Compilation 4INFO – Résumé et définitions

Guillaume PANNETIER

1 Analyse lexicale

- *Configuration d'un Automate d'états finis :* couple (E, M) où E est l'état courant et M le mot recherché.
  - Si E = état initial → configuration initiale
  - Si E $\in$ ensemble des états finaux → configuration finale
- *Diagramme de transitions :* graphe fini, orienté, value DT = <V, E, T, V_0, Vf> avec
  - V = ensemble fini de nœuds ;
  - E = ensemble fini d'arc étiquetés ;
  - $V_0 \in V$ = nœud initial ;
  - $Vf \subseteq V$ = ensemble des états finaux.
- *Crible :* outil qui :
  - Reconnaît les mots qui ont un rôle particulier dans le langage (ex : début, fin, si, …) ;
  - Elimine les mots qui peuvent être ignorés (ex : '\n', '/*') ;
  - Interprète les partie du programme qui sont des directives pour le compilateur (ex : #define) ;
  - Initialise la table des symboles pour que l'analyse syntaxique se fasse sur des unités lexicales ;
  - Le crible est appelé par l'analyseur lexical chaque fois que celui-ci trouve un mot.
- *W-chemin :* $w \in T^*$, chemin partant d'un état q jusqu'à un état p tel que la concaténation des étiquettes est égal à w.
- *AFD :* Un Automate d'états Finis est Déterministe si, pour chaque mot reconnaissable, il existe un unique w-chemin partant de l'état initial. (i.e. Δ est une fonction partielle). → Un seul état d'arrivée pour tout couple (état, caractère lu).
- *Un AF est défini de la sorte :* $M = <\Sigma, Q, \Delta, Q_0, F>$ avec :
  - Σ : alphabet ;
  - Q : ensemble d'états ;
  - $Q_0 \in Q$: état initial ;
  - $F \subseteq Q$: ensemble des états finaux ;
  - $\Delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow Q$: relation de transition. (ex : $\Delta((q1, a)) = \{q3, q6, q10\}$).
- *Classe de caractères :* Groupe (généralement intervalle) de caractères utilisable comme un seul caractère dans une grammaire ou une expression régulière. (ex : [0-9]* ; ident => <chiffre> | <lettre>).
  - Lorsque l'on utilise des classes de caractères, il faut veuillez à ce qu'elles soient disjointes deux à deux. (ex : ch = [0-9] ; ca = [a-z] ; cach = [a-z 0-9] ; à la lecture de 'a', impossible de savoir si l'on est dans ca ou dans cach).

- *Suite de définitions régulières :*
  A1 = R1
  A2 = R2
  …
  An = Rn
    - Où les Ai sont des identificateurs, deux à deux distincts, et les Ri sont des expressions régulières sur $\sum \cup \{A1, …, Ai\text{-}1\}$ ($1 \leq i \leq n$).
    - Il est à noter que chaque expression ne peut utiliser que des identificateurs définis au dessus. Cela permet de remplacer les identificateurs sans problème.
- *Exemples :*
  ch = 0-9
  ca = a-z
  intConst = ch ch*

2 Analyse syntaxique

- *Analyse syntaxique :*
    - Structure une séquence d'unités lexicales en unités syntaxiques ;
    - Détecte les erreurs de structure (ex : parenthèses, commentaires imbriqués s'ils sont gérés) ;
    - S'efforce de donner un diagnostic en cas d'erreur ;
    - Essaie de récupérer les erreurs pour continuer l'analyse.
- *Outils utilisés :* En général → GNC avec un automate à pile car les regexp ne suffisent pas (ex : structures récursives).
    - L'automate à pile est de préférence déterministe mais pas toujours (langages de type II).
    - On essaie de se ramener à des classes de grammaires déterministes (LL(1) ou LL(k) par exemple) car des techniques éprouvées existent pour ce type de classes.
- *Une analyse non déterministe peut impliquer :*
    - Un ensemble de solutions possibles ;
    - Plusieurs exécutions avec les mêmes données peuvent donner des résultats différents.
- *Syntaxe concrète :* Structure de surface, utile à l'utilisateur et à l'analyseur syntaxique pour reconnaître la structure (les priorités).
    - Contient par exemple des marqueurs purement syntaxiques (ex : parenthésage) qui permettent à l'analyseur syntaxique de décider de la priorité de dérivation.
- *Syntaxe abstraite :* Structure profonde, de représentation interne, utile à l'analyseur sémantique pour l'interprétation. → Les ambiguïtés syntaxiques ont été levées par l'analyseur syntaxique.
- *Remarques :*
    - Pour une syntaxe concrète, on peut trouver plusieurs syntaxes abstraites, et inversement ;

