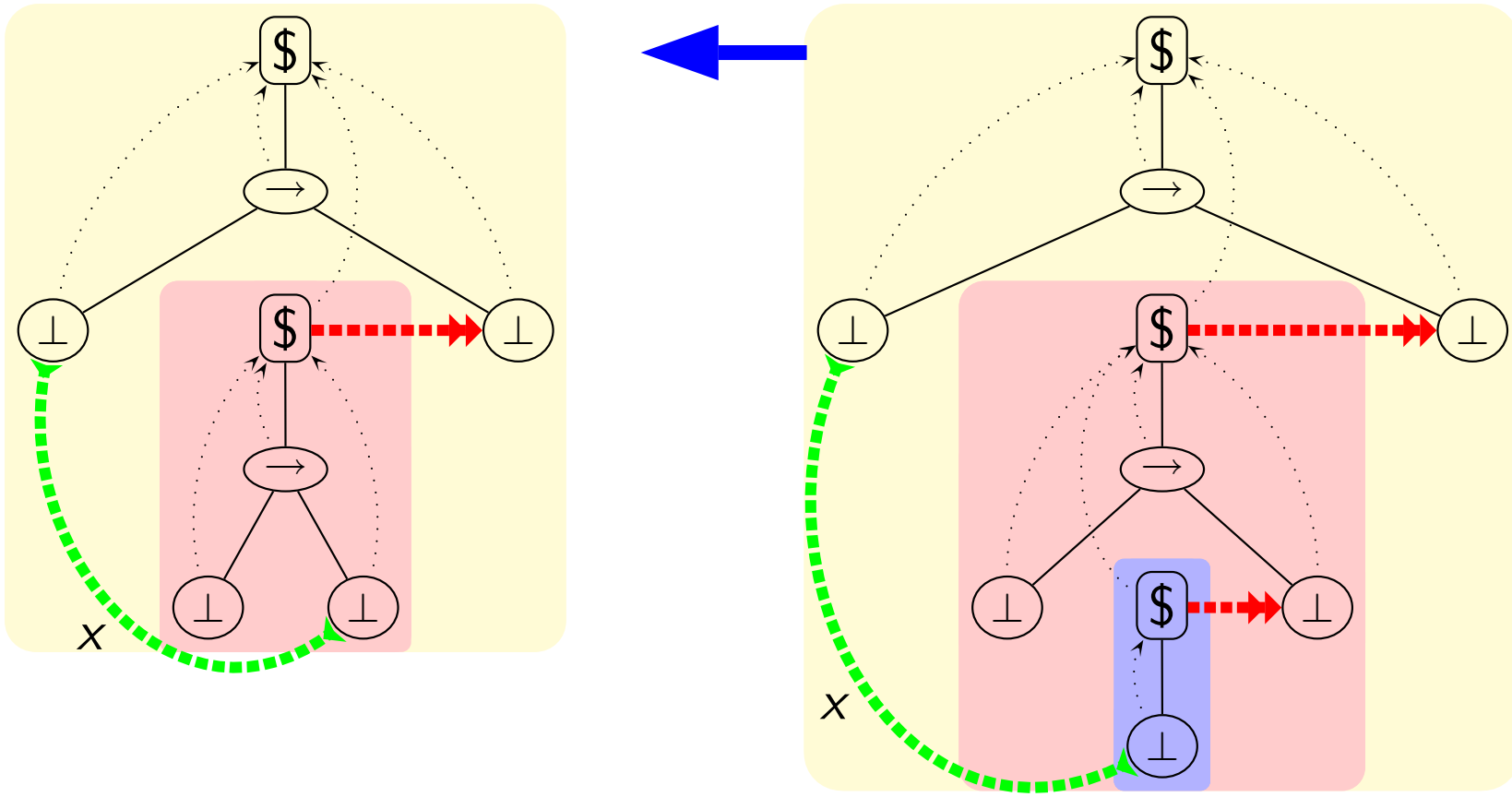
Par définition

Typage de $\lambda(x)\ \lambda(y)\ x$



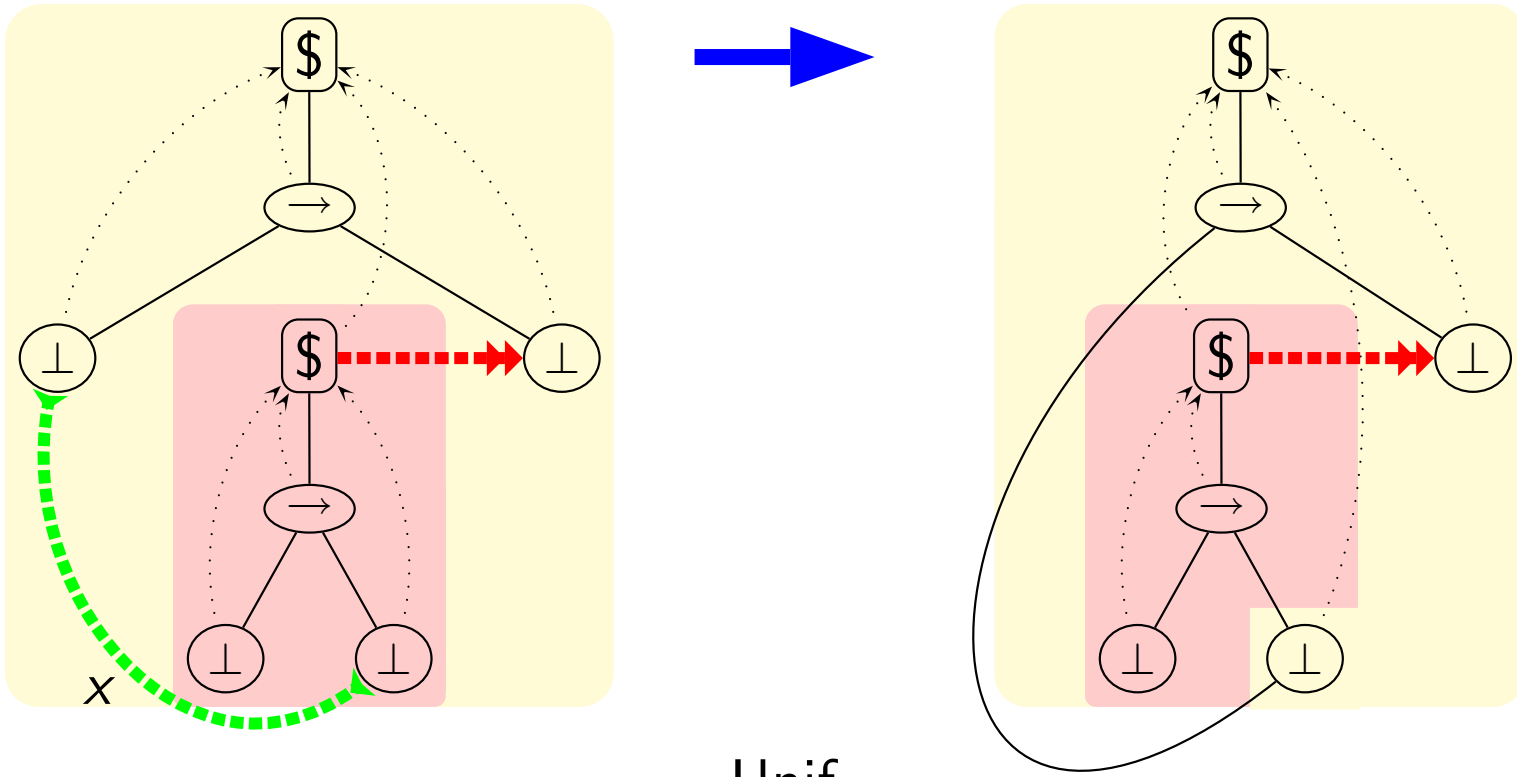
Par définition

Typage de $\lambda(x)\ \lambda(y)\ x$



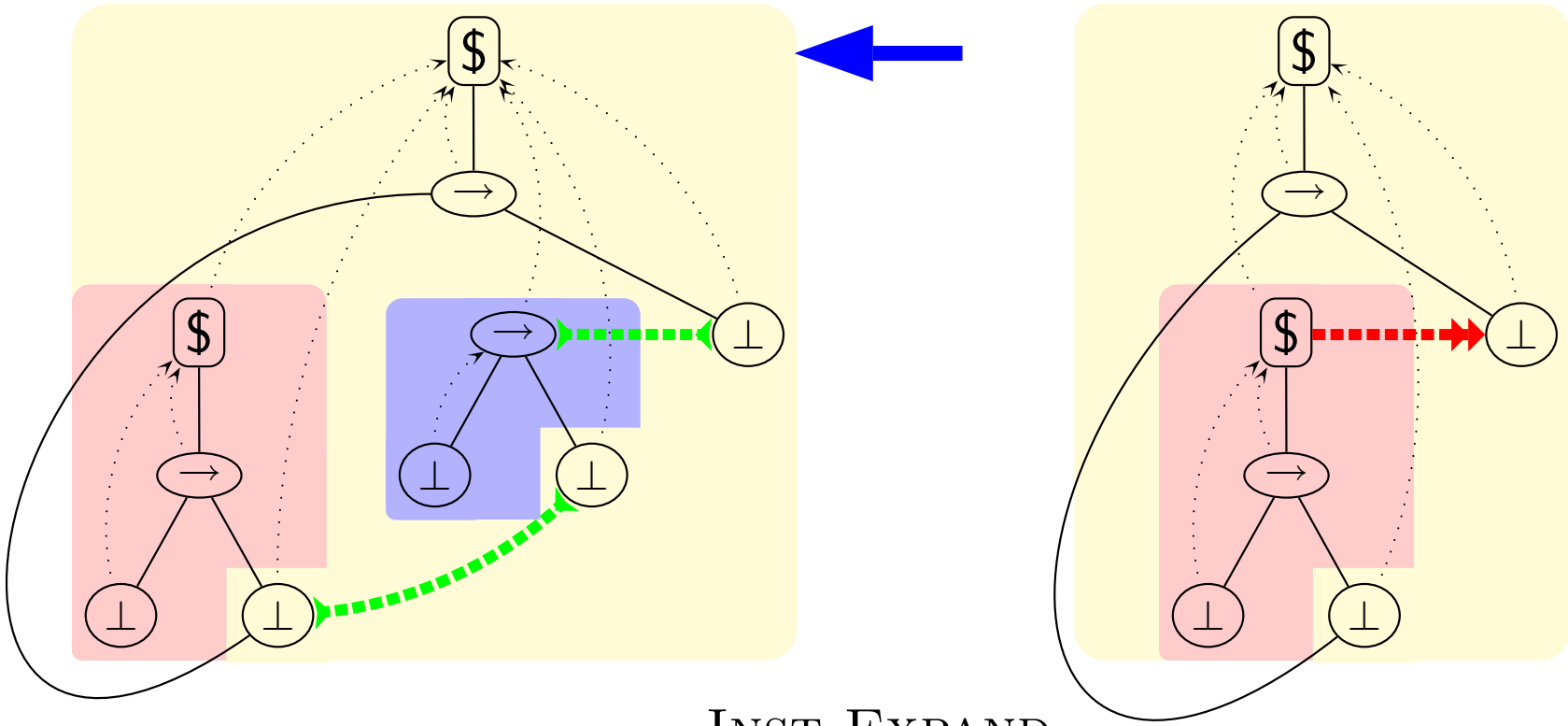
VAR-ABS

Typage de $\lambda(x)\ \lambda(y)\ x$



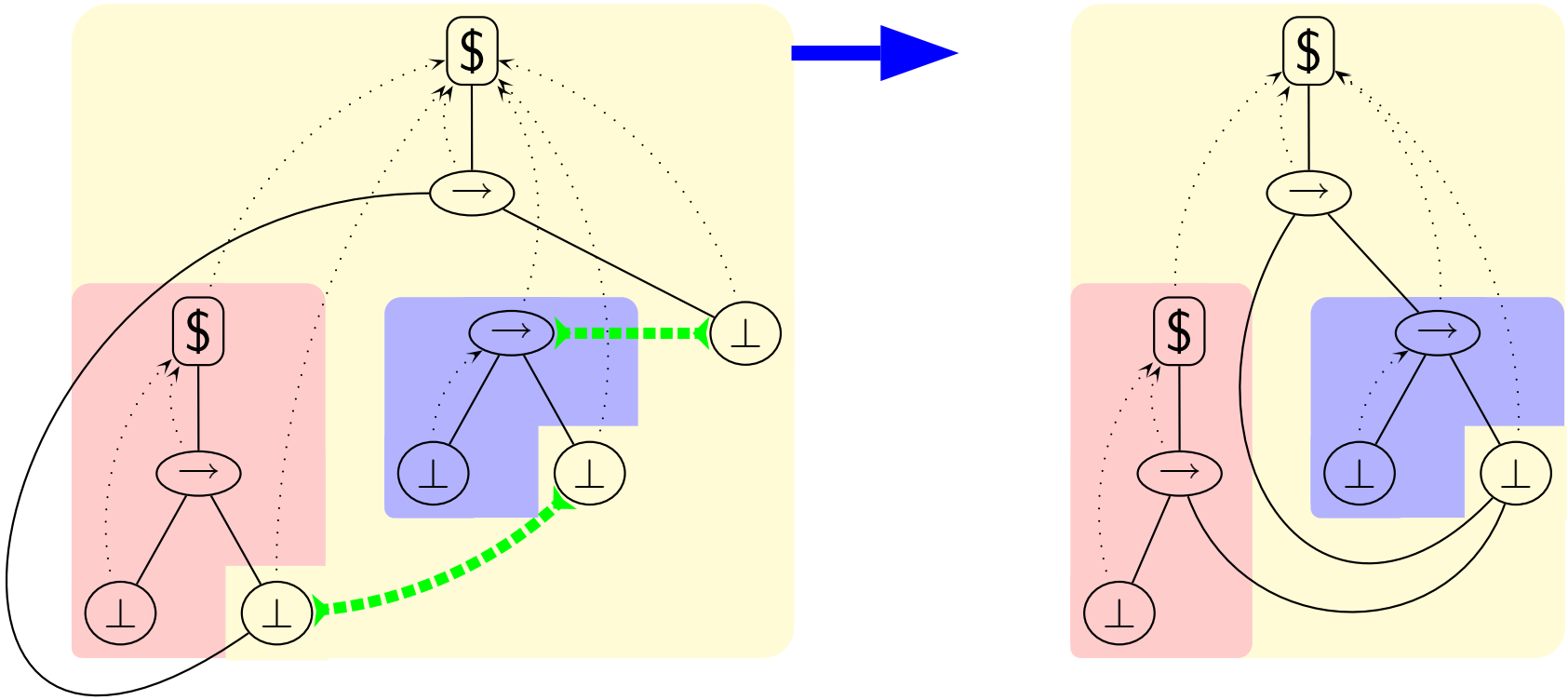
