Une écriture du théorème d'incomplétude de Kurt Gödel

P. Manoury

2005

1 Le paradoxe du menteur comme paragon du théorème

De l'aveu même de son inventeur, la preuve du *théorème d'incomplètude* de Gödel [3] reprend, dans les termes de la logique mathématique, la forme du paradoxe du menteur. De l'énoncé d'Epiménide : «*Je suis un menteur*», le logicien ne peut rien dire, c'est-à-dire, qu'il ne peut statuer sur sa vérité. Le logicien est un être étrange qui poursuit l'idéal d'un monde où tout doit être soit vrai, soit faux ; où ce qui n'est pas vrai est faux et ce qui n'est pas faux est vrai. Pour obtenir le paradoxe du menteur, le logicien fait l'irréaliste hypothèse que le monde se divise en deux catégories d'individus : ceux qui disent toujours vrai et ceux qui disent toujours faux (les *menteurs*).

Le logicien se pose donc la question de savoir ce qu'il en est de la vérité de la phrase «*Je suis un menteur*». Et il raisonne ainsi :

1. Si la phrase «*Je suis un menteur*» est vraie, alors celui qui l'énonce est un menteur (puisque c'est ce qu'affirme la phrase) et donc sa phrase est fausse (puisqu'un menteur ne dit jamais le vrai). Notre hypothèse nous amenant à sa contradiction, il n'est pas possible que la phrase soit vraie.
2. Si la phrase «*Je suis un menteur*» est fausse, alors celui qui l'énonce est un menteur (puisqu'il dit quelque chose de faux). Mais alors ce qu'il dit serait vrai. Ce qui contredit l'hypothèse logicienne sur les menteurs. Il n'est donc pas non plus possible que la phrase soit fausse.

Le paradoxe du menteur met ainsi les logiciens devant l'absurdité de leurs hypothèses sur le monde en posant une phrase qui semble ne pouvoir être ni vraie ni fausse. En fait ce que met réellement en jeu le paradoxe du menteur ce n'est pas tant la vanité du monde idéal des logiciens que leur incapacité à *décider* de la vérité d'une phrase. En effet le *paradoxe* ne réside pas dans la phrase elle-même mais dans l'échec de la tentative de déterminer, par les moyens du raisonnement, si cette phrase est vraie ou non. Ainsi, la leçon à tirer de cette histoire de menteur est qu'*il existe une phrase dont on ne peut décider par le raisonnement si elle est vraie ou fausse*. Et bien c'est ce qu'énonce *mutatis mutandis*, nous verrons comment, le théorème d'incomplétude de Gödel dans le monde des nombres, des formules et des règles du raisonnement formalisées par un *système de preuve*.

Dans le monde abstrait de la logique formalisée, il n'y a pas de locuteur qui puisse parler de ses qualités, mais uniquement des formules. Si l'on veut y reconduire le paradoxe du menteur, il y faut une phrase (une formule) qui énonce qu'elle ment, c'est-à-dire qu'elle est fausse ; il y faut une phrase comme «*Je suis une phrase fausse*». On peut alors raisonner comme suit :

1. Si la phrase «*Je suis une phrase fausse*» est vraie alors le «Je» de la phrase «*Je suis une phrase fausse*» est faux. Ce «Je» n'étant autre que la phrase «*Je suis une phrase fausse*», on obtient, en remplaçant «Je» par sa référence, que la phrase «*Je suis une phrase fausse*» est fausse . Ce résultat contradit l'hypothèse de départ. On ne peut donc supposer que la phrase est vraie.
2. Si la phrase «*Je suis une phrase fausse*» est fausse alors sa négation est vraie. Mais attention, sa négation n'est pas la phrase «*Je ne suis pas une phrase fausse*» dans laquelle le «Je» n'est plus le même

que celui de la phrase de départ. Pour tirer la conséquence paradoxale attendue de notre hypothèse, oublions un temps l'auto-référence du «Je» pour une référence plus générale que l'on appellera F et livrons nous à une petite acrobatie formelle. La négation de «*F est une phrase fausse*» est bien la phrase «*F n'est pas une phrase fausse*». Si cette phrase est vraie, son contenu est vérifié et, ce qui n'est pas faux étant vrai, on a que la phrase F est vraie. Ce que nous venons de réaliser est valide pour n'importe quelle référence F, ça l'est donc aussi pour l'auto-référence «Je» de la phrase «*Je suis une phrase fausse*». On a donc établi en particulier (ce qui a peu de sens en français, mais en a formellement) que la phrase «Je» est vraie. En remplaçant ce «Je» par celui qui nous intéresse, on obtient que la phrase «*Je suis une phrase fausse*» est vraie. Ici encore, ce résultat contredit l'hypothèse de départ. On ne peut donc supposer que la phrase est fausse.

Remarque : pour conduire à bien ce deuxième raisonnement, nous avons fait *explicitement* usage de la notion de remplacement d'une référence. Nous avons même fortement utilisé cette notion dans le second cas du raisonnement lorsque nous avons fait un détour par une phrase quelconque F, pour y substituer une certaine référence «Je» que nous avons elle-même remplacée par notre phrase. Nous verrons l'importance de l'usage explicite de l'opération de substitution dans la construction de la formule génératrice du paradoxe dans le théorème d'incomplétude.

Pour passer du paradoxe du menteur au théorème d'incomplétude, on glisse de la notion de vérité d'un énoncé à celle de sa *prouvabilité*. La prouvabilité est définie par un ensemble de faits logiques de bases (énoncés particuliers appelés *axiomes*) et de règles de déductions. Le tout constitue un *système de preuves*. Le rapport entre vérité et prouvabilité s'exprime en terme de

correction tout ce qui est prouvable est vrai.

complétude tout ce qui est vrai est prouvable.

Comme nous sommes dans le monde des logiciens où un énoncé doit être vrai ou faux sans autre possibilité, on pourrait imaginer l'on sache *décider* du statut de n'importe quel énoncé :

- ou bien il est vrai et alors il est prouvable ;
- ou bien il est faux et alors c'est sa négation qui est prouvable.

Et bien c'est précisément cette imagination que le théorème d'incomplétude bat en brèche en posant un énoncé arithmétique (ie. concernant les nombres) dont ni lui ni sa négation ne sont prouvables. L'argument logique fondamental pour obtenir cette double impossibilité est celui utilisé pour le paradoxe du menteur.

Le raisonnement conduisant au paradoxe traite d'un énoncé auto-référent. Tout le travail admirablement subtil de Gödel a été de trouver un moyen mathématique formel d'obtenir un énoncé arithmétique possédant les propriétés de l'auto-référence. En fait l'énoncé de Gödel ne dit pas «je» mais «une représentation de moi-même», ou encore : «un *codage* de moi-même». En effet, le théorème d'incomplétude repose sur la possibilité de *coder* les formules et leur système de preuves avec des nombres entiers. À chaque formule Gödel fait correspondre un nombre entier (son *numéro de Gödel*). Dès lors, il peut construire une formule qui affirme quelque chose d'un nombre qui est le nombre de Gödel d'une formule ; c'est-à-dire, que par le truchement du codage, Gödel construit des formules qui parlent des formules. De surcroît, le système de preuves est lui-même codable avec des nombres (certains nombres sont les *nombres de Gödel* de preuves). Il peut donc également construire une formule mettant en relation deux nombres : l'un représente une formule et l'autre la preuve de cette formule. Poursuivant, il écrit une formule qui exprime la propriété de *prouvabilité*, puis une formule qui exprime la non-prouvabilité. Enfin, comme toutes les formules, la formule de non-prouvabilité a un code, un nombre de Gödel. En appliquant ce nombre à la formule de non-prouvabilité, il obtient un énoncé moralement auto-référent que l'on peut paraphraser *grosso modo* en «*"n'est pas prouvable" n'est pas prouvable*».

Plan de ce qui suit

La section 2 introduit le *paysage formel* où s'énonce le théorème. On y présente ce que sont le langage des formules avec l'opération de susbtitution, le système de preuve et l'arithmétique formelle. La section 3 présente un certain nombre de traits et propriétés du système formel. En particulier, on y définit une classe

restreinte de formules jouissant de la propriété de complétude. La section 4 contient l'énoncé et la preuve du théorème d'incomplétude. La section 5 donnant le détail du codage des formules et des preuves sous forme de chaînes de caractères ainsi que la définition des fonctions et relations nécessaires à l'expression de la formule paradoxale du théorème. La section 6 achève effectivement la démonstration en posant comment le codage en terme de chaîne de caractères repose sur de simples fonctions arithmétiques.

2 Le paysage formel

Le grand œuvre des logiciens entre la toute fin du XIXième et le début du XXième siècle a été la tentative de fondation des mathématiques sur des bases purement logiques. Ce travail a abouti à la conception de la *logique formelle* où les notions de *formule* et de *preuve* acquièrent un contenu précis.

Une formule est comme une phrase qui énonce un fait (propriété, relation) à propos d'objets[1], d'ensembles d'objets ou du monde des objets lui-même (quantification). Concrètement, une formule est un *objet syntaxique* : une suite de symboles arrangés selon un ordre et des conventions syntaxiques telles qu'à chaque suite proposée de symboles on sache répondre sans ambiguïté il s'agit ou non d'une formule. Les symboles qui composent une formule sont de deux sortes : les *constantes logiques* comme les *connecteurs propositionnels* («et», «ou», «non», etc.) et les *quantificateurs* («pour tout», «il existe») ; des noms pour les propriétés (ou *prédicats*), relations, fonctions ou individus d'un monde d'objets.

Une formule peut être vraie ou fausse. On peut s'assurer de la vérité (ou fausseté) d'une formule, directement ; en confrontant ce qu'elle énonce avec la connaissance que l'on a du monde d'objets dont on veut qu'elle parle. Pour cela, on *interprète* les symboles des formules par des individus, fonctions, propriétés, relations, etc. connus du monde d'objets visé. Certaines formules sont vraies (ou fausse) dans tous les mondes possibles : leur vérité (ou fausseté) ne dépend que de l'interprétation des constantes logiques. Cet accés à la vérité est dit *sémantique*. On peut l'illustrer par la définition quasi redondante de la vérité selon Alfred Tarski :

l'énoncé «la neige est blanche» est vrai si et seulement si la neige est blanche.

Si cette définition a un sens, il est à chercher dans la distinction entre l'usage (deuxièmes occurence) et la mention (première occurence entre guillemets) de la proposition *la neige est blanche*. La vision sémantique de la vérité peut recevoir un contenu plus précis en faisant appel à de la théorie des ensembles pour construire des *modèles*. Nous n'irons pas jusque là dans cette présentation et nous contenterons de retenir qu'il existe une notion de vérité ou *validité* sémantique.

Une formule peut être vraie ou fausse en soi, mais en pratique, elle est le plus souvent la *conséquence* d'autres formules. Un théorème est rarement une seule formule, mais plutôt une phrase du genre : *si l'on suppose que l'on a ceci et cela alors on a également ça* ; ou encore : *sous les* hypothèses *que ceci et cela, on montre que ça*. On peut donner une définition sémantique de la relation de conséquence : toute interprétation qui vérifie les hypothèses vérifie également la conclusion. Mais une telle définition ressort de la vision redondante de la vérité selon Tarski : si elle dit ce qu'est la relation de conséquence, elle ne donne pas de *moyen* de l'établir. Ce moyen est le raisonnement, la *preuve*.

Une preuve est un enchaînement d'étapes de raisonnement dont chacune engendre une conséquence élémentaire des hypothèses ou des conséquences élémentaires déjà acquises. Lorsque la conclusion d'une étape élémentaire est la formule que l'on cherche à prouver, la preuve est achevée. Dans la vision formelle de la logique, une preuve n'est pas un discours qui doit emporter la conviction de son lecteur, mais un texte suffisament explicite pour que l'on puisse vérifier la validité de l'articulation des étapes menant des hypothèses à la conclusion. Concrètement, on se donne un ensemble de formules de base, les *axiomes* ainsi qu'un ensemble de *règles de déductions* associant une, voire plusieurs, conclusion à un ensemble de formules *prémisses*. Une *preuve formelle* (ou *dérivation*) est alors une suite de formules telle que chaque formule de

[1] Le terme «objet» est à prendre ici dans son acception la plus indéterminée ; on aurait pu tout aussi bien parler de «trucs» ou de «machins». Il en va de même du terme «monde» utilisé ci-après.

la suite est soit une hypothèse, un axiome logique ou le résultat de l'application d'une régle de déduction utilisant comme prémisses des formules précédantes[2] de la suite. L'ultime formule de la suite est la formule à prouver. À l'instar des formules, une preuve (formelle) est donc un *objet syntaxique* : une suite de formules arrangée de façon telle qu'à chaque suite de formule proposée, on puisse répondre sans ambiguïté s'il s'agit ou non d'une preuve.

Parallèlement à leur travail d'affinage des outils logiques, les logiciens-mathématiciens fondateurs se sont attachés à la recherche et à l'énonciation d'un ensemble de principes minimaux sur lesquels faire reposer l'ensemble des vérités mathématiques connues et à venir : les *axiomes* dont on admet *a priori* (c'est-à-dire sans preuve) la vérité. L'utilisation d'un ensemble d'axiomes pour définir un domaine des mathématiques est connu depuis Euclide qui avait fondé sa géométrie sur un tel ensemble. On accorde à l'italien Peano [1] d'avoir donné une première *arithmétique formelle*. Une arithmétique formelle est la réunion des formules exprimant les axiomes de l'arithmétique et d'un *système de preuves* (axiomes logiques et règles de déduction).

Un mot sur les entiers naturels L'arithmétique est la branche des mathématiques qui s'intéresse à l'ensemble des nombres entiers positifs 0, 1, 2, 3, etc. (que l'on appelle *entiers naturels* ou plus simplement *entiers*), à leurs opérations et à leurs propriétés.

L'ensemble des entiers naturels possède cette propriété remarquable que ses éléments sont donnés dans un certain ordre et qu'à l'exception du premier d'entre eux (0) n'importe quel nombre peut être donné comme étant celui qui vient après un autre dans la suite (1 vient après 0, 2 vient après 1, etc.) ; comme le *successeur* d'un autre. Pour obtenir l'ensemble de tous les entiers, il suffit donc de se donner le premier (0) et une fonction qui a tout entier associe son successeur.

Cette structure particulière de l'ensemble des entiers permet d'y appliquer un principe de raisonnement particulier : *le raisonnement par récurence*. Si l'on veut établir la validité d'une certaine propriété pour tout entier, il suffit de montrer qu'elle est vraie du premier et que, si on la suppose vraie pour un entier quelconque alors elle l'est aussi de son successeur.

Plan de ce qui suit Le reste de cette section est dévolue à la présentation du système de l'arithmétique formelle. On y trouve la définition du langage des formules et de l'opération de substitution (2.1), la définition des preuves formelles en logique pure (2.2) et enfin la donnée des axiomes de l'arithmétique (2.3) ce qui achève la définition de l'arithmétque formelle.

2.1 Le langage des formules

Une formule est construite à partir d'un ensemble de symboles : des symboles logiques, des symboles purement syntaxiques, des symboles dits de *constantes* et des symboles dits de *variables* dont on suppose l'ensemble infini. On distingue parmi les symboles de constantes trois catégories : les symboles d'individus, les symboles de fonctions et les symboles de prédicats ou relation. Les symboles de variables sous tous des symbole d'individus[3].

2.1.1 Lexique

Symbole syntaxiques

(parenthèse ouvrante ;
) parenthèse fermante.

Constantes logiques

¬ pour la négation, le «non» ;

[2] La relation de précédance interdit simplement d'utiliser la conclusion comme argument de sa propre preuve.
[3] C'est ce qui fait de notre langage un langage du *premier ordre*.

$\Rightarrow$ pour l'implication logique ;
$\forall$ pour la quantification universelle, le «pour tout».

Constantes de relation
$=$ pour la relation d'égalité ;
$\leq$ pour la relation d'infériorité au sens large, «inférieur ou égal».