- Si la syntaxe abstraite fait plus que supprimer les marqueurs syntaxiques (comme les parenthèses par exemple) ; alors il faut le spécifier.
- *Deux types d'analyses existent :*
  - Analyse descendante (prédictive) : grammaires déterministes (ex : LL(1)) ;
  - Analyse ascendante (réductive) : grammaires non déterministes (ex : LR(1)).
- *Non-terminal inaccessible :* pas de mot α et β tels que : $S \xrightarrow{*} \alpha A \beta$;
- *Non-terminal improductif :* pas de mot u dans Vt* tq $A \xrightarrow{*} u$;
- *Proto-phrase :* mot $g \in (Vt \cup Vn)^*$ tq $S \xrightarrow{*} g$ (proto-phrase gauche ou droite selon la dérivation) ;
- *Grammaire réduite :* grammaire qui ne contient aucun non-terminal inaccessible ou improductif ;
- *Automate à pile (AP) :* $P = <V, Q, \Delta, q0, F)$ avec :
  - V = alphabet ;
  - Q = ensemble fini d'états ;
  - $q0 \in Q$ = état initial ;
  - $F \subseteq Q$ = ensemble des états finaux ;
  - $\Delta \in Q^+ \times (V \cup \{\varepsilon\}) \times Q^*$: relations de transition. (Q^+ étant la pile)
- *Configuration d'un AP :* couple $(\gamma, w) \in Q^* \times V^*$
- *Automate à pile déterministe (APD) :*
  - Soient $(\gamma 1, a, \gamma 2) \in \Delta$; $(\gamma' 1, a', \gamma' 2) \in \Delta$, SI
  - $\gamma' 1$ suffixe de $\gamma 1$ ou $\gamma 1$ suffixe de $\gamma' 1$,
    ET
    a' préfixe de a ou a préfixe de a'
  - ALORS a=a', $\gamma 2 = \gamma' 2$, $\gamma 1 = \gamma' 1$
  - i.e. pas de compétition entre :
    - une ε-transition et une transition sur un mot de V ;
    - deux transitions sur des mots de V ;
    - sur le choix du sommet de pile.
- *Calcul de premier et suivant (même si en général un calcul formel n'est pas demandé au D.S.) :*
  - Soit P l'ensemble des règles de production de la grammaire G ;
  - Soit γ la relation « peut commencer par » sur $Vn \times (Vt \cup Vn)$; $X\gamma A \Leftrightarrow X \rightarrow wA\ldots \in P$
  - Soit α la relation « est à côté de » sur $(Vt \cup Vn)$ w $(Vt \cup Vn)$; $A\alpha B \Leftrightarrow C \rightarrow \ldots AwB\ldots \in P$
  - Soit δ la relation « peut terminer par » sur $Vn \times Vn$; $A \,\delta\, B \Leftrightarrow B \rightarrow \ldots Aw \in P$
  - γ^* étendue à $(Vt \cup Vn) \times (Vt \cup Vn)$ avec $a\gamma^*b$ et $a, b \in Vt \Leftrightarrow a=b \rightarrow$ i.e. $\gamma^* = \gamma^+ \cup Id$
  - Idem pour $\delta^* = \delta^+ \cup Id$
  - γ^* et δ^* permettent de dire « si $X \in$premier(S) et $U \in$premier(X), alors $U \in$premier(S) »
  - Premier(A) = $\gamma^+(A) \cap Vt$
  - $a \in$ suivant(T) $\Leftrightarrow S \xrightarrow{*} uTav$

$$\Leftrightarrow a \in Vt \text{ et } \exists A \in Vn \text{ tq } T\delta^*A$$

ET

$$\exists f \in (Vt \cup Vn) \text{ tq } A\alpha f$$

ET

$$f\gamma^*a$$

→ a $\in$ suivant(T)⇔a$\in$Vt ET T$\delta^*\alpha\gamma^*$a (T peut terminer qqch qui est à côté d'autre chose qui peut commencer par a).