Unif

Typage de $\lambda(x)\ \lambda(y)\ x$



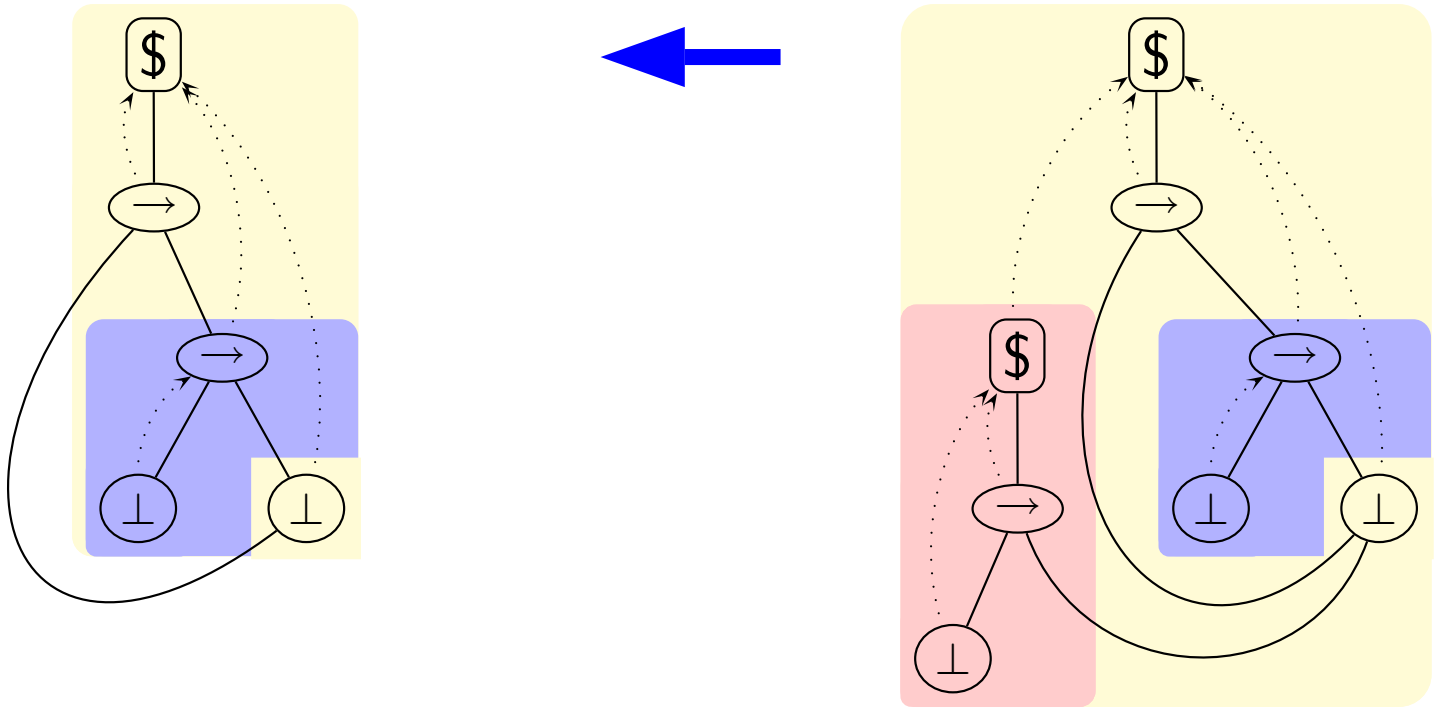
INST-EXPAND

Typage de $\lambda(x)\ \lambda(y)\ x$



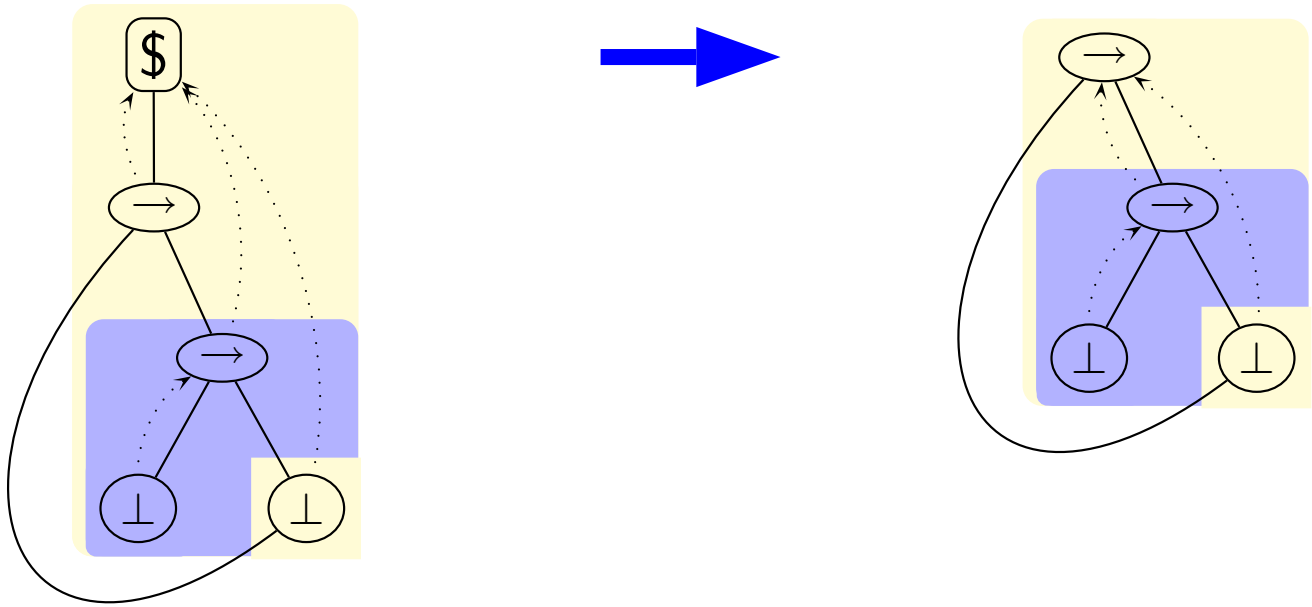
Unif $\times 2$

Typage de $\lambda(x)\ \lambda(y)\ x$



Exist-Elim

Typage de $\lambda(x)\ \lambda(y)\ x$



