Complexité descriptive et classes de graphes

Thibaut Balabonski

9 mai 2007

Ce mini-mémoire va introduire une notion de complexité indépendante des modèles de calcul, appelée "complexité descriptive", et analyser les liens avec les graphes de deux grandes classes de cette échelle de complexité.

On montrera dans un premier temps le théorème de Fagin qui, il y a trente-trois ans, a fondé la complexité descriptive en affirmant un lien fort entre la complexité classique à base de machines de Turing et la description d'objets par des formules logiques. S'ensuivra une nouvelle définition de la NP-complétude, plus restrictive mais n'évinçant pas nos problèmes naturels et classiques.

La seconde voie abordée sera une explication un peu détaillée d'un résultat auquel le cours a fait allusion, et montrera que dans une certaine classe descriptive (qu'on sait très large dans les graphes), une largeur arborescente bornée donne accès à des algorithmes linéaires. Cette explication sera l'occasion également de montrer comment la théorie des jeux permet de saisir la puissance qu'ont (ou n'ont pas) les formules logiques.

1 Logique et structures relationnelles

Cette section d'introduction définit les notions sur lesquelles se base la complexité descriptive. Il s'agira de définir formellement la question : "un graphe satisfait-il une propriété donnée ?". Cette propriété sera présentée comme une formule logique plus ou moins puissante, selon le nombre et le type des quantifications qu'elle utilise. Et c'est cette puissance qui caractérisera la "complexité" d'un problème.

Tout ceci s'applique dans le cadre plus général où l'on remplace "graphe" par "structure relationnelle". On fera régulièrement l'aller-retour entre les deux, et on finira par montrer que le pouvoir représentatif des graphes peut finalement contenir toutes ces structures.

Les documents de référence pour cette partie sont [3] et [4].

1.1 Langages relationnels et structures

Langages
Un **langage** va définir nos moyens élémentaires de description. On définit un langage relationnel (parfois appelé "signature") par :

- un ensemble de **constantes**
- un ensemble de **relations**

Chaque relation a une arité (*ie* un nombre d'arguments attendus) fixée. Elles peuvent éventuellement être typées. On utilisera également des variables pour former nos phrases, et on aura toujours une relation $=$ exprimant l'égalité de deux éléments (constantes ou variables du premier ordre).

Structures et interprétations
Une **structure** va être la formalisation (dans le langage précédemment choisi) d'un objet étudié, et est constituée de deux éléments :

- Un ensemble, appelé **domaine**, dont les éléments sont éventuellement typés.
- Une fonction d'**interprétation** des constantes et relations du langage dans le domaine.

Décrire les graphes
Prenons dès maintenant comme exemple deux langages permettant de décrire les graphes, et explicitons les interprétations associées. Dans tous les cas on se limitera à des graphes simples sans boucles (on n'a pas encore le pouvoir de l'imposer, mais ce ne sera pas difficile)

Une première description d'un graphe est "un ensemble de sommets dont certains sont reliés par une arête". Ce point de vue est traduit par le langage τ_1 constitué d'une unique relation binaire E. Le domaine représente alors les sommets et E est interprétée comme relation d'adjacence entre les sommets.

Une seconde description est "un ensemble de sommets et un ensemble d'arêtes, les arêtes étant incidentes à des sommets". On utilise ici un second langage τ_2, contenant deux relations binaires I_{cible} et I_{source}. Les éléments du domaine sont de deux types : "sommet" ou "arête". Les relations I sont typées : elles s'appliquent à un sommet et une arête, et sont interprétées comme relations d'incidence entrante et sortante. (Dans le cas non dirigé on utilise une unique relation I) Dans un souci de commodité, on prendra deux relations unaires S et A donnant le type "sommet" ou "arête".

Description d'un autre problème : SAT
Regardons un exemple supplémentaire pour décrire un problème qui n'a rien à voir avec les graphes : SAT. Une instance de SAT est "un ensemble de

clauses faisant intervenir un ensemble de variables, positivement ou négativement".

Notre traduction va être alors immédiate : on regardera des structures avec deux types déléments : des variables et des clauses. On aura dans notre langage deux relations binaires typées P et N, prenant en argument une variable et une clause. Elles dénoteront respectivement les apparitions positive et négative de la variable dans la clause.

On remarquera que ce langage est complètement isomorphe à τ_2, ils ne diffèrent que par les noms des types d'éléments et des relations. Mais bientôt ils seront interprêtés différemment (notamment on donnera des conditions de bonne formation aux graphes).

1.2 Formules logiques et satisfaction

Il faut maintenant pouvoir assembler les briques de base formées par un langage afin de former des phrases exprimant des propriétés. Cet assemblage sera purement syntaxique, mais son interprétation sera immédiate.

Formules atomiques et connecteurs booléens
On appelle **formule atomique** une brique de base : il s'agit de l'application d'un symbole de relation à des éléments (variables du premier ordre ou constantes).

On combine ces briques à l'aide des connecteurs booléens classiques pour obtenir un premier ensemble de formules, dites **formules sans quanteurs**. Définition par induction de cet ensemble :

- Une formule atomique est une formule.
- Si ϕ est une formule, alors $\neg\phi$ est une formule.
- Si ϕ et ψ sont des formules, alors $\phi \vee \psi$ et $\phi \wedge \psi$ sont des formules.

On utilisera aussi le connecteur $\rightarrow$ pour l'implication.

Quantifications des premier et second ordres
On a déjà parlé de variables du premier ordre, qui représentent des éléments du domaine. On les notera traditionnellement x,y,z... On peut lier ces variables à des quantificateurs, et on obtient les **formules du premier ordre** en ajoutant cette condition à la définition précédente (où x représente une variable du premier ordre) :

- Si ϕ est une formule, alors $\forall x\phi$ et $\exists x\phi$ sont des formules.

Les variables du second ordre ont pour l'instant été passées sous silence. Ces variables représentent des relations et sont couramment notées X,Y,Z... Chacune de ces variables a une arité !

On obtient les **formules du second ordre** d'abord en étendant la notion de formule atomique à "l'application d'un symbole de relation ou d'une

variable du second ordre à des éléments", puis la condition suivante à la définition par induction (où i est un entier et X une variable du second ordre d'arité i) :

– Si ϕ est une formule, alors $\forall^i X\phi$ et $\exists^i X\phi$ sont des formules.

Satisfaction et définition de classes

Après toutes ces définitions syntaxiques, reste à décider de quand une structure G donnée satisfait une formule ϕ donnée elle aussi, simplement en disant que tous les symboles introduits jusqu'ici ont bien la signification qu'on leur imagine. On note cette satisfaction $G \models \phi$, et on dit aussi que G est un modèle de ϕ.

Une formule définit alors une classe de structures : l'ensemble des structures qui la satisfont.

Graphes simples sans boucles

On a maintenant le pouvoir de vérifier que nos graphes sont bien simples et sans boucles.

Dans le langage τ_1, il suffit de vérifier "sans boucle", la première condition étant toujours remplie.

– $\forall x\ \neg E(x,x)$.

Dans le langage τ_2, les deux points sont à vérifier, une fois que nous serons assurés que le graphe est bien formé (chaque arête a exactement une source et une cible).