- Calcul de $\delta^*\ \alpha\ \gamma^*$:
  - 1. $\alpha\gamma^*$ = Si AαB et Bγ^+a, on ajoute le couple (A, a) ;
  - 2. $\delta^*(\alpha\gamma^*) = \delta^+\alpha\gamma^* \cup \alpha\gamma^*$.

2.1 Analyse descendante – LL(K)

- *Grammaire simplement LL(1) :* Si chaque alternative des Vn commence par un Vt différent ET pas de ε-production.
- *Grammaire LL(k) :* $\forall$ A $\in$ Vn, A→β $\in$ P, A → γ $\in$ P, β ≠ γ ➔ $premier_k(\beta_\alpha) \cap premier_k(\gamma_\alpha) = \varnothing$, $\forall\ \alpha \in$ Vt* tq S *→ wAα.
  - Grammaire ambiguë ➔ non LL(K) (pour aucun K);
  - Grammaire récursive gauche ➔ non LL(K) (pour aucun K).
- *LL(k) :* la prélecture de k unités lexicales suffit à garantir l'unicité de la règle de dérivation dans une analyse descendante ;
- *Grammaire généralisée NC :* P : Vn → Regexp ; Les regexp donnent un substitut aux récursivités gauches.
- *Détection d'erreurs avec LL(1) :*
  - Mauvais terminal en sommet de pile (≠ du symbole d'entrée courant) ;
  - Le non-terminal en sommet de pile ne permet pas de lire le symbole courant (a $\notin$ premier(A) OU a$\notin$premier(A)$\cup$Suivant(A) si null(A) avec A le sommet de pile et a le symbole courant).
- *Récupération sur erreur :*
  - Contrairement à une simple détection d'erreur qui stoppe l'analyse, la récupération sur erreur tente de trouver un diagnostic sur la suite du programme en continuant l'analyse ; (Attention : aucune correction n'est apporté, on tente juste de préciser l'erreur).
- *Récupération en mode panique :* Définir tous ce qui doit apparaître dans toutes les situations (ex en C : « ; » en fin d'instruction, « ; » ou « , » lors d'une déclaration, …) ;
- *Récupération simple :*
  - Vt en sommet : si l'unité lexicale en sommet ne peut être reconnue, simuler son insertion et continuer
  - Vn en sommet : On saute les unités lexicales en entrée jusqu'à en trouver une appartenant à un ensemble de synchronisation.
  - Construction d'un ensemble de synchronisation :
    - Suivant(A) $\subset$ synchro(A), pour sauter A ;
    - Premier(A) $\subset$ synchro(A) pour essayer de reprendre l'analyse de A ;
    - Pour construction bas niveau, ajouter symboles commençant les constructions hauts niveau, pour pouvoir reprendre l'analyse plus tôt.
- *Récupération récursive :*
  - deux modes, analyse et erreur (le mode analyse est activté depuis S mais aussi depuis le mode erreur quand on passe sur un symbole caractéristique du début d'un Vn ; cela évite de sauter de trop grandes parties d'entrées).
  - $\forall$ Vn, il faut :

        - Un ensemble de synchronisation pour reprendre le mode analyse courant depuis le mode erreur ;
        - Un ensemble de continuation pour empiler une nouvelle analyse depuis le mode erreur.

