FONDEMENT DES SYSTÈMES INFORMATIQUES

Complice

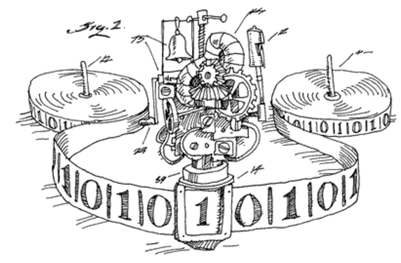
Complexité implicite, concurrence et extraction

Programme Blanc, édition 2008
Partenaires : LIP (Lyon), LIPN (Paris nord), LORIA (Nancy)

Agence Nationale de la Recherche
ANR

Programmes, fonctions et complexité

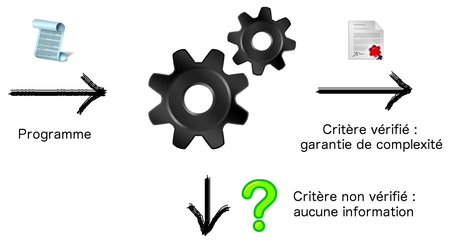
– Si $\mathbf{S}$ est le type des mots binaires, la complexité (en temps) d'un programme $P : \mathbf{S} \to \mathbf{S}$ est définie comme le temps nécessaire pour calculer $P(x)$, en fonction de la *taille* de x.
– Exemples : les programmes de complexité polynomiale (intérêt pratique) ; les programmes de complexité élémentaire, *i.e.*, de la forme $2^{2^{\cdot^{\cdot^{2^n}}}}$ (intérêt théorique).
– On peut ainsi classifier les fonctions (sur $\{0,1\}^*$) selon la complexité du programme le plus efficace qui les calcule. Par exemple, **FP** (resp. **FELEM**) est la classe des fonctions admettant un programme polynomial (resp. élémentaire).

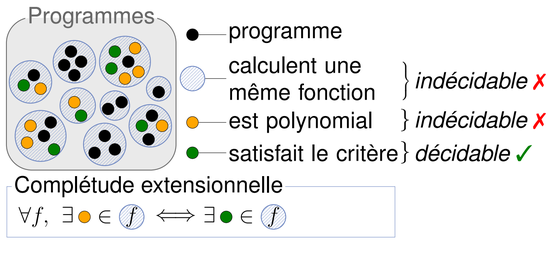


La complexité implicite

Le but de la complexité implicite est de caractériser la complexité des programmes à travers des propriétés structurelles, qui peuvent être analysées/garanties au moment de la compilation.

Ces propriétés structurelles fournissent un critère pour certifier qu'un programme est correct par rapport à une certaine classe de complexité. Ce critère ne doit pas être trop strict : il faut que toute fonction dans la classe de complexité considérée soit calculable par au moins un programme satisfaisant le critère (complétude extensionnelle).





Caractériser explicitement tous les programmes d'une complexité donnée, par exemple polynomiale, est un problème indécidable, c'est à dire qui ne peut être résolu par un programme (cela contredirait le célèbre résultat de A. Turing d'indécidabilité de l'arrêt). Ici nous caractérisons uniquement certains programmes, suffisamment bien distribués pour assurer la complétude extentionnelle. Ainsi, pour chaque programme de complexité polynomiale, nous caractérisons un programme qui lui est équivalent du point de vue de la fonction calculée (et qui calcule également en temps polynomial).

Exemple : quasi-interprétations

Programme fonctionnel (CAML). Par exemple, tri par insertion :

```
insert(a,[]) -> [a]
insert(a,b::l) -> if a < b then a::b::l else b::insert(a,l)
sort([]) -> []
sort(a::l) -> insert(a,sort(l))
```

Quasi-interprétation : association d'un polynôme $(\!|\mathtt{f}|\!)$ à chaque symbole de fonction `f` donnant une borne sur la taille des valeurs. Par exemple :

$$(\!|\mathtt{insert}|\!)(x,y) = x + y + 1$$
$$(\!|\mathtt{sort}|\!)(x) = x$$

Garantie de complexité obtenue en combinant avec un certificat de terminaison :

quasi-interprétation + **certificat de terminaison** (par exemple obtenu par un ordre récursif sur les chemins (RPO)) =

borne sur le temps d'exécution du programme

G. Bonfante, J.-Y. Marion, J.-Y. Moyen. Quasi-interpretations a way to control resources. *Theoretical Computer Science*, 412(25) :2776–2796, 2011.
P. Baillot, U. Dal Lago, J.-Y. Moyen. On quasi-interpretations, blind abstractions and implicit complexity. *Mathematical Structure in Computer Science*, 22(4) :549–580, 2012.

Exemple : la logique linéaire à niveaux

Correspondance de Curry-Howard : les formules sont des types et les démonstrations des programmes.
Logique linéaire : introduit notamment le type $\mathbf{S} \multimap \mathbf{S}$ des programmes utilisant leur entrée *exactement une fois*.

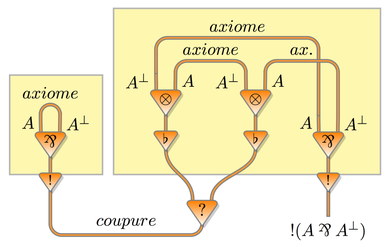
$$\mathbf{S} \to \mathbf{S} \quad = \quad !\mathbf{S} \multimap \mathbf{S}.$$

Stratification : on ajoute une modalité $\S(-)$ induisant une stratification par contraintes *d'équilibre*.

$$\dfrac{\dfrac{\dfrac{}{A \vdash A}}{\S A \vdash A}}{\S A \vdash \S A}$$
preuve équilibrée ✓

$$\dfrac{\dfrac{}{A \vdash A}}{\S A \vdash A}$$
preuve non équilibrée ✗

Caractérisation de classes de complexité : sur les réseaux de preuve, les contraintes de stratification se formulent comme des propriétés de graphes.



P. Baillot, D. Mazza. Linear logic by levels and bounded time complexity. *Theoretical Computer Science*, 411(2) :470–503, 2010.
P. Boudes, D. Mazza, L. Tortora de Falco. An abstract approach to stratification in linear logic. *Information and Computation*. À paraître.

Autres contributions du projet

Extensions à d'autres paradigmes de programmation :
– langage de *threads* fonctionnels concurrents,
– langage impératif.

Extraction de programmes à partir de preuves formelles.

Caractérisation d'autres classes de complexité : **NC**, **L**, **NL**.

Méthodes sémantiques : sémantique dénotationnelle, réalisabilité, jeux.

Un nouveau *workshop* international

DICE Developments in Implicit Computational Complexity
organisé annuellement depuis 2010 en satellite de la multiconférence
ETAPS
EUROPEAN JOINT CONFERENCE ON THEORY & PRACTICE OF SOFTWARE

4e édition, à Rome les 16 et 17 mars 2013

`http://dice2013.di.unito.it/`

Contact : Patrick Baillot <patrick.baillot@ens-lyon.fr>
Site web : http://lipn.univ-paris13.fr/complice/

cnrs advancing on frontiers — LIP — LIPN — Loria