Démonstration automatique en calcul des séquents

Gilles Dowek

Gilles Dowek

Démonstration automatique en calcul des séquents

Février 2000

Merci à Damien Doligez, Jean Goubault, Thérèse Hardin, Hugo Herbelin, Claude Kirchner et Benjamin Werner pour leurs remarques.

Gilles Dowek
INRIA-Rocquencourt
B.P. 105
78153 Le Chesnay Cedex
France
Gilles.Dowek@inria.fr
http://pauillac.inria.fr/~dowek

Chapitre 1

La démonstration automatique

1.1 Programmes de démonstration automatique

Un *programme de démonstration automatique* est un programme qui prend en entrée une proposition et des axiomes et qui retourne une démonstration de cette proposition sous ces axiomes.

Par exemple, si on demande à un tel programme de démontrer la proposition Q sous les axiomes $P \Rightarrow Q$ et P, il retournera la démonstration

$$\frac{\overline{P \Rightarrow Q, P \vdash P \Rightarrow Q} \quad \overline{P \Rightarrow Q, P \vdash P}}{P \Rightarrow Q, P \vdash Q}$$

Si on lui demande une démonstration de la proposition $4 + 0 = 0 + 4$ sous l'axiome $\forall x\ \forall y\ (x + y = y + x)$, il retournera la démonstration

$$\frac{\frac{\overline{\forall x\ \forall y\ (x + y = y + x) \vdash \forall x\ \forall y\ (x + y = y + x)}}{\forall x\ \forall y\ (x + y = y + x) \vdash \forall y\ (4 + y = y + 4)}}{\forall x\ \forall y\ (x + y = y + x) \vdash 4 + 0 = 0 + 4}$$

Si on lui demande une démonstration du théorème de Pythagore sous les axiomes de la géométrie, il retournera une démonstration de ce théorème. Si on lui demande une démonstration du théorème de Fermat sous les axiomes de la théorie des ensembles, il devrait, en théorie, retourner une démonstration de ce théorème.

1.2 Mythes et réalités de la démonstration automatique

Les pionniers qui conçurent les premiers programmes de démonstration automatique à la fin des années cinquante avaient des projets très ambitieux. Certains d'entre eux prévoyaient, qu'en dix ans au plus, les programmes de démonstration automatique remplaceraient les mathématiciens et résoudraient des problèmes mathématiques ouverts. Ils pensaient aussi que donner la faculté de raisonner aux ordinateurs en ferait des machines intelligentes, c'est-à-dire capables de résoudre de nombreux problèmes, y compris hors du champ mathématique.

Ces projets sont aujourd'hui abandonnés, au moins provisoirement. D'une part, on s'est rendu compte que l'intelligence ne se réduit pas à la faculté de raisonner formellement. L'intelligence artificielle s'est alors tournée vers d'autres thèmes : l'apprentissage à partir d'exemples, le raisonnement incertain, la révision des croyances, l'utilisation de connaissances incomplètement formalisées, etc. D'autre part, si certains programmes de démonstration automatique sont, en théorie, capables de démontrer des théorèmes difficiles, le temps qu'ils demandent pour le faire est en général exorbitant, et quand le temps de recherche dépasse l'âge de l'univers ou même quelques semaines, ces programmes n'ont pas d'intérêt pratique.

S'ils ne permettent pas de rendre les ordinateurs intelligents, ni de résoudre des problèmes mathématiques difficiles, les programmes de démonstration automatique ont, en revanche, trouvé d'autres champs d'application.

1.2.1 Les bases de données déductives

Quand on interroge une base de données qui contient les informations que Charles Martel est le père de Pépin le bref et que Pépin le bref est le père de Charlemagne, en demandant qui est le grand-père de Charlemagne, on veut obtenir la réponse qu'il s'agit de Charles Martel, même si cette information n'est pas explicite dans la base de données. Un programme de démonstration automatique peut être utilisé pour construire, en un temps acceptable, un raisonnement établissant que Charles Martel est le grand-père de Charlemagne sous les axiomes

Charles Martel est le père de Pépin le bref
Pépin le bref est le père de Charlemagne
Le grand-père est le père du père.

et donc répondre à la requête.

1.2.2 Les systèmes experts

Certains systèmes informatiques, comme le célèbre système `Mycin`, sont proposés aux médecins pour les aider dans leurs diagnostics. Le médecin communique à un tel système les symptômes qu'il observe chez son patient et le système propose des maladies possibles. Pour cela, le système exploite des règles qui indiquent les symptômes associés à différentes maladies. Un tel système peut utiliser un programme de démonstration automatique qui exploite comme axiomes ces règles et les informations sur le patient. Utiliser un programme de démonstration automatique permet de changer facilement les règles quand les connaissances évoluent. Cela permet aussi d'indiquer au médecin le raisonnement qui conduit à la conclusion.

Bien d'autres problèmes peuvent se ramener à la recherche d'un raisonnement simple : les maladies cryptogamiques des tomates, le diagnostic de pannes de voiture, la prospection minière, la recherche de formes développées de molécules, etc.

1.2.3 La programmation déclarative

Plus généralement, n'importe quel programme peut s'exprimer par un ensemble de propositions et n'importe quel calcul comme la recherche d'une démonstration. Un programme calculant x^n peut se décrire par les propositions

$$x^0 = 1$$
$$x^{n+1} = x * x^n$$

L'exécution de ce programme sur les entrées 2 et 10, c'est-à-dire le calcul de 2^{10}, se ramène à la recherche d'un nombre p et d'une démonstration de la proposition $2^{10} = p$ utilisant les axiomes ci-dessus.

Ce principe sert de base aux langages de programmation déclaratifs, comme le langage Prolog, dans lequel on programme le calcul de x^n par les deux clauses

```
e(A,0,1).
e(A,P,S) :- Q is P - 1, e(A,Q,R), S is A * R.
```

1.2.4 L'aide à la vérification de démonstrations

Certains programmes de démonstration automatique sont aussi utiles comme modules de programmes de vérification de démonstrations. Ces programmes, dont le système `Coq` est un exemple, sont moins ambitieux que les programmes de démonstration automatique, puisqu'ils prennent en argument une proposition, des

axiomes, et une démonstration, et qu'ils se contentent de vérifier que la démonstration est correcte. Quand on utilise de tels programmes, l'écriture complète d'une démonstration formelle est souvent fastidieuse, et on veut pouvoir ne démontrer que les lemmes difficiles, laissant les lemmes faciles à un programme de démonstration automatique. Ainsi, quand on veut déduire la proposition $0 \leq 3$ de $\forall x\ (0 \leq x)$, on ne veut pas indiquer qu'il faut utiliser la règle d'élimination du quantificateur universel "quel que soit" avec le terme 3, mais on veut laisser le système chercher une démonstration de $0 \leq 3$ sous l'axiome $\forall x\ (0 \leq x)$.

Il n'y a, en fait, pas de frontière nette entre programmes de vérification de démonstration et programmes de démonstration automatique. Il y a un spectre de programmes qui prennent en argument une proposition et des indications pour la démontrer et qui retournent une démonstration complète. À un bout du spectre, les programmes de démonstration automatique demandent une indication minimale, c'est-à-dire nulle, à l'autre bout, les programmes de vérification demandent une indication maximale, c'est-à-dire la démonstration complète.

1.2.5 Démonstrations fastidieuses

Certaines démonstrations, par exemple la démonstration de l'équivalence fonctionnelle de deux circuits logiques, ou de l'absence de situation de blocage dans un protocole de communication, sont plus fastidieuses que difficiles, car elles demandent l'exploration d'un grand nombre de cas simples. De telles démonstrations peuvent être produites, en un temps raisonnable, par des programmes de démonstration automatique, alors qu'il est totalement exclu de les faire à la main.

1.2.6 Champs restreints des mathématiques

Enfin, dans certains domaines restreints des mathématiques, comme la géométrie élémentaire ou la recherche de primitives, certains programmes de démonstration automatique spécialisés, les systèmes de calcul formel, excellent.

Ces applications variées, dont certaines répondent à des besoins industriels, font que, malgré l'échec des projets grandioses qui ont donné naissance à leur étude, les algorithmes de démonstration automatique font aujourd'hui partie, à l'instar des algorithmes de tris ou d'inversion de matrices, des outils classiques employés par les informaticiens.

1.3 Le théorème de Church

1.3.1 Indécidabilité du problème de la décision

La première difficulté rencontrée quand on veut concevoir un programme de démonstration automatique est que cela est impossible.

En effet, on sait qu'il existe des problèmes, dits *non calculables* ou *indécidables*, qui ne peuvent pas être résolus par le calcul. Le plus célèbre de ces problèmes est le problème de l'arrêt : il n'y a pas de programme qui prend en argument le texte d'un programme p et une donnée a et retourne la valeur *vrai* si le programme p exécuté sur la valeur a termine et la valeur *faux* sinon. En effet, si un tel programme existait, il pourrait, entre autres, analyser son propre texte. On pourrait alors construire un programme qui termine si et seulement si il ne termine pas, ce qui est contradictoire.

Le problème de la décision (*Entscheidungsproblem*) est de la même nature : il n'y a pas de programme qui prend en argument une proposition et qui indique si cette proposition est démontrable en calcul des prédicats ou non. Ce théorème a été démontré indépendamment par Alonzo Church et Alan Turing en 1936. Sa démonstration repose sur le fait qu'on peut construire dans le calcul des prédicats la proposition "le programme p termine quand on l'exécute sur l'entrée a". Si on disposait d'un programme qui résolvait le problème de la décision, on pourrait l'utiliser pour résoudre le problème de l'arrêt.

1.3.2 Énumération et test

Mais ce résultat négatif doit être nuancé par deux résultats positifs. D'abord, si on ajoute au calcul des prédicats du premier ordre des axiomes (par exemple, les axiomes de la géométrie élémentaire), il se peut, bien que ce soit rare, qu'on obtienne une théorie décidable.

Ensuite, si le problème de la décision n'est pas décidable, il est en revanche semi-décidable. Cela signifie qu'il existe un programme qui prend en argument une proposition et

— retourne la valeur *vrai* (ou une démonstration) quand la proposition est démontrable,
— peut poursuivre son exécution à l'infini (c'est-à-dire boucler) quand la proposition n'est pas démontrable.

Ici, contrairement au paragraphe précédent, on ne demande pas que le programme termine et retourne la valeur *faux* à chaque fois que la proposition n'est pas démontrable.

Il est en fait facile de construire un tel programme. Dans les systèmes à la Frege-Hilbert une démonstration est une suite de propositions telle que chaque proposition soit produite par une règle de déduction à partir de propositions situées avant elle dans cette suite (avec une règle de déduction sans prémisse pour les axiomes). En déduction naturelle, une démonstration est une suite de séquents telle que chaque séquent soit produit par une règle de déduction à partir de séquents situés avant lui dans la suite. On peut construire un programme qui prend en argument une chaîne de caractères et un séquent et indique si la chaîne de caractères est une démonstration du séquent ou non. Il suffit de s'assurer que la chaîne de caractères est une suite de séquents et de vérifier ensuite, étape par étape, que chacun de ces séquents est produit par une règle de déduction à partir de séquents situés avant lui dans la suite.