Fonctions et constante d'individu
$+$ pour l'addition ;
$\times$ pour la multiplication ;
$\hat{}$ pour l'exponentiation, l'élévation à la puissance ;
s pour la fonction successeur ;
0 pour le nombre zéro.

Symboles de variables Les variables d'individus seront écrites x_1, x_2, x_3, etc.

2.1.2 Les termes

Un individu, c'est-à-dire un entier, est désigné soit directement par un symbole, soit par une combinaison de symboles. Les symboles et combinaisons qui désignent un individu sont appelés des *termes*. L'écriture d'un terme est définie par les règles suivantes :
- 0 est un terme ;
- les variables x_1, x_2, x_3, etc. sont des termes ;
- si t est un terme alors st est un terme ;
- si t_1 et t_2 sont des termes et si $\star$ est l'un des symboles $+$, $\times$, $\hat{}$ alors $(t_1 \star t_2)$ est un terme.

Notation des entiers formels Dans notre système formel, les entiers 0, 1, 2, 3, etc. s'écriront respectivement 0, $s0$, $ss0$, $sss0$, etc. Pour écrire un entier n quelconque, on écrira s^n0, désignant par là une suite de n symboles s terminée par un 0 (par convention, $s^00 = 0$). L'écriture s^nt désigne l'application n fois de la fonction s au terme t. On a $s^0t = t$ et $s^{sn}t = ss^nt$.

Ainsi l'entier 0 correspond au terme noté s^00, l'entier 1 à s^10, l'entier 2 à s^20, etc. De façon plus générale, si n ou a désigne un entier, on note s^n0, resp. s^a0 les termes correspondants.

2.1.3 Les formules

On utilise x_i pour désigner une variable quelconque parmi x_1, x_2, x_3, etc. L'ensemble des formules est défini par les règles suivantes :
- si t_1 et t_2 sont des termes alors $t_1 = t_2$ et $t_1 \leq t_2$ sont des formules (dites *atomiques*) ;
- si A est une formule alors $\neg A$ est une formule ;
- si A_1 et A_2 sont des formules alors $(A_1 \Rightarrow A_2)$ est une formule ;
- si A est une formule et x_i une variable alors $\forall x_i\ A$ est une formule.

Les formules ne contenant pas de quantificateurs forment le *fragment propositionnel* du langage.

Nous appellerons $\mathcal{L}$ le langage définit ci-dessus.

Ensemble suffisant de symboles. Les trois symboles logiques choisis, et les trois opérations logiques qu'elles symbolisent, sont suffisants pour retrouver les autres expressions logiques usuelles. Le tableau ci-dessous donne les *abréviations* que nous pourrons utiliser.

	Notation	Définition
Disjonction («ou»)	$A_1 \vee A_2$	$\neg A_1 \Rightarrow A_2$
Conjonction («et»)	$A_1 \wedge A_2$	$\neg(A_1 \Rightarrow \neg A_2)$
Existentielle («il existe»)	$\exists x\ A$	$\neg\forall x\ \neg A$

Quantification bornée. Pour toute formule A et tout terme t, on définit les abréviations suivantes :

Notation	Définition
$\forall i \leq t.A$	$\forall i\ ((i \leq t) \Rightarrow A)$
$\exists i \leq t.A$	$\exists i\ ((i \leq t) \wedge A)$

Notons que le point utilisé dans l'abréviation disparaît dans la définition : il ne fait donc pas partie du langage.

Une quantification bornée énonce en fait une propriété pour un *nombre fini* d"entiers. Cela est avantageux du point de vue de la recherche de la validité de la propriété : intuitivement, il est toujours possible d'examiner tous les cas d'un ensemble fini de valeurs. Une propriété exprimée par une quantification bornée est toujours *décidable*. L'usage de la quantification bornée et sa décidabilité sont précisés en 3.4.

Liaison de variable Dans une formule, une variable n'a pas le même sens selon qu'elle est quantifiée ou non. Ça n'est pas la même chose de dire dans le monde arithmétique «*n est pair*» ou «*Pour tout n, n est pair*». Dans le premier cas on parle d'un entier n certe quelconque mais unique chaque fois que l'on considère l'énoncé, alors que dans le second cas on parle de tous les entiers naturels et l'énoncé doit être vrai ou faux pour tous les entiers à la fois (il est ici à l'évidence faux). Dans le second cas, on dit que la variable n est *liée*. Si une variable n'est pas liée, on dit qu'elle est *libre*, et *vice versa*. Pour être plus précis, la notion de liaison ne concerne pas une variable, mais ses *occurences* (ie. ses apparitions) dans une formule. Par exemple la variable x_i a une occurence libre (la première) et une autre liée (la seconde) dans la formule $0 \leq x_1 \vee \forall x_1\ 0 \leq x_1$. Remarquez que le symbole (on dit aussi le *nom*) utilisé pour les variables liées ne joue aucun rôle vis-à-vis de la vérité des formules. C'est-à-dire que $\forall x_1\ 0 \leq x_1$ est logiquement équivalente à $\forall x_2\ 0 \leq x_2$. On a ainsi que $0 \leq x_1 \vee \forall x_1\ 0 \leq x_1$ est logiquement équivalente à $0 \leq x_1 \vee \forall x_2\ 0 \leq x_2$. Mais attention $0 \leq x_1 \vee \forall x_1\ 0 \leq x_1$ *n'est pas* logiquement équivalente à $0 \leq x_2 \vee \forall x_2\ 0 \leq x_2$. On pourra donc s'autoriser (librement :) à changer le nom des variables liées mais on ne touchera pas le nom des variables libres.

On définit la notion *d'occurence libre* d'une variable x_i dans une formule A en suivant les règles de formation des formules :

1. si A est une formule atomique, les occurences (si elles existent) de x_i y sont libres ;
2. si A est la formule $\neg A_1$, les occurences libres de x_i dans A sont les occurences libres de x_i dans A_1 ;
3. si A est la formule $A_1 \Rightarrow A_2$, les occurences libres de x_i dans A sont les occurences libres de x_i dans A_1 ainsi que les occurences libres de x_i dans A_2 ;
4. si A est la formule $\forall x_i\ A_1$ alors x n'a pas d'occurence libre dans A ;
5. si A est la formule $\forall x_j\ A_1$ alors les occurences libres de $x$avec $i \neq j$ dans A sont les occurences libres de x_i dans A_1.

On note $A(x_i)$ une formule A où x_i a une occurence libre, $A(x_i, x_j)$ une formule A où les variables x_i et x_j ont une occurence libre, etc.

Si une variable x n'a pas d'occurence libre dans la formule A, alors toutes ses occurences, si elle en a, sont liées.

Une formule qui ne contient pas de variable libre est appelée un *énoncé*.

2.1.4 La substitution

Nous avons mentionné l'importance de l'usage de l'opération de remplacement ou de substitution d'une référence dans un énoncé pour obtenir le paradoxe de la version purement logique du menteur. Dans le langage des formules, le rôle de référence est tenu par les variables. La substitution est en fait le remplacement *des occurences libres* d'une variable x_i dans une formule A par un terme t. On note $A[t/x_i]$ qui se lit «A *dans laquelle* t *remplace* x_i». On définit cette opération d'abord sur les termes (on note $t[u/x_i]$) puis sur les formules en suivant les règles de leurs formations.

Sur les termes :

1. si t est la constante 0 alors $t[u/x_i]$ est égal à t.
2. si t est une variable alors
   (a) si t est la variable x_i alors $t[u/x_i]$ est égal à u ;
   (b) sinon, $t[u/x_i]$ est égal à t.
3. si t est de la forme $t_1 \star t_2$ (avec $\star$ l'un des $+$, $\times$, $\hat{}$) alors $t[u/x_i]$ est égal à $t_1[u/x_i] \star t_2[u/x_i])$.

Sur les formules :

1. si A est une formule atomique $t_1 \diamond t_2$ (avec $\diamond$ l'un des $=$, $\leq$) alors $A[t/x_i]$ est égale à $t_1[u/x_i] \diamond t_n[u/x_i]$;
2. si A est la formule $\neg A_1$ alors $A[t/x_i]$ est égale à $\neg A_1[t/x_i]$;
3. si A est la formule $A_1 \Rightarrow A_2$ alors $A[t/x_i]$ est égale à $A_1[t/x_i] \Rightarrow A_2[t/x_i]$;
4. si A est la formule $\forall x_i\ A_1$ alors $A[t/x_i]$ est égale à A (on ne fait rien) ;
5. si A est la formule $\forall x_j\ A_1$ avec $i \neq j$ alors il faut considérer deux cas :
   (a) ou bien le symbole x_j n'a pas d'occurence dans t et alors $A[t/x_i]$ est égale à $\forall x_j\ A_1[t/x_i]$;
   (b) ou bien x_j apparaît dans t auquel cas, on change le nom de la variable liée x_j puis on procède à la substitution. Ce qui donne que $A[t/x_j]$ est égale à $\forall x_k\ A_1[x_k/x_j][t/x_i]$ en choisissant[4] un k tel que x_k n'a pas d'occurence ni dans A_1 ni dans t.

Note : si, dans le dernier cas (b), l'on ne prend pas cette précaution de *renommage*, on court le risque qu'une variable libre dans un terme devienne liée dans la formule obtenue. Ce phénomène est appelé *capture de variable*.

2.2 Preuves formelles

Un système formel de preuves est une *théorie de la vérité* donnée par un ensemble d'axiomes et un ensemble de *règles de déduction*. Les axiomes sont des formules *universellement valides* (ie. vraies dans tous les mondes possibles - du moins les mondes qui obéissent aux lois de la logique :), les régles sont chargées de produire de nouvelles vérités à partir de vérités déjà établies.

L'histoire de la logique formelle a produit plusieurs systèmes de preuves : citons les systèmes *à la Hilbert*, le calcul des séquents [], la déduction naturelle [], etc. Gödel a montré son théorème pour le système présenté par Whitehead et Russell dans leurs *Principia mathematica* [2], c'est un système *à la Hilbert*. Mais l'incomplétude n'est pas propre à ce système. Nous utiliserons pour notre part un système *à la Hilbert* faisant plus ou moins partie du folklore. Il est peu naturel pour développer réellement des preuves, mais notre but n'est pas de le manipuler mais d'en explorer les propriétés.

[4] C'est ici qu'il est primordial d'avoir supposé un nombre infini de symboles de variables. Cela permet de toujours pouvoir choisir un tel k.

2.2.1 Axiomes logiques

Les axiomes de la logique sont en fait des *schémas d'axiomes*. Les vrais axiomes sont obtenus en remplaçant les symboles A_1, A_2 et A_3 utilisés ci-dessous par n'importe quelle formule. On dit que l'on a alors une *instance* d'un schéma.

Axiomes propositionnels ces trois axiomes sont des *tautologies* du fragment propositionnelles, c'est-à-dire des formules vraies quelque soit la *valeur de vérité* des formules A_1, A_2 et A_3.

P1 : $A_1 \Rightarrow (A_2 \Rightarrow A_1)$
P2 : $(A_1 \Rightarrow (A_2 \Rightarrow A_3)) \Rightarrow ((A_1 \Rightarrow A_2) \Rightarrow (A_1 \Rightarrow A_3))$
P3 : $(\neg A_1 \Rightarrow \neg A_2) \Rightarrow (A_2 \Rightarrow A_1)$

Axiomes pour la quantification Il y a deux axiomes logiques concernant la quantification. Le premier dit simplement que si une propriété est vrai pour tous, elle est vraie pour un particulier, le second est plus technique et permet de «rentrer » un quantificateur dans une implication sous certaines conditions.

Q1 : $(\forall x_1\ A_1) \Rightarrow A_1[t/x_1]$
Q2 : $\forall x_1\ (A_1 \Rightarrow A_2) \Rightarrow (A_1 \Rightarrow \forall x_1\ A_2)$, si x_1 n'a pas d'occurence libre dans A_1.

2.2.2 L'égalité

Une relation joue un rôle privilégié en mathématique : l'égalité. Abstraitement, l'égalité est une relation réfléxive, symétrique et transitive (ie. une relation d'équivalence) qui *passe au contexte* ; c'est-à-dire qu'en remplaçant, dans un terme ou une formule, un terme par un égal, on obtient un nouveau terme égal au premier ou une formule équivalente à la première. On donne la théorie de l'égalité en posant les axiomes :

E1 : $x_1 = x_1$
E2 : $x_1 = x_2 \Rightarrow x_2 = x_1$
E3 : $x_1 = x_2 \Rightarrow (x_2 = x_3 \Rightarrow x_1 = x_3)$
E4 : $x_1 = x_2 \Rightarrow sx_1 = sx_2$
E5-7 : $x_1 = x_2 \Rightarrow (x_3 = x_4 \Rightarrow (x_1 \star x_3) = (x_2 \star x_4))$, où $\star$ est respectivement l'un des $+$, $\times$, $\hat{}$.
E8-9 : $x_1 = x_2 \Rightarrow (x_3 = x_4 \Rightarrow (x_1 \diamond x_3 \Rightarrow x_2 \diamond x_4))$, où $\diamond$ est respectivement l'un des $=$, $\leq$.

Ces axiomes sont implicitement universellement clos ; ils valent pour tout x_1, x_2, x_3.

2.2.3 Règles de déduction

Une règle de déduction, à l'instar des fonctions, est une opération qui produit une nouvelle formule à partir d'autres formules. On se donne deux règles de déduction, le *modus ponens* et la règle de *généralisation* :

R1 de $A_1 \Rightarrow A_2$ et si A_1 on déduit A_2.
R2 de A_1 on déduit $\forall x_i\ A_1$.

On dit que A_2 et $\forall x_i\ A_1$ sont des *conséquences immédiates* de, rspectivement, $A_1 \Rightarrow A_2$, A_1 et A_1. Nous avions employé le terme «élémentaire»dans notre introduction

2.2.4 Preuves formelles

Nous pouvons préciser ce que l'on entend par *preuve formelle*. Une preuve (formelle) d'une formule A est une suite de formules $A_1, \ldots, A_n$ telles que

1. la formule A est l'un des A_i pour $1 \leq i \leq n$[5] ;
2. toute formule A_i de la suite est
   - soit l'instance d'un axiome ;
   - soit obtenue par application de la régle de déduction R1 à deux formules A_j, A_k avec $j < i$ et $k < i$;

[5] Il suffirait en fait de prendre $A = A_n$ mais cette défintion plus générale facilitera la suite.

– soit obtenue par application de la régle de déduction R2 à une formule A_j avec $j < i$.

Correction la correction d'une preuve formelle (i.e. toute formule prouvée formellement est sémantiquement vraie) est garantie d'une part par la validité sémantique des axiomes et d'autre part par la correction des règles de déduction : si les prémisses sont sémantiquement valides alors la conclusion l'est aussi.

Complétude ce système de logique pure (i.e. sans les axiomes propres de l'arithmétique) est *complet*. C'est-à-dire que toute formule sémantiquement vraie est prouvable et pour toute formule sémantiquement fausse, sa négation est prouvable.

C'est à Kurt Gödel que l'on doit la première preuve de complètude d'un système formel. Il ne s'agit pas exactement du système donné ici pour lequel il faudrait en toute rigueur refaire la preuve de complètude. Nous nous épargnerons cet effort dans la mesure où le jeu d'axiomes et les règles de notre systèmes font partie du folklore.

La complètude du système de logique pure donne en particulier le fait suivant que nous utiliserons par la suite lorsque nous auront besoin de montrer l'existence d'une preuve :

Fait (1) toute tautologie du fragment propositionnel est prouvable.

2.3 Arithmétique formelle

Pour obtenir une arithmétique formelle, il suffit d'ajouter au système de preuves ci-dessus des axiomes propres à l'arithmétique qui énoncent les propriétés attendues de 0, s, $+$, $\times$, $\hat{}$ et $\leq$.