Résultat : $\forall(\alpha)\ \forall(\beta \geqslant \forall(\gamma)\ \gamma \to \alpha)\ \beta \to \alpha$

ou $\forall(\alpha)\ (\forall(\gamma)\ \gamma \to \alpha) \to \alpha$

Typage de ω

- $\lambda(x)\ x\ x\,?$

Typage de ω

- $\lambda(x)\ x\ x$? Échec

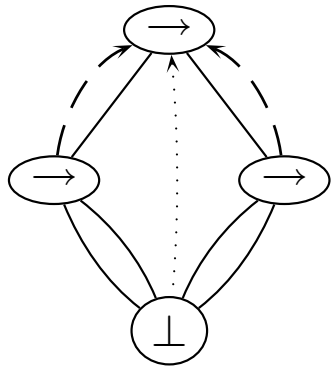
Typage de ω

- $\lambda(x)\ x\ x$? Échec

- $\lambda(x : \alpha \to \alpha)\ x\ x$?

S'expanse en $\lambda(x)$ let $x = (\alpha \to \alpha)\ x$ in $x\ x$

Fonction de coercion :



Typage de ω

- $\lambda(x)\ x\ x$? Échec

- $\lambda(x : \alpha \to \alpha)\ x\ x$?

  S'expanse en $\lambda(x)$ let $x = (\alpha \to \alpha)\ x$ in $x\ x$