On peut ensuite construire un programme qui énumère toutes les chaînes de caractères d'une lettre : a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, puis toutes les chaînes de caractères de deux lettres : aa, ab, ac, ad, ae, af, ag, ah, ai, aj, ak, al, am, an, ao, ap, aq, ar, as, at, au, av, aw, ax, ay, az, ba, bb, bc, bd, be, bf, bg, bh, bi, bj, bk, bl, bm, bn, bo, bp, bq, br, bs, bt, bu, bv, bw, bx, by, bz, ca, cb, cc, cd, ce, cf, cg, ch, ci, cj, ck, cl, cm, cn, co, cp, cq, cr, cs, ct, cu, cv, cw, cx, cy, cz, da, db, dc, dd, de, df, dg, dh, di, dj, dk, dl, dm, dn, do, dp, dq, dr, ds, dt, du, dv, dw, dx, dy, dz, ea, eb, ec, ed, ee, ef, eg, eh, ei, ej, ek, el, em, en, eo, ep, eq, er, es, et, eu, ev, ew, ex, ey, ez, fa, fb, fc, fd, fe, ff, fg, fh, fi, fj, fk, fl, fm, fn, fo, fp, fq, fr, fs, ft, fu, fv, fw, fx, fy, fz, ga, gb, gc, gd, ge, gf, gg, gh, gi, gj, gk, gl, gm, gn, go, gp, gq, gr, gs, gt, gu, gv, gw, gx, gy, gz, ha, hb, hc, hd, he, hf, hg, hh, hi, hj, hk, hl, hm, hn, ho, hp, hq, hr, hs, ht, hu, hv, hw, hx, hy, hz, ia, ib, ic, id, ie, if, ig, ih, ii, ij, ik, il, im, in, io, ip, iq, ir, is, it, iu, iv, iw, ix, iy, iz, ja, jb, jc, jd, je, jf, jg, jh, ji, jj, jk, jl, jm, jn, jo, jp, jq, jr, js, jt, ju, jv, jw, jx, jy, jz, ka, kb, kc, kd, ke, kf, kg, kh, ki, kj, kk, kl, km, kn, ko, kp, kq, kr, ks, kt, ku, kv, kw, kx, ky, kz, la, lb, lc, ld, le, lf, lg, lh, li, lj, lk, ll, lm, ln, lo, lp, lq, lr, ls, lt, lu, lv, lw, lx, ly, lz, ma, mb, mc, md, me, mf, mg, mh, mi, mj, mk, ml, mm, mn, mo, mp, mq, mr, ms, mt, mu, mv, mw, mx, my, mz, na, nb, nc, nd, ne, nf, ng, nh, ni, nj, nk, nl, nm, nn, no, np, nq, nr, ns, nt, nu, nv, nw, nx, ny, nz, oa, ob, oc, od, oe, of, og, oh, oi, oj, ok, ol, om, on, oo, op, oq, or, os, ot, ou, ov, ow, ox, oy, oz, pa, pb, pc, pd, pe, pf, pg, ph, pi, pj, pk, pl, pm, pn, po, pp, pq, pr, ps, pt, pu, pv, pw, px, py, pz, qa, qb, qc, qd, qe, qf, qg, qh, qi, qj, qk, ql, qm, qn, qo, qp, qq, qr, qs, qt, qu, qv, qw, qx, qy, qz, ra, rb, rc, rd, re, rf, rg, rh, ri, rj, rk, rl, rm, rn, ro, rp, rq, rr, rs, rt, ru, rv, rw, rx, ry, rz, sa, sb, sc, sd, se, sf, sg, sh, si, sj, sk, sl, sm, sn, so, sp, sq, sr, ss, st, su, sv, sw, sx, sy, sz, ta, tb, tc, td, te, tf, tg, th, ti, tj, tk, tl, tm, tn, to, tp, tq, tr, ts, tt, tu, tv, tw, tx, ty, tz, ua, ub, uc, ud, ue, uf, ug, uh, ui, uj, uk, ul, um, un, uo, up, uq, ur, us, ut, uu, uv, uw, ux, uy, uz, va, vb, vc, vd, ve, vf, vg, vh, vi, vj, vk, vl, vm, vn, vo, vp, vq, vr, vs, vt, vu, vv, vw, vx, vy, vz, wa, wb, wc, wd, we, wf, wg, wh, wi, wj, wk, wl, wm, wn, wo, wp, wq, wr, ws, wt, wu, wv, ww, wx, wy, wz, xa, xb, xc, xd, xe, xf, xg, xh, xi, xj, xk, xl, xm, xn, xo, xp, xq, xr, xs, xt, xu, xv, xw, xx, xy, xz, ya, yb, yc, yd, ye, yf, yg, yh, yi, yj, yk, yl, ym, yn, yo, yp, yq, yr, ys, yt, yu, yv, yw, yx, yy, yz, za, zb, zc, zd, ze, zf, zg, zh, zi, zj, zk, zl, zm, zn, zo, zp, zq, zr, zs, zt, zu, zv, zw, zx, zy, zz, puis toutes les chaînes de caractères de trois lettres, etc. En énumérant ainsi toutes les chaînes de caractères, on énumère tous les textes qu'il est possible d'écrire (on a déjà produit les mots “a”, “bu”, “de”, “do”, “fa”, “il”, “je”, “la”, “le”, “mi”, “oh”, “or”, “os”, “re”, “ri”, “si”, “un”, etc. *Voyelles* apparaîtra quand nous aurons atteint quelques centaines de lettres, *la Bibliothèque de Babel* quelques milliers et *Bouvard et Pécuchet* un peu plus tard). En étendant l'alphabet, on peut ajouter les symboles mathématiques que l'on veut.

On peut alors pour chacune de ces chaînes de caractères se demander si c'est une démonstration du séquent à démontrer ou non. Si le séquent a une démonstration celle-ci finira par apparaître dans l'énumération et il suffira alors d'afficher la valeur *vrai* (ou la démonstration elle-même). Sinon l'énumération se poursuivra à l'infini.

1.3.3 Recherche simultanée d'une démonstration de la négation

On peut améliorer cette méthode en cherchant simultanément une démonstration, sous les axiomes, de la proposition à démontrer P et de sa négation $\neg P$. Quatre cas peuvent alors se produire

- si la proposition P est démontrable, mais pas sa négation $\neg P$, on trouvera une démonstration de P et la recherche s'arrêtera,
- si la négation $\neg P$ est démontrable, mais pas la proposition P, on trouvera une démonstration de $\neg P$ et la recherche s'arrêtera,
- si la proposition P et sa négation $\neg P$ sont toutes deux démontrables, on trouvera une démonstration de l'une ou de l'autre en fonction de l'ordre d'énumération et la recherche s'arrêtera,
- si ni la proposition P, ni sa négation $\neg P$ ne sont démontrables, la recherche se poursuivra à l'infini.

Le troisième cas est pathologique : quand, sous un ensemble d'axiomes, une proposition et sa négation sont toutes deux démontrables l'ensemble d'axiomes est dit *contradictoire*. C'est par exemple le cas de l'ensemble comprenant à la fois l'axiome "Pépin le bref est le père de Charlemagne" et "Pépin le bref n'est pas le père de Charlemagne". Si on se place sous un ensemble d'axiomes cohérent (c'est-à-dire non contradictoire), ce cas ne peut pas se produire.

Sous un ensemble d'axiomes cohérent, si on trouve une démonstration de $\neg P$ on peut en déduire que P n'est pas démontrable. Le programme indique donc que la proposition P est démontrable à chaque fois qu'elle l'est, il indique qu'elle n'est pas démontrable à chaque fois qu'elle ne l'est pas et que sa négation $\neg P$ l'est. Le seul cas de non terminaison se produit quand la proposition P est indéterminée sous les axiomes, c'est-à-dire quand elle n'est pas démontrable et que sa négation ne l'est pas non plus.

Par exemple, si A est une variable et qu'on cherche à démontrer $A \Rightarrow A$ sans axiomes, le programme indiquera que cette proposition est démontrable. Si on cherche à démontrer $A \wedge \neg A$, le programme indiquera que la proposition n'est pas démontrable (car sa négation l'est et l'ensemble vide d'axiomes est cohérent). Si on cherche à démontrer la proposition A, le programme ne terminera pas. En effet ni la proposition A, ni $\neg A$ ne sont démontrables sans axiomes. De même, la proposition $2 + 2 = 4$ n'est pas démontrable si on ne pose pas quelques axiomes.

Le raisonnement ci-dessus permet de montrer au passage que pour tous les ensembles d'axiomes cohérents et indécidables il y a des propositions indéterminées, sinon le quatrième cas ne se produirait jamais et ce programme serait un programme de décision.

On trouvera un exposé détaillé sur toutes ces questions dans [1].

1.3.4 Vers une méthode utilisable

Cette *méthode d'énumération et test* a le mérite d'exister et d'être un outil pour montrer la semi-décidabilité du calcul des prédicats, mais ce sont bien là ses seuls mérites. En pratique, elle engendre tant de chaînes de caractères inutiles, que même pour démontrer un théorème facile, elle demande un temps exorbitant.

Néanmoins, cette idée d'énumération et de test est à la base des méthodes plus efficaces qui sont utilisables en pratique. Simplement, ces méthodes utilisent une procédure d'énumération moins grossière. Par exemple, elles engendrent uniquement les suites de séquents, ou même uniquement les suites de séquents qui sont des débuts possibles de démonstrations (il est inutile d'engendrer toutes les suites qui commencent par un séquent qui n'est pas produit par la règle *axiome*, aucune d'elles ne sera une démonstration), ou encore uniquement les suites de séquents qui sont des fins possibles de démonstration (il est inutile d'engendrer toutes les suites qui ne se terminent pas par le séquent à démontrer, aucune d'elles ne sera une démonstration de ce séquent).

Anticiper l'échec du test, pour brider l'énumération est un problème central en démonstration automatique.

Cette idée d'énumération (bridée) et de test se retrouve également quand on étudie la manière dont un lycéen, ou un mathématicien, recherche une démonstration "à la main". Pour démontrer qu'un triangle est isocèle, il pourra ou bien chercher à démontrer que deux cotés sont égaux, ou bien que deux angles sont égaux. Bien souvent, il teste la première idée, puis la seconde si la première échoue.

Chapitre 2

La logique du premier ordre

2.1 Les langages du premier ordre

Un langage logique permet de désigner des choses (la Lune, Pépin le bref, le nombre deux, l'ensemble des nombres pairs, ...) et d'exprimer des faits (la Lune est un satellite de la Terre, Pépin le bref est le père de Charlemagne, le nombre deux est pair, l'ensemble des nombres pairs est infini, ...). Une expression qui permet d'exprimer une chose est appelée un *terme*, une expression qui permet d'exprimer un fait une *proposition*.

Un langage logique est constitué de *variables*, de *symboles d'individu*, de *symboles de fonction* et de *symboles de prédicat*. Les variables doivent être en nombre infini. À chaque symbole de fonction et de prédicat on associe son nombre d'arguments ou *arité*.

Par exemple, le langage de l'*arithmétique* est constitué
— de variables x, x_0, x_1, x_2, ..., y, y_0, ...
— du symbole d'individu 0,
— des symboles de fonction S (*successeur*), $+$ et $\times$,
— du symbole de prédicat $=$.
L'arité du symbole S est 1, celle des symboles $+$, $\times$ et $=$ est 2.

Les symboles d'individu peuvent être vus comme des symboles de fonction d'arité nulle. Les symboles de prédicat d'arité nulle sont parfois appelés *symboles de proposition*. Un exemple de symbole de proposition en français est le symbole "pleut" dans la phrase "(il) pleut".

2.1.1 Les termes

Les *termes* sont formés par les deux règles suivantes :
— les variables sont des termes,
— les symboles d'individu sont des termes,
— si f est un symbole de fonction d'arité n et $t_1, ..., t_n$ sont des termes alors l'expression $f(t_1, ..., t_n)$ est un terme.

Par exemple, en arithmétique, les expressions suivantes sont des termes 0, $S(0)$ (plus informellement notée 1), $S(S(0))$, (plus informellement notée 2), $S(S(S(0)))$, (plus informellement notée 3), $+(0, 0)$ (plus informellement notée $0 + 0$), x, $+(x, y)$, $+(\times(x, S(S(0))), S(y))$, ...

On a, bien souvent, d'une définition plus rigoureuse de la notion de terme. On doit alors définir les termes comme des suites finies de symboles pris dans l'ensemble constitué des variables, des symboles d'individu, de fonction et de prédicat et des *symboles impropres* "(", ")", ",". Une définition alternative consiste à les définir comme des arbres dont les nœuds internes sont labellés par des symboles de fonction et les feuilles par des variables et des symboles d'individu.

Un terme, tel $+(\times(x, S(S(0))), S(y))$, qui contient des variables est dit *ouvert*, un terme qui n'en contient pas est dit *clos*.