2.3.1 Axiomes pour l'arithmétique

Ici aussi il faut lire les axiomes comme valant pour toute x_1, x_2.
A1 $\neg(0 = sx_1)$
A2 $sx_1 = sx_2 \Rightarrow x_1 = x_2$
A3 $0 + x_2 = x_2$
A4 $sx_1 + x_2 = s(x_1 + x_2)$
A5 $0 \times x_2 = 0$
A6 $sx_1 \times x_2 = x_2 + (x_1 \times x_2)$
A7 $x_1 \hat{} 0 = s^1 0$
A8 $x_1 \hat{} sx_2 = x_1 \times (x_1 \hat{} x_2)$
A9 $(x_1 \leq 0) \Leftrightarrow (x_1 = 0)$
A10 $(x_1 \leq sx_2) \Leftrightarrow (x_1 \leq x_2 \vee x_1 = sx_2)$
A11 $(x_1 \leq x_2) \vee (x_2 \leq x_1)$
A12 pour toute formule A, $A[0/x_1] \Rightarrow (\forall x_1\ (A \Rightarrow A[sx_1/x_1]) \Rightarrow \forall x_1\ A)$

Ce dernier axiome est en fait un *schéma* d'axiome, comme le sont les axiomes logiques 2.2.1. C'est l'axiome dit de *récurrence*, celui qui permet le raisonnement par récurrence en arithmétique formelle. Fait notable, cet axiome n'est pas nécessaire à l'établissement du théorème d'incomplétude. Nous ne le donnons ici que pour être exhaustif vis-à-vis de l'arithmétique.

Désormais les preuves formelles peuvent faire usage des instances des axiomes A1-12. Appelons $\mathcal{S}$ le système formel formé du langage de formule donné en 2.1 ; des axiomes P1-P3 et Q1-Q2 donnés en 2.2.1 ; des axiomes E1-E9 donnés en 2.2.2 ; des axiomes A1-A11 donnés en 2.3 et des régles de déductions données en 2.2.3. Une formule A est dite *prouvable* (implicitement, dans $\mathcal{S}$) s'il existe une preuve formelle (au sens de 2.2.4 étendu aux axiomes A1-A11) de A.

Désormais, notre système est *incomplet* : on va construire une formule qui ne sera pas prouvable et dont la négation ne le sera pas non plus.

3 Expressivité, définissabilité, prouvabilité

3.1 Ensembles exprimables

On dit qu'un ensemble d'entiers E est *exprimable* s'il existe une formule $F(x)$ de $\mathcal{L}$ à une variable libre x telle que pour tout entier n,

$$n \in E \text{ ssi } F(s^n 0) \text{ est vraie}$$

Plus généralement, un ensemble de k-uplets d'entiers E est exprimable s'il existe une formule $F(x_1, \ldots, x_k)$ à k variables libre $x_1, \ldots, x_k$ telle que pour tout entier $n_1, \ldots, n_k$

$$(n_1, \ldots, n_k) \in E \text{ ssi } F(s^{n_1}0, \ldots, s^{n_k}0) \text{ est vraie}$$

Sémantiquement, une relation k-aire n'est autre qu'un ensemble de k-uplets : la notation relationnelle $R(n_1, \ldots, n_k)$ est équivalente à la notation ensembliste $(n_1, \ldots, n_k) \in R$. Une relation k-aire $R(n_1, \ldots, n_k)$ est exprimable s'il existe une formule $F(x_1, \ldots, x_k)$ telle que

$$R(n_1, \ldots, n_k) \text{ ssi } F(s^{n_1}0, \ldots, s^{n_k}0) \text{ est vraie}$$

3.2 Définissabilité

À l'inverse, une formule $F(x_1, \ldots, x_k)$ à k variables libre $x_1, \ldots, x_k$ définit une relation k-aire[6]. On nommera la relation ainsi définie R en posant :

$$R(x_1, \ldots, x_k) := F(x_1, \ldots, x_k)$$

Une relation (k+1)-aire est dite *fonctionnelle* lorsque, pour tout k-uplet $(n_1, \ldots, n_k)$ il existe un unique n_{k+1} tel que $R(n_1, \ldots, n_k, n_{k+1})$. Pour manifester ce caractère fonctionnel, on utilise la notation $n_{k+1} = r(n_1, \ldots, n_k)$. Ainsi, lorsque l'on définira une relation fonctionnelle, on posera

$$y = r(x_1, \ldots, x_k) := F(x_1, \ldots, x_k, y)$$

Intuitivement, si un (k+1)-uplet $(n_1, \ldots, n_k, n_{k+1})$, la valeur de n_{k+1} est uniquement déterminée par (i.e. ne dépend que de) $n_1, \ldots, n_k$. C'est ce qui légitime l'usage du symbole d'égalité et du symbole de fonction r. *Mais attention* : l'usage du signe $=$ dans $y = r(x_1, \ldots, x_k)$ est un abus de langage : il ne s'agit pas du symbole binaire $=$ du langage $\mathcal{L}$. De même, le symbole de fonction r ne fait pas partie de $\mathcal{L}$. On ne s'autorise cet abus que lorsque l'on a affaire à une relation fonctionnelle qui légitime la notation[7].

Ce premier abus de langage en autorise un second. Soit une formule $F(\ldots, x, \ldots)$ une formule où x est libre. Si l'on a défini une relation fonctionnelle $y = r(x_1, \ldots, x_k)$, on pose l'abréviation suivante :

Notation	Définition
$F(\ldots, r(x_1, \ldots, x_k), \ldots)$	$\forall x(x = r(x_1, \ldots, x_k) \Rightarrow F(\ldots, x, \ldots))$

Cette notation n'est légitime que si aucune des variables $x_1, \ldots, x_k$ ne devient liée dans $F(\ldots, r(x_1, \ldots, x_k), \ldots)$. De plus, elle est compositionnelle : $F(\ldots, r_1(\ldots, r_2(x_1, \ldots, x_k), \ldots), \ldots)$ est une abréviation de

$$\forall y_2(y_2 = r_2(x_1, \ldots, x_k) \Rightarrow \forall y_1(y_1 = r_1(\ldots, y_2, \ldots) \Rightarrow F(\ldots, y_1, \ldots)))$$

[6] Cette relation peut être (l'ensemble) *vide* si $F(s^{n_1}0, s^{n_k}0)$ n'est vraie pour aucun k-uplet d'entiers $(n_1, \ldots, n_k)$

[7] Une façon syntaxique de dire les choses est de dire que le nom de la relation est $.. = r(..)$.

3.3 Preuves et équations

Fait (2) Soit t un terme clos, il existe un entier n tel que l'égalité $t = s^n 0$ soit prouvable.

Fait (3) Soient t, u et v trois termes, soit x une variable. On a que si $t = u$ est prouvable alors $v[u/x] = v[t/x]$ est prouvable.

Fait (4) Soient t et u deux termes, x une variable et A une formule. On a que si $t = u$ est prouvable alors $A[u/x] \Rightarrow A[t/x]$ est prouvable.

Des faits (2) et (4) on déduit le corolaire :

Fait (5) Soient A une formule, $x_1, \ldots, x_k$ des variables, $t_1, \ldots, t_k$ des termes. Pour prouver $A[t_1/x, \ldots, t_k/x_k]$, il suffit de prouver $A[s^{n_1}0/x_1, \ldots, s^{n_k}0/x_k]$ avec $t_1 = s^{n_1}0$, ..., $t_k = s^{s_k}0$.

3.4 Fragment Σ_0

On distingue parmi toutes les formules la classe particulière des formules dites Σ_0 ainsi définie :

1. les formules atomiques $t_1 = t_2$ et $t_1 \leq t_2$ sont Σ_0.
2. si A_1 et A_2 sont Σ_0 alors $\neg A_1$ et $A_1 \Rightarrow A_2$ aussi.
3. si A_1 est Σ_0 alors $\forall x \leq t_1.A_1$ avec x n'apparaissant pas dans t_1 est aussi Σ_0.

Proposition (6) Σ_0-complétude : Soit A est un énoncé (formule close) Σ_0. Si A est vraie alors A est prouvable et si A est fausse alors $\neg A$ est prouvable.

Preuve (idée de) On montre la proposition par induction sur A.

- Si A est un énoncé atomique, il est de la forme $t_1 \diamond t_2$ où t_1, ni t_2 ne contiennent de variable. Par 5, il suffit d'établir la provabilité des énoncés de la forme $s^{n_1}0 \diamond s^{n_2}0$.
  - Si $\diamond$ est l'égalité $(=)$ et $s^{n_1}0 = s^{n_2}0$ est vraie alors $n_1 = n_2$ et les termes $s^{n_1}0$ et $s^{n_2}0$ sont le même terme. L'axiome E1 permet de construire la preuve cherchée.
  - Si $\diamond$ est la relation d'infériorité $(\leq)$ et $s^{n_1}0 \leq s^{n_2}0$ est vraie alors
    - Si $n_2 = 0$, c'est-à-dire $t_2 = s^0 0 = 0$ alors, puisque $s^{n_1}0 \leq s^{n_2}0$ est vraie, t_1 est nécessairement égal à 0 et on utilise A9.
    - Sinon $n_2 = n_2' + 1$, par induction sur n_2' : ou bien $n_1 = n_2' + 1$ ou bien $s^{n_1}0 \leq s^{n_2'}0$ (qui est vraie) est prouvable ; on utile l'axiome A10.
- si A est de la forme $A_1 \Rightarrow A_2$, alors
  ou bien A_2 est vraie et donc, par hypothèse d'induction, prouvable. On peut utiliser l'axiome P1.
  ou bien A_2 est fausse alors pour que $A_1 \Rightarrow A_2$ soit vraie, il faut que A_1 soit aussi fausse. Par complètude de la logique pure (2.2.4), $\neg A_1$ est prouvable ainsi que $\neg A_1 \vee A_2$ ainsi que $(\neg A_1 \vee A_2) \Rightarrow (A_1 \Rightarrow A_2)$. On utilise alors le modus ponens.
- si A est de la forme $\forall x \leq t.A_1$. Le terme t est clos. On peut «calculer» sa valeur et prouver l'équivalence de A avec $\forall x \leq s^n 0.A_1$ pour un certain n. Par induction sur n, on peut montrer que pour chaque n, on peut prouver l'équivalence de $\forall x \leq s^n 0.A_1$ avec la proposition $A_1[0/x] \wedge \ldots \wedge A_1[s^n 0/x]$. Or $A_1[0/x] \wedge \ldots \wedge A_1[s^n 0/x]$ est vraie (correction du système de preuve) donc chacune des $A_1[0/x], \ldots A_1[s^n 0/x]$ est vraie ; donc prouvable, par hypothèse d'induction. Par complètude de la logique pure la conjonction $A_1[0/x] \wedge \ldots \wedge A_1[s^n 0/x]$ est aussi prouvable. Ce qui permet, pour chaque n de construire une preuve de A.

4 Le théorème

Nous énonçons dans ce paragraphe le théorème d'incomplétude de Gödel et donnons l'essentiel de sa preuve. Pour cela nous posons la possibilité d'*exprimer* le système $\mathcal{S}$ dans l'arithmétique. Cette possibilité sera rendue effective par la suite (5).

4.1 Codage et prouvabilité

On sait associer à une variable, un terme ou une formule un entier unique que l'on appelle son *numéro de Gödel*, son *nombre de Gödel* voire, plus simplement, son *code*. On sait également associer à une suite de termes ou de formules un numéro de Gödel. On note $\widehat{x_i}$ le code (numéro) de la variable x_i. Plus généralement, on note $\widehat{e}$ le code de n'importe quelle expression (terme ou formule) e du langage.

Soit χ_1 le code de la première variable du langage $\mathcal{L}$ ($\chi_1 = \widehat{x_1}$).

On sait exprimer la relation fonctionnelle entre un entier n et le code de l'entier formel (terme) s^n0. Soit $y = nCode(x)$ cette relation (y est le code de s^x0, i.e. $y = \widehat{s^x0}$).

On sait écrire, dans le langage du système $\mathcal{S}$, une formule à deux variables libres (disons x_1 et x_2) qui soit vraie si et seulement si x_2 code une preuve de la formule (de numéro) x_1. Autrement dit, la relation *«est une preuve de »* est exprimable. Notons $Proves(x_2, x_1)$ cette formule.

Cette formule jouit de la propriété de complétude suivante :

Fait (7) (Complétude restreinte) pour tous entiers n et m, si $Proves(s^n0, s^m0)$ est vraie alors $Proves(s^n0, s^m0)$ est prouvable et si $Proves(s^n0, s^m0)$ est fausse alors $\neg Proves(s^n0, s^m0)$ est prouvable.

Ce fait est une conséquence de la complétude du fragment Σ_0 (cf 3.4). En effet, la formule $Proves(x_2, x_1)$ qui sera définie en fin de section 5 est une formule Σ_0.

4.2 Auto-référence

Nous avons exposé en introduction comment le théorème de Gödel est proche du paradoxe du menteur et nous avons vu comment ce paradoxe repose sur le phénomène d'*auto-référence*. Il nous faut donc obtenir une formule «auto-référente».

Donnons nous une formule A avec une variable libre unique : x. C'est une formule qui parle de x. En remplaçant x dans A par A elle-même, on aurait une formule auto-référente (qui parle d'elle-même). On pourrait imaginer de la construire en faisant appel à la substitution (cf 2.1.4) : $A[A/x]$. On ne peut cependant pas procéder aussi directement car dans la définition de la substitution $A[t/x]$, t est un terme et non une formule.

Mais nous avons dit ci-dessus qu'à chaque formule on savait associer un nombre entier : son code. Soit donc α le code de la formule A ($\alpha = \widehat{A}$). Le terme $s^\alpha 0$ est la représentation formelle de l'entier α. La formule $A[s^\alpha 0/x]$ est correcte et elle *exprime* l'auto-référence de A.

Il nous reste encore un pas à franchir : notre intention est d'exprimer la prouvabilité de $A[s^\alpha 0/x]$ en utilisant la formule $Proves(x_2, x_1)$ où l'on remplacerait x_1 par la formule auto-référente. Mais ici encore l'expression que l'on écrirait $Proves(x_2, A[s^\alpha 0/x])$ n'est pas une formule correcte (il faut un terme à la place de x_1 et $A[s^\alpha 0/x]$ est une formule).

La substitution $A[t/x]$ définit une relation fonctionnelle quaternaire entre la formule A, la variable x, le terme t et la formule résultant de la substitution. Formules, variables et termes sont codables : on peut donc associer une relation fonctionnelle sur les entiers à la relation fonctionnelle définie par la substitution. Notons $y = subst(x_1, x_2, x_3)$ cette relation (y est le résultat de la substitution du terme x_3 aux occurences libres de la variable x_2 dans la formule x_1). L'autoréférence recherchée est alors exprimée par $y = subst(\alpha, \widehat{x}, nCode(\alpha))$, où $\widehat{x}$ est le code de la variable x.

Nous utilisons l'abus d'écriture pour les relations fonctionnelles (cf 3.2) dans notre usage des relations $nCode$ et $subst$.

4.3 ω-cohérence

Le théorème de Gödel fait naturellement l'hypothèse de cohérence du système $\mathcal{S}$: quelque soit la formule A, on ne peut avoir à la fois que A est prouvable et que $\neg A$ est prouvable. En fait, dans son article original [3], Gödel fait l'hypothèse d'une cohérence plus exigeante, l'*ω-cohérence*, qui s'énonce : quelque soit la formule $A(x_i)$ on ne peut avoir à la fois que pour tout n, $A(s^n 0)$ est prouvable et que $\neg\forall x_i F(x_i)$ est prouvable. Notons que tout système ω-cohérent est cohérent.

4.4 Énoncé et preuve du théorème

Énoncé Si $\mathcal{S}$ est ω-cohérent alors il existe une formule G telle que ni G ni $\neg G$ n'est prouvable.

Preuve La preuve consiste à construire la formule G puis montrer qu'elle satisfait l'énoncé.