– $\forall_{arete} x\ \exists_{sommet} y\ I_{source}(y,x)$
– $\forall_{arete} x\ \forall_{sommet} y \forall_{sommet} z\ (I_{source}(y,x) \wedge I_{source}(z,x) \rightarrow y = z)$
– $\forall_{arete} x\ \exists_{sommet} y\ I_{cible}(y,x)$
– $\forall_{arete} x\ \forall_{sommet} y \forall_{sommet} z\ (I_{cible}(y,x) \wedge I_{cible}(z,x) \rightarrow y = z)$
– $\forall_{sommet} x\ \neg\exists_{arete} y\ I_{source}(x,y) \wedge I_{cible}(x,y)$
– $\forall_{arete} x \forall_{arete} y\ (\exists_{sommet} u \exists_{sommet} v\ I_{source}(u,x) \wedge I_{source}(u,y) \wedge I_{cible}(v,x) \wedge I_{cible}(v,y)) \rightarrow x = y$

SAT

L'ensemble des instances satisfiables de SAT est défini par une formule du second ordre que l'on donne ici :

$$\exists^1 A\ \forall_{clause} c\ \exists_{variable} x\ (A(x) \wedge P(x,c)) \vee (\neg A(x) \wedge N(x,c))$$

Il faut la lire ainsi : il existe une assignation des variables A telle que dans chaque clause apparaît un littéral interprêté à vrai (une variable apparaissant positivement à qui on a assigné la valeur "vrai" ou une variable apparaissant négativement et à qui on a assigné la valeur "faux").

Graphes 3-colorables
Voici un dernier exemple pour synthétiser toute cette partie : une formule du second ordre définissant la classe des graphes 3-colorables dans le langage τ_1.

$$\exists^1 R\ \exists^1 V\ \exists^1 B$$
$$(\forall s\ R(s) \vee V(s) \vee B(s))$$
$$\wedge$$
$$(\forall s\ \forall t\ E(s,t) \rightarrow \neg((R(s) \wedge R(t)) \vee (V(s) \wedge V(t)) \vee (B(s) \wedge B(t))))$$

On l'interprête ainsi : existent trois couleurs R, V et B, représentées par des relations unaires (ce qui équivaut à des ensembles, chaque relation définissant donc l'ensemble des sommets possédant ladite couleur), telles que :

- Chaque élément a (au moins) une couleur.
- Deux éléments adjacents n'admettent pas la même couleur.

1.3 Réductions du premier ordre

On va ici définir une classe de fonctions entre deux structures, au très faible pouvoir de calcul. Ces fonctions de réductions seront basées sur des formules du premier ordre, et sont dans la hiérarchie Turing strictement incluses dans LOGSPACE (et donc dans P).

On a une structure de départ G de domaine D sur un langage donné $L = \langle c_1, \ldots, c_s;\ R_1, \ldots, R_t \rangle$. On va aboutir à une nouvelle structure G' de domaine D' inclus dans D^m pour un certain entier m, dans un nouveau langage $L' = \langle d_1, \ldots, d_u;\ Q_1, \ldots, Q_v \rangle$

Parties définissables
Le domaine de G' va être une **partie définissable du premier ordre** de D^m. Cela signifie qu'il existe une formule du premier ordre $\phi(x_1, \cdots, x_m)$ sur L, qui est vérifiée dans G par l'uplet $(a_1, \cdots, a_m)$ si et seulement si ce dernier est dans D'.

Réduction des relations
On va de même définir les interprétations des relations de L' à partir des constituants de L, en utilisant des formules du premier ordre. Ainsi pour chaque Q_i n-aire de L' on a une formule du premier ordre $\phi_i(x_{1,1}, \cdots, x_{n,m})$ vérifiée par $a_{1,1}, \cdots, a_{n,m}$ si et seulement si $Q_i((a_{1,1}, \cdots, a_{1,m}), \cdots, (a_{n,1}, \cdots, a_{n,m}))$ est interprétée à vrai.

Réduction des constantes, ou ce que je laisserais bien en exercice
Il faut de la même façon une formule du premier ordre par constante de L' !

Réductions du premier ordre entre τ_1 et τ_2
Appliquons ceci pour aller d'une représentation à l'autre. De τ_2 à τ_1 pour commencer.

Il faut d'abord choisir un m puis définir le sous-ensemble de D^m qu'on conserve avec une formule ϕ, et ensuite définir la relation d'adjacence E avec une seconde formule ψ.

- On prend $m = 1$, et $\phi(x) \equiv S(x)$ (on garde les éléments de type "sommet").
- On retrouve E avec $\psi(x, y) \equiv \exists z\ A(z) \wedge I_{source}(x, z) \wedge I_{cible}(y, z)$ (on a un arc de x à y s'il existe un arc ayant x pour source et y pour cible)

Pour aller de τ_1 à τ_2, il va cette fois falloir agrandir le domaine ($m = 2$ fonctionnera). Ensuite, il faudra retenir tous les couples représentant des sommets et uniquement ceux représentant des arêtes existant, par une formule ϕ. Deux formules ψ_{source} et ψ_{cible} traduiront les relations d'incidence. On retrouvera aussi θ_{sommet} et θ_{arete} pour typer les éléments.

- $\phi(x, y) \equiv (x = y) \vee E(x, y)$
- $\theta_{sommet}(x, y) \equiv x = y$
- $\theta_{arete}(x, y) \equiv \neg x = y$
- $\psi_{source}((x, y), (u, v)) \equiv x = u \quad (\wedge\ x = y \wedge \neg u = v)$
- $\psi_{cible}((x, y), (u, v)) \equiv x = v \quad (\wedge\ x = y \wedge \neg u = v)$

Ici on a représenté le sommet x par le couple (x, x) (graphes sans boucles donc pas de collision avec la suite) et l'arête xy par le couple (x, y).

1.4 Universalité des graphes

On va donner ici l'idée de deux réductions du premier ordre réciproques I_σ et I_σ^{-1}. I_σ va transformer toute structure G sur σ en un graphe. Et ceci en conservant toutes les informations : on aura

$$I_\sigma(I_\sigma^{-1}(G)) \cong G$$

.

La moralité sera : modulo réduction du premier ordre, toutes les structures sont des graphes. Ce résultat est un exercice tiré de [4].

Réduction à un graphe
L'idée va être de transformer chaque élément du langage d'origine en un gadget de graphes. Les éléments du domaine D de G pourront être représentés par des cliques à trois éléments et chaque relation par des cliques d'un nombre d'élément différent.

Pour une relation R n-aire, représentée par la clique à p éléments, pour chaque n-uplet $(a_1, \cdots, a_n)$ de D vérifiant $R(a_1, \cdots, a_n)$ on reliera une nouvelle clique K_p aux n triangles représentant les a_i (on donne à chacun de ces

gadgets un sommet supplémentaire d'entrée pour centraliser tous ces liens et distinguer un point de l'ensemble).

L'écriture de la formule correpondante est assez infâme, car nécessite de nombreux codages, mais le premier ordre suffit.

Réduction inverse
Le passage inverse est plus facile, car on a tout fait avant pour conserver l'information. On garde dans le domaine le point distingué de chaque élément, et on retrouve les relations avec des formules du premier ordre ("il existe une clique, de la bonne taille exactement, reliée exactement aux éléments qu'on veut", chaque "exactement" masquant une quantification universelle du premier ordre).

1.5 Les différentes puissances descriptives

Avec ce qu'on a vu jusqu'ici, on a déjà trois degrés de complexité descriptive :

- Degré zéro sans quanteurs. Sur les graphes on ne fait rien.
- Premier ordre, on obtient la classe AC^0, définie classiquement à partir des circuits booléens de profondeur bornée. C'est strictement inclus dans LOGSPACE et donc dans P.
- Second ordre, on obtient toute la hiérarchie polynomiale, ce qui commence à être très puissant car flirtant avec PSPACE.

On va retrouver des échelons intermédiaires en restreignant nos quantifications du second ordre, et ce de deux manières :

- En contrôlant l'alternance des quanteurs existentiels et universels, on contrôle la montée dans la hiérarchie polynomiale (dont les premiers niveaux sont P et NP).
- L'arité des variables du second ordre, est liée au degré du polynôme donnant la complexité classique en temps.

Les deux grandes parties suivantes vont avoir chacune pour thème l'une de ces limitations. On s'attaquera dans chaque cas à la plus forte restriction, c'est-à-dire :

- On obtiendra la logique existentielle du second ordre en interdisant toute alternance des quanteurs du second ordre et en choisissant comme unique quanteur l'existentiel.
- On obtiendra la logique monadique du second ordre en se limitant à des variables du second ordre unaires (on quantifiera donc uniquement sur des relations unaires, *ie* des ensembles).