2.1.2 Les propositions

Le moyen le plus simple de former une proposition est d'appliquer un symbole de prédicat à un certain nombre de termes, on peut, par exemple, former ainsi les propositions $= (+(S(0), S(0)), S(S(0)))$, c'est-à-dire $1 + 1 = 2$, ou $= (+(S(0), S(0)), S(S(S(0))))$, c'est-à-dire $1 + 1 = 3$. On a aussi besoin de former des propositions plus complexes, pour cela on utilise les connecteurs $\bot$ ("contradiction"), $\neg$ ("non"), $\wedge$ ("et"), $\vee$ ("ou"), $\Rightarrow$ ("implique"), $\Leftrightarrow$ ("si et seulement si"), et de quantificateurs $\forall$ ("pour tout") et $\exists$ ("il existe").

Les quantificateurs servent à exprimer que tous les objets du discours vérifient une propriété, ou qu'au moins un objet vérifie une propriété, mais sans spécifier lequel. Pour exprimer de telles propositions, les langues naturelles, comme le français, utilisent des pronoms indéfinis, par exemple "tous" et "quelques" : "tous les hommes sont mortels", "tous les nombres entiers sont supérieurs à 0", ... Formellement on écrirait cette dernière phrase

$$\text{tout} \geq 0$$

dans laquelle le symbole "tout" prend la place d'un terme comme argument du symbole de prédicat.

Hélas, ce mécanisme est ambigu quand plusieurs termes sont remplacés par de tels symboles. Par exemple la phrase "il y a un nombre entier supérieur à tout nombre entier" peut signifier ou bien que pour chaque nombre entier, il existe un nombre entier qui lui est supérieur (ce qui est vrai) ou bien qu'il existe un nombre qui est est supérieur à tous les nombres entiers (ce qui est faux).

On préfère donc mettre une variable comme argument du prédicat

$$x \geq 0$$

et indiquer ensuite la signification et la portée de cette variable par un quantificateur

$$\forall x \ (x \geq 0)$$

Ainsi, on distingue les propositions

$$\forall x \ \exists y \ (y \geq x)$$

qui est vraie et

$$\exists y \ \forall x \ (y \geq x)$$

qui est fausse.

Les *propositions* sont donc formés par les règles suivantes :

— si P est un symbole de prédicat d'arité n et $t_1, ..., t_n$ sont des termes alors l'expression $P(t_1, ..., t_n)$ est une proposition,

— $\bot$ est une proposition, si A est une proposition alors $\neg A$ est une proposition et si A et B sont des propositions alors $A \wedge B$, $A \vee B$, $A \Rightarrow B$ et $A \Leftrightarrow B$ sont des propositions,

— si A est une proposition et x une variable alors $\forall x \ A$ et $\exists x \ A$ sont des propositions.

2.2 Les démonstrations

On veut maintenant se donner les outils qui permettent de démontrer des propositions.

2.2.1 Les axiomes et les règles de déduction

Une démonstration d'une proposition A est une suite finie de propositions telle que la dernière proposition de la suite est A et chaque proposition de la suite est ou bien un *axiome*, c'est-à-dire une proposition dont la vérité est admise sans argumentation, ou bien produite par une *règle de déduction* à partir de propositions la précédant dans la suite.

Par exemple dans un système logique dans lequel on a trois axiomes

$$P \Rightarrow (Q \Rightarrow R)$$

$$P$$

$$Q$$

et une règle de déduction

$$\text{de } A \Rightarrow B \text{ et } A \text{ déduire } B$$

qu'on écrit

$$\frac{A \Rightarrow B \quad A}{B}$$

on a la démonstration suivante de la proposition R :

$$P \Rightarrow (Q \Rightarrow R), P, Q \Rightarrow R, Q, R$$

En effet, la première proposition de cette suite ($P \Rightarrow (Q \Rightarrow R)$) est un axiome, la deuxième (P) aussi, la troisième $Q \Rightarrow R$ est produite par la règle de déduction ci-dessus à partir de la première et de la deuxième, la quatrième (Q) est un axiome et la cinquième R est produite par la règle de déduction ci-dessus à partir de la troisième et de la quatrième.

2.2.2 Les suites et les arbres

Dans ce même système on a aussi la démonstration

$$Q, P \Rightarrow (Q \Rightarrow R), P, Q \Rightarrow R, R$$

En toutes rigueur ces deux démonstrations sont différentes : ce sont deux suites différentes de propositions. Pourtant elles expriment le même argument intuitif. Qu'on invoque l'axiome Q après avoir démontré la proposition $Q \Rightarrow R$ ou avant n'a pas d'importance.

Pour ne pas introduire un ordre arbitraire entre les propositions indépendantes, on préfère souvent définir les démonstration comme des arbres.

Une démonstration d'une proposition A se définit alors comme arbre dont chaque nœud est étiqueté par une proposition tel que la racine soit étiquetée par la proposition A et que la proposition étiquetant un nœud soit produite par une règle de déduction à partir des propositions étiquettes de ses fils.

Ainsi dans le système ci-dessus on a la démonstration

$$\frac{\dfrac{P \Rightarrow (Q \Rightarrow R) \quad P}{Q \Rightarrow R} \qquad Q}{R}$$

2.2.3 L'introduction d'hypothèses

Pour démontrer que $(n = 0) \Rightarrow (n + 1 = 1)$, on veut pouvoir supposer que $n = 0$ puis démontrer que $n + 1 = 1$. La notion de démonstration du chapitre précédent ne permet pas de faire cela car une règle de déduction transforme la proposition à démontrer mais pas les axiomes qu'on peut utiliser. Si on veut pouvoir poser l'introduction d'hypothèse comme une règle de déduction, il faut changer la notion de règle de déduction de manière à ce qu'une règle p puisse modifier aussi bien la proposition P que la liste d'axiomes Γ.

Une démonstration n'est donc plus un arbre de propositions, mais un arbre de couples (Γ, P) où Γ est une liste de propositions et P une proposition. Un tel couple se note $\Gamma \vdash P$ (lire "Gamma thèse P") et est appelé un séquent

La règle d'introduction d'hypothèses se formule alors ainsi

$$\text{du séquent } \Gamma, A \vdash B, \text{ déduire le séquent } \Gamma \vdash A \Rightarrow B$$

On la note

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B}$$

On peut alors maintenant donner les règles de déduction correspondant à tous les connecteurs et quantificateurs. Ces règles sont classées en fonction du connecteur ou quantificateur qu'elles concernent. Parmi les règles concernant un même quantificateur, on les classe en *règle d'élimination* et *règle d'introduction* selon que ce connectuer ou quantificateur apparait en prémisse ou en conclusion de la règle.

2.2.4 Les Règles de la déduction naturelle

$$\frac{}{\Gamma \vdash A}\ \text{axiome} \quad \text{si } A \in \Gamma$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B}\ \Rightarrow\text{-intro}$$

$$\frac{\Gamma \vdash A \Rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B}\ \Rightarrow\text{-élim}$$

$$\frac{\Gamma, A \vdash B \quad \Gamma, B \vdash A}{\Gamma \vdash A \Leftrightarrow B}\ \Leftrightarrow\text{-intro}$$

$$\frac{\Gamma \vdash A \Leftrightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B}\ \Leftrightarrow\text{-élim}$$

$$\frac{\Gamma \vdash A \Leftrightarrow B \quad \Gamma \vdash B}{\Gamma \vdash A}\ \Leftrightarrow\text{-élim}$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\ \wedge\text{-intro}$$

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}\ \wedge\text{-élim}$$

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}\ \wedge\text{-élim}$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\ \vee\text{-intro}$$

$$\frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\ \vee\text{-intro}$$

$$\frac{\Gamma \vdash A \vee B \quad \Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma \vdash C}\ \vee\text{-élim}$$

$$\frac{\Gamma, A \vdash \bot}{\Gamma \vdash \neg A}\ \neg\text{-intro}$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash \neg A}{\Gamma \vdash \bot}\ \neg\text{-élim}$$

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash A}\ \bot\text{-élim}$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash \forall x\ A}\ \forall\text{-intro} \quad \text{si } x \text{ n'apparaît pas dans } \Gamma$$

$$\frac{\Gamma \vdash \forall x\ A}{\Gamma \vdash A[x \leftarrow t]}\ \forall\text{-élim}$$

$$\frac{\Gamma \vdash A[x \leftarrow t]}{\Gamma \vdash \exists x\ A}\ \exists\text{-intro}$$

$$\frac{\Gamma \vdash \exists x\ A \quad \Gamma, A \vdash B}{\Gamma \vdash B}\ \exists\text{-élim} \quad \text{si } x \text{ n'apparaît pas dans } \Gamma \text{ ni dans } B$$

$$\frac{}{\Gamma \vdash A \vee \neg A}\ \text{tiers exclu}$$

La notation $A[x \leftarrow t]$ désigne la proposition A dans laquelle la variable x a été substituée par le terme t. Cette proposition se définit ainsi par récurrence sur la structure de A :

- $x[x \leftarrow t] = t$,
- si y est une variable distincte de x, alors $y[x \leftarrow t] = y$,
- $f(a_1, ..., a_n)[x \leftarrow t] = f(a_1[x \leftarrow t], ..., a_n[x \leftarrow t])$,
- $P(a_1, ..., a_n)[x \leftarrow t] = P(a_1[x \leftarrow t], ..., a_n[x \leftarrow t])$,
- $\bot[x \leftarrow t] = \bot$
- $(\neg A)[x \leftarrow t] = \neg A[x \leftarrow t]$
- $(A \wedge B)[x \leftarrow t] = A[x \leftarrow t] \wedge B[x \leftarrow t]$, $(A \vee B)[x \leftarrow t] = A[x \leftarrow t] \vee B[x \leftarrow t]$, $(A \Rightarrow B)[x \leftarrow t] = A[x \leftarrow t] \Rightarrow B[x \leftarrow t]$, $(A \Leftrightarrow B)[x \leftarrow t] = A[x \leftarrow t] \Leftrightarrow B[x \leftarrow t]$,
- $(\forall x\ A)[x \leftarrow t] = \forall x\ A$, $(\exists x\ A)[x \leftarrow t] = \exists y\ A$,
- si y est une variable distincte de x, $(\forall y\ A)[x \leftarrow t] = \forall y\ A[x \leftarrow t]$, $(\exists y\ A)[x \leftarrow t] = \exists y\ A[x \leftarrow t]$. Dans ce cas, il faut que y n'apparaisse pas dans t. Si c'est le cas, il est nécessaire de renommer la variable y dans $\forall y\ A$ ou $\exists y\ A$ avec une nouvelle variable.

Une démonstration d'une proposition A sous les axiomes Γ se définit alors comme un arbre dont chaque nœud est étiqueté par un séquent tel que la racine soit étiquetée par le séquent $\Gamma \vdash A$ et que le séquent étiquetant un nœud soit produit par une règle de déduction naturelle à partir des séquents étiquetant de ses fils.

Chapitre 3

Le calcul des séquents

3.1 Recherche de démonstration en déduction naturelle

Au lieu d'énumérer toutes les chaînes de caractères pour trier ensuite celles qui sont des démonstrations de celles qui n'en sont pas, essayons d'énumérer directement toutes les fins possibles de démonstrations d'un séquent $\Gamma \vdash A$. Cette méthode s'appelle le *chaînage arrière* car la progression s'effectue de la conclusion vers les axiomes. Elle s'oppose à l'énumération des débuts possibles de démonstrations, le *chaînage avant* qui procède des axiomes vers la conclusion.

