Histoire de λ-calcul

Qu'est-ce que c'est un λ-calcul ? Les codeurs sont familiers avec une notion de λ-fonction - une fonction anonyme qui est utilisée dans les morceaux du code qui ne méritent pas d'avoir un méthode nommé : clé de trie, les petits transformation dans les requêtes à la sql etc. Cépendant, la notion de λ-fonction a été pris d'une système de calcul aussi puisant que la machine de Turing (est inventé dans les mêmes années 30s). Dans cet article on va discuter l'histoire de son invention pour mieux comprendre le concept.

Plan

1. Crise des fondaments
2. Axiomes de Peano
3. 1ere version de Lambda calculs
4. Theoreme de Goedel
5. Machine de Turing et calculabilité
6. Thèse de Church-Turing
7. Applications

Crise des fondaments

Disons qu'un ensemble est **simple** s'il n'appartient pas à lui-même. Par exemple, l'ensemble des tout les gens est simple, car cet ensemble n'est pas une personne. Ainsi, l'ensemble des tout les ensemble n'est pas simple par son définition. *L'ensemble de Russel* est un ensemble qui contient tout les ensembles simples et rien d'autre. Est-ce qu'un ensemble de Russel est simple ? Si c'est le cas, par construction il contient lui-même. Donc il n'est pas simple. Mais s'il n'est pas simple il doit contenir lui-même, ce que signifie qu'il est simple. *Contradiction.*

Ce paradoxe a été indépendement trouvé par Russel et Zermelo au début de XX siècle. Ce paradoxe a beaucoup d'autre formulations plutôt didactiques : *paradoxe du menteur*, *paradoxe du barbier*, etc. Cépendant dans une version de Russel ce paradoxe n'utilise que les constructions formelles de mathématique. Cela signifie que telles constructions sont contradictoires elles-mêmes : si nous avons prouvé qu'une formule propositionnelle est à la fois vrai et fausse, le même peut avoir lieu pour n'importe quel théorème. Si on ajoute qu'au début de XXème siècle paradoxe de Russel n'a pas été le seul

paradoxe connu, on voit bien qu'est-ce que c'était le *crise de fondaments* en mathématique.

Ce crise a été refleté sous le numéro 2 dans une liste des 23 fameux problèmes de Hilbert détérminants le développement du mathématique en XXème siècle.

Problème (2ème problème de Hilbert)**.** *Déterminer la consistence de l'arithmétique.*

Même si l'enoncé est simple, on peut dire que pour résoudre ce problème, il faut passer par les étapes suivants.

- Formuler les axiomes de l'arithmétique - i.e., trouver les proposition "minimales" telles que on peut en déduire tout ce que l'on connait jusqu'qu présent.
- Prouver que en partant de ces axiomes, il n'existe pas d'une proposition X, tel que les axiome implique X ainsi que "non X".

Axiomes de Peano

Le résultat ? 1) Aux années 1889, Peano a proposé les axiomes de l'arithmétique les plus connues : ...

1ère version de Lambda calcul

2) Vers les années 1932 Church a proposé une autre construction qui a été appelé lambda-calcul. Malheureusement, son étudiant, Kleene a prouvé que cette construction n'a pas été consistente.

λ-calcul a formalisé une application d'une fonction. L'écriture envisage la compréhension d'une fonction comme une "règle". Et l'écriture classique $f(x)$ pointe plutôt sur le résultat de ce règle.

Théorème de Gödel

3) Le résultat le plus connu, a été prouvé par Kurt Gödel : il a prouvé que la consistence d'une système d'axiomes ne peut pas être prouvée en n'utilisent que ces axiomes. Donc pour prouver la consistence d'arithmétique il faut ajouter les axiomes supplémentaires (qui a été vite fait, en 1936). Le seul problème est que maintenant il faut prouver une autre système...

Pour résumer le sujet de l'arithmétique, disons que lambda-calcul a été l'un des modèles qui pourrait formaliser les axiomes de l'arithmétique. Son version actuel a été prouvée consiétente et publiée en 1936. Cette construction devait rester un sujet purement théorique qui a intéressé les rares genies

de mathématique qui a étudié ses fondaments (on rappel que beaucoup des personnages de cet article se sont finis mal...).

Machine de Turing et calculabilité

Cépéndant, comme il est souvent en science, il faudrait étudier le même domaine de point de vue un peu différent. Cela a été fait sur l'autre continent par un jeun étudiant Alan Turing. Il a cherché une solution pour une problème de la décision posé en 1928 par Hilbert et Ackermann : "trouver un algorithme qui détermine dans un temps fini, s'il un énoncé est vrai ou faux". La formalisation d'un terme algorithme a conduit au concept de machine de Turing connu par tout le monde. Entre outre, le théorème de Gödel a été reformuler en thermes d'une machine de Turing. Le résultat a été aussi négative, connu comme une théorème de Turing-Church: "il existe les énoncés pour lesquels on ne peux pas déterminer" (vérifier l'enoncé et le nom d'un théorème).

Thèse de Church-Turing

Le résultat positif. S'il existe les fonctions, qu'il peuvent pas être décidées, on se pose la question, qu'est-ce que ce sont les fonction simple, i.e. les fonctions que l'on peut effectivement calculer. Intuitivement, c'est dont la valeur peut être calculée avec un crayon si on a suffisament de papier et du temps. Mais vous comprenez déjà que les mathématiciens n'acceptent pas les solutions intuitives... Le problème de décision est lié avec une problème de calculabilité. Qu'est-ce que signifie qu'une fonction peut être calculée ? Souvent on se refère sur "des méthodes d'un crayon et de papier". Indépendement, chaque des deux a proposé que toute fonction calculable en thèrmes de crayon et papier peut être calculé par son méthode (lambda-calcul ou la machine de Turing). Les deux propositions - ne sont pas les théorèmes, peuvent pas être prouvées car on ne peut pas formaliser autrement calculabilité. (On doit remarquer ici qu'il y avait le troisième mechanisme de détérminer la calculabilité - les fonction récursive primitive). Relativement vite il a été prouvé que tout les 3 méchanismes sont équivalent. Donc, n'importe lequel peut être utilisé comme une définition de function effectivement calculable.

Impacts de λ-calculs

1. Formalisation d'une notion de calculabilité.

2. Preuves de calculabilité.

3. Preuves formelles.

4. Programmation fonctionnelle.