2 Classe NP et logique existentielle du second ordre

Dans ce premier développement on regardera la première restriction, et on se limitera aux formules de la forme $\exists \bar{X} \psi(\bar{X})$, avec ψ du premier ordre, *ie* aux formules dont les seules quantifications du second ordre sont existentielles et apparaissent en tête.

On verra d'abord le grand résultat de Fagin identifiant cette puissance de description à la classe NP, puis on regardera la complétude en prenant non la réduction polynomiale habituelle, mais la plus faible réduction du premier ordre. Le but sera de prouver la NP-complétude de problèmes de graphes simplement en exhibant quelques formules. Cette partie utilise beaucoup [4].

2.1 Définition descriptive de NP : théorème de Fagin

Théorême de Fagin
La classe NP est égale aux problèmes réductibles à des propriétés de graphes exprimables en logique existentielle du second ordre.

Inclusion simple
Partant d'une formule existentielle, la construction d'une machine non déterministe en temps polynomial décidant si un modèle satisfait la formule est rapide : on "devine" d'abord les relations correspondant aux variables du second ordre (ceci revient à écrire une certaine suite de bits sur le ruban de travail : pour une relation k-aire, on énumère tous les k-uplets et on note 0 ou 1 selon qu'ils satisfont la relation ou non ; le temps et l'espace requis sont la k-ième puissance du cardinal du domaine, ce qui est bien polynomial), puis il reste à évaluer une formule du premier ordre.

Mimer la machine de Turing
Le deuxième sens est plus délicat. Il s'agit, à partir d'une machine de Turing non déterministe en temps polynomial, de construire une formule équivalente. L'idée va être : "il existe un calcul acceptant". Le tout étant alors de modéliser "un calcul" avec des variables du second ordre, et vérifier "le calcul est acceptant" avec une formule du premier ordre.

Regardons un calcul Turing comme un tableau : chaque ligne représente le ruban de la machine à un instant donné. Ces lignes doivent donc être infinies dans l'absolu, mais on va pouvoir se restreindre à un tableau carré fini. En effet, prenons k tel que la machine travaille en temps borné par $n^k - 1$ sur les graphes à n sommets (pour la clarté, on va considérer que la taille de l'entrée est le nombre de sommets, même si le codage utilise plus de caractères).

Le premier constat est que notre tableau aura n^k lignes. De plus, l'espace utilisé par le calcul ne peut en aucun cas être supérieur au temps. Ainsi on

peut se limiter à une fenêtre de largeur n^k sur les rubans. Ce que l'on analyse est donc un tableau carré de côté n^k, où k est un entier connu et n le nombre de sommets du graphe donné en entrée.

On peut alors vérifier qu'un tableau est bien formé avec ces trois conditions :

- La première ligne est le ruban initial : elle représente le graphe donné en entrée et la tête de lecture apparaît une unique fois, sur la première case et dans l'état initial.
- La dernière ligne représente un état acceptant : la tête de lecture y apparaît quelque part dans son état d'acceptation (si on le désire, on peut imposer que la tête soit sur la première case et que le reste du ruban ait été effacé)
- Chaque ligne est bien une descendante possible de la précédente : pour chaque élément de la ligne, on regarde les cases de la ligne précédentes susceptibles de l'affecter (la case correspondant à la même cellule, et ses deux voisines). Si la tête de lecture n'y apparaît pas, la cellule doit rester identique, sinon il faut comparer avec la règle de transition appliquée.

Ici apparaissent deux petits problèmes que les paragraphes suivants vont éclaircir :

- Il va falloir quantifier sur les lignes du tableau et sur les positions dans ces lignes. Hors, nous n'avons le droit de quantifier que sur les sommets du graphe et pas sur des nombres !
- Nous regardons une machine de Turing non déterministe, donc plusieurs choix sont potentiellement possibles à chaque itération, et les conditions précédentes ne suffisent plus à assurer la bonne formation du calcul. Il faut décider à l'avance quel choix est fait à chaque étape. On peut gratuitement considérer que seuls deux choix sont possibles à chaque pas (et même exactement deux choix, en prenant deux règles identiques quand cela est nécessaire), mais une astuce sera quand même nécessaire.

Problème de l'ordre

On veut qu'une suite raisonnable de quantifications du premier ordre nous donne un nombre p entre 0 et $n^k - 1$. La manière classique de faire est de prendre k éléments de notre structure (*ie* k sommets du graphe), correspondant chacun à un nombre entre 0 et $n - 1$, et de les prendre comme décomposition en base n de p.

Ceci revient à ordonner les sommets du graphes. Ici, c'est facile, un ordre sur les sommets n'étant qu'une relation binaire qu'on peut introduire existentiellement, et les conditions assurant que cette relation $\preceq$ est bien un ordre total sont immédiates :

– $\forall x\ x \preceq x$ (relation symétrique)
– $\forall x \forall y\ (x \preceq y \wedge y \preceq x) \rightarrow x = y$ (relation irréflexive)
– $\forall x \forall y \forall z\ (x \preceq y \wedge y \preceq z) \rightarrow x \preceq z$ (relation transitive)
– $\forall x \forall y\ x \preceq y \vee y \preceq x$ (relation totale)

Cependant, sans droit à une quantification du second ordre d'arité deux, cette possibilité s'écroule. On peut y voir un défaut inhérent au modèle des machines de Turing (ce qui serait une justification supplémentaire à la recherche de définitions indépendantes des modèles de calcul, comme la complexité descriptive). En effet, une machine de Turing qui travaille sur un graphe travaille en fait sur sa représentation en tant que chaîne de caractères. Ainsi pour les machines de Turing les sommets des graphes sont ordonnés de fait par l'ordre dans lequel ils apparaissent dans le codage. Il s'ensuit qu'une telle machine peut se comporter de manière complétement différente sur deux codages du même graphe ne différant que par l'ordre de prise en compte de certains sommets ou arêtes.

Sur quoi quantifier
Le dernier point à régler est "il existe un calcul". On doit donc créer un tableau et une liste de bits correspondant aux choix effectués à chaque pas du calcul. On va introduire des relations sur les éléments du tableau. Chaque point du tableau représente soit une case du ruban, soit une case du ruban plus la tête de lecture. En faisant donc le produit de l'alphabet de travail par les états possibles de la tête de lecture (plus l'état "absent"), on a l'ensemble des configurations possibles en un point du tableau. On quantifie alors sur une "coloration" du tableau à la manière de l'exemple de 3-COLORATION montré précédemment, ce qui fait autant de relations $2k$-aires que d'états possibles pour une case. Par rapport à l'exemple de 3-COLORATION, il faut de plus vérifier que chaque case n'est que d'une seule couleur.

Il faut de plus quantifier sur une relation k-aire guidant les choix de calcul. La signification donnée à cette relation est : elle est satisfaite par le k-uplet $(a_1, \cdots, a_k)$ de sommets si et seulement si à la p-ième étape (où p est le nombre correspondant Ã l'uplet, comme ci-dessus) on a fait le premier choix. Cette dernière relation sépare donc les étapes auxquelles la machine a fait le premier choix de celles auxquelles elle a fait le second.

Cette longue construction aboutit finalement à une formule caractérisant une machine de Turing non déterministe. On remarquera que cette formule est de la forme $\exists \bar{X} \psi(\bar{X})$ comme on l'attendait. Mais mieux que cela, les seuls quanteurs du premier ordre utilisés étaient universels. On obtient donc en forme prénexe (*ie* avec tous les quantificateurs en tête, mais attention aux transformations occasionnées par les connecteurs $\neg$!) une formule de la forme : $\exists \bar{X}\ \forall \bar{x}\ \theta(\bar{X}, \bar{x})$ où θ est une formule sans quanteurs. Ce détail aura

son importance dans la preuve de NP-complétude de SAT qui entame la section suivante.