Soit $Q(x_2, x_1)$ **:=** $Proves(x_2, sub(x_1, \chi_1, nCode(x_1))$ i.e. x_2 est une preuve de la formule («autoréférente») $sub(x_1, \chi_1, nCode(x_1))$.

Soit $G_0(x_1)$ **:=** $\forall x_2\, \neg Q(x_2, x_1)$ **:=** $\forall x_2\, \neg Proves(x_2, sub(x_1, \chi_1, nCode(x_1))$ i.e. la formule («autoréférente») $sub(x_1, \chi_1, nCode(x_1))$ n'est pas prouvable. Soit γ_0 le code (le numéro de Gödel) de G_0 ; rappelons que le code de la variable x_1 est χ_1.

Soit G **:=** $G_0(\gamma_0)$ **:=** $\forall x_2\, \neg Q(x_2, \gamma_0)$ intuitivement : *«je ne suis pas prouvable »*. Notons que le code de G est $sub(\gamma_0, \chi_1, nCode(\gamma_0))$ (le code de γ_0 est $nCode(\gamma_0)$).

- G n'est pas prouvable : supposons qu'au contraire G soit prouvable. Il existerait alors une preuve (suite de formules) de G. Soit alors π_0 le code de cette preuve. La formule $Proves(\pi_0, sub(\gamma_0, \chi_1, nCode(\gamma_0)))$ est un énoncé vrai, et donc, par 7 (complétude restreinte), elle est prouvable.
  Mais la formule G est égale à $\forall x_2\, \neg Proves(x_2, sub(x_1, \chi_1, nCode(x_1))$. Si elle est prouvable, alors le cas particulier $\neg Proves(\pi_0, sub(\gamma_0, \chi_1, nCode(\gamma_0)))$ est également prouvable (règle d'instanciation).
  On aurait donc qu'à la fois $Q(\pi_0, \gamma_0)$ et $\neg\ Q(\pi_0, \gamma_0)$ serait toutes deux prouvables. Ce qui contredit l'hypothèse de cohérence de la théorie.
- $\neg G$ n'est pas prouvable : supposons qu'au contraire $\neg G$ est prouvable, c'est-à-dire (par définition de G) que $\neg\forall x_1\, \neg Q(x_1, \gamma_0)$ est prouvable.
  Mais nous venons de voir que G n'est pas prouvable. On a donc que pour tout entier π, la formule $\neg Proves(\pi, sub(\gamma_0, \chi_1, nCode(\gamma_0)))$, c'est-à-dire $\neg Q(\pi, \gamma_0)$, est un énoncé vrai, donc prouvable (complètude restreinte).
  On aurait donc qu'à la fois pour tout entier π, $\neg Q(\pi, \gamma_0)$ et que $\neg\forall x_2\, \neg Q(x_2, \gamma_0)$. Ce qui contredit l'hypothèse d'ω-cohérence.

5 Codage

La formule auto-référente construite par Gödel utilise le truchement d'un *codage* du système logique dans le monde des nombres (l'arithmétique). Au vrai, ce qui est codé par des nombres sont les formules et les suites de formules (les preuves). Pour obtenir la formule auto-référente, il faut également savoir exprimer certaines propriétés sur les nombres telles *«un nombre est le code d'une formule»*, *«une formule est une implication»*, *«un nombre est le code d'une suite de formules»*, *«une suite de formules est une preuve»*, etc. Le travail préparatoire à l'énoncé de la formule auto-référente consiste donc à

1. déterminer un encodage des formules et des suites de formules ;
2. donner la définition des formules nécessaires à l'expression de la propriété *«n'est pas prouvable»*.

Afin de faciliter et rendre plus intuitif l'introduction du codage nous procèderons en deux étapes. Dans un premier temps, nous montrerons comment écrire les formules et les preuves comme des suites ou *chaînes* de caractères et nous définirons un ensemble de relation sur les chaînes aboutissant à l'expression de la prouvabilité (la relation *Proves* introduite en 4.1). Nous procèderons au passage à un «absconsissement» de la syntaxe. C'est dans un second temps, à la section 6, que nous montrerons comment les relations de base sur les chaînes de caractères peuvent se réduire à de pures fonctions ou relations sur les nombres pour peu que l'on donne à chaque caractère une valeur (un code) numérique.

Rappel Le langage des formules donné en 2.1 est résumé par les règles de grammaires suivantes :

$$\begin{array}{lcll} \text{Variables} & v & ::= & x_1 \mid x_2 \mid \ldots \\ \text{Termes} & t & ::= & 0 \mid v \mid st \mid t+t \mid t \times t \mid t \hat{} t \\ \text{Formules} & f & ::= & (t=t) \mid (t \leq t) \mid (\neg f) \mid (f \Rightarrow f) \mid \forall v(f) \end{array}$$

5.1 Absconsissement

Le tableau ci-dessous donne le jeu de caractères utilisé pour l'écriture absconse des formules ainsi que la correspondance que nous utiliserons entre symboles usuels, caractères et, par anticipation, codes numériques.

symbole usuel	caractère abscons	valeur numérique
0	`'Z'`	1
s	`'S'`	2
$+$	`'P'`	3
$\times$	`'M'`	4
$\hat{}$	`'X'`	5
$=$	`'E'`	6
$\leq$	`'L'`	7
$\neg$	`'N'`	8
$\Rightarrow$	`'I'`	9
$\forall$	`'A'`	10
(	`'('`	11
)	`')'`	12
v	`'V'`	13
!	`'!'`	14

Tab. 1

Toutes les variables sont écrites avec le seul symbole `'V'` : x_1 devient `V`, x_2 devient `VV`, etc. On n'a ainsi plus besoin d'un ensemble infini de symboles de variables tout en conservant un ensemble infini de variables.

L'introduction dans notre langage du caractère `'!'` sera justifiée en 5.3.

Le tableau ci-dessous redéfinit les ensembles de variable (v) de termes (t) et de formules (f) comme des suites de caractères choisis parmi les 13 premiers du tableau ci-dessus.

$$\begin{array}{lcl} v & ::= & \texttt{V} \mid \texttt{V}v \\ t & ::= & \texttt{Z} \mid v \mid \texttt{S}t \mid \texttt{P(}t\texttt{)(}t\texttt{)} \mid \texttt{M(}t\texttt{)(}t\texttt{)} \mid \texttt{X(}t\texttt{)(}t\texttt{)} \\ f & ::= & \texttt{E(}t\texttt{)(}t\texttt{)} \mid \texttt{L(}t\texttt{)(}t\texttt{)} \mid \texttt{N(}f\texttt{)} \mid \texttt{I(}f\texttt{)(}f\texttt{)} \mid \texttt{A}v\texttt{(}f\texttt{)} \end{array}$$

La syntaxe est légèrement revue. Par exemple, l'axiome A1 s'écrit désormais : `N(E(Z)(SV))`.

5.2 Les chaînes de caractères

Une chaîne de caractères est tout simplement une suite de caractères, un mot dans un alphabet donné (les caractères). L'opération de construction des chaînes est la *concaténation* : mise bout à bout de deux

chaînes de caractères. On note $x \cdot y$ la chaîne résultant de la concaténation de x et y.

Il est usuel (en informatique) de dénoter une chaîne de caractères par la suite des caractères qui la compose mis entre guillemets (anglais : double apostrophe). Nous utiliserons cette notation. Par exemple, la chaîne qui code l'axiome A1 s'écrit : `"N(E(Z)(SV))"`.

Relations sur les chaînes

Nous serons assez laxistes sur l'usage des noms de variables, mais gardons en tête qu'en toute rigueur il faudrait les (re)nommer dans $x_1, x_2, x_3, \ldots$. Nous n'insisterons pas sur le caractère fonctionnel des relations définies. Le lecteur peut, s'il le désire le vérifier.

Caractère $Char(x)$: x est un caractère. Cette formule sera définie en 6.

Soit A une formule et x une variable, on définit l'abréviation

Notation	Définition
$\forall x : Char.A$	$\forall x.(Char(x) \Rightarrow A)$

Longueur d'une chaîne $n = len(x)$: (la chaîne) x est de longueur n ; ou encore, la chaîne x contient n caractères. Cette formule sera définie en 6.

Soit A une formule, pour toutes variables i et x, on définit les abréviations suivantes :

Notation	Définition
$\forall i \leq len(x).A$	$\forall n \leq x.(n = len(x) \Rightarrow \forall i \leq n.A)$
$\exists i \leq len(x).A$	$\forall n \leq x.(n = len(x) \Rightarrow \exists i \leq n.A)$

Concaténation $y = x_1 \cdot x_2$: y est le résultat de la concaténation de x_1 et x_2. Cette relation (fonctionnelle) sera définie en 6.

Concernant la concaténation, nous utiliserons systématiquement l'abus de notation introduit en 3.2 pour les relations fonctionnelles. L'opération de concaténation étant de surcroît associative ($(x_1 \cdot (x_2 \cdot x_3) = (x_1 \cdot x_2) \cdot x_3$), on omettra les parenthèses pour écrire simplement $x_1 \cdot x_2 \cdot x_3$.

i-ième caractère d'une chaîne (index) $c = strNth(x, i)$: le caractère c est le i-ième caractère de x, ou c apparaît dans x en position i. La première position dans une chaîne est 0.

$$c = strNth(x, i) := Char(c) \wedge \exists x_1 \leq x. \exists x_2 \leq x.(i = len(x_1) \wedge (x = x_1 \cdot c \cdot x_2))$$

Dernier caractère $c = strLast(x)$: c est le dernier caractère de x.

$$c = strLast(x) := Char(c) \wedge \exists x_1 \leq x. x = x_1 \cdot c$$

Appartenance $StrMem(c, x)$: le caractère c a une occurrence dans la chaîne x.

$$StrMem(c, x) := \exists i \leq len(x).Nth(x, i, c)$$

5.3 Listes indéxées

Une liste indexée est une suite de couples de la forme $(0, a_0)(1, a_1) \ldots (n, a_n)$. On dit que a_i est *l'élément d'indice i* d'une telle suite (avec $i \leq n$). L'emploi de l'article défini (l') est légitime dans la mesure où, par définition, les indices sont tous différents : la relation *«élément d'indice .. de .. »* est fonctionnelle. On dit simplement que x *est élément* d'une telle suite lorsqu'il existe i tel que x est l'élément d'indice i de la suite.

Pour coder ces suites, on utilise un caractère *séparateur*. Comme on ne codera que des suites de termes ou de formules, on peut prendre comme caractère séparateur `' ! '` : il n'intervient pas dans le codage des formules. La suite $(0, a_0)(1, a_1) \dots (n, a_n)$ est alors codée par la chaîne résultant de la concaténation

$$\texttt{" ! !"} \cdot \widehat{0} \cdot \texttt{" !"} \cdot \widehat{a_0} \cdot \texttt{" ! !"} \cdot \widehat{1} \cdot \texttt{" !"} \cdot \widehat{a_1} \cdot \texttt{" ! !"} \cdot \dots \cdot \texttt{" ! !"} \cdot \widehat{n} \cdot \texttt{" !"} \cdot \widehat{a_n} \cdot \texttt{" ! !"}$$

Listes de couples (reconnaisseur) $PList(x) : x$ est une liste de couples.

$$\begin{aligned} PList(x) \quad \textbf{:=} \quad & \neg StrMem(\texttt{" ! ! !"}, x) \wedge \\ & \forall y \leq x.(\neg StrMem(\texttt{' !'}, y) \wedge StrMem(\texttt{" !"} \cdot y \cdot \texttt{" !"}, x) \\ & \quad \Rightarrow \exists z \leq x.(\neg StrMem(\texttt{' !'}, z) \wedge \\ & \qquad (StrMem(\texttt{" ! !"} \cdot y \cdot \texttt{" !"} \cdot z \cdot \texttt{" ! !"}, x) \vee StrMem(\texttt{" ! !"} \cdot z \cdot \texttt{" !"} \cdot y \cdot \texttt{" ! !"}, x)))) \end{aligned}$$

En français : x représente une liste indexée si elle ne contient jamais trois occurences consécutives du caractère `' !'` et si, à chaque fois qu'y apparaît une sous chaîne de la forme $\texttt{" !"} \cdot y \cdot \texttt{" !"}$, avec y ne contenant pas `' !'`, alors elle contient une sous-chaîne de la forme $\texttt{" ! !"} \cdot y \cdot \texttt{" !"} \cdot z \cdot \texttt{" ! !"}$ ou une sous-chaîne de la forme $\texttt{" ! !"} \cdot z \cdot \texttt{" !"} \cdot y \cdot \texttt{" ! !"}$ pour un certain z qui ne contient pas `' !'`.

Listes indexées (reconnaisseur) $IList(x) : x$ est une liste de couples telle que la première composante du premier couple est (le code de) 0 et si deux couples (n_1, x_1) et (n_2, x_2) sont consécutifs dans x alors $n_2 = sn_1$.

$$\begin{aligned} IList(x) \quad \textbf{:=} \quad & PList(x) \wedge \\ & \exists y \leq x.\exists x' \leq x.(x = \texttt{" ! !"} \cdot \texttt{"Z"} \cdot \texttt{" !"} \cdot y \cdot \texttt{" ! !"} \cdot x') \wedge \\ & \forall n_1 \leq x.\forall y_1 \leq x.\forall n_2 \leq x.\forall y_2 \leq x. \\ & \quad (StrMem(\texttt{" ! !"} \cdot n_1 \cdot \texttt{" !"} \cdot y_1 \cdot \texttt{" ! !"} \cdot n_2 \cdot \texttt{" !"} \cdot y_2 \cdot \texttt{" ! !"}, x) \\ & \qquad \Rightarrow n_2 = \texttt{"S"} \cdot n_1) \end{aligned}$$

Notons que, par définition, si (n_1, x_1) est un couple de x (i.e. $\texttt{" ! !"} \cdot n_1 \cdot \texttt{" !"} \cdot x_1 \cdot \texttt{" ! !"}$ une sous-chaîne de x) alors n_1 est nécessairement (le code d') un entier (formel).

i-ème élément d'une liste indexée $x = iListNth(s, i) : x$ est l'élément d'indice i de s.

$$x = iListNth(s, i) \textbf{ := } StrMem(\texttt{" ! !"} \cdot nCode(i) \cdot \texttt{" !"} \cdot x \cdot \texttt{" ! !"}, s)$$

Note $nCode(i)$ est défini en section 5.4.

Appartenance à une liste indexée $IListMem(y, x) : y$ est un élément de la liste indexée x.

$$IListMem(y, x) \textbf{ := } \exists i \leq x.y = iListNth(x, i)$$

Prédécesseur dans une liste : $IListPred(y_1, y_2, x) : y_1$ est un élément précédent y_2 dans la liste indexée x.

$$IListPred(y_1, y_2, x) \textbf{ := } \forall i_2 \leq x.(y_2 = iListNth(x, i_2) \Rightarrow \exists i_1 \leq i_2.y_1 = iLstNth(x, i_1))$$

5.4 Termes

Termes (constructeurs/abréviations)

Zéro :	$zero$	**:=**	$\texttt{"Z"}$
Successeur :	$succ(x_1)$	**:=**	$\texttt{"S"} \cdot x_1$
Addition :	$plus(x_1, x_2)$	**:=**	$\texttt{"A("} \cdot x_1 \cdot \texttt{")("} \cdot x_2 \cdot \texttt{")"}$
Multiplication :	$mult(x_1, x_2)$	**:=**	$\texttt{"M("} \cdot x_1 \cdot \texttt{")("} \cdot x_2 \cdot \texttt{")"}$
Exponentiation :	$exp(x_1, x_2)$	**:=**	$\texttt{"X("} \cdot x_1 \cdot \texttt{")("} \cdot x_2 \cdot \texttt{")"}$

Variable (reconnaisseur) $Var(x)$: x est (le code d') une variable : x *ne contient que le caractère* `'V'`.

$$Var(x) \coloneqq \forall c : Char.(StrMem(c, x) \Rightarrow (c = \texttt{'V'}))$$

Entier formel (reconnaisseur) $Num(x)$: x est (le code d') un entier formel : *le dernier caractère de* x *est* `'Z'` *et tous les autres sont* `'S'`.

$$Num(x) \coloneqq \forall c : Char.(StrMem(c, x) \Rightarrow ((c = strLast(x) \wedge (c = \texttt{'Z'})) \vee (\neg(c = strLast(x)) \wedge (c = \texttt{'S'}))))$$

Entier formel (code d'un) $x = nCode(n)$: x est le code de l'entier formel s^n0.

$$x = nCode(n) \coloneqq Num(x) \wedge len(x) = sn$$

Opération $Op(x_1, x_2, x)$: x a la forme d'une opération arithmétique entre x_1 et x_2.

$$Op(x_1, x_2, x) \coloneqq (x = plus(x_1, x_2)) \vee (x = mult(x_1, x_2)) \vee (x = exp(x_1, x_2))$$

5.4.1 Syntaxe

Dans ce paragraphe, nous allons voir comment reconnaître dans les chaînes de caractères celles d'entre elles qui correspondent aux termes de notre langage. Ce processus s'appelle *l'analyse syntaxique*. Pour analyser une chaîne en vue d'y reconnaître une construction syntaxique d'un terme, on s'appuie sur la définition des termes donnée en 2.1.2. La difficulté que nous devons affronter ici est le caractère *récursif* de cette définition : rien dans notre théorie formalisée de l'arithmétique ne nous autorise l'usage des définitions récursives. Il va falloir ruser.