2.2 Analyse ascendante – Langages LR(K)

- *Principe :* On tente de réduire une chaîne vers l'axiome de la grammaire. A chaque étape de réduction, une sous-chaîne correspondant à la partie droite d'une production est remplacée par le symbole de la partie gauche de cette règle de production.
- *Manche de proto-phrase droite γ :*
    - Soit une production A→β ;
    - Un manche de la proto-phrase droite γ est la production A→β ainsi que la position dans γ où la chaîne β peut être trouvée et remplacée par A pour produite la proto-phrase droite précédente dans une dérivation droite de γ ;
    - S *→ w1Aw2 → w1βw2.
    - Exemple :
        - S → adEfg
        - E → eee ;
        - γ = adeeefg proto-phrase droite ;
        - Le manche est donc ici le couple (E→eee, position 3) ;
    - (Attention, ce n'est parce qu'on peut réduire que c'est un manche, il faut ensuite pouvoir remonter jusqu'à la racine)

      →Si une grammaire est non ambiguë, chaque proto-phrase droite de cette grammaire est exactement un manche.
- *Implémentation :*
    - décaler les caractères d'entrées vers la pile jusqu'à trouver un manche β en sommet de pile.
    - Réduire β vers la partie gauche de la production ;
    - Répéter jusqu'à fin.
    - Décaler : mettre le symbole courant du tampon dans la pile ;
    - Réduire : Remplacer le manche en sommet de pile par le Vn correspondant.
- *Algorithme :*
    - Le théorème nous dit qu'il n'y a qu'un seul manche
      → Réduire dès que l'on peut pour ne pas le rater, sinon décaler.
      → Retourner en arrière en cas d'impasse (et cette fois-ci décaler à la place de réduire)
- *Analyse LR(K) :*
    - Avantages :
        - Couvre quasiment toutes les grammaires NC ;
        - Détecte les erreurs plus tôt qu'une analyse descendante ;
        - Efficace.
    - Inconvénients :
        - Tables d'analyse fastidieuses ;
        - Récupération d'erreurs pénible à spécifier.

2.3 Analyse ascendante – Construction des tables

- *Rappels :* Analyse LR = ascendante et déterministe. Tous les analyseurs LR ont le même comportement, seules les tables changent.
- *Grammaire augmentée :* S' nouvel axiome. On ajoute la production S' → S. Permet de simplifier le traitement de S → …S…|… .
- *Préfixe viable :* Le but est de réduire le plus tôt possible pour ne pas rater le manche.
  - Réduction le plus à gauche → Dérivation inverse le plus à droite ;
  - Eviter les impasses.
  - ∀ proto-phrase droite w1βw2, chaque préfixe de w1β est appelé préfixe viable. ex : S *→ w1βw2.   β = manche
  - Les préfixes viables sont ceux qui peuvent apparaître sur la pile d'un analyseur par décalage réduction.

2.4 Analyse ascendante – Construction des tables SLR à base d'items LR(0)

- *Item LR(0) :* Production avec un point repérant une position dans sa partie droite.
- *Item valide :* A→β1 . β2 est valide pour un préfixe viable αβ1 ⇔ ∃ une dérivation la plus à droite S *→ αAw → αβ1β2w.
  - Ex : A→β1.β2 item valide ;
  - Si non(null(β2)) alors l'analyse de la partie droite n'est pas finie → Décalage.
  - Si null(β2) alors le manche semble être A→β1 ⇒ Réduction.
  - Avec les items, on va effectuer des dérivation « symboliques » afin de préparer les réductions ⇒ 2 opérations : Fermeture et transition.
- *Fermeture (calcul de point fixe) :* I ensemble d'items, Fermeture(I) est l'ensemble d'Items construit par deux règles :
  - 1. Placer les items de I dans Fermeture(I) ;
  - 2. Si A→α.Bβ ∈ Fermeture(I) ET
Si B → γ ∈ P alors
ajouter B→.γ à Fermeture(I)
- *Transition :* I ensemble d'items, X ∈ (Vt ∪ Vn) , Transition(I, X) = Fermeture(J) où J = {A→αX.β tq A→α.Xβ ∈ I}
  - Remarque : Si I ensemble d'items valides pour un préfixe viable γ alors Transition(I, X) ensemble d'items valides pour le préfixe viable γX.
- *AFD Produit :* Si G est LR(0), l'ensemble des items valides pour le préfixe viable γ est exactement l'ensemble des items atteints depuis l'état initial le long d'un chemin étiqueté γ dans l'AFD construit à partir de la collection canonique LR(0).
- *Algorithme de construction des tables :*
  - Table action :
    - Pour [A→α.aβ] ∈ Ii tq Transition(Ii, a) = Ik et a ∈ Vt alors Action[i, a] = décaler(k) ;
    - Pour [A→α.] ∈ Ii tq A ≠ S' et ∀ a ∈ suivant(A), Action[i, a] = réduire par A→α ;
    - Si [S' → S.] ∈ Ii alors Action[i, $] = Accepter ;
    - Pour tout le reste → Erreur. Si ces règles produisent des conflits, la grammaire n'est pas SLR et aucun analyseur n'est produit.

