



HAL
open science

Le lambda-calcul

Olivier Ridoux

► **To cite this version:**

| Olivier Ridoux. Le lambda-calcul. Master. France. 2021. hal-03155381

HAL Id: hal-03155381

<https://univ-rennes.hal.science/hal-03155381v1>

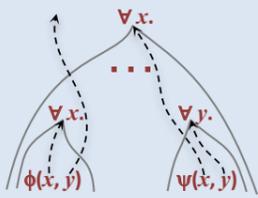
Submitted on 1 Mar 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

1 Les quantificateurs et leur portée

Le concept de **quantificateur** évoque le plus souvent celui des quantificateurs du calcul des prédicats, \forall et \exists . Au delà de leur sens **logique**, ces quantificateurs ont un sens purement **syntactique**. Ils **lient** des variables et leur donnent une **portée**. Dans une formule $\forall x. \phi$ ou $\exists x. \phi$, on dit que le quantificateur **lie** la variable x dans la sous-formule ϕ . Cela signifie que les occurrences libres (c-à-d. pas sous une autre quantification) de x dans ϕ sont indépendantes d'autres occurrences qui seraient liées par un autre quantificateur.



En fait, plutôt que de dire que \forall lie x dans ϕ , on ferait mieux de dire que x est le nom d'une instance du quantificateur \forall , et que les occurrences de x dans ϕ désignent l'instance du quantificateur qui les lie (flèches pointillées dans le diagramme). De cette façon, une variable désigne simplement la première instance de quantification englobante qui porte le même nom, d'où une notion de **portée**. Et tout autre moyen qu'un nom, ex. un rang, rendrait ce service de désigner une instance englobante particulière.

Les noms des variables liées par un quantificateur sont tellement anecdotiques qu'on peut les changer arbitrairement, mais dans la limite de ne pas confondre les instances de quantificateur. Le renommage est formalisé par une substitution, notée $[y/x]$, qui exprime que les occurrences libres de x peuvent être remplacées par y , mais seulement si ça ne cause pas une liaison accidentelle de y . Ex. $\forall x. \exists y. x + y = 0$ peut subir la substitution $[z/x]$ mais pas la substitution $[y/x]$ car l'occurrence de x dans $x + y$ se trouverait liée par la quantification de y .

Les quantifications logiques ne sont absolument pas les seules à montrer ce comportement. Les sommes, ex. $\sum_{i \in [1, n]} x_i$, les produits, ex. $\prod_{j \in [1, n]} x_j$, les intégrales, ex. $\int f(x). dx$, les dérivées, ex. $\partial f(y) / \partial y$, les notations d'ensemble en compréhension, ex. $\{2 \times n \mid n \in \mathbb{N}\}$, ont toutes le même type de comportement. Ce sont toutes des quantifications qui lient respectivement i, j, x, y et n . Les déclarations des paramètres de fonction dans les langages de programmation se comportent elles-aussi comme des quantifications ; les déclarations tout court, les boucles FOR aussi. Dans tous les cas, le sens d'une variable de programme est déterminé par **qui la quantifie**. Les requêtes SQL aussi montrent ces phénomènes, et ça se manifeste de façon critique dans la construction des **requêtes imbriquées**.

Toutes les quantifications introduisent des noms, les variables sont en fait des noms de quantification, et on peut les renommer en leur appliquant une substitution à condition de ne pas utiliser le nom d'une quantification qui englobe les occurrences de la variable à renommer. En appliquant une substitution $[E/x]$ qui généralise une substitution $[y/x]$ dans un contexte K , soit $K_{[E/x]}$, aucune occurrence de variable libre dans E ne doit être liée accidentellement par une quantification qui englobe une occurrence de x dans K . On ne peut pas changer les noms des occurrences de variables libres de E car ils référencent des quantifications qu'on ne voit pas. Par contre, on peut toujours changer les noms des quantifications englobant x dans K .