Représentation des relations récursives Intuitivement, l'analyse d'une chaîne comme `"P(x)(s0)"` pour y reconnaître un terme se déroule ainsi :

1. `"P(x)(s0)"` commence par `'P'`, ce sera un terme si `"x"` et `"s0"` sont des termes.
2. `"x"` est une variable, donc c'est un terme.
3. `"s0"` est un entier formel (c'est donc un terme).

L'analyse a ainsi décomposé la chaîne de départ en une suite de trois chaînes (`"x"`, `"s0"`, `"P(x)(s0)"`) telle que chaque élément est soit une variable (`"x"`), soit un entier formel (`"s0"`), soit obtenu par application d'une des *règles de formation* des termes appliquée à des éléments venant avant dans la suite (`"P(x)(s0)"`).

On peut donc dire qu'une chaîne x représente un terme s'il existe une suite (liste indexée) s telle que :
- x est élément de s ;
- et pour chaque élément y_1 de la suite :
  - ou bien il s'agit (du code) d'une variable ;
  - ou bien il s'agit (du code) d'un entier formel ;
  - ou bien il existe deux éléments y_2 et y_3 précédant y_1 dans s tels que y_1 est une opération arithmétique entre y_2 et y_3.

On appelle une telle suite une *suite de construction* de terme.

Taille maximale Si x code effectivement un terme, il est possible de déterminer, en fonction de x, la taille que ne peut excéder une suite de construction de terme s pour x. Notons σ_x cette borne. Posons la égale à $x^4 + x^3 + 3x^2 + 2$. Notons qu'elle s'exprime en fonction de x et de nos trois opérations arithmétiques. Elle excéde à l'évidence le minimum nécessaire. L'important ici n'est pas tant qu'elle soit juste mais qu'elle existe. La justification de cette formule est la suivante :

1. une suite de construction d'un terme x est une suite de couples $(0, x_0)(1, x_1) \dots (n, x_n)$ où chacun des x_i est une sous-chaîne de x.

2. le nombre de sous-chaînes de x est égal à $x + (x-1) + \ldots + 1$ qui est le nombre de sous-chaînes de longueur 1 plus le nombre de sous-chaînes de longueur 2 plus ...plus le nombre de sous-chaînes de longueur $len(x)$.
3. $x + (x-1) + \ldots + 1$, qui contient x termes, est bornée par x^2.
4. les indices de la suite de construction s sont bornés par le nombre de sous-chaînes, soit x^2.
5. chaque sous-chaîne de x apparaissant dans s est bornée par x.
6. la taille d'un couple de s est bornée par $x^2 + x + 3$ (taille de l'indice, plus taille de la sous-chaîne, plus 3 pour les `'!'`).
7. la taille de s est donc bornée par $x^2(x^2+x+3)+2$ (plus 2 pour les `'!'` extérieurs), soit $x^4+x^3+3x^2+2$.

Suite de construction de terme $TermSeq(x)$: x est une suite de construction de terme : *x est une liste indexée dont chaque élément est soit une variable, soit un entier formel soit une opération arithmétique.*

$$\begin{aligned} TermSeq(x) \quad \textbf{:=} \quad & IList(x) \wedge \\ & \forall y \leq x.(IListMem(y,x) \Rightarrow \\ & \quad (\ Var(x) \vee \\ & \quad\ Num(x) \vee \\ & \quad\ \exists y_1 \leq x.\exists y_2 \leq x.(IListPred(y_1,y,x) \wedge IListPred(y_2,y,x) \wedge Op(y_1,y_2,y)))) \end{aligned}$$

Terme $Term(x)$: x est un terme : *il existe une suite de construction de terme dont x est un élément.*

$$Term(x) \textbf{ := } \exists s \leq \sigma_x.(TermSeq(s) \wedge IListMem(x,s))$$

5.5 Formules

On reconduit pour les formules ce qui a été fait pour les termes.

Formules (constructeurs/abréviations)

Égalité :	$eq(x_1,x_2)$	:=	`"E("` $\cdot x_1 \cdot$ `")("` $\cdot x_2 \cdot$ `")"`
Infériorité :	$le(x_1,x_2)$	:=	`"L("` $\cdot x_1 \cdot$ `")("` $\cdot x_2 \cdot$ `")"`
Négation :	$neg(x_1)$	:=	`"N("` $\cdot x_1 \cdot$ `")"`
Implication :	$impl(x_1,x_2)$	:=	`"I("` $\cdot x_1 \cdot$ `")("` $\cdot x_2 \cdot$ `")"`
Quantification :	$all(x_1,x_2)$	:=	`"A"` $\cdot x_1 \cdot$ `"("` $\cdot x_2 \cdot$ `")"`

Formule atomique (reconnaisseur) $Atom(x)$: x est une formule atomique.

$$Atom(x) \textbf{ := } \exists x_1 \leq x.\exists x_2 \leq x.(Term(x_1) \wedge Term(x_2) \wedge ((x = eq(x_1,x_2)) \vee (x = le(x_1,x_2))))$$

Suite de construction de formule $FormSeq(x)$: x est une suite de construction de formule : *x est une liste indexée dont chaque élément est soit une formule atomique, soit la négation d'un élément précédent de la liste, soit l'implication entre deux éléments précédents de la liste soit la quantification d'un élément précédent de la liste.*

$$\begin{aligned} FormSeq(x) \quad \textbf{:=} \quad & List(x) \wedge \\ & \forall y \leq x.(IListMem(y,x) \Rightarrow \\ & \quad (\ Atom(y) \vee \\ & \quad\ (\exists y_1 \leq x.(IListPred(y_1,y,x) \wedge (y = neg(y_1)))) \vee \\ & \quad\ (\exists y_1 \leq x.\exists y_2 \leq x.(IListPred(y_1,y,x) \wedge IListPred(y_2,y,x) \wedge (y = imp(y_1,y_2)))) \vee \\ & \quad\ (\exists y_1 \leq x.\exists z \leq x.(IListPred(y_1,y,x) \wedge Var(z) \wedge (y = all(z,y_1)))))) \end{aligned}$$

Suite de construction d'une formule $FormSeqOf(s,x)$: s est une suite de construction de formule pour x.

$$FormSeqOf(s,x) := FormSeq(s) \wedge IListMem(x,s)$$

Formule (honnête) $Form(x)$: x est une formule ; elle est «honnête» en ce qu'aucune de ses variables libres n'y a également d'occurence liée. On exprime l'honnêteté en utilisant les suites de construction de formule : si une formule de la forme $\forall x.A$ apparaît à un moment dans une suite de construction s, alors la variable x ne peut plus apparaître ni dans une formule atomique (i.e. x n'apparaît dans aucune suite de construction de la formule atomique) ni comme variable quantifiée d'une autre formule de la forme $\forall x'.A'$ dans s.

$$\begin{array}{lll} Form(x) & := & \exists s \leq \sigma_x.FormSeqOf(s,x) \wedge \\ & & \quad \forall y \in s.\forall v \leq y.\forall w \leq y.(y = all(v,w) \Rightarrow \\ & & \quad\quad \forall y' \in s.(IListPred(y,y',s) \Rightarrow \\ & & \quad\quad\quad (Atom(y') \Rightarrow \forall s' \leq s.(FormSeqOf(s',y') \Rightarrow \neg IListMem(v,s'))) \wedge \\ & & \quad\quad\quad (\forall v' \leq y'.\forall w' \leq y'.(y' = all(v',w') \Rightarrow \neg(v = v'))))) \end{array}$$

Réclamer l'honnêteté des formules vise à faciliter l'expression de l'opération de substitution.

Couple honnête pour la substitution $FaitPair(x,y)$: x est une formule, y un terme et aucune variable de y n'a d'occurence liée dans x : on ruse en demandant que $impl(x, eq(y, zero))$ soit une formule (honnête).

$$FairPair(x,y) := Form(impl(x, eq(y, zero)))$$

Si (A,t) forme un couple honnête pour la substitution, on peut remplacer une variable x de A par t sans avoir à se soucier de renommage car comme on a garanti qu'aucune variable de t n'a d'occurence liée dans A, il n'y a pas de risque de capture de nom de variable. De surcroît, dans A, un variable libre x ne peut y avoir d'ocurence liée, autrement dit, toute occurence de x dans A est nécessairement libre.

Variable libre $Free(v,x)$: la variable v est libre dans la formule x. On utilise les suites de construction de formules : v est libre dans x si x apparaît dans une suite de construction de x et n'est jamais quantifiée. C'est suffisant avec notre hypothèse d'honnêteté des formules.

$$\begin{array}{lll} Free(v,x) & := & Var(v) \wedge Form(x) \wedge \\ & & \forall s \leq \sigma_x.(FormSeqOf(s,x) \wedge IlistMem(v,s) \Rightarrow \\ & & \quad \forall y \leq s.((IlistMem(y,s) \wedge \exists y' \leq y.\exists v' \leq y.y = all(v',y')) \Rightarrow \neg(v = v'))) \end{array}$$

5.6 Substitution

Pour définir la substitution $t' = t[u/x]$, l'idée est d'utiliser une suite de construction s du terme t pour construire une suite *parallèle* de termes s' pour t' telle que : à chaque fois que x apparaît dans s, u apparaît dans s', si une variable $y \neq x$ apparaît dans s alors elle apparaît dans s' et si un terme t_i apparaît dans s comme résultant d'une opération entre deux prédécesseurs de t_i dans s alors un terme t'_i apparaît dans s' comme résultant de la même opération entre deux éléments de s' de mêmes indices. Cette idée est reconduite pour définir la substitution dans les formules. On fait alors également intervenir la notion définie ci-dessus de couple honnête.

Suite de substitution (pour les termes) $s' = tSubstSeq(s,x,u)$: s' est une suite de substitution pour la suite (de termes) s ; pour chaque indice i de s, si t_i est le terme d'indice i de s alors il existe un terme t'_i d'indice i de s' tel que

1. si t_i est la variable x alors $t'_i = u$.

2. si t_i est une variable différente de x alors $t'_i = t_i$.
3. si t_i est un entier formel alors $t'_i = t_i$.
4. si t_i est une opération entre deux prédécesseurs t_{i_1}, t_{i_2} dans s alors t'_i est la même opération entre deux prédécesseurs de mêmes indices t'_{i_1} et t'_{i_2} de t'_i dans s'.

$$
\begin{array}{lll}
s' = tSubstSeq(s,x,u) & := & \forall i \leq len(s).\forall t \leq s.(t = iListNth(s,i) \\
& & \quad \Rightarrow \exists t' \leq s'.(t' = iListNth(s',i) \wedge \\
& & \quad\quad (t = x \Rightarrow t' = u) \wedge \\
& & \quad\quad (Var(t) \wedge \neg(t = x) \Rightarrow t' = t) \wedge \\
& & \quad\quad (Num(t) \Rightarrow t' = t) \wedge \\
& & \quad\quad (\forall i_1 \leq i.\forall t_1 \leq s.\forall i_2 \leq i.\forall t_2 \leq s. \\
& & \quad\quad\quad (t_1 = iListNth(s,i_1) \wedge t_2 = iListNth(s,i_2) \\
& & \quad\quad\quad\quad \Rightarrow \exists t'_1 \leq s'.\exists t'_2 \leq s'. \\
& & \quad\quad\quad\quad\quad (t'_1 = iListNth(s',i_1) \wedge t'_2 = iListNth(s',i_2) \wedge \\
& & \quad\quad\quad\quad\quad ((t = plus(t_1,t_2) \wedge t' = plus(t'_1,t'_2)) \vee \\
& & \quad\quad\quad\quad\quad\quad (t = mult(t_1,t_2) \wedge t' = mult(t'_1,t'_2)) \vee \\
& & \quad\quad\quad\quad\quad\quad (t' = exp(t'_1,t'_2) \wedge t' = exp(t'_1,t'_2)))))))))
\end{array}
$$

En conséquence, par définition de $s' = tSubstSeq(s,x,u)$ et de la substitution, pour tout terme t_i dans s, on a t'_i de même indice dans s' tel que $t'_i = t_i[u/x]$.

Substitution (pour les termes) $t' = tSubst(t,x,u)$: t' est le terme résultat du remplacement des occurences de la variable x dans le terme t par le terme u.

$$
\begin{array}{lll}
t' = tSubst(t,x,u) & := & Term(t) \wedge Var(x) \wedge Term(u) \wedge \\
& & \forall s \leq \sigma_t.\forall s' \sigma_{t'}.(TermSeq(s) \wedge IListMem(t,s) \wedge s' = tSubstSeq(s,x,u) \\
& & \quad \Rightarrow \forall i \leq len(s).(t = iListNth(s,i) \Rightarrow t' = iListNth(s',i)))
\end{array}
$$

Substitution atomique $a' = subst0(a,x,t)$: a est une formule atomique et a' est le résultat de la substitution du terme t à la variable x dans a.

$$
\begin{array}{lll}
a' = subst0(a,x,t) & := & Atom(a) \wedge \\
& & \forall t_1 \leq a.\forall t_2 \leq a.\exists t'_1 \leq a'.\exists t'_2 \leq a'. \\
& & \quad (t'_1 = tSubst(t_1,x,t) \wedge t'_2 = tSubst(t_2,x,t) \wedge \\
& & \quad\quad ((a = eq(t_1,t_2) \wedge a' = eq(t'_1,t'_2)) \vee \\
& & \quad\quad\quad (a = le(t_1,t_2) \wedge a' = le(t'_1,t'_2)))
\end{array}
$$

Suite de substitution (pour les formules) $s' = substSeq(s,x,t)$: s' est une suite de substitution pour la suite (de construction de formule) s telle que pour tout y d'indice i de s, il existe un élément y' d'indice i dans s' tel que y' est le résultat de la substitution du terme t à la variable x dans y. Comme pour les termes,

on suit les clauses de la définition de la substitution.