  Échec

Typage de ω

- $\lambda(x)\ x\ x$? Échec

- $\lambda(x : \alpha \to \alpha)\ x\ x$?

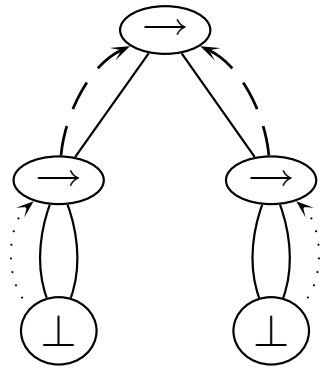
  S'expanse en $\lambda(x)$ let $x = (\alpha \to \alpha)\ x$ in $x\ x$

  Échec

- $\lambda(x : \forall\,(\alpha)\ \alpha \to \alpha)\ x\ x$?

  S'expanse en $\lambda(x)$ let $x = (\forall\,(\alpha)\ \alpha \to \alpha)\ x$ in $x\ x$

Fonction de coercion :



Typage de ω

- $\lambda(x)\ x\ x$? Échec

- $\lambda(x : \alpha \to \alpha)\ x\ x$?

  S'expanse en $\lambda(x)$ let $x = (\alpha \to \alpha)\ x$ in $x\ x$

  Échec

- $\lambda(x : \forall\,(\alpha)\ \alpha \to \alpha)\ x\ x$?