Cela donne l'idée de scinder une quantification en d'une part son rôle strictement syntaxique, le même pour toutes les quantifications imaginables, et son rôle sémantique, très différent d'une quantification à une autre. Il faut disposer pour cela d'une quantification neutre, qu'on pourrait noter λ , comme un **individu** λ , et de symboles distincts pour chaque sorte de quantification. On écrirait alors $\forall(\lambda x. \phi)$, $\exists(\lambda x. \phi)$, $\int(\lambda x. f(x))$, $\partial(\lambda y. f(y))$ et $\sum(\lambda i. (x_i \mid i \in [1, n]))$. C'est d'ailleurs ce qu'on ferait en Python si on n'y disposait pas déjà de listes en compréhension, ex. `2*x for x in range(10)` ; on écrirait `list(map(lambda x: 2*x, range(10)))`.

Conclusion : les quantifications sont beaucoup plus variées que les seules quantifications du calcul des prédicats, mais présentent toutes un aspect syntaxique commun, celui de lier des variables à des contextes nommés. Les noms des contextes n'importent pas, du moment qu'on ne crée pas de confusion entre contextes. On peut donc imaginer une quantification neutre, $\lambda x. \dots$, sans sémantique particulière si ce n'est de créer un contexte, avec pour seule opération la possibilité de substituer quelque chose aux variables liées, $[E/x]$, en respectant les contraintes des contextes.

3 Le paradoxe de Russell

Gottlob Frege publie en 1879 (puis 1884, 1893 et 1903) la première présentation d'une logique avec **quantificateurs**. Il propose au passage une première **théorie des ensembles** qui fondent les quantifications. Mais cet édifice est sapé par Bertrand Russell qui présente un paradoxe qui rend cette logique **inconsistante**.

Le paradoxe repose sur la possibilité d'un **ensemble de tous les ensembles** et de caractériser des ensembles d'ensembles en **compréhension**. L'ensemble de tous les ensembles est forcément un élément de lui-même. Mais tous les ensembles ne sont pas éléments d'eux-mêmes. Ex. l'ensemble des entiers n'est pas un entier. Le prédicat « s'appartenir » partitionne donc l'ensemble de tous les ensembles en deux sous-ensembles. Question : que dire du sous-ensemble des ensembles qui ne s'appartiennent pas ? S'appartient-il, ou non ?

*Si il s'appartient, il n'est pas un ensemble qui ne s'appartient pas, donc il ne s'appartient pas. Si il ne s'appartient pas, il appartient à l'ensemble des ensembles qui ne s'appartiennent pas, donc à lui-même. D'où une **inconsistance**.*

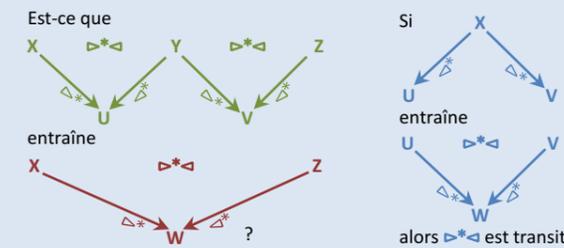
On perçoit facilement que le paradoxe vient de la trop grande facilité avec laquelle on a formé des ensembles d'ensembles en utilisant le **principe de compréhension**. Ce principe dit qu'il suffit qu'un prédicat soit défini sur un ensemble pour que cela caractérise deux sous-ensembles, celui des éléments pour qui le prédicat est vrai, et les autres. D'où de nombreuses tentatives de restreindre cette faculté, en particulier en stratifiant le langage des ensembles, des ensembles d'ensembles, etc.