$$
\begin{array}{lcl}
s' = substSeq(s,x,t) & := & \forall i \leq len(s).\forall y \leq s.(y = iListNth(s,i) \Rightarrow \\
& & \quad \exists y' \leq s'.(y' = iListNth(s',i) \wedge \\
& & \quad\quad (Atom(y) \Rightarrow y' = subst0(y,x,t)) \wedge \\
& & \quad\quad (\forall i_1 \leq i.\forall y_1 \leq s.\forall y'_1 \leq s'. \\
& & \quad\quad\quad (y_1 = iListNth(s,i_1) \wedge y = neg(y_1) \wedge y'_1 = iListNth(s',i_1) \\
& & \quad\quad\quad\quad \Rightarrow y' = neg(y'_1))) \\
& & \quad\quad (\forall i_1 \leq i.\forall y_1 \leq s.\forall y'_1 \leq s'.\forall i_2 \leq i.\forall y_2 \leq s.\forall y'_2 \leq s'. \\
& & \quad\quad\quad (y_1 = iListNth(s,i_1) \wedge x_2 = iListNth(s,i_2) \wedge y = impl(y_1,y_2) \wedge \\
& & \quad\quad\quad\ y'_1 = iListNth(s',i_1) \wedge y'_2 = iListNth(s',i_2) \\
& & \quad\quad\quad\quad \Rightarrow y' = impl(y'_1,y'_2))) \\
& & \quad\quad (\forall i_1 \leq i.\forall y_1 \leq s.\forall x' \leq s.\forall y'_1 \leq s'. \\
& & \quad\quad\quad (y_1 = iListNth(s,i_1) \wedge Var(x') \wedge y = all(x',y_1) \wedge y'_1 = iListNth(s',i_1) \\
& & \quad\quad\quad\quad \Rightarrow y' = all(x',y'_1)))
\end{array}
$$

Substitution $x' = subst(x,v,t)$ la formule x' est le résultat de la substitution du terme t à la variable v dans la formule x.

$$
\begin{array}{lcl}
x' = subst(x,v,t) & := & Form(x) \wedge Var(v) \wedge Term(t) \wedge FairPair(x,t) \wedge \\
& & \forall s \leq \sigma_x.\forall s' \leq \sigma_{x'}.(FormSeq(s) \wedge IListMem(x,s) \wedge s' = substSeq(s,v,t) \Rightarrow \\
& & \quad \forall i \leq len(s).(x = iListNth(s,i) \Rightarrow x' = IListNth(s',i)))
\end{array}
$$

5.7 Preuves

Un entier x code une preuve s'il code une suite de formules obéissant aux règles édictées en 2.2. On sait reconnaître une suite de formules, reste à vérifier que chacune des formules est soit l'instance d'un axiome soit l'application d'une des régles de déduction. Pour reconnaître qu'une formule est l'instance d'un axiome, il suffit d'y reconnaître la forme de cet axiome. Par exemple pour P1, on a que A est une instance de P1 s'il exsite deux formules A_1 et $A-2$ telles que A est égale à $A_1 \Rightarrow (A_2 \Rightarrow A_1)$. Une formule A est le résultat de l'application d'un modus ponens entre A_1 et A_2 si A_1 est égale à $A_2 \Rightarrow A$. Une formule A est le résultat de l'application d'une généralisation s'il existe une formule A_1 et une varaibel x telle que A est égale à $\forall x\ A_1$.

La reconnaissance des axiomes est formulée dans le long paragraphe qui suit. La reconnaissance de l'application d'une règle est directement donnée dans la définition du prédicat $Proof$ (*«est une preuve»*).

Axiomes

$$
\begin{array}{lcl}
AxP1(x) & := & \exists x_1 \leq x.\exists x_2 \leq x.(x = impl(x_1, impl(x_2,x_1))) \\
AxP2(x) & := & \exists x_1 \leq x.\exists x_2 \leq x.\exists x_3 \leq x. \\
& & \quad (x = impl(impl(x_1,impl(x_2,x_3)),impl(impl(x_1,x_2),impl(x_1,x_3)))) \\
AxP3(x) & := & \exists x_1 \leq x.\exists x_2 \leq x.\exists x_3 \leq x.(x = impl(impl(neg(x_1),neg(x_2)),impl(x_2,x_1))) \\
AxQ1(x) & := & \exists x_1 \leq x.\exists v \leq x.\exists t \leq x.(x = impl(all(v,x_1),subst(x_1,v,t))) \\
AxQ2(x) & := & \exists x_1 \leq x.\exists x_2 \leq x.\exists v \leq x. \\
& & \quad ((x = impl(all(v,impl(x_1,x_2)),impl(x_1,all(v,x_2)))) \wedge \neg Free(v,x_1))
\end{array}
$$

$$
\begin{array}{lcl}
AxE1(x) & := & \exists x_1 \leq x.(x = eq(x_1, x_1)) \\
AxE2(x) & := & \exists x_1 \leq x.\exists x_2 \leq x.(x = impl(eq(x_1, x_2), eq(x_2, x_1))) \\
AxE3(x) & := & \exists x_1 \leq x.\exists x_2 \leq x.\exists x_3 \leq x.(x = impl(eq(x_1, x_2), impl(eq(x_2, x_3), impl(eq(x_1, x_3))))) \\
AxE4(x) & := & \exists x_1 \leq x.\exists x_2 \leq x.(x = impl(eq(x_1, x_2), eq(succ(x_1), succ(x_2)))) \\
AxE5(x) & := & \exists x_1 \leq x.\exists x_2 \leq x.\exists x_3 \leq x.\exists x_4 \leq x. \\
 & & (x = impl(eq(x_1, x_2), impl(eq(x_3, x_4), eq(plus(x_1, x_3), plus(x_2, x_4))))) \\
AxE6(x) & := & \exists x_1 \leq x.\exists x_2 \leq x.\exists x_3 \leq x.\exists x_4 \leq x. \\
 & & (x = impl(eq(x_1, x_2), impl(eq(x_3, x_4), eq(mult(x_1, x_3), mult(x_2, x_4))))) \\
AxE7(x) & := & \exists x_1 \leq x.\exists x_2 \leq x.\exists x_3 \leq x.\exists x_4 \leq x. \\
 & & (x = impl(eq(x_1, x_2), impl(eq(x_3, x_4), eq(exp(x_1, x_3), exp(x_2, x_4))))) \\
AxE8(x) & := & \exists x_1 \leq x.\exists x_2 \leq x.\exists x_3 \leq x.\exists x_4 \leq x. \\
 & & (x = impl(eq(x_1, x_2), impl(eq(x_3, x_4), impl(eq(x_1, x_3), eq(x_2, x_4))))) \\
AxE9(x) & := & \exists x_1 \leq x.\exists x_2 \leq x.\exists x_3 \leq x.\exists x_4 \leq x. \\
 & & (x = impl(eq(x_1, x_2), impl(eq(x_3, x_4), impl(le(x_1, x_3), le(x_2, x_4))))) \\
AxA1(x) & := & \exists x_1 \leq x.x = neg(eq(zero, succ(x_1))) \\
AxA2(x) & := & \exists x_1 \leq x.\exists x_2 \leq x.x = impl(eq(succ(x_1), succ(x_2)), eq(x_1, x_2)) \\
AxA3(x) & := & \exists x_1 \leq x.x = eq(plus(zero, x_1), x_1) \\
AxA4(x) & := & \exists x_1 \leq x.\exists x_2 \leq x.x = eq(plus(succ(x_1), x_2), succ(plus(x_1, x_2))) \\
AxA5(x) & := & \exists x_1 \leq x.x = eq(mult(zero, x_1), zero) \\
AxA6(x) & := & \exists x_1 \leq x.\exists x_2 \leq x.x = eq(mult(succ(x_1), x_2), plus(x_2, mult(x_1, x_2))) \\
AxA7(x) & := & \exists x_1 \leq x.x = eq(exp(x_1, zero), s^1 0) \\
AxA8(x) & := & \exists x_1 \leq x.\exists x_2 \leq x.x = eq(exp(x_1, succ(x_2)), mult(x_1, exp(x_1, x_2))) \\
AxA9(x) & := & \exists x_1 \leq x.x = neg(impl(le(x_1, zero), \neg(eq(x_1, zero))))^\dagger \\
AxA10(x) & := & \exists x_1 \leq x.\exists x_2 \leq x.x = neg(impl(le(x_1, succ(x_2)), neg(impl(\neg(le(x_1, x_2), eq(x_1, x_2)))^\ddagger)))^\dagger \\
AxA11(x) & := & \exists x_1 \leq x.\exists x_2 \leq x.impl(\neg(le(x_1, x_2)), \leq (x_2, x_1))^\ddagger
\end{array}
$$

†équivalence logique ‡disjonction

$$Ax(x) \quad := \quad Form(x) \wedge (AxP1(x) \vee \ldots \vee AxA11(x))$$

Preuve $Proof(x)$: x est une suite de formules qui forme une preuve.

$$
\begin{array}{lcl}
Proof(x) & := & FormSeq(x) \wedge \\
 & & \forall x_1 \leq x.(IListMem(x_1, x) \Rightarrow \\
 & & \quad (\ Ax(x_1) \vee \\
 & & \quad\quad (\exists x_2 \leq x.\exists x_2 \leq x. \\
 & & \quad\quad\quad (IListPred(x_2, x_1, x) \wedge IListPred(x_3, x_1, x) \wedge (x_3 = impl(x_2, x_1)))) \vee \\
 & & \quad\quad (\exists x_2 \leq x.\exists z \leq x.(IListPred(x_2, x_1, x) \wedge Var(z) \wedge (x_1 = all(z, x_2))))))
\end{array}
$$

Prouvabilité $Proves(x, y)$: la suite de formules x est une preuve qui contient y.

$$Proves(x, y) := Proof(x) \wedge IListMem(y, x)$$

6 Arithmétisation

Une façon de noter les entiers est la notation décimale qui utilise les dix chifres 0 1 2 3 4 5 6 7 8 9. Une autre façon est de n'utiliser que deux chiffres 0 et 1 : c'est la notation binaire. En fait, on peut utiliser n'importe quel nombre de chiffres pour noter les entiers. En particulier, on peut tout aussi bien utiliser 15 chiffres : nous nous sommes donnés 14 caractères pour coder formules et preuves ; le premier caractère ayant valeur 1 et le dernier 14 ; il nous faut également un zéro pour avoir une numération complète. Le nombre 15 sera donc notre *base de numération* : notons la b.

Si la base de numération est 10, multiplier n par 10 revient à rajouter un 0 en fin de la notation de n : par exemple, $674 \times 10 = 6740$. En base b, multiplier n par b revient à ajouter un 0 en fin de la notation de n : par exemple $674 \times \mathsf{b} = 6740$. Pour rajouter 2 zéros,en base 10, il faut multiplier par $100 = 10^2$: $674 \times 10^2 = 67400$. Pour rajouter 2 zéros en base b, on mulptiplie par b^2. De façon générale, pour rajouter k zéros en fin d'un nombre noté en base b, on multiplie par b^k.

Pour ajouter un chiffre en fin d'un nombre, noté en base 10, on multiplie par 10 puis on ajoute le chiffre : par exemple, $(674 \times 10) + 1 = 6740 + 1 = 6741$. Le principe est le même en base b. Pour ajouter un nombre de 2 chiffres à un nombre noté en base 10 on multiplie par $10^2 = 100$ puis on additionne : par exemple, $1234 \times 10^2 + 56 = 123400 + 56 = 123456$. De façon générale, en base 10, pour ajouter un nombre m de k chiffres à n, on fait $n \times 10^k + m$. Si la base est b, on fait $n \times \mathsf{b}^k + m$.

L'opération de concaténation $x_1 \cdot x_2$ où x_1 et x_2 sont écrits en base b revient à calculer $x_1 \times \mathsf{b}^{len(x_2)} + x_2$ où $len(x_2)$ est le nombre de chiffres nécessaires à l'écriture de x_2.

Caractère $Char(x)$: x est un caractère.

$$Char(x) := x \leq \mathsf{b}$$

L'abréviation $\forall c : Char.A$ est donc en fait la quantification bornée $\forall c \leq \mathsf{b}.A$.

Longueur (d'une chaîne) $n = len(x)$: n est la longueur de (la chaîne) x ; i.e. n est le nombre de chiffres nécessaires à l'écriture de x en base b ; i.e. n est le plus petit entier telle que $x \leq \mathsf{b}^n$.

$$n = len(x) := x \leq \mathsf{b}^n \wedge \neg\exists m \leq n.(x \leq \mathsf{b}^m)$$

Concatènation $x = x_1 \cdot x_2$: x est le résultat de la concatènation de x_1 et x_2.

$$(x = x_1 \cdot x_2) := \forall n \leq x_2.(n = len(x_2) \Leftrightarrow x = x_1 \times \mathsf{b}^n + x_2)$$

7 Pour finir

Gödel pousse le bouchon un peu plus loin en posant une formule, codable, qui exprime la cohérence de l'arithmétique formelle : il n'existe pas de formule A telle que A et $\neg A$ soient prouvables. Il montre que cette formule n'est pas prouvable avec les ressources de l'arithmétique formelle interdisant ainsi la réalisation du *programme de Hilbert* proposé à la communauté des mathématiciens par D. Hilbert au Congrès internationnal de mathématique de Paris en 1900 qui visait à établir la consistance de l'aritmétique (et par là de la théorie des nombres réels et de la géométrie, etc.) avec les seuls moyens de l'arithmétique formelle.

A Complétude

Nous nous proposons de (ne) donner (que) la démonstration de la complètude du fragment propositionnel, celle en fait qu'énonce le fait 1. Pour ce, nous étendons la notion de preuves formelles aux *preuves sous hypothèses*.

Nous ne considérons dans cette section que les formules écrites avec les seuls connecteurs $\Rightarrow$ et $\neg$. On appelle *proposition* de telles formules. On appelle *atomes propositionnels* les formules atomiques qui interviennent dans l'écriture d'une proposition.

A.1 Sémantique

Pour définir la *valeur de vérité* d'une proposition, on se donne une application δ, appelée *distribution de valeurs de vérité*, de l'ensemble des atomes propositionels dans l'ensemble à deux éléments $\{0, 1\}$ (intuitivement : 0 pour faux et 1 pour vrai). On étend cette application aux propositions de la façon suivante : soit A_1 et A_2 deux propositions

- si $\delta(A_1) = 1$ alors $\delta(\neg A_1) = 0$;
- si $\delta(A_1) = 0$ alors $\delta(\neg A_1) = 1$;
- si $\delta(A_1) = 1$ et $\delta(A_2) = 1$ alors $\delta(A_1 \Rightarrow A_2) = 1$;
- si $\delta(A_1) = 1$ et $\delta(A_2) = 0$ alors $\delta(A_1 \Rightarrow A_2) = 0$;
- si $\delta(A_1) = 0$ alors $\delta(A_1 \Rightarrow A_2) = 1$ quelque soit la valeur de $\delta(A_2)$.

Un proposition A est une *tautologie* si pour toute distribution de valeurs de vérité δ, on a $\delta(A) = 1$.

Conséquence sémantique

Soit Σ un ensemble de propositions, soit A une formule. On dit que A est une conséquence sémantique de Σ, ce que l'on note $\Sigma \models A$, si pour toute distribution de valeurs de vérité δ telle que pour toute proposition A_i de Σ, $\delta(A_i) = 1$, on a $\delta(A) = 1$.

A.2 Syntaxe et extensions

Preuves sous hypothèses (fragment propositionnel)

Soit Σ un ensemble de formules, soit A une formule. Une preuve de A sous hypothèses Σ est une suite de couples $(\Sigma, A_1); \ldots ; (\Sigma, A_n)$ tels que

1. la formule A est égale à A_n.
2. toute formule A_i de la suite est
   - soit l'une des formules de Σ ;
   - soit l'instance d'un axiome parmi P1, P2 ou P3 ;
   - soit obtenue par application de la régle de déduction R1 à deux formules A_j, A_k avec $j < i$ et $k < i$;

On note $\Sigma \vdash A$ les couples constitués d'un ensemble de formules (Σ) et d'une formule (A). On ne prouve donc plus des formules, mais des couples $\Sigma \vdash A$ que l'on appelle parfois *séquents*. Lorsque l'ensemble Σ est vide, on note simplement $\vdash A$.

On dit que $\Sigma \vdash A$ *est prouvable* s'il existe une preuve (au sens ci-dessus) du séquent $\Sigma \vdash A$. Les tautologies sont les propositions A telles que $\vdash A$ est prouvable.

Notation des preuves

Nous noterons les preuves sous forme d'une suite de lignes, chacune constituée des trois éléments suivants :

- un numéro de ligne ;
- un séquent ;

– une justification.