2.2 Réductions du premier ordre et NP-complétude

On va ici montrer que quelques problèmes "naturels" classiquement NP-complets le sont encore en se limitant aux réductions du premier ordre. On suivra l'exemple de Cook et on traitera SAT d'abord, en repartant de la fin de la preuve précédente. On aura ensuite les outils pour faire d'autres preuves de NP-complétude en n'utilisant que des formules logiques.

NP-complétude de SAT
On sait déjà que SAT est dans NP, par une formule donnée plus tôt. On prouve ici la complétude de ce problème.

Soit A un problème de NP. On a vu qu'on pouvait l'exprimer par une formule de la forme $\exists \bar{X}\ \forall \bar{x}\ \theta(\bar{X}, \bar{x})$ avec θ sans quanteurs. On suppose de plus que θ est sous forme normale conjonctive :

$$\theta(\bar{X}, \bar{x}) \equiv \bigwedge_{j=1}^{r} C_j(\bar{X}, \bar{x})$$

où chaque C_j est une disjonction de formules atomiques ou de négations de formules atomiques.

Ces disjonctions C_j vont naturellement devenir les clauses de notre problème SAT. On se rappelle que dans SAT une clause C s'écrit $\bigvee_{i=1}^{k} y_i$ où chaque y_i est une variable ou la négation d'une variable. Les variables de SAT vont donc correspondre à des formules atomiques.

Plus précisément, ici les formules atomiques sont de la forme $R(\bar{x})$ avec R une relation du langage, ou $X_i(\bar{x})$ avec X_i une variable du second ordre. Dans toute structure, on peut directement interprêter les premières et les remplacer par vrai ou faux dans les formules (la réduction travaille à partir d'une structure G, donc ceci est bien fixé !). En revanche, les X_i sont introduites existentiellement dans notre formule. Leurs valeurs en chaque uplet sont donc à définir, et c'est là qu'apparaissent les variables de SAT : chaque $X_i(a_1, \cdots, a_p)$ pour p arité de X_i et a_j élément du domaine de G forme une variable.

Cette construction donne bien à partir d'une structure G une instance G' de SAT, qui est satisfiable si et seulement si $G \models A$. C'est bien une réduction du premier ordre, mais on se passera ici du détail des formules. En revanche, l'exemple suivant montrera à quoi ressemble une réduction complexe et sera traité intégralement.

Chemins hamiltoniens
Le problème regardé ici est : étant donné un graphe, admet-il un chemin hamiltonien ? (c'est-à-dire un chemin passant par chaque sommet une fois et une seule)

On montre d'abord que ce problème est dans NP, en donnant une formule de la logique monadique du second ordre le caractérisant. L'idée est la suivante : "il existe un ordre total sur les sommets, et les sommets pris dans cet ordre forment un chemin". On peut raffiner en "il existe un ordre total sur les sommets, on en déduit une fonction successeur, et chaque sommet est lié à son successeur (sauf le dernier qui n'en possède pas)".

On a déjà vu comment introduire un ordre total. Déduisons-en d'abord une fonction successeur *succ*, qu'on représentera par son graphe *Succ* (*ie* *Succ* est la relation caractérisant les couples antécédent/image par *succ* : $Succ(x, y)$ si et seulement si $y = succ(x)$).

$$\begin{gathered} \exists^2 Succ \\ Succ(x, y) \\ \longleftrightarrow \\ (x \preceq y) \wedge (x \neq y) \wedge (\forall z \ (z \neq x \wedge x \preceq z) \rightarrow y \preceq z) \end{gathered}$$

Ceci exprime que y est le plus petit majorant de x, et la relation *Succ* remplit alors bien son rôle.

Il ne reste qu'à dire que cette relation de succession forme bien un chemin :

$$\forall x \forall y \ Succ(x, y) \rightarrow E(x, y)$$

Soit une instance S de SAT, on va la réduire à une instance équivalente G de CHEMIN_HAMILTONIEN. Le graphe G formé va posséder un sommet par clause de S, et un gadget sera utilisé pour chaque variable.

Le gadget possède $k + 3$ sommets où k est le nombre de clauses de S. Il est constitué d'un sommet d'entrée, d'un sommet de sortie, et de $k + 1$ sommets formant une liste doublement chaînée dont chaque extrémité est liée à l'entrée et à la sortie. Un chemin hamiltonien de G traversera le gadget de la façon suivante : il arrive par l'entrée, puis va à l'une des extrémité de la chaîne, la parcourt et enfin va à la sortie une fois l'autre extrémité de la chaîne atteinte. Il y a un choix binaire à effectuer, qui est le sens de parcours de la chaîne, et qui correspondra au choix de l'assignation de la valeur "vrai" ou "faux" à la variable de S correspondant au gadget.

On relie les sommets (de G) correspondant aux clauses (de S) aux gadgets des variables (de S) apparaissant dans la clause. Si une variable apparaît positivement dans la i-ème clause, on aura une arête allant du i-ème élément

de la chaîne vers le sommet de la clause, puis une repartant de la clause vers le $i+1$-ème élément de la chaîne. Si la variable apparaît négativement, on place les arcs dans l'autre sens. Ainsi, on ne peut intégrer le sommet de la clause à notre chemin que si la chaîne est parcourue dans le bon sens, ceci pour au moins une des variables apparaissant dans la clause. On relie la sortie d'une variable à l'entrée de la variable suivante.

Les chemins hamiltoniens éventuels de G commencent donc à la première entrée et terminent à la dernière sortie, choisissent une orientation à chaque variable, et font un saut vers chaque clause à un moment où l'autre. Ils correspondent exactement à des certificats de satisfiabilité de S.

Essayons maintenant de donner une idée propre de cette réduction un peu compliquée de S (sur l'alphabet $\langle P, N, C, V\rangle$ -apparaît positivement, apparaît négativement, est une clause, est une variable) vers G (sur l'alphabet $\langle E\rangle$ -sont liés par un arc). On se rend compte au passage de quelques artifices qui seront nécessaires :

- Pour que les gadgets soient assemblés en un fil, il faut un ordre sur les variables.
- Exactement de la même manière, pour que les noeuds des gadgets qui correspondent aux clauses forment bien une châine, il faut un autre ordre sur les clauses.
- Pour distinguer certains sommets il va falloir avoir des "constantes" ou plus généralement certains éléments distingués dans le problème SAT. Cela peut se faire grâce aux ordres précédemment introduits.

Reprenons la définition d'une réduction : les sommets de G vont être des m-uplets d'éléments de S (de deux types : clause ou variable). Il va d'abord falloir décider un tel m, puis définir par une formule du premier ordre une partie de S^m (on confond ici comme souvent la structure et son domaine).

Prenons ici $m = 2$ et expliquons comment nous allons interprêter les couples. Dans les cas suivants, c représente une clause, v une variable, et les v_i représentent trois variables distinguées (par exemple : les trois premières dans un ordre qu'on introduit ; on conviendra que nos problèmes comportent au moins trois variables).

- Le couple (c, c) représente le sommet correspondant à la clause c.
- Le couple (v, c) représente le sommet correspondant à c du gadget correspondant à v.
- Le couple (v, v_i) représente l'un des trois autres sommets du gadget associé à v.
- Aucun autre couple ne sera accepté dans le domaine de G.

Ceci décidé, la définition par une formule est immédiate : elle se limite à des tests de type et dégalité :

$$\phi(x, y) \equiv (C(x) \wedge C(y)) \vee (V(x) \wedge C(y)) \vee (V(x) \wedge \bigvee_{i=1}^{3} y = v_i)$$

Il faut maintenant définir la relation d'adjacence E avec une formule du premier ordre utilisant le langage de SAT. On commence par introduire comme précédemment des ordres $\preceq_{clauses}$ et $\preceq_{variables}$, totaux respectivement sur les clauses et sur les variables de S. On en déduit de même les relations $Succ_{clauses}$ et $Succ_{variables}$. On garde la notation $succ$ pour la fonction correspondante pour simplifier l'écriture de l'explication. On note aussi c_min et c_max les première et dernière clauses.