2 Réduction, passage au contexte, équivalence, confluence, convergence

Considérons \mathcal{A} un langage d'arbres finis, soit sous forme graphique, soit sous forme textuelle non-ambiguë, ex. le langage des **expressions arithmétiques**. On appellera **schéma d'arbre** une notation d'arbre comportant des **méta-variables** qui représentent n'importe quel arbre qu'il serait légal de placer là. Les arbres représentés par un schéma sont appelés les **instances** du schéma d'arbre. Ex. $A + B$ est le schéma des expressions arithmétiques qui sont des sommes, et $2 + 3$ est une instance de ce schéma. Dans un schéma les multiples occurrences d'une même méta-variable sont significatives de ce que toutes ces occurrences représentent strictement le même arbre (au contraire des multiples occurrences d'un même non-terminal dans une règle de grammaire algébrique qui n'indiquent pas autant d'arbres identiques, mais autant d'arbres de même catégorie syntaxique). Ex. $A + A$ est le schéma des expressions arithmétiques qui sont des sommes d'une expression et d'elle-même. Par extension, un arbre est un schéma d'arbre qui ne comporte pas de méta-variable. On appellera **contexte** d'une ou plusieurs occurrences dans un arbre cet arbre où les **positions** de ces occurrences ont été relevées. On notera $K_{[A]}$ un contexte qui contient une ou plusieurs occurrences de A où A peut être un schéma d'arbre : ex. $K_{[A+B]}$ serait le contexte des sommes dans une expression. Cette notation est ambiguë, ex. si il a des sommes dans des sommes, et on devra préciser le cas échéant les occurrences qui sont visées. On notera $K_{[B/A]}$ le contexte $K_{[A]}$ où on a remplacé les occurrences de A par autant de B .

On définit des **règles de réduction**, c-à-d. des règles qui expriment le remplacement d'un arbre, le **rédex**, par un autre, le **réduit**. Une règle de réduction sera notée **schéma de redex** \triangleright **schéma de réduit** où toutes les méta-variables du schéma de réduit doivent être des méta-variables du schéma de redex ; c'est donc en fait un schéma de règle. Comme pour les schémas d'arbre, les multiples occurrences d'une même méta-variable dans un schéma de règle représente strictement le même arbre, ex. $A + A \triangleright 2 \times A$ qui exprime une sorte de factorisation. On peut assortir une règle de réduction de conditions annexes pour exprimer des contraintes d'application, ex. $A + B \triangleright B + A$, si $\|A\| > \|B\|$ où $\|\cdot\|$ représente le nombre de nœuds d'un arbre.

Finalement, on définit une **relation de réduction** entre 2 arbres X et Y , notée $X \triangleright Y$, par **passage au contexte** (on dit aussi **par congruence**) : $X \triangleright Y$ si $X = K_{[U]}$ et $Y = K_{[V]}$ et $U \triangleright V$ est une instance d'une règle de réduction et U représente une occurrence unique dans $K_{[U]}$. Ex. (1) $(5 + 5) + (5 + 5) \triangleright (5 + 5) + (2 \times 5) \triangleright (2 \times 5) + (2 \times 5) \triangleright 2 \times (2 \times 5)$, et (2) $(5 + 5) + (5 + 5) \triangleright 2 \times (5 + 5) \triangleright 2 \times (2 \times 5)$. On note \triangleright^* un enchaînement de réductions, y compris aucune (**fermeture réflexive-transitive**), ex. $(5 + 5) + (5 + 5) \triangleright^* 2 \times (2 \times 5)$ et $2 \times 5 \triangleright^* 2 \times 5$. Si il y a plusieurs règles de réduction, et en l'absence de précision, \triangleright désigne l'une d'entre elles.



On définit ensuite une relation qu'on voudrait être d'équivalence : $X \triangleright^* \triangleleft Y$ si il existe U tel que $X \triangleright^* U$ et $Y \triangleright^* U$. Pour que ce soit une relation d'équivalence, il faudrait que ce soit une relation transitive, et cette définition ne le garantit pas. Ex. on peut avoir $X \triangleright^* U$ et $Y \triangleright^* U$, et $Y \triangleright^* V$ et $Z \triangleright^* V$, donc $X \triangleright^* \triangleleft Y$ et $Y \triangleright^* \triangleleft Z$, mais si on n'a pas $U \triangleright^* \triangleleft V$ on risque de ne pas avoir $X \triangleright^* \triangleleft Z$. On appelle **confluence** la propriété que si $X \triangleright^* U$ et $X \triangleright^* V$ alors $U \triangleright^* \triangleleft V$. La confluence est dite **faible** (on dit parfois locale) si les \triangleright^* partant de X sont remplacés par des \triangleright . Cette propriété dépend du système de règles de réduction adopté, et elle est suffisante pour que $\triangleright^* \triangleleft$ soit une relation d'équivalence, et on la notera alors \equiv . On aurait pu construire une relation d'équivalence par fermeture **réflexive-transitive-symétrique** de \triangleright , mais des règles de réduction comme $0 \times A \triangleright 0$ ne se prêtent pas à être lues symétriquement car toutes les méta-variables de ce que serait le réduit ne sont pas des méta-variables de ce que serait le redex. En fait, la littérature offre beaucoup de variations sur ce thème : (1) définir une relation d'équivalence puis se demander si une relation de co-convergence la représente bien, (2) définir une relation de co-convergence puis se demander si elle est une relation d'équivalence, et de nombreuses variantes entre les deux.