Cherchons une démonstration de la proposition $Q \Rightarrow (P \wedge Q)$ sous l'axiome P. Nous cherchons donc une démonstration du séquent $P \vdash Q \Rightarrow (P \wedge Q)$. Une démonstration en déduction naturelle de ce séquent doit se terminer par l'une des règles de ce système. Explorons les possibilités l'une après l'autre. Parmi ces possibilités, nous considérons le cas où la règle utilisée est la règle d'introduction de l'implication

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B} \Rightarrow\text{-intro}$$

Dans ce cas, la seule possibilité est d'avoir $\Gamma = [P], A = Q, B = P \wedge Q$, nous nous ramenons donc à la recherche d'une démonstration du séquent $P, Q \vdash P \wedge Q$. Pour cela, énumérons encore toutes les règles possibles. Parmi ces possibilités, nous considérons le cas où la règle utilisée est la règle d'introduction de la conjonction "et"

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \wedge\text{-intro}$$

La seule possibilité est d'avoir $\Gamma = [P, Q], A = P, B = Q$, nous nous ramenons donc à la recherche d'une démonstration des séquents $P, Q \vdash P$ et $P, Q \vdash Q$. Cherchons d'abord une démonstration du premier. Nous énumérons encore toutes les règles possibles et parmi elles la règle *axiome*

$$\frac{}{\Gamma, A \vdash A} \text{ axiome}$$

qui permet de conclure. La même règle permet également de démontrer le second séquent. On aboutit donc à la démonstration

$$\frac{\dfrac{\dfrac{}{P, Q \vdash P}\text{ axiome} \quad \dfrac{}{P, Q \vdash Q}\text{ axiome}}{P, Q \vdash P \wedge Q} \wedge\text{-intro}}{P \vdash Q \Rightarrow (P \wedge Q)} \Rightarrow\text{-intro}$$

Quand on énumère toutes les règles possibles pour démontrer le séquent $P, Q \vdash P \wedge Q$, parmi toutes les règles d'introduction, seule l'introduction de la conjonction "et" est possible. En effet, l'introduction de la

disjonction “ou”, par exemple,

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \vee\text{-intro}$$

ne permet de démontrer que des propositions de la forme $A \vee B$, donc elle ne peut pas être utilisée pour démontrer la proposition $P \wedge Q$. En revanche, à chaque étape de la recherche toutes les règles d'élimination peuvent être utilisées. Par exemple la règle

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \wedge\text{-élim}$$

peut être utilisée, quelle que soit la forme de la proposition à démontrer.

De plus, quand on utilise cette règle, le séquent $P, Q \vdash P \wedge Q$, nous suggère de prendre $\Gamma = [P, Q]$ et $A = P \wedge Q$, mais il ne suggère rien pour B qui n'apparaît pas dans la conclusion. Pour que la recherche soit complète, il faut donc énumérer toutes les propositions possibles pour B.

De ce fait, quand on cherche à démontrer le séquent $P \wedge Q \vdash P$, dont la démonstration utilise une règle d'élimination

$$\frac{\dfrac{}{P \wedge Q \vdash P \wedge Q}\,\text{axiome}}{P \wedge Q \vdash P} \wedge\text{-élim}$$

rien n'indique qu'il faut prendre $B = Q$ et une énumération des propositions est nécessaire. (En chaînage avant, l'énumération serait ailleurs mais tout aussi importante.) Pourtant, en regardant les hypothèses du séquent $P \wedge Q \vdash P$, on voit bien qu'il faut prendre $B = Q$ pour se ramener au séquent $P \wedge Q \vdash P \wedge Q$, et conclure.

3.2 Le calcul des séquents

3.2.1 Règles gauches

En chaînage arrière, la déduction naturelle permet d'exploiter la forme de la conclusion du séquent pour guider la recherche, mais pas celle des hypothèses. L'idée du *calcul des séquents* est de conserver les règles d'introduction de la déduction naturelle, qu'on appelle désormais *règles droites* et de remplacer les règles d'élimination par des règles d'introduction sur les hypothèses du séquent : les *règles gauches*. Par exemple la règle d'élimination de la conjonction “et” est remplacée par la règle

$$\frac{\Gamma, A, B \vdash C}{\Gamma, A \wedge B \vdash C} \wedge\text{-gauche}$$

Pour démontrer, en calcul des séquents, le séquent $P \wedge Q \vdash P$, il suffit d'appliquer cette règle pour se ramener à la recherche d'une démonstration du séquent $P, Q \vdash P$ dont la conclusion figure parmi les hypothèses.

Dans la démonstration en déduction naturelle,

$$\frac{\dfrac{}{P \wedge Q \vdash P \wedge Q}\,\text{axiome}}{P \wedge Q \vdash P} \wedge\text{-élim}$$

la règle *axiome* est utilisée avec la proposition $P \wedge Q$. Alors que dans la démonstration en calcul des séquents

$$\frac{\dfrac{}{P, Q \vdash P}\,\text{axiome}}{P \wedge Q \vdash P} \wedge\text{-gauche}$$

la règle *axiome* est utilisée avec la proposition P.

Les règles gauches sont donc les suivantes

$$\frac{\Gamma \vdash A \quad \Gamma, B \vdash C}{\Gamma, A \Rightarrow B \vdash C} \Rightarrow\text{-gauche}$$

$$\frac{\Gamma, A, B \vdash C}{\Gamma, A \wedge B \vdash C} \wedge\text{-gauche}$$

$$\frac{\Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma, A \vee B \vdash C} \vee\text{-gauche}$$

$$\frac{\Gamma \vdash A}{\Gamma, \neg A \vdash B} \neg\text{-gauche}$$

$$\frac{}{\Gamma, \bot \vdash A} \bot\text{-gauche}$$

$$\frac{\Gamma, A[x \mapsto t] \vdash B}{\Gamma, \forall x\ A \vdash B} \forall\text{-gauche}$$

$$\frac{\Gamma, A \vdash B}{\Gamma, \exists x\ A \vdash B} \exists\text{-gauche} \quad \text{si } x \text{ n'apparaît pas dans } \Gamma B$$

auxquelles s'ajoutent les règles droites

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B} \Rightarrow\text{-droite}$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \wedge\text{-droite}$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \vee\text{-droite}$$

$$\frac{\Gamma \vdash B}{\Gamma \vdash A \vee B} \vee\text{-droite}$$

$$\frac{\Gamma, A \vdash \bot}{\Gamma \vdash \neg A} \neg\text{-droite}$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash \forall x\ A} \forall\text{-droite} \quad \text{si } x \text{ n'apparaît pas dans } \Gamma$$

$$\frac{\Gamma \vdash A[x \mapsto t]}{\Gamma \vdash \exists x\ A} \exists\text{-droite}$$

et la règle *axiome*

$$\frac{}{\Gamma, A \vdash A} \text{axiome}$$

3.2.2 Le tiers exclu

Le *tiers exclu* est une règle du raisonnement qui permet de démontrer la proposition $A \vee \neg A$ même si on ne sait pas démontrer A ni $\neg A$. Ainsi, on peut démontrer que "Il neigera à Paris le premier Janvier 3000 ou il ne neigera pas à Paris le premier Janvier 3000" même si on ignore tout des prévisions météorologiques à cette date. La pertinence de cette règle a été, et reste, l'objet de vifs débats philosophiques et mathématiques. On qualifie un système logique de "classique" s'il vérifie cette règle.

En déduction naturelle, le tiers exclu est une règle spéciale qui s'ajoute aux règles d'introduction de la disjonction $\vee$:

$$\frac{}{\Gamma \vdash A \vee \neg A}$$

on peut aussi le remplacer par la règle

$$\frac{\Gamma \vdash \neg\neg A}{\Gamma \vdash A}$$

Essayons de démontrer le séquent $\neg\neg(P \Rightarrow Q), P \vdash Q$. Si on applique la règle gauche de la négation on se ramène au séquent $P \vdash \neg(P \Rightarrow Q)$ qui n'est pas démontrable. L'idée est alors d'appliquer le tiers exclu pour se ramener à $\neg\neg(P \Rightarrow Q), P \vdash \neg\neg Q$ puis d'appliquer la règle droite de la négation de manière à obtenir $\neg\neg(P \Rightarrow Q), P, \neg Q \vdash \bot$ puis d'appliquer enfin la règle gauche de la négation comme ci-dessus. On obtient alors la démonstration suivante (qui se comprend mieux si on la lit de bas en haut, c'est-à-dire en partant de la conclusion).

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{P \vdash P}\text{ axiome} \quad \dfrac{}{P, Q \vdash Q}\text{ axiome}}{P, P \Rightarrow Q \vdash Q}\Rightarrow\text{-gauche}}{P, \neg Q, P \Rightarrow Q \vdash \bot}\neg\text{-gauche}}{P, \neg Q \vdash \neg(P \Rightarrow Q)}\neg\text{-droite}}{\neg\neg(P \Rightarrow Q), P, \neg Q \vdash \bot}\neg\text{-gauche}}{\neg\neg(P \Rightarrow Q), P \vdash \neg\neg Q}\neg\text{-droite}}{\neg\neg(P \Rightarrow Q), P \vdash Q}\text{ tiers exclu}$$

On peut comprendre l'utilisation du tiers exclu dans cette démonstration, comme un moyen d'éviter la destruction de la proposition Q lors de l'utilisation de la règle gauche de la négation, en stockant sa négation dans la liste d'hypothèses. Une idée alternative consiste à laisser la proposition Q dans la partie droite du séquent. Cela demande donc de considérer des séquents étendus dans lesquels il y a plusieurs hypothèses *et aussi plusieurs conclusions*. Ainsi la démonstration ci-dessus peut se réécrire

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{P \vdash P, Q}\text{ axiome} \quad \dfrac{}{P, Q \vdash Q}\text{ axiome}}{P, P \Rightarrow Q \vdash Q}\Rightarrow\text{-gauche}}{P \vdash \neg(P \Rightarrow Q), Q}\neg\text{-droite}}{\neg\neg(P \Rightarrow Q), P \vdash Q}\neg\text{-gauche}$$

Le tiers exclu peut donc s'exprimer par cette possibilité d'avoir plusieurs conclusions dans un séquent (cette idée, habituelle en calcul des séquents, peut aussi être utilisée en déduction naturelle).

Il est également possible d'avoir zéro proposition en conclusion, dans ce cas il faut interpréter le séquent $\Gamma \vdash$ comme $\Gamma \vdash \bot$. De ce fait, on peut se passer du symbole $\bot$.

3.2.3 La contraction

Pour que le calcul des séquents soit équivalent à la déduction naturelle, on doit, en outre, ajouter une règle permettant de faire une copie d'une proposition avant de l'utiliser, afin de pouvoir l'utiliser une seconde fois

$$\frac{\Gamma, A, A \vdash \Delta}{\Gamma, A \vdash \Delta}\text{ contraction-gauche}$$

$$\frac{\Gamma \vdash A, A, \Delta}{\Gamma \vdash A, \Delta}\text{ contraction-droite}$$

Ces règles sont indispensables, par exemple pour démontrer la proposition $\exists x\ ((P\ x) \Rightarrow (P\ (f\ x)))$. Si on applique la règle droite du quantificateur existentiel "il existe", on obtient un séquent $\vdash (P\ t) \Rightarrow (P\ (f\ t))$ qui

n'est pas démontrable. En revanche, en dupliquant cette proposition, on peut construire une démonstration.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(P\ a), (P\ (f\ a)) \vdash (P\ (f\ a)), (P\ (f\ (f\ a)))}\text{ axiome}}{(P\ a) \vdash (P\ (f\ a)), (P\ (f\ a)) \Rightarrow (P\ (f\ (f\ a)))}\Rightarrow\text{-droite}}{\vdash (P\ a) \Rightarrow (P\ (f\ a)), (P\ (f\ a)) \Rightarrow (P\ (f\ (f\ a)))}\Rightarrow\text{-droite}}{\vdash (P\ a) \Rightarrow (P\ (f\ a)), \exists x\ ((P\ x) \Rightarrow (P\ (f\ x)))}\exists\text{-droite}}{\vdash \exists x\ ((P\ x) \Rightarrow (P\ (f\ x))), \exists x\ ((P\ x) \Rightarrow (P\ (f\ x)))}\exists\text{-droite}}{\vdash \exists x\ ((P\ x) \Rightarrow (P\ (f\ x)))}\text{ contraction-droite}$$