On a alors plusieurs cas dans lesquels il faut mettre un arc entre deux sommets :

- De (v, c) à $(v, succ(c))$ et de $(v, succ(c))$ à (v, c) (chaîne à l'intérieur du gadget associé à v).
  $\psi_1((x, y), (u, t)) \equiv x = u \wedge (Succ_{clauses}(y, t) \vee Succ_{clauses}(t, y))$
- De (v, c_max) à (v, v_2) et inversement (dernier maillon de la chaîne).
  $\psi_2((x, y), (u, t)) \equiv x = u \wedge ((y = c_max \wedge t = v_2) \vee (t = v_2 \wedge y = c_max))$
- De (v, v_1) à (v, c_min) et à (v, v_2) (l'entrée du gadget va aux deux extrémités de la chaîne).
  $\psi_3((x, y), (u, t)) \equiv x = u \wedge y = v_1 \wedge (t = c_min \vee t = v_2)$
- De (v, c_min) et de (v, v_2) à (v, v_3) (des deux extrémités de la chaîne à la sortie du gadget).
  $\psi_4((x, y), (u, t)) \equiv x = u \wedge t = v_3 \wedge (y = c_min \vee y = v_2)$
- De (v, v_3) à $(succ(v), v_1)$ (de la sortie d'un gadget à l'entrée du suivant).
  $\psi_5((x, y), (u, t)) \equiv Succ_{variables}(x, u) \wedge y = v_3 \wedge t = v_1$
- De (v, c) à (c, c) si v apparaît positivement dans c.
  $\psi_6((x, y), (u, t)) \equiv x = u = t \wedge P(x, y)$
- De (c, c) à $(v, succ(c))$ si v apparaît positivement dans c.
  $\psi_7((x, y), (u, t)) \equiv x = y \wedge Succ_{clauses}(x, t) \wedge P(u, y)$
- De $(v, succ(c))$ à (c, c) si v apparaît négativement dans c.
  $\psi_8((x, y), (u, t)) \equiv u = t \wedge Succ_{clauses}(t, y) \wedge N(x, t)$
- De (c, c) à (v, c) si v apparaît négativement dans c.
  $\psi_9((x, y), (u, t)) \equiv x = y = t \wedge N(u, t)$

En prenant la disjonction de tous les ψ_i explicités ici, on obtient la définition de E !* Preuve est maintenant faite que le problème de l'existence d'un chemin hamiltonien est encore NP-complet via des réductions plus faibles que P en général, et ce sans plus aucun appel aux machines de Turing, fût-il implicite.

*Pour être complètement certain du résultat, il faudrait ajouter dans chaque cas le test du type de chaque variable, qui a été omis ici. Un cas a aussi été à moitié omis ici : celui des variables apparaissant dans la dernière clause, pour lesquelles on n'a introduit qu'un des deux arcs voulus. On rend ceci correct en considérant que $succ(c_max) = v_2$.

3 Largeur arborescente et logique monadique du second ordre

Nous allons maintenant regarder la deuxième manière de restreindre la logique du second ordre : nous allons dans la suite limiter l'arité des variables du second ordre.

On peut d'abord citer un théorème de Lynch qui donne une inclusion de classes de complexité :

$$DTIME(n^k) \subseteq SOL^k$$

où SOL^k désigne la restriction à des variables du second ordre d'arité k. L'inclusion inverse est quant à elle un problème ouvert.

Cette partie va se limiter à la logique monadique du second ordre (MSOL), *ie* aux variables du second ordre unaires. En substance, on ne saura quantifier que sur des éléments ou des ensembles déléments, mais plus sur des relations plus compliquées. En particulier on ne pourra plus introduire de relation d'ordre quand cela viendra à nous manquer.

Dans un premier temps on comparera l'expressivité de cette logique sur nos deux représentations des graphes. Ensuite, après rappel de quelques faits sur les deux paramètres de graphes que sont largeur arborescente et largeur de clique, on regardera comment logique monadique du second ordre et largeur bornée assemblées permettent de construire des algorithmes de graphes linéaires.

3.1 Expressivités de MS_1 et MS_2

On note MS_i la classe des propriétés de graphes exprimables en logique monadique du second ordre à l'aide du langage τ_i, pour $i \in \{1, 2\}$.

On a déjà vu que la 3-colorabilité était dans MS_1. On sait aussi transformer une formule de MS_1 en une formule de MS_2, simplement en remplaçant chaque $E(x, y)$ en $\exists z\ I(x, z) \wedge I(y, z)$. (dans l'autre sens, on a une réduction du premier ordre, ce qui n'est pas une traduction directe)

On va en revanche montrer que l'inclusion de MS_1 dans MS_2 est stricte. Prenons comme exemple les couplages parfaits, et exhibons une formule MS_2 définissant les graphes admettant un couplage parfait :

$$\exists_{aretes} E$$
$$\forall_{sommet} x \exists_{arete} e\ (E(e) \wedge I(x, e))$$
$$\wedge$$
$$\forall_{arete} e \forall_{arete} f\ (\exists_{sommet} x\ I(x, e) \wedge I(y, e) \wedge E(e) \wedge E(f)) \rightarrow e = f$$

L'idée en est : il existe un ensemble d'arêtes couvrant tous les sommets, chaque sommet étant couvert une seule fois. On a ici explicitement utilisé ce

qui fait la principale différence entre MS_1 et MS_2 : la quantification sur les arêtes. On va voir qu'on ne pouvait faire autrement, car la propriété n'est pas exprimable dans MS_1.

On va maintenant aborder la méthode classique pour montrer qu'une propriété est ou n'est pas exprimable, à partir des jeux d'Ehrenfeucht-Fraisse, présentés dans [3] et [4]

Rang de quantification d'une formule et m-équivalence de modèles

On appelle rang de quantification d'une formule son nombre maximal de quanteurs imbriqués (on ne distingue pas premier et second ordres). Formellement on a la définition inductive suivante :

- Le rang de quantification (rq) d'une formule atomique est 0.
- $rq(\neg\phi) = rq(\phi)$
- $rq(\phi \wedge \psi) = rq(\phi \vee \psi) = max(rq(\phi), rq(\psi))$
- $rq(\forall x\phi) = rq(\exists x\phi) = rq(\phi) + 1$

Si on écrit une formule comme un arbre, on prend la branche partant de la racine qui contient le plus de quanteurs, elle nous donne le rang de quantification.

Soient alors deux structures A et B (disons, deux graphes). Ces structures sont dites m-équivalentes, noté $A \equiv_m B$, si elles satisfont les mêmes formules de rang de quantification au plus k. On verra bientôt une caractérisation en terme de jeux de cette propriété, mais donnons d'abord le résultat qui va nous permettre de prouver que certaines propriétés ne sont pas expressibles dans certaines logiques :

Soit K une classe de structures. Si pour tout entier m il existe deux structures A et B m-équivalentes et telles que $A \in K$ mais $B \notin K$, alors la classe K n'est pas définissable par une formule.

Jeux, va-et-vient, et m-équivalence

Nous définissons un jeu à deux joueurs (dénommés ∀belard[†] et ∃loise[‡]). On prend deux structures A et B. ∀belard joue en premier et choisit un élément de A ou de B. Le but est alors pour ∃loise de trouver dans l'autre structure un élément "équivalent", *ie* qui satisfasse exactement les mêmes formules atomiques.

[†]Pierre Abélard (1079-1142), philosophe et théologien, dont le doute méthodique et l'analyse logique du langage -et en particulier des textes bibliques- ont ouvert la voie à la méthode scolastique.

[‡]Héloise (1100-1164), élève d'Abélard, brillante par de nombreux aspects, on laissera au lecteur le soin de se documenter sur la passion de ces deux personnages, les rebondissements tragiques de leur histoire, le mythe qui s'en est dégagé, et les divers symbolismes que peut revêtir l'emprunt de leurs noms ici.