    - Table successeurs :
        - Pour A ∈ Vn tq Transition(Ii, A) = Ik, alors successeur(i, A) = k.
- *Conflits :* Les conflits que l'on peut trouver sont décaler-réduire ou réduire-décaler, mais jamais décaler-décaler grâce au regroupement dans les ensembles d'items.
- *Détection des erreurs :* Erreur lorsqu'il n'y a plus de continuation valide. Une erreur est donc une entrée « vide » dans la table action → Même méthode de récupération que pour LL.
- *Grammaires ambiguës :* Une grammaire ambiguë n'est pas LR → Génération de conflits.
    - On peut souvent lever ces ambiguïtés avec des règles définissant des propriétés d'opérateurs, des priorités d'opérateurs ou des cas particuliers.
- *Priorité :* id+id*id, *>+ donc on décale pour gérer d'abord le *
- *Associativité :* id+id+id, + est associatif à gauche donc on réduit pour gérer d'abord la partie gauche.
- *Cas particulier :* Pour gérer les conflits dus à des cas particuliers, on donne toujours la priorité à ces derniers.

3 Analyse sémantique

- *Analyseur sémantique :* Prend en entrée un arbre syntaxique abstrait et rend un arbre décoré. Vérifie les propriétés contextuelles (déclarations de variables, arité des procédures, cohérences des types).
- *Propriété statique :* Propriété (sémantique) statique ⇔
    - ∀ occurrences de cette construction, la valeur de la propriété est la même dans toutes les exécutions
    - Cette propriété peut être calculée pour chaque occurrence de la construction dans un programme correct
      → Calculable entièrement à la compilation.
- *Grammaire attribuée :* Extension des GNC. Technique de spécification de propriétés sémantiques statiques.
    - Associent aux symboles d'une GNC des attributs comme support d'informations sémantiques.
    - Elles permettent d'exprimer des dépendances fonctionnelles entre les attributs.
    - Sous certaines conditions sur ces dépendances, il est possible d'évaluer les attributs dans l'arbre syntaxique d'un programme correct. Les attributs permettent de faire transiter les informations d'un endroit du programme à un autre.
    - Remarque : Les attributs sont associés à des équations pour le pas être directionnels.
- *Attributs :*
    - synthétisé : Attribut évalué en commençant par les feuilles et en remontant.
    - Hérité : Attribut évalué en commençant par un nœud haut de l'arbre et en descendant.
    - Attention : Jamais synthétisé ET hérité.
    - Remarque : Un attribut est associé à un seul symbole
        - Ex : L.typeH est le nom de l'attribut, typeH est un suffixe de l'attribut.