3.2.4 Les règles

$$\frac{}{\Gamma, A \vdash A, \Delta}\text{ axiome}$$

$$\frac{\Gamma, A, A \vdash \Delta}{\Gamma, A \vdash \Delta}\text{ contraction-gauche}$$

$$\frac{\Gamma \vdash A, A, \Delta}{\Gamma \vdash A, \Delta}\text{ contraction-droite}$$

$$\frac{\Gamma \vdash A, \Delta \quad \Gamma, B \vdash \Delta}{\Gamma, A \Rightarrow B \vdash \Delta}\Rightarrow\text{-gauche}$$

$$\frac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A \Rightarrow B, \Delta}\Rightarrow\text{-droite}$$

$$\frac{\Gamma, A, B \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta}\wedge\text{-gauche}$$

$$\frac{\Gamma \vdash A, \Delta \quad \Gamma \vdash B, \Delta}{\Gamma \vdash A \wedge B, \Delta}\wedge\text{-droite}$$

$$\frac{\Gamma, A \vdash \Delta \quad \Gamma, B \vdash \Delta}{\Gamma, A \vee B \vdash \Delta}\vee\text{-gauche}$$

$$\frac{\Gamma \vdash A, B, \Delta}{\Gamma \vdash A \vee B, \Delta}\vee\text{-droite}$$

$$\frac{\Gamma \vdash A, \Delta}{\Gamma, \neg A \vdash \Delta}\neg\text{-gauche}$$

$$\frac{\Gamma, A \vdash \Delta}{\Gamma \vdash \neg A, \Delta}\neg\text{-droite}$$

$$\frac{\Gamma, A[x \mapsto t] \vdash \Delta}{\Gamma, \forall x\ A \vdash \Delta}\forall\text{-gauche}$$

$$\frac{\Gamma \vdash A, \Delta}{\Gamma \vdash \forall x\ A, \Delta}\forall\text{-droite}\quad \text{si } x \text{ n'apparaît pas dans } \Gamma\Delta$$

$$\frac{\Gamma, A \vdash \Delta}{\Gamma, \exists x\ A \vdash \Delta}\exists\text{-gauche}\quad \text{si } x \text{ n'apparaît pas dans } \Gamma\Delta$$

$$\frac{\Gamma \vdash A[x \mapsto t], \Delta}{\Gamma \vdash \exists x\ A, \Delta}\exists\text{-droite}$$

La condition "x n'apparaît pas dans $\Gamma\Delta$" dans la règle gauche du quantificateur existentiel "il existe" et dans la règle droite du quantificateur universel "quel que soit" est appelée *condition de fraîcheur des variables*. Si elle n'est pas vérifiée, il est nécessaire de renommer la variable liée avant d'appliquer la règle.

Dans toutes ces règles, il faut considérer les listes de propositions comme invariantes modulo permutation. Si on préfère utiliser des listes ordonnées, il faut ajouter des règles de permutation

$$\frac{\Gamma, A, B, \Gamma' \vdash \Delta}{\Gamma, B, A, \Gamma' \vdash \Delta}\ \text{permutation-gauche}$$

$$\frac{\Gamma \vdash \Delta, A, B, \Delta'}{\Gamma \vdash \Delta, B, A, \Delta'}\ \text{permutation-droite}$$

ou bien transformer les règles de manière à pouvoir appliquer une règle à une proposition quelle que soit sa position. La règle droite de la conjonction "et", par exemple, deviendrait

$$\frac{\Gamma \vdash \Delta, A, \Delta' \quad \Gamma \vdash \Delta, B, \Delta'}{\Gamma \vdash \Delta, A \wedge B, \Delta'}\ \wedge\text{-droite}$$

On peut ajouter deux règles d'affaiblissement qui sont redondantes

$$\frac{\Gamma \vdash \Delta}{\Gamma, A \vdash \Delta}\ \text{affaiblissement-gauche}$$

$$\frac{\Gamma \vdash \Delta}{\Gamma \vdash A, \Delta}\ \text{affaiblissement-droite}$$

Avec les règles d'affaiblissement, on peut aussi ne prendre qu'une forme plus faible de la règle *axiome*

$$\frac{}{A \vdash A}\ \text{axiome}$$

3.2.5 La règle de coupure

On a maintenant tous les outils pour formuler le théorème d'équivalence de la déduction naturelle et du calcul des séquents.

Théorème Un séquent de la forme $\Gamma \vdash P$ est démontrable en déduction naturelle si et seulement si il est démontrable en calcul des séquents.

Une démonstration de ce théorème peut se trouver dans [3]. Cette démonstration procède en traduisant pas à pas une démonstration d'un système dans l'autre. En fait, pour construire ces traductions, on a besoin d'une règle supplémentaire dans le calcul des séquents, la règle de coupure

$$\frac{\Gamma \vdash A, \Delta \quad \Gamma, A \vdash \Delta}{\Gamma \vdash \Delta}\ \text{coupure}$$

Et on démontre ensuite le théorème d'élimination des coupures :

Théorème Un séquent est démontrable dans le calcul des séquents avec coupures si et seulement si il est démontrable dans le calcul des séquents sans coupures.

Une démonstration de ce théorème peut se trouver dans [3].

3.2.6 Quelques restrictions équivalentes

Remarquons enfin qu'on garde un système équivalent si on se restreint à n'utiliser la règle *axiome* que sur des propositions atomiques, c'est-à-dire des propositions sans connecteurs ni quantificateurs. On peut, de

même, se restreindre à n'utiliser la règle *axiome* que quand toutes les propositions du séquent sont atomiques. Dans un tel système, le séquent $P \wedge Q \vdash P \wedge Q$ se démontre ainsi

$$\dfrac{\dfrac{\dfrac{}{P, Q \vdash P}\,\text{axiome} \quad \dfrac{}{P, Q \vdash Q}\,\text{axiome}}{P, Q \vdash P \wedge Q}\,\wedge\text{-droite}}{P \wedge Q \vdash P \wedge Q}\,\wedge\text{-gauche}$$

On peut, de même, se restreindre à n'utiliser la règle de contraction que sur des propositions non atomiques.

3.3 Recherche de démonstration en calcul des séquents

Voyons maintenant comment s'organise la recherche d'une démonstration d'un séquent $\Gamma \vdash \Delta$ en calcul des séquents.

3.3.1 Un exemple

Cherchons, par exemple, une démonstration de la proposition $(P \wedge Q) \Rightarrow (P \wedge (Q \vee R))$, c'est-à-dire du séquent $\vdash (P \wedge Q) \Rightarrow (P \wedge (Q \vee R))$.

Seules deux règles peuvent s'appliquer : la règle droite de l'implication et la contraction droite. Il y a donc deux possibilités à explorer. La première de ces possibilités donne le séquent $P \wedge Q \vdash P \wedge (Q \vee R)$.

Maintenant que nous avons plusieurs propositions, nous pouvons appliquer une règle ou bien à la proposition $P \wedge Q$ ou bien à la proposition $P \wedge (Q \vee R)$, dans chacun de ces deux cas deux règles s'appliquent : la règle logique correspondant au connecteur ou quantificateur principal et à la latéralité de la proposition (règle gauche de la conjonction "et" pour $P \wedge Q$, règle droite de la conjonction "et" pour $P \wedge (Q \vee R)$) et la règle de contraction correspondant à la latéralité de la proposition. Il y a donc quatre possibilités à explorer. La règle gauche de la conjonction "et" donne le séquent $P, Q \vdash P \wedge (Q \vee R)$.

Deux règles peuvent alors s'appliquer : la règle droite de la conjonction "et" et la contraction droite. La première de ces règles donne les deux séquents $P, Q \vdash P$ et $P, Q \vdash Q \vee R$.

Ici, un nouveaux choix se présente, chercher d'abord une démonstration du premier séquent, ou bien d'abord une démonstration du second.

Si on commence par le premier la seule règle qui peut s'appliquer est la règle *axiome*. Elle s'applique. Il ne nous reste plus qu'à chercher une démonstration pour le second séquent $P, Q \vdash Q \vee R$.

Pour ce séquent, deux règles peuvent s'appliquer : la règle droite de la disjonction "ou" ou la contraction droite. La première de ces règles donne le séquent $P, Q \vdash Q, R$ et la démonstration se conclut avec la règle *axiome*.

3.3.2 Les choix

À chaque étape de la recherche, on a donc un ensemble de séquents à démontrer, il faut donc choisir le séquent à traiter, ensuite il faut choisir la proposition à traiter dans ce séquent. Une fois cette proposition choisie, deux règles peuvent s'appliquer : la règle logique correspondant à son connecteur ou quantificateur principal et à sa latéralité et la règle de contraction correspondant à sa latéralité. Une fois le séquent, la proposition et la règle choisie, il n'y a plus de choix à faire comme c'était le cas en déduction naturelle, où même après avoir décidé d'appliquer la règle d'élimination de la conjonction "et"

$$\dfrac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}\,\wedge\text{-élim}$$

au séquent $P \wedge Q \vdash P$, il fallait encore choisir la proposition B.

Malgré tout, quand on applique la règle gauche du quantificateur universel “quel que soit” ou la règle droite du quantificateur existentiel “il existe”

$$\frac{\Gamma, A[x \mapsto t] \vdash \Delta}{\Gamma, \forall x\ A \vdash \Delta}\ \forall\text{-gauche}$$

$$\frac{\Gamma \vdash A[x \mapsto t], \Delta}{\Gamma \vdash \exists x\ A, \Delta}\ \exists\text{-droite}$$

il y a un dernier choix à faire : celui du terme t. Par exemple, si on veut cherche une démonstration du séquent $\forall x\ (0 \leq x) \vdash (0 \leq 3)$ il faut appliquer la règle gauche du quantificateur universel “quel que soit” avec le terme 3.

À chaque étape de la démonstration, il faut donc choisir
— le séquent dont on cherche une démonstration,
— la proposition à laquelle on applique une règle,
— la règle : logique ou contraction,
— le terme si cette règle est la règle gauche du quantificateur universel “quel que soit” ou la règle droite du quantificateur existentiel “il existe”.

3.3.3 Choix arborescent et choix indifférent

Dans l'organisation d'une recherche, quand on se trouve face à un choix : explorer la voie A ou explorer la voie B et qu'on choisit la voie A, deux situations peuvent se produire. En général, si la voie A mène à une impasse, il faut explorer la voie B. C'est le choix dans le labyrinthe. (Dans certains cas, il faut même commencer à explorer la voie B avant d'avoir abouti à un constat d'échec dans la voie A, car l'exploration de cette voie peut ne pas terminer.) On parle alors de *choix arborescent*.

Dans d'autres cas, si la voie A mène à un échec, on sait que la voie B mènera aussi à un échec, on parle alors de choix *indifférent*. Ainsi, pour faire des œufs mimosa, on peut commencer par faire la mayonnaise ou par faire cuire les œufs. Si on commence par faire la mayonnaise et qu'on échoue, il est inutile d'essayer de faire cuire les œufs d'abord, cela ne changera rien pour la mayonnaise.

3.3.4 La nature des choix lors de la recherche d'une démonstration

Le choix du séquent

Dans l'organisation de la recherche dans le calcul des séquents, le choix du séquent à traiter en premier est indifférent : l'ordre dans lequel on cherche les démonstrations de $P, Q \vdash P$ et $P, Q \vdash Q \vee R$ n'a pas d'importance, car chercher une démonstration d'un séquent ne change pas l'autre.

En revanche, les trois autres choix peuvent être arborescents.

Le choix de la proposition

Dans certains cas, le choix de la proposition est indifférent. Par exemple, quand on cherche une démonstration du séquent $P \wedge Q \vdash Q \vee R$, qu'on applique la règle gauche de la conjonction “et” ou la règle droite de la disjonction “ou” en premier, on se ramène toujours au séquent $P, Q \vdash Q, R$.