Dans une preuve, les numéros de ligne doivent être tous distincts. Ils correspondront en général à l'ordre de succession des lignes mais il n'y a pas ici de caractère d'obligation. Les justifications permettent d'identifier quelle règle légitime chaque ligne. On utilise pour ce un *identificateur de règle* auquel on peut adjoindre un ou plusieurs *arguments*. Les arguments sont des *références* aux formules ou séquents requis par les régles. Les justifications de base sont : $Ax(\alpha)$, où α est le nom d'un des axiomes du système formel ; $Hyp(i)$ lorsque l'on invoque une hypothèse où i est le numéro de l'hypothèse invoquée (les hypothèses sont numérotées de gauche à droite, la première a le numéro 1) ; $MP(n_1)(n_2)$ lorsque l'on applique la règle du *modus ponens* entre deux formules obtenues en lignes n_1 et n_2.

Règle Hyp Dans la pratique, on n'utilisera pas explicitement la règle Hyp. On mentionnera l'usage d'une hypothèse en la nommant comme argument de règle. L'hypothèse numéro i sera nommée Hi. Par exemple, un modus ponens entre une formule obtenue à l'étape n et l'hypothèse Hi sera notée $MP(n)(Hi)$. Une référence pourra donc être soit un numéro de ligne, soit un nom d'hypothèse.

Modus ponens Il est fréquent d'utiliser une séquence d'applications de modus ponens : une première application, disons entre les références α et β donne une formule sur laquelle on utilise un deuxième modus ponens avec la référence δ, etc. Dans ce cas, on notera une seule ligne donnant le résultat final avec l'identificateur de règle étendu $MP(\alpha)(\beta, \delta, etc.)$.

Exemple À titre de premier exemple, voici comment on écrit la preuve de la tautologie triviale $A \Rightarrow A$:

Tautologie (1) $A \Rightarrow A$

(1)	$\vdash A \Rightarrow ((A \Rightarrow A) \Rightarrow A)$	Ax(P1)
(2)	$\vdash A \Rightarrow (A \Rightarrow A)$	Ax(P1)
(3)	$\vdash (A \Rightarrow ((A \Rightarrow A) \Rightarrow A)) \Rightarrow ((A \Rightarrow (A \Rightarrow A)) \Rightarrow (A \Rightarrow A))$	Ax(P2)
(4)	$\vdash A \Rightarrow A$	$MP(3)(1,2)$

***Pseudo règle*, extensions du jeu de règles**

Pour faciliter leur conception et alléger l'écriture des preuves, on étend l'ensemble des règles (ou justifications) utilisables dans la construction des preuves formelles. Voici un premier exemple d'une telle extension.

On peut montrer des résultats généraux sur l'existence des preuves formelles. Par exemple :

Fait (8) si $\Sigma \vdash A$ alors, pour tout B, $\Sigma \cup \{B\} \vdash A$.

Il est en effet clair que l'on peut toujours se donner plus d'hypothèses que nécessaire.

Donnons nous le nouvel identificateur Wk (pour *weakening* en anglais, affaiblissement en français) ; et posons :

$$\begin{array}{lll} & \vdots & \\ (n) & \Sigma \vdash A & .. \\ & \vdots & \\ (m) & \Sigma \cup \{B\} \vdash A & Wk(n) \end{array}$$

L'indice rappelé aprés le nom Wk fait référence au séquent dont on affaiblit l'ensemble d'hypothèses. Notez que l'on ne force pas la succession des lignes (n) et (m)

Généricité, réutilisation de preuve

Soient A une formule propositionnelle et Σ un ensemble de formules propositionelles ; soient X un atome propositionnel et B une formule. Notons, respectivement, $\Sigma[B/X]$ et $A[B/X]$ l'ensemble de formules, respectivement, la formule, obtenus en remplaçant[8] X par B dans les formules de Σ et A. Il est clair que

Fait (9) si $\Sigma \vdash A$ alors $\Sigma[B/X] \vdash A[B/X]$.

En d'autres termes, une formule dont on connaît une dérivation peut jouer le rôle d'un axiome. En particulier, si $\vdash A$ est prouvable alors $\vdash A[B/X]$ l'est également. Soient alors l'identificateur Thm et α un moyen quelconque d'identifier la dérivation – et sa conclusion – de $\vdash A$. On s'autorise à utiliser dans toute preuve une ligne de la forme :

$$(n) \quad \Sigma \vdash A[B/X] \quad Thm(\alpha)$$

Justification : posons $\Sigma = \{A_1, \ldots, A_k\}$. On prologe la preuve de $\vdash A$ en itérant la règle Wk :

$$\begin{array}{lll} & \vdots & \\ (n) & \vdash A & .. \\ (n') & \vdash A[B/X] & .. \\ (n_1) & A_1 \vdash A[B/X] & Wk(n) \\ & \vdots & \\ (n_k)\star & A_1, \ldots, A_k \vdash A[B/X] & Wk(n_k - 1) \end{array}$$

On a marqué d'une étoile ($\star$) la ligne correspondant à celle de la pseudo règle. Pour d'autres pseudo règles, on pourra marquer plusieurs lignes d'une étoile.

Vrai et faux formels

Il est parfois utile, parce que plus intuitif, de faire référence dans une preuve formelle au *vrai* ou au *faux*. On les représente respectivement par les symboles $\top$ et $\bot$.

Vrai formel Le *vrai* peut être dénoté par n'importe quelle tautologie, formule toujours vraie du point de vue sémantique. Prenons l'axiome P1 et posons

$$\top := A \Rightarrow (A \Rightarrow A)$$

Il est clair qu'en utilisant l'affaiblissement, pour tout Σ, on a $\Sigma \vdash \top$. On peut alors en déduire la pseudo règle $True$ qui donne que pour tout Σ, pour toute formule A, on a que si $\Sigma \vdash \top \Rightarrow A$ alors $\Sigma \vdash A$:

$$\begin{array}{lll} & \vdots & \\ (n) & \Sigma \vdash \top \Rightarrow A & .. \\ & \vdots & \\ (m) & \Sigma \vdash A & True(n) \end{array}$$

Justification :

$$\begin{array}{lll} & \vdots & \\ (n)\star & \Sigma \vdash \top \Rightarrow A & .. \\ & \vdots & \\ (m') & \Sigma \vdash \top & Ax(\text{P1}) \\ (m)\star & \Sigma \vdash A & MP(n)(m') \end{array}$$

[8] La définition précise (par induction sur la forme des formules) de l'opération de remplacement est laissée au lecteur.

Le *faux*, en bonne logique, est la négation du *vrai*. Posons

$$\bot := \neg\top$$

Les termes *conradiction* ou *absurdité* sont parfois utilisés dans le vernaculaire logico-mathématique comme synonyme du faux.

A.3 Lemme de déduction

Pour faire court, on note Σ, A pour $\Sigma \cup \{A\}$.

Lemme (10) si $\Sigma, A \vdash B$ est prouvable alors $\Sigma \vdash A \Rightarrow B$ l'est aussi.

Preuve soit $\Sigma, A \vdash B_1; \ldots; \Sigma, A \vdash B_n$ une preuve de $\Sigma, A \vdash B$ (on a B_n égale à B). On raisonne par induction sur n.

- si $n = 1$ alors
  - si B est égal à A, on a immédiatement :

$$(1) \quad \Sigma \vdash A \Rightarrow A \quad Thm(1)$$

  - B est un élément de Σ ou l'instance d'un axiome. On construit la preuve suivante :

$$\begin{array}{lll} (1) & \Sigma \vdash B & Hyp/Ax \\ (2) & \Sigma \vdash B \Rightarrow (A \Rightarrow B) & Ax(\text{P1}) \\ (3) & \Sigma \vdash A \Rightarrow B & MP(2)(1) \end{array}$$

- si $n > 1$, alors
  - Le cas où B est A ou un élément de Σ ou l'instance d'un axiome se traite de façon analogue à ce qui a été fait pour $n = 1$.
  - si B est le résultat d'un modus ponens, il existe deux entiers i et j inférieurs à n tels que B_j soit égal à $B_i \Rightarrow B$. Notons que les suites $\Sigma, A \vdash B_1; \ldots; \Sigma, A \vdash B_i$ et $\Sigma, A \vdash B_1; \ldots; \Sigma, A \vdash B_j$ sont des preuves respectives de $\Sigma, A \vdash B_i$ et $\Sigma, A \vdash B_i \Rightarrow B$. Par hypothèse d'induction, $\Sigma \vdash A \Rightarrow B_i$ et $\Sigma \vdash A \Rightarrow (B_i \Rightarrow B)$, sont prouvables. On construit alors

$$\begin{array}{lll} & \vdots & \\ (n_1) & \Sigma \vdash A \Rightarrow B_i & .. \\ & \vdots & \\ (n_2) & \Sigma \vdash A \Rightarrow (B_i \Rightarrow B) & .. \\ (n) & \Sigma \vdash (A \Rightarrow (B_i \Rightarrow B)) \Rightarrow ((A \Rightarrow B_i) \Rightarrow (A \Rightarrow B)) & Ax(\text{P2}) \\ (n') & \Sigma \vdash A \Rightarrow B & MP(n)(n_2, n_1) \end{array}$$

Pseudo règle *Ded*

Le lemme de la déduction permet de poser la *pseudo règle Ded* :

$$\begin{array}{lll} & \vdots & \\ (n) & \Sigma, A \vdash B & .. \\ & \vdots & \\ (m) & \Sigma \vdash A \Rightarrow B & Ded(n) \end{array}$$

Ici également, on explicite le numéro de ligne d'où l'on tire la nouvelle étape de preuve.

Il est fréquent d'utiliser une séquence de la règle Ded qui a pour résultat d'*"épuiser"* l'ensemble des hypothèses. On résume cette itération par Ded^* :

$$\begin{array}{lll} & \vdots & \\ (n) & A_1, \ldots, A_n \vdash B & .. \\ & \vdots & \\ (m) & \vdash A_1 \Rightarrow (\ldots \Rightarrow (A_n \Rightarrow B)\ldots) & Ded^*(n) \end{array}$$

A.4 Quelques tautologies utiles

Nous prouvons formellement dans cette section un certain nombre de tautologies du calcul des propositions. L'utilité de ce faire est qu'à l'instar des axiomes, nous pourrons invoquer toute instance de ces tautologies dans la construction une preuve formelle.

Notons que nous montrons la prouvabilité formelle de ces tautologies *avant* que ne soit établit le théorème de complétude, plus, la prouvabilité de ces tautologies sera utilisée (directement ou non) pour établir la complétude. Ces preuves d'existence de preuves formelles n'est donc pas une inutile trivialité.

Certaines de ces tautologies expriment des figures usuelles du raisonnement. Nous les utiliserons pour poser des pseudo règles correspondant à ces figures.

Tautologie (2) $\neg A \Rightarrow (A \Rightarrow B)$

Intuitivement : d'une contradiction ($\neg A$ et A) on peut déduire n'importe quoi (B) : en latin *ex falsum quod libet.*

$$\begin{array}{lll} (1) & \neg A \vdash \neg A \Rightarrow (\neg B \Rightarrow \neg A) & Ax(\text{P1}) \\ (2) & \neg A \vdash \neg B \Rightarrow \neg A & MP(1)(H1) \\ (3) & \neg A \vdash (\neg B \Rightarrow \neg A) \Rightarrow (A \Rightarrow B) & Ax(\text{P3}) \\ (4) & \neg A \vdash A \Rightarrow B & MP(3)(2) \\ (5) & \vdash \neg A \Rightarrow (A \Rightarrow B) & Ded(4) \end{array}$$

Pseudo règle On tire de cette tautologie la pseudo règle $Contrad$:

$$\begin{array}{lll} & \vdots & \\ (n_1) & \Sigma \vdash A & .. \\ & \vdots & \\ (n_2) & \Sigma \vdash \neg A & .. \\ & \vdots & \\ (m) & \Sigma \vdash B & Contrad(n_1)(n_2) \end{array}$$

Justification :

$$\begin{array}{lll} & \vdots & \\ (n_1)\star & \Sigma \vdash A & .. \\ & \vdots & \\ (n_2)\star & \Sigma \vdash \neg A & .. \\ & \vdots & \\ (m') & \Sigma \vdash \neg A \Rightarrow (A \Rightarrow B) & Thm(2) \\ (m'') & \Sigma \vdash A \Rightarrow B & MP(m')(n_2) \\ (m)\star & \Sigma \vdash B & MP(m'')(n_1) \end{array}$$

Notons que l'on a comme cas particulier de *Contrad* que de $\Sigma \vdash A$ et $\Sigma \vdash \neg A$, on tire $\Sigma \vdash \bot$.

Tautologie (3) $\neg\neg A \Rightarrow A$

Cette tautologie donne un sens de l'*idempotence* de la négation : une double négation revient à une affirmation. L'autre sens sera montré en 8.

(1)	$\neg\neg A \vdash \neg\neg A \Rightarrow (\neg A \Rightarrow \neg\top)$	$Thm(2)$
(2)	$\neg\neg A \vdash \neg A \Rightarrow \neg\top$	$MP(1)(H1)$
(3)	$\neg\neg A \vdash (\neg A \Rightarrow \neg\top) \Rightarrow (\top \Rightarrow A)$	Ax(P3)
(4)	$\neg\neg A \vdash \top \Rightarrow A$	$MP(3)(2)$
(5)	$\neg\neg A \vdash A$	$True(4)$
(6)	$\vdash \neg\neg A \Rightarrow A$	$Ded(5)$

Tautologie (4) $(\neg A \Rightarrow \bot) \Rightarrow A$

Cette tautologie exprime la figure du raisonnement par l'absurde : pour montrer A supposons $\neg A$ et cherchons une contradiction ($\bot$).

Rappelons que $\bot$ **:=** $\neg\top$. Ce qu'il faut montrer est donc en fait $(\neg A \Rightarrow \neg\top) \Rightarrow A$

(1)	$\neg A \Rightarrow \neg\top \vdash (\neg A \Rightarrow \neg\top) \Rightarrow (\top \Rightarrow A)$	Ax(P3)
(2)	$\neg A \Rightarrow \neg\top \vdash \top \Rightarrow A$	$MP(1)(H1)$
(3)	$\neg A \Rightarrow \neg\top \vdash A$	$True(2)$
(4)	$\vdash (\neg A \Rightarrow \neg\top) \Rightarrow A$	$Ded(3)$

Pseudo règle *Abs* :

	$\vdots$	
(n)	$\Sigma, \neg A \vdash \bot$	..
	$\vdots$	
(m)	$\Sigma \vdash A$	$Abs(n)$

Justification :

	$\vdots$	
$(n)\star$	$\Sigma, \neg A \vdash \bot$	..
	$\vdots$	
(n')	$\Sigma \vdash \neg A \Rightarrow \bot$	$Ded(n)$
(n'')	$\Sigma \vdash (\neg A \Rightarrow \bot) \Rightarrow A$	$Thm(4)$
$(m)\star$	$\Sigma \vdash A$	$MP(n'')(n')$

Tautologie (5) $(A \Rightarrow \bot) \Rightarrow \neg A$

Intuitivement, cette tautologie correspond à la figure de raisonnement suivante : si de A on déduit une contradiction ($A \Rightarrow \bot$) alors A est faux ($\neg A$). C'est une façon *naturelle* de montrer une formule négative.

On l'obtient en combinant un raisonnement par l'absurde avec le fait que $\neg\neg A$ implique A.