- *Définition d'une GA :* GA = (G, Attr, D, F, R) où
  - G = GNC de départ ;
  - Attr = ensemble d'attributs (synthétisés ou hérités) ; Her(X) = ensemble d'attributs hérités attachés à X ; Syn(X) = ensemble d'attributs synthétisés attachés à X ;
  - D = ensemble de domaines de valeurs des attributs (D = {Da tq a ∈ Attr}) ;
  - F = ensemble des fonctions dites « sémantiques » → utilisées dans les règles de sémantiques ;
  - R = règles sémantiques :
    - ∀ ai ∈ Her(X), $1 \leq i \leq np$; ai = $f_{p,ai}(b_{j1,1}, \ldots, b_{jk,k})$
    - ∀ a0 ∈ Syn(X0), a0 = $f_{p,a0}(b_{j1,1}, \ldots, b_{jk,k})$ avec
      p = X0 → x1…xnp ;
      $0 \leq jl \leq np$ et $1 \leq l \leq K$;
      $b_{jl,l} \in Attr(X_{jl})$;
      $f_{p,ai} \in F$;
      $f_{p,ai} \in Da_{j1,1} \cup \ldots \cup Da_{jk,k}$
- *Fonction sémantique :* Fonction mathématiques qui retourne un résultat mais n'a aucun effet de bord. Les seules variables autorisées comme paramètre sont les attributs littéraux de la règle de production.
- *Occurrence d'attribut :* On note Oc(p) l'ensemble des occurrences d'attributs dans p. (p règle de production)
  - Soit aj ∈ Oc(p), si a ∈ Her(Xi) ou a ∈ Syn(X0) alors aj = occurrence de définition, sinon occurrence d'utilisation.
- *Forme normale :* Si un attribut est paramètre d'une fonction, alors c'est une occurrence d'utilisation, sinon occurrence de définition.
- *Sémantique d'une GA :* But = affecter une valeur aux attributs de chaque nœud.
  - Soit n un nœud de t, symb(n) son étiquette, si symb(n) ∈ Vn, soit prod(n) la production appliquée en n, alors ∀ a ∈ Attr(symb(n)) se trouve en n une instance d'attribut a_n.
- *Occurrence et instance :* Les occurrences existent dès que la grammaire est écrite alors que les instances n'existent qu'au moment de la compilation (au calcul des attributs).
  - Pour l'ensemble de toutes les instances d'attributs dans t, V(t) = {am | m ∈ t, a ∈ Attr(symb(m)) }.
  - On obtient un système d'équations dont les inconnues sont les am.
  - Si ce système est récursif, il peut y avoir aucune solution ou bien plusieurs. Sinon → une unique solution, une valeur bien définie par l'instance d'attribut.
- *GA bien formée :* Si aucun système n'est récursif. Dans ce cas, la sémantique de la GA est l'affectation non ambiguë de valeur d'attribut en chaque nœud de l'arbre.

3.1 Analyse sémantique – Inférence et vérification de types

- *Système de Milner :*
  - Fortement typé (i.e. aucune erreur de typage possible à l'exécution) ;
  - Polymorphe (variables de type autorisées) ;
  - Statique (décidable à la compilation) ;

- Extensible.

- *Exemple :*

$$\frac{TS \vdash E{:}T1 \rightarrow T2 \quad TS \vdash E'{:}T1}{TS \vdash E\,E'{:}T2}$$

Si , sous l'ensemble d'hypothèses TS, E est de type « T1 donne T2 » et que, sous l'ensemble d'hypothèses TS, E' est de type T1, alors, sous l'ensemble d'hypothèses TS, E appliquée à E' est de type T2.

- *Polymorphisme :* fonction x → x' : 'a → 'a. 'a est une variable de type. Si monomorphe, il faudrait déclarer une fonction par type de paramètre.
- *Type principal :* Type le plus général associé à une fonction.
  Ex : function(x, y) → x
  (int, int) → int
  (int, bool) → int
  ('a, 'a) → 'a
  ('a, 'b) → 'a : Type principal, les autres types présentés ci-dessus sont plus spécifiques.
- *Substitution :* Ensemble de paires (X/T) où X est une variable de type et T un type, possible variable.
  Ex : ϴ = {'a/'b ; 'b/int ; 'c/bool}
- *Instance de type :* T' est une instance du type T_0 ⇔ ∃ une substitution ϴ telle que T' = $ϴT_0$ (Pour une expression donnée, toutes ses occurrences doivent avoir un type qui est instance du type principal).
- *Unification :* Deux types T1 et T2 s'unifient si ∃ une substitution ϴ telle que ϴT1 = ϴT2.
- *Algorithme d'unification :*
  - Initialisation :
    - ϴ = ∅ ;
    - pile = [T1 = T2] ;
    - echec = false.
  - Tant que (!pile.estVide() && !echec) faire

```
dépiler(X = Y) ;
si (X variable qui n'apparaît pas dans Y) alors
    substituer X par Y dans la pile ;
    substituer X par Y dans ϴ ;
    ajouter X/Y dans ϴ ;
sinon si (Y variable qui n'apparaît pas dans X) alors
    substituer Y par X dans la pile ;
    substituer Y par X dans ϴ ;
    ajouter Y/X dans ϴ ;
sinon si (X et Y sont des constantes ou variables identiques) alors
   continue ;
sinon si (X = f(x1, …, xn) et Y = f(y1, …, yn) pour un foncteur f et n > 0) alors
   pour i = 1 à n faire
      empiler xi=yi ;
  fpour
```