∀belard joue alors une fois de plus, et ∃loise doit continuer à trouver des èléments équivalents, sachant qu'il faut tenir compte de tous les éléments précédemment choisis. C'est-à-dire qu'on regarde les formules atomiques faisant appel à tous les éléments retenus. On parle de va-et-vient pour désigner cette suite de coups et de réponses.

On appelle jeu d'Ehrenfeucht-Fraisse de longueur m le jeu précédent se déroulant pendant m tours. On dit qu'∃loise a une stratégie pour ce jeu si elle peut gagner à coup sûr (quels que soient les choix d'∀belard).

Le théorème alors important est le suivant : ∃loise a une stratégie pour le jeu de longueur m si et seulement si les deux structures sont m-équivalentes.

Équivalence de graphes linéaires

On regarde des graphes linéaires exprimés dans le langage τ_1. J'appelle graphe linéaire P_n le fil (ou chemin) orienté à n sommets (on a déjà souvent parlé de P_4). On va montrer que P_i et P_j sont m-équivalents si $i = j$ ou si i et j sont tous deux strictement supérieurs à 2^{m+1}.

Si $i = j$, les deux graphes sont égaux et la question de leur équivalence ne se pose pas. Mais il va falloir faire intervenir un jeu pour prouver le cas intéressant. Pour ceci, il faut analyser les coups possibles d'∀belard et choisir dans chaque cas une réponse adaptée pour ∃loise. On remarque pour commencer qu'il y a deux sommets distingués : le départ et l'arrivée. ∀belard commence comme toujours, et choisit un sommet quelconque a de P_i.

- Si a est à une distance inférieure à 2^m d'une des extrémités, ∃loise choisit b à la même distance de la même extrémité dans P_j. On remarque que a et b sont tous deux à une distance strictement plus grande de l'autre extrêmité.
- Si a est à une distance au moins 2^m de chaque extrémité, ∃loise doit aussi trouver un point suffisamment éloigné des deux extrémités. Elle choisit le milieu (ou éventuellement "un milieu") du fil, qui est à distance au moins 2^m de chaque extrémité.

On continue par récurrence, il ne reste plus que $m - 1$ coups à jouer, et on a coupé P_i et P_j de l'une des manières suivantes :

- Deux fils de strictement plus de 2^m sommets.
- Un fil de strictement plus de 2^m sommets et l'autre d'une taille identique pour P_i et P_j.

Donc ∃loise a une stratégie : P_i et P_j sont m-équivalents.

Jeux d'ordre supérieur

On a défini ici le jeu du premier ordre : les joueurs choisissaient des éléments dans chaque structure. Ce jeu correspond à la logique du premier ordre. On obtient le jeu correspondant à la logique monadique du second ordre en autorisant les joueurs à choisir non seulement des éléments, mais également des ensembles (c'est-à-dire des relations unaires, rien de surprenant). ∀belard

choisit des éléments et des ensembles, et ∃loise répond à chaque fois par des objets du même type.

Dans ces jeux du second ordre, les théorèmes et méthodes utilisées ici sont valables exactement de la même façon.

L'existence d'un couplage parfait n'est pas exprimable dans MS_1
Appliquons le jeu précédent pour montrer que la propriété "admettre un couplage parfait" est hors de portée de la logique MS_1. On va montrer que les cliques K_{2^m} et K_{2^m+1} sont m-équivalentes. Pourtant la première admet des couplages parfaits, mais pas la seconde pour une bête question de parité.

On utilise pour la preuve un jeu comme dans l'exemple précédent. Remarquons simplement que prendre m ensembles de sommets ne permet de distinguer que 2^m sommets (à chaque nouvel ensemble, au pire on coupe en deux chaque ensemble déjà existant). Comme nos structures ont toutes deux au moins ce nombre de sommets, on sait qu'on va pouvoir faire gagner ∃loise. Prenons deux cliques K_i et K_j, avec i et j au moins égaux à 2^m.

- Si ∀belard choisit un sommet, ∃loise peut prendre n'importe quel sommet.
- Si ∀belard choisit un ensemble de sommets A tel que le cardinal de A ou le cardinal du complément de A est strictement inférieur à 2^{m-1}, alors ∃loise fait de même.
- Si ∀belard choisit un ensemble de sommets suffisamment éloigné à la fois de l'ensemble vide et du graphe complet, ∃loise prend la moitié de sa clique.

Dans chacun des deux derniers cas, ∃loise doit faire attention à prendre ou ne pas prendre chaque sommet des sommets déjà distingués, en fonction de la configuration dans l'autre graphe.

On continue comme la première fois par récurrence, avec la simple remarque supplémentaire que le nombre de jeux peut se multiplier à chaque étape : un jeu de longueur m, deux de longueur $m-1$ correspondant au découpage induit par le choix d'un ensemble (et donc de son complément), puis 2^k jeux de longueur $m-k$, jusqu'à arriver à zéro.

3.2 Largeur arborescente, largeur de clique et opérations sur les graphes

On va maintenant rappeler quelques opérations de transformation et de combinaison de graphes k-colorés, et redonner des définitions de la largeur arborescente et de la largeur de clique à partir de ces opérations. Ceci vient des articles [2], et [1]. On appelle ici graphe k-coloré un graphe dont les sommets sont partitionnés en k ensembles, chacun caractérisé par une couleur

dans $\{1, \cdots, k\}$. Notation standard pour un tel graphe : $\langle V, E, V_1, \cdots, V_k \rangle$ où V_i est l'ensemble des sommets de couleur i.

Union disjointe $(\oplus)$
Deux graphes sont simplément posés côte à côte. Nous avons appelé en cours cette opération "opération parallèle". En supposant V et V' disjoints on peut lécrire :

$$G \oplus G' = \langle V \cup V', E \cup E', V_1 \cup V'_1, \ldots, V_n \cup V'_n \rangle$$

Changement de couleur $(\rho_{i \to j})$
On donne la couleur j à tous les sommets initialement de couleur i.

$$\rho_{i \to j}(G) = \langle V, E, V_1, \ldots, V'_i = \emptyset, \ldots, V'_j = V_j \cup V_i, \ldots, V_n \rangle$$

Contraction $(Contr_i)$
Tous les sommets de couleur i sont contractés en un seul, qui collecte toutes les arêtes de l'ensemble. On pose $G' = Contr_i(G)$, on note v_i un nouveau sommet (hors de V), et on désigne une arête par le couple de ses extrémités. L'opération se formalise alors :

- $V' = (V \setminus V') \cup \{v_i\}$
- $V'_i = \{v_i\}$
- Pour tout $j \neq i$, $V'_j = V_j$
- $E' = (E \cap V'^2) \cup \{(u, v_i), \exists v \in V_i, (u, v) \in E\} \cup \{(v_i, v), \exists u \in V_i, (u, v) \in E\}$

Connexion croisée $(\eta_{i \to j})$
Tous les sommets de couleur i sont liés à tous les sommets de couleur j. On pose toujours $G' = \eta_{i \to j}(G)$ et on écrit :

- $V' = V$
- Pour tout i, $V'_i = V_i$
- $E' = E \cup \{(u, v), u \in V_i et v \in V_j\}$

On peut remarquer que toutes les transformations données ici sont trivialement définissables avec les réductions que nous avons déjà mentionnées. On se place indifféremment dans τ_1 et τ_2 et on ajoute k relations unaires pour les V_i. Il faut tout de même un peu d'astuce pour la contraction, où l'on doit pouvoir "élire" le sommet restant de V_i. C'est facile avec des sommets ordonnés car on peut distinguer un élément minimal, mais ici il faut faire un petit peu plus compliqué, et prendre $m = 2$. Ce qu'il est important de remarquer pour la suite est que ce cas excepté, ces réductions n'utilisent aucun quantificateur. On peut donc sans aucun souci transformer des formules sans en changer le rang de quantification. Ce point sera critique dans l'algorithme final.