Mais dans le cas général, ce choix est arborescent à cause des conditions de fraîcheur des variables dans la règle droite du quantificateur universel “quel que soit” et la règle gauche du quantificateur existentiel “il existe”.

$$\frac{\Gamma \vdash A, \Delta}{\Gamma \vdash \forall x\ A, \Delta}\ \forall\text{-droite}\quad \text{si } x \text{ n'apparaît pas dans } \Gamma\Delta$$

$$\frac{\Gamma A \vdash \Delta}{\Gamma, \exists x\ A \vdash \Delta}\ \exists\text{-gauche}\quad \text{si } x \text{ n'apparaît pas dans } \Gamma\Delta$$

Si on cherche, par exemple, une démonstration du séquent $\vdash (\forall x\ (P\ x)) \vee (\exists y\ \neg(P\ y))$, on commence par appliquer la règle droite de la disjonction "ou" pour obtenir le séquent $\vdash \forall x\ (P\ x), \exists y\ \neg(P\ y)$, ensuite il faut appliquer les règles droites des deux quantificateurs de manière à obtenir le séquent $\vdash (P\ x), \neg(P\ x)$ qui se démontre avec la règle droite de la négation et la règle *axiome*. Si on commence par la règle droite du quantificateur universel "quel que soit", on obtient le séquent $\vdash (P\ x), \exists y\ \neg(P\ y)$ on applique ensuite la règle droite du quantificateur existentiel "il existe", on obtient $\vdash (P\ x), \neg(P\ x)$ et on conclut.

En revanche, si on applique d'abord la règle droite du quantificateur existentiel "il existe", on obtient le séquent $\vdash \forall x\ (P\ x), \neg(P\ x)$, la variable x apparaît libre dans le séquent, on ne peut donc plus appliquer la règle du quantificateur universel "quel que soit" sans renommer la variable x, on obtient donc le séquent $\vdash (P\ x'), \neg(P\ x)$ qui n'est pas démontrable.

Le choix de la règle

Le choix d'appliquer une règle logique ou une contraction est également parfois indifférent, par exemple si on duplique à gauche une proposition de la forme $P \wedge Q$ pour avoir $P \wedge Q$, $P \wedge Q$ pour appliquer ensuite deux fois la règle gauche de la conjonction "et" et obtenir P, Q, P, Q, on peut aussi bien commencer par appliquer la règle gauche de la conjonction "et" P, Q puis dupliquer ces deux propositions P, P, Q, Q.

En revanche, si on veut démontrer la proposition $\exists x\ ((P\ x) \Rightarrow (P\ (f\ x)))$, il faut commencer par appliquer la règle de contraction : si on applique la règle droite du quantificateur existentiel "il existe", on obtient un séquent qui n'est pas démontrable (voir ci-dessus).

Le choix du terme

Le choix du terme est naturellement arborescent, si on veut démontrer $\exists x\ ((P\ x) \Rightarrow (P\ 0))$ il faut substituer x par 0, si on le substitue par exemple avec 1, on obtient le séquent $\vdash (P\ 1) \Rightarrow (P\ 0)$ qui n'est pas démontrable.

Chapitre 4

Calcul propositionnel

4.1 Permutations des règles

On veut montrer que quand on a une démonstration d'un séquent qui contient une proposition non atomique, il est souvent possible de transformer cette démonstration en permutant des règles de manière à commencer la démonstration en appliquant à cette proposition la règle correspondant à son connecteur principal et à sa latéralité.

Cela est vrai sauf pour les propositions de la forme $\exists x\ A$ à droite et $\forall x\ A$ à gauche. Dans ces deux cas, la règle droite du quantificateur existentiel "il existe" et la règle gauche du quantificateur universel "quel que soit" ne permutent pas toujours avec la règle de contraction ni avec la règle droite du quantificateur universel "quel que soit" et la règle gauche du quantificateur existentiel "il existe".

— Une démonstration dont les deux premières règles s'appliquent à des propositions distinctes

$$\begin{array}{l} -\ R \\ -\ R' \end{array}$$

ou

$$\begin{array}{ll} -\ R & -\ R \\ \multicolumn{2}{l}{\text{———}\ R'} \end{array}$$

(si les deux occurrences de la règle R s'appliquent à la même proposition) peut se transformer en une démonstration qui commence par la règle R, ou éventuellement par une suite de contractions suivie de la règle R, sauf dans le cas où R' est la règle droite du quantificateur universel "quel que soit" ou la règle gauche du quantificateur existentiel "il existe" et R est la règle gauche du quantificateur universel "quel que soit" ou la règle droite du quantificateur existentiel "il existe". Par exemple, la démonstration

$$\dfrac{\dfrac{\dfrac{}{P, Q \vdash Q, R}\,\text{axiome}}{P \wedge Q \vdash Q, R}\,\wedge\text{-gauche}}{P \wedge Q \vdash Q \vee R}\,\vee\text{-droite}$$

se transforme en

$$\dfrac{\dfrac{\dfrac{}{P, Q \vdash Q, R}\,\text{axiome}}{P, Q \vdash Q \vee R}\,\vee\text{-droite}}{P \wedge Q \vdash Q \vee R}\,\wedge\text{-gauche}$$

mais la démonstration

$$\dfrac{\dfrac{\dfrac{\dfrac{}{(P\ x) \vdash (P\ x)}\ \text{axiome}}{\vdash (P\ x), \neg(P\ x)}\ \neg\text{-droite}}{\vdash (P\ x), \exists x\ \neg(P\ x)}\ \exists\text{-droite}}{\vdash \forall x\ (P\ x), \exists x\ \neg(P\ x)}\ \forall\text{-droite}$$

ne se transforme pas en

$$\dfrac{\dfrac{\dfrac{\dfrac{}{(P\ x) \vdash (P\ x)}\ \text{axiome}}{\vdash (P\ x), \neg(P\ x)}\ \neg\text{-droite}}{\vdash \forall x\ (P\ x), \neg(P\ x)}\ \forall\text{-droite}}{\vdash \forall x\ (P\ x), \exists x\ \neg(P\ x)}\ \exists\text{-droite}$$

qui viole la condition de fraîcheur des variables.
Les règles binaires peuvent demander d'introduire une contraction. Par exemple, la démonstration

$$\dfrac{\dfrac{\dfrac{}{(P\ a) \vdash (P\ a)}\ \text{axiome}}{(P\ a) \vdash \exists x\ (P\ x)}\ \exists\text{-droite} \qquad \dfrac{\dfrac{}{(P\ b) \vdash (P\ b)}\ \text{axiome}}{(P\ b) \vdash \exists x\ (P\ x)}\ \exists\text{-droite}}{(P\ a) \vee (P\ b) \vdash \exists x\ (P\ x)}\ \vee\text{-gauche}$$

se transforme en

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(P\ a) \vdash (P\ a), (P\ b)}\ \text{axiome}}{(P\ a) \vee (P\ b) \vdash (P\ a), (P\ b)}}{(P\ a) \vee (P\ b) \vdash (P\ a), \exists x\ (P\ x)}\ \exists\text{-droite} \qquad \dfrac{\dfrac{}{(P\ b) \vdash (P\ a), (P\ b)}\ \text{axiome}}{(P\ a) \vee (P\ b) \vdash (P\ a), \exists x\ (P\ x)}\ \vee\text{-gauche}}{(P\ a) \vee (P\ b) \vdash \exists x\ (P\ x), \exists x\ (P\ x)}\ \exists\text{-droite}}{(P\ a) \vee (P\ b) \vdash \exists x\ (P\ x)}\ \text{contraction}$$

— Une démonstration qui commence par une contraction suivie d'une règle logique sur l'une des propositions contractées peut se transformer de manière à commencer par cette règle logique, sauf si cette règle est la règle gauche du quantificateur universel "quel que soit" ou la règle droite du quantificateur existentiel "il existe". Par exemple, si on part d'une démonstration de la forme

$$\dfrac{\dfrac{\dfrac{D}{\Gamma, P, Q, P \wedge Q \vdash \Delta}}{\Gamma, P \wedge Q, P \wedge Q \vdash \Delta}\ \wedge\text{-gauche}}{\Gamma, P \wedge Q \vdash \Delta}\ \text{contraction}$$

on commence par transformer la démonstration D de manière à la faire commencer par la règle gauche de la conjonction "et"

$$\dfrac{\dfrac{\dfrac{\dfrac{D'}{\Gamma, P, Q, P, Q \vdash \Delta}}{\Gamma, P, Q, P \wedge Q \vdash \Delta}\ \wedge\text{-gauche}}{\Gamma, P \wedge Q, P \wedge Q \vdash \Delta}\ \wedge\text{-gauche}}{\Gamma, P \wedge Q \vdash \Delta}\ \text{contraction}$$

puis on transforme cette démonstration en

$$\dfrac{\dfrac{\dfrac{\dfrac{D'}{\Gamma, P, P, Q, Q \vdash \Delta}}{\Gamma, P, P, Q \vdash \Delta}\ \text{contraction}}{\Gamma, P, Q \vdash \Delta}\ \text{contraction}}{\Gamma, P \wedge Q \vdash \Delta}\ \wedge\text{-gauche}$$

Mais, comme nous l'avons vu, il n'y a pas moyen de transformer la démonstration

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(P\ a), (P\ (f\ a)) \vdash (P\ (f\ a)), (P\ (f\ (f\ a)))}\text{ axiome}}{(P\ a) \vdash (P\ (f\ a)), (P\ (f\ a)) \Rightarrow (P\ (f\ (f\ a)))}\Rightarrow\text{-droite}}{\vdash (P\ a) \Rightarrow (P\ (f\ a)), (P\ (f\ a)) \Rightarrow (P\ (f\ (f\ a)))}\Rightarrow\text{-droite}}{\vdash (P\ a) \Rightarrow (P\ (f\ a)), \exists x\ ((P\ x) \Rightarrow (P\ (f\ x)))}\exists\text{-droite}}{\vdash \exists x\ ((P\ x) \Rightarrow (P\ (f\ x))), \exists x\ ((P\ x) \Rightarrow (P\ (f\ x)))}\exists\text{-droite}}{\vdash \exists x\ ((P\ x) \Rightarrow (P\ (f\ x)))}\text{ contraction-droite}$$

de manière à commencer par la règle droite du quantificateur existentiel "il existe".

Si un séquent contient, par exemple, une proposition de la forme $A \wedge B$ à droite, et qu'il a une démonstration, on peut utiliser les propriété de permutation ci-dessus pour transformer cette démonstration en une autre qui commence par la règle droite de la conjonction "et". De ce fait, quand on cherche une démonstration d'un séquent qui contient une proposition de la forme $A \wedge B$ à droite, et qu'on applique la règle droite de la conjonction "et" à cette proposition, le choix du séquent, de la proposition et de la règle est indifférent. En effet si les séquents de l'ensemble obtenu ne sont pas tous démontrables, cela signifie que les séquents de l'ensemble de départ ne sont pas tous démontrables non plus et qu'il est inutile d'explorer une autre voie.

Un raisonnement similaire s'applique pour toutes les règles logiques du calcul des séquents sauf la règle gauche du quantificateur universel "quel que soit" et la règle droite du quantificateur existentiel "il existe".

Toutes ces règles sont dites *réversibles* car, comme nous venons de le voir, leurs prémisses sont démontrables si et seulement si leur conclusion l'est.

4.2 Le calcul propositionnel

Le calcul propositionnel est la restriction du calcul des prédicats obtenue en ne considérant que des symboles de prédicats sans arguments et en supprimant les quantificateurs et les règles sur les quantificateurs. Toutes les règles logiques du calcul des séquents propositionnel sont donc réversibles. Tant que dans un séquent, il y a une proposition non atomique, on peut donc appliquer la règle logique correspondante, sans jamais avoir à explorer d'autre voie.

Ce processus termine car il produit sans cesse des séquents comprenant moins de connecteurs logiques, et il donne un ensemble de séquents dont toutes les propositions sont atomiques. La seule règle susceptible de démontrer de tels séquents est la règle *axiome*, donc si ces séquents ont chacun une proposition commune à gauche et à droite, la proposition de départ est démontrable, sinon elle ne n'est pas.

Cet argument est à la base d'une démonstration du théorème de décidabilité du calcul propositionnel.

Théorème Le calcul propositionnel est décidable.