(1)	$A \Rightarrow \bot, \neg\neg A \vdash \neg\neg A \Rightarrow A$	$Thm(3)$
(2)	$A \Rightarrow \bot, \neg\neg A \vdash A$	$MP(1)(H2)$
(3)	$A \Rightarrow \bot, \neg\neg A \vdash \bot$	$MP(H1)(2)$
(4)	$A \Rightarrow \bot \vdash \neg A$	$Abs(3)$
(5)	$\vdash (A \Rightarrow \bot) \Rightarrow \neg A$	$Ded(4)$

Pseudo règle Neg :

$$\begin{array}{lll} & \vdots & \\ (n) & \Sigma, A \vdash \bot & .. \\ & \vdots & \\ (m) & \Sigma \vdash \neg A & Neg(n) \end{array}$$

Justification

$$\begin{array}{lll} & \vdots & \\ (n)\star & \Sigma, A \vdash \bot & .. \\ & \vdots & \\ (n') & \Sigma, \vdash A \Rightarrow \bot & Ded(n) \\ (n'') & \Sigma \vdash (A \Rightarrow \bot) \Rightarrow \neg A & Thm(5) \\ (m)\star & \Sigma \vdash \neg A & Neg(n) \end{array}$$

Tautologie (6) $(A \Rightarrow B) \Rightarrow (\neg B \Rightarrow \neg A)$

La formule $\neg B \Rightarrow \neg A$ est appelée *contraposée* de $A \Rightarrow B$. En fait, ces deux formules sont logiquement équivalentes. La tautologie montrée dans ce paragraphe est l'autre sens de l'axiome P3.

$$\begin{array}{lll} (1) & A \Rightarrow B, \neg B, A \vdash B & MP(H1)(H2) \\ (2) & A \Rightarrow B, \neg B, A \vdash \bot & Contrad(1)(H2) \\ (3) & A \Rightarrow B, \neg B \vdash \neg A & Neg(2) \\ (4) & \vdash (A \Rightarrow B) \Rightarrow (\neg B \Rightarrow \neg A) & Ded^*(3) \end{array}$$

Tautologie (7) $(A \Rightarrow B) \Rightarrow ((\neg A \Rightarrow B) \Rightarrow B)$

Intuitivement, si B est vrai que A soit vrai ou non, alors B est toujours vrai.

$$\begin{array}{lll} (1) & A \Rightarrow B, \neg A \Rightarrow B, \neg B \vdash (\neg A \Rightarrow B) \Rightarrow (\neg B \Rightarrow \neg\neg A) & Thm(6) \\ (2) & A \Rightarrow B, \neg A \Rightarrow B, \neg B \vdash \neg\neg A & MP(1)(H2, H3) \\ (3) & A \Rightarrow B, \neg A \Rightarrow B, \neg B \vdash \neg\neg A \Rightarrow A & Thm(3) \\ (4) & A \Rightarrow B, \neg A \Rightarrow B, \neg B \vdash A & MP(3)(2) \\ (5) & A \Rightarrow B, \neg A \Rightarrow B, \neg B \vdash B & MP(H1)(4) \\ (6) & A \Rightarrow B, \neg A \Rightarrow B, \neg B \vdash \bot & Contrad(H1)(5) \\ (7) & A \Rightarrow B, \neg A \Rightarrow B \vdash B & Neg(6) \\ (8) & \vdash (A \Rightarrow B) \Rightarrow ((\neg A \Rightarrow B) \Rightarrow B) & Ded^*(7) \end{array}$$

Tautologie (8) $A \Rightarrow \neg\neg A$

$$\begin{array}{lll} (1) & A, \neg A \vdash A & Hyp \\ (2) & A, \neg A \vdash \neg A & Hyp \\ (3) & A, \neg A \vdash \bot & Contrad(1)(2) \\ (4) & A \vdash \neg A \Rightarrow \bot & Ded(3) \\ (5) & A \vdash \neg\neg A & Neg(4) \\ (6) & \vdash A \Rightarrow \neg\neg A & Ded(5) \end{array}$$

Tautologie (9) $A \Rightarrow (\neg B \Rightarrow \neg(A \Rightarrow B))$

Cette tautologie est une formalisation de ce qu'une formule vraie ne peut pas impliquer une formule fausse : si l'on suppose A et $\neg B$ alors on n'a pas $A \Rightarrow B$ (puisqu'alors on a $\neg(A \Rightarrow B)$).

$$\begin{array}{lll} (1) & A, \neg B, A \Rightarrow B \vdash B & MP(H3)(H1) \\ (2) & A, \neg B, A \Rightarrow B \vdash \bot & Contrad(1)(H2) \\ (3) & A, \neg B \vdash \neg(A \Rightarrow B) & Neg(2) \\ (4) & \vdash A \Rightarrow (\neg B \Rightarrow \neg(A \Rightarrow B)) & Ded^*(3) \end{array}$$

A.5 Le théorème

Nous allons montrer que si une formule A est une tautologie alors il existe une preuve formelle de $\vdash A$.

L'idée de la démonstration est la suivante : si une formule A est une tautologie alors quelque soit la valeur de vérité que l'on accorde à ses atomes propositionnels, la valeur de vérité de A est toujours 1 (le vrai). Soient donc $P_1, \ldots, P_n$ les atomes propositionnels de A. À chaque distribution de la valeur de vérité δ, on associe l'ensemble noté $P_1^\delta, \ldots, P_n^\delta$ tel que pour tout $i \in [1 \ldots n]$,

P_i^δ **:=** P_i si $\delta(P_i) = 1$ et

P_i^δ **:=** $\neg P_i$ si $\delta(P_i) = 0$.

On a ainsi trouvé un moyen formel d'énumérer les distributions de valeur de vérité. On fait pour A ce que l'on a fait pour les P_i en posant

$A^\delta = A$ si $\delta(A) = 1$ et

$A^\delta = \neg A$, sinon.

On montre dans un premier temps que A^δ est prouvable sous les hypothèses $P_1^\delta, \ldots, P_n^\delta$ (ie $P_1^\delta, \ldots, P_n^\delta \vdash A^\delta$). Puis, dans le cas où A est une tautologie, puisque pour tout δ, $\delta(A) = 1$, on a $A^\delta = A$; on montre alors qu'on peut éliminer les hypothèses $P_1^\delta, \ldots, P_n^\delta$ pour obtenir une preuve de $\vdash A$.

Lemme (11) Soient $P_1, \ldots, P_n$ l'ensemble des atomes propositionnels d'une formule A, soit δ une distribution de valeurs de vérité, soit $\Pi = \{P_1^\delta, \ldots, P_n^\delta\}$, il existe une preuve formelle de $\Pi \vdash A^\delta$.

Preuve par induction sur la forme de A :

- si A est un atome propositionnel, c'est l'un des P_i. Il suffit alors d'utiliser la règle Hyp pour obtenir une preuve de $\Pi \vdash A^\delta$.
- si A est égale à $\neg A_1$, on a, par hypothèse d'induction qu'il existe une preuve formelle de $\Pi \vdash A_1^\delta$ et on considère les deux cas
  - soit $\delta(A_1) = 0$, alors $\delta(A) = 1$ et $A_1^\delta = \neg A_1 = A = A^\delta$.
    On a, par hypothèse d'induction que $\Pi \vdash A_1^\delta$ est prouvable, c'est-à-dire $\Pi \vdash A^\delta$.
  - soit $\delta(A_1) = 1$, alors $(i) A_1^\delta = A_1$ et $\delta(A) = 0$ d'où $(ii) A^\delta = \neg A = \neg\neg A_1$.
    On a par hypothèse d'induction que $\Pi \vdash A_1^\delta$ est prouvable, c'est-à-dire $\Pi \vdash A_1$ (par (i)). On prolonge la preuve de $\Pi \vdash A_1$ de la façon suivante

$$\begin{array}{lll} & \vdots & \\ (n) & \Pi \vdash A_1 & .. \\ (n') & \Pi \vdash A_1 \Rightarrow \neg\neg A_1 & Thm(8) \\ (n'') & \Pi \vdash \neg\neg A_1 & MP(n')(n) \end{array}$$

    On a donc, par (ii) une preuve de $\Pi \vdash A^\delta$.
- si A est égale à $A_1 \Rightarrow A_2$, on a par hypothèse d'induction que $(hr1) \Pi \vdash A_1^\delta$ et $(hr2) \Pi \vdash A_2^\delta$ sont prouvables. On considère les trois cas suivants
  - si $\delta(A_1) = 0$ alors $(i) A_1^\delta = \neg A_1$ et $\delta(A) = 1$ d'où $(ii) A^\delta = A = A_1 \Rightarrow A_2$.
    On a par $(hr1)$ que $\Pi \vdash A_1^\delta$ est prouvable, c'est-à-dire, $\Pi \vdash \neg A_1$ (par (i)). On prolonge la preuve de $\Pi \vdash A_1^\delta$:

$$\begin{array}{lll} & \vdots & \\ (n) & \Pi \vdash \neg A_1 & .. \\ (n') & \Pi \vdash \neg A_1 \Rightarrow (A_1 \Rightarrow A_2) & Thm(2) \\ (n'') & \Pi \vdash A_1 \Rightarrow A_2 & MP(n')(n) \end{array}$$

    On a donc, par (ii), que $\Pi \vdash A^\delta$ est prouvable.
  - si $\delta(A_1) = 1$ et $\delta(A_2) = 0$ alors $(i) A_1^\delta = A_1$, $(ii) A_2^\delta = \neg A_2$ et $\delta(A) = 0$ d'où $(iii) A^\delta = \neg A = \neg(A_1 \Rightarrow A_2)$.
    On a par $(hr1)$ que $\Pi \vdash A_1^\delta$ est prouvable, c'est-à-dire, $\Pi \vdash A_1$ (par (i)) et que $\Pi \vdash A_2^\delta$ est prouvable,

c'est-à-dire, $\Pi \vdash \neg A_2$ (par (ii)). On construit alors la preuve

$$\begin{array}{lll} & \vdots & \\ (n_1) & \Pi \vdash A_1 & .. \\ & \vdots & \\ (n_2) & \Pi \vdash \neg A_2 & .. \\ (m) & \Pi \vdash A_1 \Rightarrow (\neg A_2 \Rightarrow \neg(A_1 \Rightarrow A_2) & Thm(9) \\ (m') & \Pi \vdash \neg(A_1 \Rightarrow A_2) & MP(m)(n_1, n_2) \end{array}$$

On a donc, par (iii), que $\Pi \vdash A^\delta$ est prouvable.

– si $\delta(A_1) = 1$ et $\delta(A_2) = 1$ alors $(i) A_2^\delta = A_2$ et $\delta(A) = 1$ d'où $(ii) A^\delta = A = A_1 \Rightarrow A_2$.
On a, par $(hr2)$ que $\Pi \vdash A_2^\delta$ est prouvable, c'est-à-dire, par (i), $\Pi \vdash A_2$. On construit alors la preuve

$$\begin{array}{lll} & \vdots & \\ (n) & \Pi \vdash A_2 & .. \\ (n') & \Pi \vdash A_2 \Rightarrow (A_1 \Rightarrow A_2) & Ax(\mathrm{P1}) \\ (n'') & \Pi \vdash A_1 \Rightarrow A_2 & MP(n')(n) \end{array}$$

On a donc, par (ii) que $\Pi \vdash A^\delta$ est prouvable.

Preuve de complétude Le fait 1 énonçait que toute tautologie du fragment propositionnel est prouvable. Il s'énonce maintenant un peu plus formellement comme : *si A est une tautologie alors $\vdash A$ est prouvable.*

La preuve se base sur le lemme clef 11 et procède en trois étapes : le lemme 12 qui permet d'éliminer une hypothèse P_i^δ ; le lemme 13 qui permet d'éliminer un nombre quelconque d'hypothèses P_i^δ dans la preuve d'une tautologie ; comme corollaire immédiat du lemme 13, on élimine toutes les hypothèses P_i^δ pour obtenir une preuve de $\vdash A$.

Lemme (12) pour toute formule A et tout ensemble d'atomes propositionnels, si pour toute distribution δ, $P_1^\delta, \ldots, P_n^\delta \vdash A$ est prouvable alors $P_1^\delta, \ldots, P_{n-1}^\delta \vdash A$ aussi.

Preuve Soit δ_0 une distribution. Soient alors δ_1 et δ_2 telles que
(a) $\delta_1(P_i) = \delta_0(P_i)$ si $i \in [1..n-1]$ et $\delta_1(P_n) = 1$. On a alors $P_i^{\delta_1} = P_i^{\delta_0}$ si $i \in [1..n-1]$ et $P_n^{\delta_1} = P_n$;
(b) $\delta_2(P_i) = \delta_0(P_i)$ si $i \in [1..n-1]$ et $\delta_2(P_n) = 0$. On a alors $P_i^{\delta_2} = P_i^{\delta_0}$ si $i \in [1..n-1]$ et $P_n^{\delta_2} = \neg P_n$.
Par hypothèse, $P_1^\delta, \ldots, P_n^\delta \vdash A$ est prouvable pour toute distribution δ. On a donc, en particulier que $P_1^{\delta_1}, \ldots, P_n^{\delta_1} \vdash A$ est prouvable, ainsi que $P_2^{\delta_1}, \ldots, P_n^{\delta_2} \vdash A$. C'est-à-dire, par (a) et (b) que

$$P_1^\delta, \ldots, P_{n-1}^\delta, P_n \vdash A \text{ et } P_1^\delta, \ldots, P_{n-1}^\delta, \neg P_n \vdash A \text{ sont prouvables}$$

En posant $\Pi = \{P_1^\delta, \ldots, P_{n-1}^\delta\}$, on construit alors la preuve suivante

$$\begin{array}{lll} & \vdots & \\ (n_1) & \Pi, P_n \vdash A & .. \\ (n_1') & \Pi \vdash P_n \Rightarrow A & Ded(n_1) \\ & \vdots & \\ (n_2) & \Pi, \neg P_n \vdash A & .. \\ (n_2') & \Pi \vdash \neg P_n \Rightarrow A & Ded(n_2) \\ (m) & \Pi \vdash (P_n \Rightarrow A) \Rightarrow ((\neg P_n \Rightarrow A) \Rightarrow A) & Thm(7) \\ (m') & \Pi \vdash A & MP(m)(n_1', n_2') \end{array}$$

Lemme (13) Soit A une tautologie dont les atomes propositionnels sont $P_1, \ldots, P_n$ alors, pour toute distribution δ, pour tout $k \in [0..n]$, $P_1^\delta, \ldots, P_{n-k}^\delta \vdash A$.

Preuve Par induction sur k :

– si $k = 0$ alors $n - k = n$. Comme A est une tautologie, pour toute distribution δ, on a
(i) $P_1^\delta, \ldots, P_n^\delta \vdash A^\delta$ est prouvable (lemme 11) ;
(ii) $\delta(A) = 1$ et donc $A^\delta = A$.
De (i) et (ii), on tire que $P_1^\delta, \ldots, P_n^\delta \vdash A$ est prouvable.

– si $k = k' + 1$, on a $n - k = n - (k' + 1) = (n - k') - 1$. Par hypothèse de récurrence, $P_1^\delta, \ldots, P_{n-k'}^\delta \vdash A$ est prouvable (pour toute distribution δ) d'où, par le lemme 12, $P_1^\delta, \ldots, P_{(n-k')-1}^\delta \vdash A$ est prouvable.

On achève la démonstration du théorème de complétude en utilisant le lemme 13 dans le cas particulier où $k = n$, ce qui vide l'ensemble des hypothèses P_i^δ et donne que $\vdash A$ est prouvable.

Références

[1] G. Peano, *The principles of arithmetic, presented by a new method* (1889), in [4].

[2] A. Whitehead B. Russel, *Principia mathematica*, vol. 1 (1910), vol. 2 (1912), vol. 3 (1913), Cambridge University Press.

[3] K. Gödel, *On formally undecidable propositions of* Principia Mathematica *and related systems I* (1931), in [4].

[4] J. van Heijenort ed., *From Frege to Gödel A source book in Mathematical Logic, 1879-1931*, Harvard University Press, 1981.

[5] R. Cori D. Lascar, *Logique mathématique Cours et exercices*, Tome II, Masson, 1993.

[6] E. Bouscaren, *Théorème de complètude du calcul propositionnel, Notes complèmentaires (0)*, notes de cours, 2003-04.

Table des matières