Largeurs
On peut caractériser les largeurs arborescente et de clique à partir de ces opérations. Cette dernière sera en fait définie par la présente caractérisation.

Les graphes de largeur arborescente au plus k sont exactement la clôture par union disjointe, changement de couleur et contraction des graphes $k+1$-colorés à au plus $k+1$ sommets. [1]

On définit l'ensemble des graphes de largeur de clique au plus k comme la clôture par union disjointe, changement de couleur et connexion croisée des singletons k-colorés. [1]

Arbres de décomposition
Un graphe de largeur au plus k possède un arbre de décomposition : c'est-à-dire un arbre qui rend compte de l'ensemble des opérations effectuées pour obtenir le graphe à partir des cas de base donnés au-dessus. L'analyse de ces arbres va être déterminante dans la recherche d'algorithmes polynomiaux et même linéaires pour résoudre des problèmes autrement compliqués.

L'arbre de décomposition est enraciné et ses feuilles sont étiquetées par des graphes de base (graphes de moins de $k+1$ sommets ou singletons selon la largeur observée). Quant à ses noeuds internes, ils sont étiquetés chacun par une des opération précédentes.

Dans la suite, on va se servir de ces arbres de décomposition, en considérant qu'on les possède. Discutons alors brièvement de la complexité de la recherche de tels arbres. On sait tout d'abord que la détermination de la largeur arborescente d'un graphe est un problème NP-complet. En revanche, Bodlaender nous a donné un algorithme qui, sur une valeur de k fixée, nous donne en temps linéaire un arbre de décomposition témoignant de la largeur arborescente au plus k d'un graphe (l'algorithme échoue si le graphe n'est pas de largeur arborescente au plus k). On notera que cet algorithme n'est pas implémentable, mais il reste linéaire en la taille du graphe.

Pour la largeur de clique, le problème de la détermination polynomiale d'un arbre de décomposition reste ouvert.

3.3 Théorème de Feferman-Vaught-Shelah

Cette section va introduire un ensemble de formules appelées formules de Hintikka. Ces formules permettent de caractériser (à équivalence logique près) toutes les formules de rang de quantification inférieur à un k donné. Qui plus est, pour chaque k elles sont en nombre fini. On se basera sur [3]. On pourra donc plus tard construire l'intégralité de ces formules et de leurs

combinaisons. Le coût d'une telle opération est prohibitif, mais ne fait intervenir que k. C'est donc à partir de ceci que sera construite la monstrueuse constante qui accompagnera nos algorithmes linéaires en n...

Une fois ces formules connues, le théorème de Feferman-Vaught-Shelah va leur affirmer un bon comportement face aux unions disjointes de graphes, ce qui aboutira aux algorithmes de la section suivante.

Positions gagnantes dans les jeux d'Ehrenfeucht-Fraisse
Pour une structure G et un ensemble d'éléments $\bar{a}$ donnés (un graphe et des sommets ou sommets et arêtes), on va construire une série de formules regroupant toutes les informations sur $\langle G, \bar{a}\rangle$ en termes de jeux. On va plus précisément construire un ensemble $\{\phi^m_{G,\bar{a}}$, m entier$\}$ où chaque $\phi^m_{G,\bar{a}}$ caractérise l'ensemble des structures $\langle G', \bar{b}$ pour lesquelles $\exists$loise a une stratégie pour le jeu à m tours partant de $\langle G, \bar{a}\rangle$ et $\langle G', \bar{b}$.

Autrement dit, la formule $\phi^m_{G,\bar{a}}$ caractérise l'ensemble des structures m-équivalentes à $\langle G, \bar{a}\rangle$, soit l'ensemble des structures vérifiant les mêmes formules de rang de quantification au plus m.

En prenant les formules avec $\bar{a} = \emptyset$ on obtient la caractérisation de G. On note les formules correspondantes ϕ^m_G.

On définit ces ensembles par récurrence sur m, ici pour le premier ordre :

$$\phi^0_{G,\bar{a}}(\bar{x}) \equiv \bigwedge\{\psi(\bar{x}) | \psi \textit{ atomique ou atomique niée},\ G \models \psi(\bar{a})\}$$

$$\phi^{m+1}_{G,\bar{a}}(\bar{x}) \equiv (\bigwedge_{s\in G} \exists y\ \phi^m_{G,\bar{a}s}(\bar{x}y)) \wedge (\forall x \bigvee_{s\in G} \phi^m_{G,\bar{a}s}(\bar{x}y))$$

Le cas de base ($\phi^0_{G,\bar{a}}$) prend l'ensemble des formules atomiques ou atomiques niées vérifiées par $\langle G, \bar{a}\rangle$ et en donne la conjonction. Ainsi cette formule caractérise les structures vérifiant ces formules atomiques. Ces structures vérifient plus généralement exactement les mêmes formules sans quanteurs (*ie* combinaisons booléennes de formules sans quanteurs).

La construction de nouvelles formules par récurrence est calquée sur les jeux d'Ehrenfeucht : $\phi^{m+1}_{G,\bar{a}}$ caractérise les structures $\langle G', \bar{b}\rangle$ vérifiant deux propriétés.

- Quelque soit s dans G, on a un t dans G' tel que $\langle G, \bar{a}s\rangle$ et $\langle G', \bar{b}t$ soient m-équivalents.
- Quelque soit t dans G', on a un s dans G tel que $\langle G, \bar{a}s\rangle$ et $\langle G', \bar{b}t$ soient m-équivalents.

Autrement dit, quelque soit le coup d'$\forall$belard, choisi dans G ou G', $\exists$loise a une réponse qui lui permette à coup sur de jouer encore au moins m fois.

Pour interpréter ces formules ainsi que cela vient d'être fait, il suffit de remarquer que la quantification universelle est la même chose qu'une

conjonction sur tous les éléments, et qu'une quantification existentielle est en revanche équivalente à une disjonction sur tous les éléments. Ici, on connaît G et on caractérise G', les quantifications sont donc sur G', et les conjonctions et disjonctions sur G.

Une fois cette définition ainsi disséquée, on peut en déduire ce que sont les formules de Hintikka de la logique monadique du second ordre : dans la construction par récurrence, il faut encore faire la conjonction avec deux formules exprimant que si ∀belard choisit un ensemble de sommets de G ou G', ∃loise peut en trouver un équivalent dans l'autre structure. Ceci se fait avec des quantificateurs monadiques du second ordre sur G', et des conjonctions et disjonctions sur l'ensemble des parties de G.

On a finalement un ensemble de formules caractérisant chacune une classe d'équivalence pour la relation de m-équivalence.

Formules de Hintikka
On va appeler formules de Hintikka l'ensemble des formules $\phi^m_{G,\bar{a}}$ pour toutes les structures $\langle G, \bar{a}\rangle$. On note $p = |\bar{a}|$, qui correspond au nombre de variables libres des formules $\phi^m_{G,\bar{a}}$. Le m sera appelé rang de la formule. On peut prouver rapidement un ensemble de propriétés sur ces formules :

- Pour m et p fixés, il n'existe qu'un nombre fini de formules de Hintikka. Ceci est immédiat par induction, en remarquant d'abord que les formules atomiques faisant apparaître au plus p variables sont en nombre fini. (le nombre de formules créées est quand même exponentiel en le nombre de formules atomiques, comme on peut prendre toutes les conjonctions de ces dernières...)
- Deux formules de Hintikka distinctes sont incompatibles.
  En effet, deux formules différentes caractérisent deux classes de m-équivalence différentes, et donc disjointes.
- Toute formule ψ de rang de quantification au plus m est équivalente à une disjonction de formules de Hintikka.
  Plus précisément, on prend toutes les structures $\langle G, \bar{a}\rangle$ vérifiant ψ. C'est la disjonction des $\phi^m_{G,\bar{a}}$ correspondantes qui nous intéresse.