4.3 Une méthode de décision pour le calcul propositionnel

On commence par définir un type pour les propositions du calcul propositionnel.

```
type proposition = VarP of string
                 | NegP of proposition
                 | ImpP of proposition * proposition
                 | ConjP of proposition * proposition
                 | DisjP of proposition * proposition;;
```

On construit ensuite une fonction qui prend en argument une liste de propositions et qui permute ces propositions de manière à mettre en tête une proposition non atomique, si la liste en comporte une.

```
let rec permuteP l = permuteP_rec [] l
and permuteP_rec l' l = match l with
    [] -> l'
  | ((VarP (str))::l'') -> permuteP_rec ((VarP(str))::l') l''
  | (p::l'') -> p::(l'@l'');;
```

On construit ensuite une fonction qui prend en argument un séquent $\Gamma \vdash \Delta$ et retourne une liste des séquents dont toutes les propositions sont atomiques, obtenue en appliquant les règles logiques tant que cela est possible. Pour cela, cette fonction commence par chercher une proposition non atomique dans la liste Γ ou dans la liste Δ, si elle en trouve une, elle applique la règle correspondante, s'appelle récursivement sur les prémisses et concatène les listes de séquents obtenues. Si toutes les propositions de Γ et de Δ sont atomiques, la fonction retourne simplement le séquent $\Gamma \vdash \Delta$.

```
let rec derP gamma delta = match (permuteP gamma,permuteP delta) with
  ((NegP a)::gamma',delta')     -> derP gamma' (a::delta')
| ((ImpP(a,b))::gamma',delta')  -> (derP gamma' (a::delta'))@(derP (b::gamma') delta')
| ((ConjP(a,b))::gamma',delta') -> derP (a::b::gamma') delta'
| ((DisjP(a,b))::gamma',delta') -> (derP (a::gamma') delta')@(derP (b::gamma') delta')
| (gamma',(NegP a)::delta')     -> derP (a::gamma') delta'
| (gamma',(ImpP(a,b))::delta')  -> derP (a::gamma') (b::delta')
| (gamma',(ConjP(a,b))::delta') -> (derP gamma' (a::delta'))@(derP gamma' (b::delta'))
| (gamma',(DisjP(a,b))::delta') -> derP gamma' (a::b::delta')
| (gamma',delta')               -> [(gamma', delta')];;
```

On construit ensuite une fonction `connectP` qui prend en argument une liste de séquents et retourne le booléen `true` si chacun de ces séquents a une proposition commune à gauche et à droite et le booléen `false` sinon. Cette fonction utilise pour cela une fonction `connectP'` qui prend en argument un séquent et retourne le booléen `true` si ce séquent a une proposition commune à gauche et à droite et le booléen `false` sinon.

```
let rec connectP l = match l with
    [] -> true
  | ((gamma,delta)::l') -> (connectP' gamma delta) & (connectP l')
and connectP' gamma delta = match gamma with
    [] -> false
  | (a::gamma') -> connectP'' a delta or connectP' gamma' delta
and connectP'' a delta = match delta with
    [] -> false
  | (b::delta') -> a = b or connectP'' a delta';;
```

Enfin on construit une fonction qui prend en argument une proposition `p` et retourne le booléen `true` si cette proposition est démontrable et le booléen `false` sinon. Cette fonction calcule d'abord une liste de séquents dont toutes les propositions sont atomiques obtenue en appliquant les règles logiques tant que cela est possible au séquent $\vdash$ `p`, puis retourne le booléen `true` si chacun de ces séquent a une proposition commune à gauche et à droite et le booléen `false` sinon.

```
let derivableP p = connectP (derP [] [p]);;
```

Exemples

On peut ensuite utiliser ce programme pour chercher des démonstrations de la proposition $A \Rightarrow A$

```
# derivableP (ImpP (VarP "A",VarP "A"));;
- : bool = true
```

de la proposition $A \Rightarrow (B \Rightarrow A)$

```
# derivableP (ImpP (VarP "A",ImpP (VarP "B",VarP "A")));;
- : bool = true
```

de la proposition $A \vee \neg A$

```
# derivableP (DisjP (VarP "A", (NegP (VarP "A"))));;
- : bool = true
```

de la proposition $((A \Rightarrow B) \wedge (B \Rightarrow C)) \Rightarrow (A \Rightarrow C)$

```
# derivableP (ImpP (ConjP(ImpP(VarP "A",VarP "B"),ImpP(VarP "B",VarP "C")),
                    ImpP(VarP "A",VarP "C")));;
- : bool = true
```

de la proposition (non démontrable) $A \Rightarrow B$

```
# derivableP (ImpP (VarP "A",VarP "B"));;
- : bool = false
```

et de la proposition (non démontrable) $((A \Rightarrow B) \wedge (B \Rightarrow C)) \Rightarrow (A \Rightarrow D)$

```
# derivableP (ImpP (ConjP(ImpP(VarP "A",VarP "B"),ImpP(VarP "B",VarP "C")),
                    ImpP(VarP "A",VarP "D")));;
- : bool = false
```

Chapitre 5

Unification

Avant d'aborder le cas général de la recherche de démonstrations en calcul des prédicats, concentrons-nous sur une deuxième restriction : celle des propositions existentielles. Une proposition existentielle est une proposition de la forme $\exists x_1 \ ... \ \exists x_n \ A$ où A est une proposition sans quantificateur. L'intérêt de ce fragment du calcul des prédicats est double. D'une part, les propositions existentielles nous permettront d'introduire deux méthodes : l'unification et le contrôle de la contraction par une multiplicité, que nous retrouverons dans le cas général. D'autre part, à partir de n'importe quel séquent $\Gamma \vdash A$, on peut construire une proposition existentielle B telle que le séquent $\Gamma \vdash A$ soit démontrable si et seulement si le séquent $\vdash B$ l'est aussi. Une méthode de démonstration automatique dans le calcul des prédicats consiste donc à se ramener à une proposition existentielle. Dans un premier temps nous commençons par chercher des démonstrations sans contraction.

Nous partons avec un séquent de la forme $\vdash \exists x_1 \ ... \ \exists x_n \ A$. Puisque nous cherchons une démonstration sans contraction, la seule règle applicable est la règle droite du quantificateur existentiel "il existe", qui donne un séquent de la forme $\vdash \exists x_2 \ ... \ \exists x_n \ A'$ sur laquelle la seule règle applicable est encore la règle droite du quantificateur existentiel, etc. Après n applications de la règle droite de l'existentiel, on obtient un séquent de la forme $\vdash B$ où B est une proposition sans quantificateurs à laquelle on peut appliquer la méthode du chapitre précédent.

Le seul choix arborescent est celui du terme à substituer dans la règle droite du quantificateur existentiel "il existe". Par exemple, si on cherche une démonstration du séquent $\vdash \exists x \ ((P \ (f \ x)) \Rightarrow (P \ (f \ (f \ a))))$, il faut substituer x par le terme $(f \ a)$ de manière à se ramener au séquent $\vdash (P \ (f \ (f \ a))) \Rightarrow (P \ (f \ (f \ a)))$ qui est démontrable. Si on substitue x, par exemple, par le terme a on obtient le séquent $\vdash (P \ (f \ a)) \Rightarrow (P \ (f \ (f \ a)))$ n'est pas démontrable.

Plutôt que d'énumérer tous les termes possibles, on retarde le choix du terme en substituant à x une variable X (que pour distinguer des variables ordinaires nous appelons *métavariable* et notons par une lettre majuscule) que nous substituerons à son tour quand cela sera nécessaire.

Nous décomposons donc la démonstration en deux informations

- un *schéma de démonstration* se distingue d'une démonstration par le fait que
  - la règle droite du quantificateur existentiel "il existe" introduit une métavariable et non un terme,
  - l'arbre peut contenir des feuilles qui ne sont pas démontrées par la règle *axiome*,
- une substitution σ qui associe un terme à chaque métavariable du schéma de démonstration, et qui appliquée à chaque feuille du schéma de démonstration donne un séquent qui a une proposition commune à gauche et à droite.

La démonstration

$$\dfrac{\dfrac{\dfrac{}{(P \ (f \ (f \ a))) \vdash (P \ (f \ (f \ a)))}\text{ axiome}}{\vdash (P \ (f \ (f \ a))) \Rightarrow (P \ (f \ (f \ a)))}\Rightarrow\text{-droite}}{\vdash \exists x \ ((P \ (f \ x)) \Rightarrow (P \ (f \ (f \ a))))}\exists\text{-droite}$$

se décompose donc en un schéma de démonstration

$$\dfrac{\dfrac{(P\ (f\ X)) \vdash (P\ (f\ (f\ a)))}{\vdash (P\ (f\ X)) \Rightarrow (P\ (f\ (f\ a)))}\Rightarrow\text{-droite}}{\vdash \exists x\ ((P\ (f\ x)) \Rightarrow (P\ (f\ (f\ a))))}\exists\text{-droite}$$

et une substitution $X \mapsto (f\ a)$.

Nous définissons donc d'abord le type des termes et des propositions, avec métavariables.

```
type term = Var of string
          | Meta of string
          | Funct of string * term list;;

type prop = Atomic of string * term list
          | Neg of prop
          | Imp of prop * prop
          | Conj of prop * prop
          | Disj of prop * prop
          | Exist of string * prop
          | Univ of string * prop;;
```

Une substitution est une fonction qui associe un terme à certaines variables et métavariables, on la représente par une liste d'association. On construit une fonction qui prend en argument une substitution σ et un terme t et calcule le terme σt obtenu en substituant dans le terme t les variables et les métavariables concernées par les termes correspondants. Par exemple, si on applique la substitution $\{X \mapsto a, Y \mapsto (f\ a)\}$ au terme $(g\ X\ Y\ Z)$ on obtient le terme $(g\ a\ (f\ a)\ Z)$.

```
let rec subst s t = match t with
    (Var str) -> (try List.assoc (Var str) s with Not_found -> t)
  | (Meta str) -> (try List.assoc (Meta str) s with Not_found -> t)
  | (Funct (str,l)) -> Funct (str,List.map (subst s) l);;
```

On construit ensuite une fonction qui prend en argument une substitution σ et une proposition P et calcule le terme σP obtenu en substituant dans la proposition P les variables et les métavariables concernées par les termes correspondants. On suppose ici que les variables substituées par un terme et les variables libres des termes substitués ne sont pas liées par un quantificateur dans la proposition, ce qui est le cas dans les utilisations de cette fonction ci-dessous.

```
let rec subst_prop s p = match p with
  (Atomic (str,l)) -> (Atomic (str,List.map (subst s) l))
| (Neg a) -> Neg (subst_prop s a)
| (Imp (a,b))  -> Imp (subst_prop s a,subst_prop s b)
| (Conj (a,b)) -> Conj (subst_prop s a,subst_prop s b)
| (Disj (a,b)) -> Disj (subst_prop s a,subst_prop s b)
| (Exist (str,b)) -> Exist (str,subst_prop s b)
| (Univ (str,b)) -> Univ (str,subst_prop s b);;
```

Comme au chapitre précédent, on construit une fonction qui prend en argument un séquent $\Gamma \vdash \Delta$ et retourne une liste de séquents dont toutes les propositions sont atomiques, obtenue en appliquant les règles logiques tant que cela est possible. La seule différence avec le chapitre précédent est qu'on peut maintenant avoir des quantificateurs existentiels "il existe" à droite du séquent, dans ce cas, on engendre une nouvelle métavariable qu'on substitue à la variable quantifiée.

```
exception Error;;

let rec permute l = permute_rec [] l
and permute_rec l'  l = match l with
    [] -> l'
  | ((Atomic (str,arg))::l'') -> permute_rec ((Atomic(str,arg))::l') l''
  | (p::l'') -> p::(l'@l'');;

let rec der gamma delta = match (permute gamma, permute delta) with
  ((Neg a)::gamma',delta')        -> der gamma' (a::delta')
| ((Imp(a,b))::gamma',delta')     -> (der gamma' (a::delta'))@(der (b::gamma') delta')
| ((Conj(a,b))::gamma',delta')    -> der (a::b::gamma') delta'
| ((Disj(a,b))::gamma',delta')    -> (der (a::gamma') delta')@(der (b::gamma') delta')
| ((Exist _)::_,_)                -> raise Error
| ((Univ _)::_,_)                 -> raise Error
| (gamma',(Neg a)::delta')        -> der (a::gamma') delta'
| (gamma',(Imp(a,b))::delta')     -> der (a::gamma') (b::delta')
| (gamma',(Conj(a,b))::delta')    -> (der gamma' (a::delta'))@(der gamma' (b::delta'))
| (gamma',(Disj(a,b))::delta')    -> der gamma' (a::b::delta')
| (gamma',(Exist(str,a))::delta') ->
                              der gamma' ((subst_prop [(Var str,Meta str)] a)::delta')
| (_,(Univ _)::_)                 -> raise Error
| (gamma',delta')                 -> [(gamma',delta')];;
```

Nous obtenons une liste de séquents dont toutes les propositions sont atomiques, mais qui peuvent comporter des métavariables. Par exemple en partant de la proposition $\exists x\ ((P\ (f\ x)) \Rightarrow (P\ (f\ (f\ a))))$ nous obtenons le séquent $(P\ (f\ X)) \vdash (P\ (f\ (f\ a)))$.