sinon
echec = true ;
fsi
ftq

- *Vérification de type :*
  - TS ⊢ E :T se lit « Sous les hypothèses TS, l'expression E est de type T » ;
  - TS(x) = T pour une variable ou une constante ;
  - La partie haute d'une règle est appelée antécédent. La partie basse conséquent. Une règle sans antécédent est un fait.
  - Attention :
    - Dans un système monomorphe, « est de type … » = « a un type identique à … »
    - Dans un système polymorphe, « est de type … » = « a un type qui est une instance de … »
- *Analyse du type :* Le principe est de partir du typage à prouver et remonter jusqu'à n'avoir que des faits.
  - Initialisation :
    - S = ensemble des typages à prouver = ∅ ;
    - Placer le typage à prouver (TS ⊢ E : T) dans S.
  - Tant que (S != ∅ et ϴ cohérent) faire
    choisir une règle R dont le conséquent C fait partie de S à une substitution près ;
    ajouter les antécédents de R à S, après substitution ;
    mettre ϴ à jour ;
    enlever C de S ;
    ftq
- *Exemple :*
  - Typage à prouver : [3 : ent] ⊢ Soit f = fn x ⇒ x dans (f3) fin : ent
  - Initialisation :
    - S = {[3 : ent] ⊢ Soit f = fn x ⇒ x dans (f3) fin : ent}
  - Itération 1 :

$$LET\ \frac{[3{:}\,ent] \vdash fn\ x \Rightarrow x : fonc(T1, T1) \quad [3{:}\,ent] + [f{:}\,fonc(T1, T1)] \vdash f3{:}\,ent}{[3{:}\,ent] \vdash Soit\ f = fn\ x \Rightarrow x\ dans\ (f3) fin : ent}$$

    - ϴ = {T1/ent}
    - S = {[3 :ent] ⊢ fn x ⇒ x :fonc(T1, T1) ; [3 : ent] + [f :fonc(T1, T1)] ⊢ f 3 : ent}
  - Itération 2 :

$$ABS\ \frac{[3{:}\,ent] + [x{:}\,T1] \vdash x{:}\,T1}{[3 : ent] \vdash fn\ x \Rightarrow x : fonc(T1, T1)}$$

    - S = {[3 : ent] + [f :fonc(T1, T1)] ⊢ f 3 : ent ; [3 :ent] + [x :T1] ⊢ x :T1}

- Itération 3 :

$$APPLI\ \frac{[3: ent; f\ : fonc(T1, T1)] \vdash f\ : fonc(ent, ent) \quad TS\ \vdash 3 : ent}{[3: ent] +\ [f: fonc(T1, T1)] \vdash f3 : ent}$$

  - S = {[3 :ent] + [x :T1] ⊢ x :T1 ; [3 :ent ; f : fonc(T1, T1)] ⊢ f : fonc(ent, ent) ; TS⊢ 3 :ent}

- Itération 3 :

$$VAR\ \frac{}{[3: ent] +\ [x: T1] \vdash x : T1}\ si\ TS(x) =\ T1$$

  - S = {[3 :ent ; f : fonc(T1, T1)] ⊢ f : fonc(ent, ent) ; TS⊢ 3 :ent}

- Itération 4 :

$$VAR\ \frac{}{[3: ent; f\ : fonc(T1, T1)] \vdash f: fonc(ent, ent)}$$

  - S = {TS⊢ 3 :ent}

- Itération 5 :

$$CONS\ \frac{}{TS\ \vdash 3: ent}$$

  - S = ∅