Théorème de Feferman-Vaught-Shelah
Pour toute formule $\psi(\bar{x}, \bar{y}, \bar{Z})$ de rang de quantification au plus k (on considère que $\bar{x}$ et $\bar{y}$ contiennent des variables du premier ordre, et $\bar{Z}$ du second ordre, comme d'habitude), il existe un ensemble fini de couples de formules de Hintikka de rang k $\{(h_{1,q}(\bar{x}, \bar{Z}), h_{2,q}(\bar{y}, \bar{Z})), q \leq Q\}$ tel que pour tout graphe $G = G_1 \oplus G_2$, tous ensembles $\bar{a}$ et $\bar{b}$ de sommets de G et tout ensemble $\bar{C}$ de parties de G, on ait :

$$G \models \psi(\bar{a}, \bar{b}, \bar{C})$$

si et seulement si il existe un q tel que

$$G_1 \models h_{1,q}(\bar{a}, \bar{C}) \; et \; G_2 \models h_{2,q}(\bar{b}, \bar{C})$$

Pour préciser les notations, $\bar{x}$ et $\bar{y}$ sont des variables concernant respectivement G_1 et G_2. $\bar{Z}$ est un ensemble de variables du second ordre, représentant des ensembles qui peuvent être à cheval sur G_1 et G_2, donc par exemple dans $h_{1,q}(\bar{a}, \bar{C})$ on considère avoir fait l'intersection entre les A_i et G_1.

Justifications
La démonstration de ce théorème passe par un lemme qui lui ressemble un peu mais qui n'est plus uniforme en $G = G_1 + G_2$.

Soit $G = G_1 + G_2$, pour toute formule de Hintikka $h(\bar{x}, \bar{y}, \bar{Z})$, existent deux formules de Hintikka uniques du même rang $h_1(\bar{x}, \bar{Z})$ et $h_2(\bar{y}, \bar{Z})$ telles que pour tous $\bar{a}$, $\bar{b}$ et $\bar{C}$ comme précédemment on ait :

$$G \models h(\bar{a}, \bar{b}, \bar{C})$$

si et seulement si

$$G_1 \models h_1(\bar{a}, \bar{C}) \; et \; G_2 \models h_2(\bar{b}, \bar{C})$$

Ce lemme se prouve avec l'interprétation des formules de Hintikka en termes de jeux. En effet, $\exists$loise ne peut avoir de stratégie gagnante sur $G = G_1 + G_2$ que si elle a une stratégie pour G_1 et une autre pour G_2, et cette condition et nécessaire et suffisante.

Une fois le lemme établi, il faut utiliser les propriétés des formules de Hintikka pour obtenir le théorème.

3.4 Algorithmes linéaires : théorème de Courcelle

On va maintenant utiliser conjointement les propriétés générales des formules de Hintikka, le théorème de Feferman-Vaught-Shelah, et les arbres de décomposition pour obtenir des algorithmes linéaires pour le calcul de fonction a priori difficiles.

Ceci a été fait par Courcelle dans [2].

On s'intéresse au dénombrement des occurences d'une propriété donnée (compter le nombre de couplages parfaits) ou à l'évaluation du poid total de ces occurences sur un graphe pondéré (poids total de tous les chemins hamiltoniens). Dans tous les cas on résoud le probème de décision associé.

En considérant que le poids d'une occurence est le produit des poids de ses composants (attention, dans d'autres contextes on fait souvent la somme !), et que le poids total est la somme des poids des occurences, on retrouve la première question à partir de la seconde en ramenant tous les poids à 1. Ce choix de faire le produit plutôt que la somme est aussi lié à des interprétations de ces fonctions (comme la notion de permanent ou de hamiltonien d'une matrice). Dans la suite on s'occupe arbitrairement de l'un ou de l'autre.

On s'intéresse toujours à des propriétés définissables dans la logique monadique du second ordre, et la fonction qu'on calcule a donc la forme :

$$\sum_{\Psi(G'),\ G' \subseteq G} \prod_{a \in G'} \omega(a)$$

où ω est la fonction de poids, et le produit peut être remplacé par une somme.

Pré-traitrements
L'algorithme générique qu'on présente ici nécessite un énorme pré-traitement qui dépendant uniquement de Ψ. Sa complexité impressionante ne compte donc que pour une constante dans la complexité finale (qui elle est considérée en fonction de la taille n du graphe).

On commence par regarder le rang de quantification m de Ψ. Il faut alors faire la liste de toutes les formules de Hintikka de ce rang. Pour chacune de ces formules, il faut également déterminer la liste de formules donnée par le théorème de Feferman-Vaught-Shelah.

Il faut ensuite prendre toutes les formules de rang de quantification au plus m (à équivalence logique près) et pour chacune, trouver la disjonction de formules de Hintikka équivalente.

Ces deux étapes sont d'une complexité considérable, et ne sont pas directement faisables avec le seul matériel présenté dans ce document.

Calcul
Une fois le graphe G donné, de largeur inférieure à k, on a besoin d'un arbre de décomposition témoignant de cette faible largeur. On rappelle qu'on a un algorithme linéaire (bien que difficilement implémentable en pratique) pour la largeur arborescente et que le problème reste ouvert pour la largeur de clique. Si on travaille avec ce second paramètre on considère donc qu'un arbre de décomposition est donné ou "facilement" calculable.

Maintenant que tout est près, il faut évaluer notre fonction en suivant la décomposition du graphe.

– Pour $G = G_1 \oplus G_2$, $\sum_{\Psi(G'),\ G' \subseteq G}$ est remplacé par $\prod_q \prod_{h_{1,q}(G'_1),\ G'_1 \subseteq G_1} \dot{\prod}^{h_{2,q}(G'_2),\ G'_2 \subseteq G_2}$ (où l'ensemble des produits peut être remplacé d'un seul bloc par des sommes).
– Pour G image de H par une des autres opérations, on opère une traduction de Ψ en une formule Ψ_H vérifiant : $\sum_{\Psi(G'),\ G' \subseteq G} = \sum_{\Psi_H(H'),\ H' \subseteq H}$ (cf. remarque lors de la définition de ces opérations)

Cette dernière étape est linéaire, car consiste simplement en une exploration de l'arbre de décomposition du graphe.

Bilan

On a introduit dans ces pages une notion de complexité se basant non sur la résolution des problèmes mais sur leur description. Cette approche a l'avantage d'être indépendante des modèles de calcul. Néanmoins, les classes de complexité que l'on construit ainsi ne sont pas dénuées de liens avec les classes de complexité Turing.

On a notamment vu deux développements de ce domaine. Le premier était la définition entièrement descriptive des classes des problèmes NP et NP-complets, le second plus "pragmatique" visait à la définition de larges de classes de problèmes pour lesquelles la construction d'algorithmes polynomiaux était envisageable.

Une part de la complexité descriptive qui n'a pas du tout été abordée est celle où l'on s'autorise à définir une relation comme point-fixe d'une équation (ladite équation étant comme toujours définie par une formule dans un langage). Cette approche était le seul moyen de définir descriptivement la classe P avant les développements de la logique linéaire de Girard. Ceci permet aussi d'aller jusqu'au plus haut de la hiérarchie Turing, au-delà même du domaine du décidable.

Références

[1] B. Courcelle and J.A. Makowsky. Fusion on relational structures and the verification of monadic second order properties. *Mathematical Structures in Computer Science*, 2002.

[2] B. Courcelle, J.A. Makowsky, and U. Rotics. On the fixed paramater complexity of graph enumeration problems definable in monadic second-order logic. *Discrete Applied Mathematics*, (108) :23–52, 2001.

[3] H-D. Ebbinghaus and J. Flum. *Finite Model Theory.* Springer, Perspectives in Mathematical Logic, 1999.

[4] N. Immerman. *Descriptive Complexity.* Springer, Graduate Texts in Computer Science, 1999.