  S'expanse en $\lambda(x)$ let $x = (\forall\,(\alpha)\ \alpha \to \alpha)\ x$ in $x\ x$

Succès : $(\forall\,(\alpha)\ \alpha \to \alpha) \to (\forall\,(\alpha)\ \alpha \to \alpha)$

Typage de ω

- $\lambda(x)\ x\ x$? Échec

- $\lambda(x : \alpha \to \alpha)\ x\ x$?

  S'expanse en $\lambda(x)$ let $x = (\alpha \to \alpha)\ x$ in $x\ x$

  Échec

- $\lambda(x : \forall\,(\alpha)\ \alpha \to \alpha)\ x\ x$?

  S'expanse en $\lambda(x)$ let $x = (\forall\,(\alpha)\ \alpha \to \alpha)\ x$ in $x\ x$

  Succès : $(\forall\,(\alpha)\ \alpha \to \alpha) \to (\forall\,(\alpha)\ \alpha \to \alpha)$

- $\lambda(x : \forall\,(\alpha)\ (\alpha \to \alpha) \to (\alpha \to \alpha))\ x\ x$?

Typage de ω

- $\lambda(x)\ x\ x$? Échec

- $\lambda(x : \alpha \to \alpha)\ x\ x$?

  S'expanse en $\lambda(x)$ let $x = (\alpha \to \alpha)\ x$ in $x\ x$

  Échec

- $\lambda(x : \forall\,(\alpha)\ \alpha \to \alpha)\ x\ x$?

  S'expanse en $\lambda(x)$ let $x = (\forall\,(\alpha)\ \alpha \to \alpha)\ x$ in $x\ x$

  Succès : $(\forall\,(\alpha)\ \alpha \to \alpha) \to (\forall\,(\alpha)\ \alpha \to \alpha)$

- $\lambda(x : \forall\,(\alpha)\ (\alpha \to \alpha) \to (\alpha \to \alpha))\ x\ x$?

  Succès :
  $(\forall\,(\alpha)\ (\alpha \to \alpha) \to (\alpha \to \alpha)) \to (\forall\,(\alpha)\ (\alpha \to \alpha) \to (\alpha \to \alpha))$

Complexité

- ML : l'inférence est $\mathrm{DExp\text{-}Time}$ complète
  (si on n'imprime pas les types)

Complexité

- ML : l'inférence est $\mathrm{DExp\text{-}Time}$ complète
  (si on n'imprime pas les types)

- [McAllester 2003] : inférence en $O(kn(\alpha(kn) + d))$
  - la taille des schémas est bornée (par k)
  - d est l'imbrication maximale des schémas

  En ML, d est l'imbrication maximale des let à gauche
  (*i.e.* let $x =$ (let $y = \ldots$ in $\ldots$) in $\ldots$)

Complexité

- ML : l'inférence est $\mathrm{DExp\text{-}Time}$ complète
  (si on n'imprime pas les types)

- [McAllester 2003] : inférence en $O(kn(\alpha(kn) + d))$
  - la taille des schémas est bornée (par k)
  - d est l'imbrication maximale des schémas

- en ML^{F}, même complexité pour l'unification, mais plus de schémas :

$$
\begin{aligned}
d(x) &= 0 \\
d(\lambda(x)\ a) &= d(a) + 1 \\
d(a\,b) &= \max(d(a), d(b)) + 1 \\
d(\text{let } \overline{x_i = a_i} \text{ in } b) &= \max(d(a_i) + 1, d(b))
\end{aligned}
$$

d est invariant par imbrication droite des let, et est (grossièrement) borné par la hauteur de la plus grosse fonction du programme.

Résultats

- Ajout des contraintes très simple et naturel
  petite extension des types graphiques
- Pas de problèmes d'α-conversion, de commutation de lieurs...
- Approche très similaire à l'inférence pour ML avec rangs
- Excellente complexité de l'inférence
- Le bon système pour étudier des extensions de ML^{F}

Futur

Subject reduction (système implicite ou explicite ?)

Extensions simples

- Typage plus primitif de la récursion
- Équivalence entre ML et le fragment sans bornes = de ML^{F}
- Propagation “inverse” des contraintes le long d’un arc d’instance, pour améliorer les messages d’erreurs

Plus ambitieux

- F^{ω}
- Types récursifs