On dit d'un arbre qui ne contient pas de redex qu'il est **irréductible** ou **normal**, et on se demande alors si toutes les réductions convergent vers des arbres normaux ; c'est la propriété de **normalisation forte**. Cela dépend encore du système de règles de réduction. Ex. la règle de réduction $A + B \triangleright B + A$ sans condition annexe permet d'entretenir indéfiniment une réduction qui ne converge pas. On peut alors se demander si pour chaque arbre au moins une réduction converge, c'est la **normalisation faible**. Le fait que des réductions convergent et d'autres non mène à se demander si des stratégies permettent de toujours trouver une convergence si elle existe, et à quel prix.

Il est courant de donner des noms aux règles de réduction et aux systèmes qu'elles forment. On pourra alors en préfixer les mots **réduction**, **équivalence**, **normal**, etc. et en indiquer les symboles \triangleright , \triangleright^* , \equiv , etc. : ex. ξ -**réduction** pour réduction selon la règle ou le système ξ , ou \equiv_ξ pour équivalence selon la règle ou le système ξ .

4 Gottlob, Bertrand, Haskell, et les autres

Gottlob Frege (1848–1925, Allemagne) : depuis son *Begriffsschrift* (1879), *Die Grundlagen der Arithmetik* (1884), jusqu'à *Die Grundgesetze der Arithmetik* (1893–1903), Frege consacre sa carrière à une première présentation d'un calcul des prédicats, son **idéographie**, et d'une théorie des ensembles.

Bertrand Russel (1872–1970, Royaume-Unis) : on lui doit directement peu de chose, mais il a été un des grands animateurs des travaux de fondation de la logique moderne, ex. dans son travail avec **Alfred North Whitehead**. Il est aussi **Prix Nobel de Littérature** et un **pacifiste**.

Haskell Curry (1900–1982, États-Unis) : co-inventeur, avec **Moses Schönfinkel**, du modèle de calcul des **combinateurs**. Ce calcul peut être entièrement simulé par le λ -calcul, et vice-versa.

Alonzo Church (1903–1995, États-Unis) : inventeur du λ -calcul pour contrer les inconsistance que Russell, et d'autres, avait exhibées dans la théorie des ensembles de Frege, puis découvreur avec Kleene et Rosser de la puissance calculatoire insoupçonnée de ce calcul.

J. Barkley Rosser (1907–1989, États-Unis) : étudiant puis collaborateur de Church à l'occasion des premiers travaux sur le λ -calcul.

Stephen Cole Kleene (1909–1994, États-Unis) : étudiant de Church à l'occasion des premiers travaux sur le λ -calcul. On lui doit des présentations consolidées de beaucoup des systèmes utilisés aujourd'hui en informatique : ex. en **théorie des langages** ou en théorie de la **calculabilité**.

Corrado Böhm (1923–2017, Italie) : grand explorateur des capacités du λ -calcul en programmation, compilation, etc.

Henk Barendregt (1947, Pays-Bas) : auteur d'une présentation consolidée du λ -calcul qui est une bible du domaine, ainsi que d'une présentation harmonisée des systèmes de λ -calcul typés qui convergent vers le calcul des constructions : le **λ -cube**, ou **Cube de Barendregt**.