Contrairement au chapitre précédent, nous ne devons plus uniquement vérifier que chaque séquent comporte une proposition commune à gauche et à droite, mais qu'il existe une substitution des métavariables telle que ce soit le cas. Il est évident que si nous trouvons une telle substitution alors le séquent de départ a une démonstration sans contraction. Réciproquement, si le séquent de départ a une démonstration sans contraction, alors il y a une telle substitution.

Par exemple en substituant la variable X par le terme $(f\ a)$, le séquent ci-dessus devient $(P\ (f\ (f\ a))) \vdash (P\ (f\ (f\ a)))$ qui est démontrable par la règle *axiome*. C'est en comparant les deux propositions, $(P\ (f\ X))$ et $(P\ (f\ (f\ a)))$ que nous trouvons le terme $(f\ a)$. Le terme $(f\ a)$ est la solution de l'équation entre propositions

$$(P\ (f\ X)) = (P\ (f\ (f\ a)))$$

Une telle équation (ou plus généralement un système de telles équations) s'appelle un *problème d'unification* et l'*algorithme d'unification* qui résout ces systèmes procède ainsi. Les solutions de l'équation

$$(P\ (f\ X)) = (P\ (f\ (f\ a)))$$

sont les mêmes que celles de l'équation

$$(f\ X) = (f\ (f\ a))$$

qui sont les mêmes que celles de l'équation

$$X = (f\ a)$$

Cette équation a une solution qui est la substitution $X \mapsto (f\ a)$.

Dans le cas général l'algorithme d'unification commence par choisir une équation dans le système.

— Si cette équation a la forme $(P\ t_1\ ...\ t_n) = (P\ u_1\ ...\ u_n)$ où P est un symbole de prédicat, un symbole de fonction ou une variable, il supprime cette équation du système, ajoute les équations $t_1 = u_1$, ..., $t_n = u_n$ et résout le système obtenu.

— Si cette équation a la forme $(P\ t_1\ ...\ t_n) = (Q\ u_1\ ...\ u_p)$ où P et Q sont des symboles de prédicat ou des symboles de fonction ou des variables distincts, l'algorithme indique que le système n'a pas de solution.
— Si cette équation a la forme $X = X$ alors il supprime cette équation du système et résout le système obtenu.
— Si cette équation a la forme $X = t$ (ou $t = X$), que X a une apparaît dans t et que t est distinct de X alors l'algorithme indique que le système n'a pas de solution.
— Si cette équation a la forme $X = t$ (ou $t = X$) et que X n'apparaît pas dans t alors, il substitue X par t dans tout le système, résout le système obtenu, ce qui donne une substitution σ et il retourne la substitution $\sigma \cup \{X \mapsto \sigma t\}$

La seule subtilité est dans la quatrième clause : une équation de la forme $X = (f\ X)$, par exemple, ne peut pas avoir de solution. Si elle en avait une par exemple le terme u, celui ci devrait être égal au terme $(f\ u)$ et donc le nombre de symboles de ce terme devrait vérifier l'équation $n = n + 1$. Ce test s'appelle le *test d'occurrence*, il est essentiel pour assurer la terminaison de l'algorithme d'unification. En effet dans le cinquième cas, la terminaison est assurée par le fait que la variable X disparaît quand on substitue X par t et donc que l'algorithme se rappelle récursivement sur un système qui comporte moins de variables.

```
let subst_eq s (a,b) = (subst s a, subst s b);;

let rec pairlist l m = match (l,m) with
    ([],[]) -> []
  | (a::l',b::m') -> (a,b)::(pairlist l' m')
  | _ -> raise Error;;

let rec occur x t = match t with
    (Var _) -> false
  | (Meta y) -> x = y
  | (Funct (_,l)) -> occur_list x l
and occur_list x l = match l with
    [] -> false
  | (t::l') -> occur x t or occur_list x l';;

exception Unif;;

let rec unif l = match l with
    [] -> []
  | ((Meta x,Meta y)::l') -> if x = y then unif l'
      else let s = (unif (List.map (subst_eq [(Meta x,Meta y)]) l'))
           in (Meta x,subst s (Meta y))::s
  | ((Meta x, t)::l') ->
     if not (occur x t)
     then let s = (unif (List.map (subst_eq [(Meta x,t)]) l'))
          in (Meta x,subst s t)::s
     else raise Unif
  | ((t, Meta x)::l') ->
     if not (occur x t)
     then let s = unif (List.map (subst_eq [(Meta x,t)]) l')
          in (Meta x,subst s t)::s
     else raise Unif
  | ((Funct (str1,l1),Funct (str2,l2))::l') ->
     if str1 = str2 then unif (l'@(pairlist l1 l2)) else raise Unif
```

```
  | ((Var str1,Var str2)::l') ->
     if str1 = str2 then unif l' else raise Unif
  | _ -> raise Unif;;

let rec unif_atomic_prop a1 a2 = match (a1,a2) with
    (Atomic (str1,l1), Atomic (str2,l2)) ->
     if str1 = str2 then (unif (pairlist l1 l2)) else raise Unif
  |  _ -> raise Error;;
```

Cet algorithme d'unification termine toujours. Il échoue si le système d'équations n'a pas de solution et il retourne une solution si le système en a une. Il se peut que le système ait plusieurs solutions. Par exemple, l'équation $X = (f\ Y)$ a, entre autres, les solutions

$$\{X \mapsto (f\ a), Y \mapsto a\}$$

$$\{X \mapsto (f\ b), Y \mapsto b\}$$

$$\{X \mapsto (f\ Z), Y \mapsto Z\}$$

$$\{X \mapsto (f\ Y)\}$$

La composition $\tau \circ \sigma$ de deux substitutions σ et τ est la substitution qui associe le terme $\tau\sigma x$ à la variable ou métavariable x si celle-ci est liée dans σ ou τ.

Une solution σ est dite *principale* si pour chaque solution τ il existe une substitution η telle que $\tau = \eta \circ \sigma$. Par exemple les deux dernières substitutions ci-dessus sont principales, mais pas les deux premières. On peut démontrer qu'un problème d'unification qui a une solution a toujours une solution principale, et que la solution calculée par l'algorithme d'unification est principale.

On peut, maintenant, construire la fonction qui cherche à substituer les métavariables de manière à ce que chacun de ces séquents ait une proposition commune à droite et à gauche.

```
let subst_seq s (gamma,delta) = (List.map (subst_prop s) gamma, List.map (subst_prop s) delta);;

let rec connect l = match l with
    [] -> true
  | ((gamma,delta)::l') -> connect' gamma delta l'
and connect' gamma delta l = match gamma with
    [] -> false
  | (a::gamma') -> connect'' a delta l or connect' gamma' delta l
and connect'' a delta l = match delta with
    [] -> false
  | (b::delta') ->
        (try let s = unif_atomic_prop a b in connect (List.map (subst_seq s) l)
        with Unif -> false)
        or (connect'' a delta' l);;
```

Cette fonction cherche à apparier les propositions à gauche et à droite des séquents les uns après les autres. Si elle construit une telle substitution alors cette substitution existe trivialement. Réciproquement s'il y a une substitution τ qui vérifie la propriété, alors cette substitution appariera une proposition à gauche et une proposition à droite du premier séquent. Ces deux propositions sont unifiables et si l'algorithme d'unification retourne une solution principale σ, alors il existe (du fait de la principalité de σ) une substitution η telle que pour chacun des autres séquents s de la liste, $\tau s = \eta(\sigma s)$. La liste obtenue en appliquant σ à chacun de ces séquents admet donc une substitution η tel que en appliquant η à chacun des séquents de cette liste on obtienne une liste de séquents qui a une proposition commune à gauche et à droite et donc la fonction ci-dessus construira une substitution.

Cela permet d'éviter l'énumération de toutes les solutions de chacun des problèmes d'unification, et de se contenter d'une solution principale.

En revanche, le choix des propositions à apparier est arborescent. En effet si on a la liste composée des deux séquents $(P\ (f\ X)), (P\ X) \vdash (P\ (f\ a))$ et $(Q\ X), (Q\ (f\ X)) \vdash (Q\ (f\ (f\ a)))$, le premier séquent peut se résoudre en appariant la conclusion avec la première hypothèse, ce qui donne $X \mapsto a$ ou avec la seconde ce qui donne $X \mapsto (f\ a)$. Dans le premier cas le second séquent devient $(Q\ a), (Q\ (f\ a)) \vdash (Q\ (f\ (f\ a)))$ qui n'est pas démontrable. En revanche avec la substitution $X \mapsto (f\ a)$, ce second séquent devient $(Q\ (f\ a)), (Q\ (f\ (f\ a))) \vdash (Q\ (f\ (f\ a)))$ qui a une proposition commune à gauche et à droite.

Enfin on conclut le programme en construisant la fonction qui indique si une proposition est démontrable sans contraction ou non.

```
let derivable p = connect (der [] [p]);;
```

Exemples

On peut maintenant chercher des démonstrations (sans contraction) de la proposition $\exists x\ ((P\ x) \Rightarrow (P\ x))$

```
# derivable
  (Exist ("x",(Imp (Atomic ("P",[Var "x"]),Atomic ("P",[Var "x"])))));;
- : bool = true
```

de la proposition $\exists x\ \exists y\ ((P\ x) \Rightarrow (P\ (f\ y)))$

```
# derivable
   (Exist ("x", Exist ("y",
     (Imp (Atomic ("P",[Var "x"]), Atomic ("P",[Funct ("f",[Var "y"])])))));;
- : bool = true
```

de la proposition $\exists x\ (((P\ x) \Rightarrow (P\ (f\ x))) \wedge (P\ a)) \Rightarrow (P\ (f\ a)))$

```
# derivable
  (Exist("x",Imp
     (Conj
       (Imp (Atomic("P",[Var "x"]),Atomic("P",[Funct("f",[Var "x"])])),
       Atomic("P",[Funct ("a",[])])),
     Atomic("P",[Funct ("f",[Funct ("a",[])])]))));;
- : bool = true
```

ainsi que de la proposition (non démontrable) $\exists x\ ((P\ (f\ x)) \Rightarrow (P\ (g\ x)))$

```
# derivable
   (Exist ("x",
     (Imp (Atomic ("P",[Funct ("f",[Var "x"])]),
           Atomic ("P",[Funct ("g",[Var "x"])])))));;
- : bool = false
```

et de la proposition (non démontrable sans contraction) $\exists x\ ((P\ x) \Rightarrow (P\ (f\ x)))$

```
# derivable
   (Exist("x",Imp (Atomic("P",[Var "x"]),Atomic("P",[Funct("f",[Var "x"])])))));;
- : bool = false
```

Chapitre 6

Multiplicité

Cette dernière proposition $\exists x\ ((P\ x) \Rightarrow (P\ (f\ x)))$ n'est donc pas démontrable sans utiliser la contraction. Mais elle est pourtant démontrable en utilisant cette règle.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(P\ a), (P\ (f\ a)) \vdash (P\ (f\ a)), (P\ (f\ (f\ a)))}\text{ axiome}}{(P\ a) \vdash (P\ (f\ a)), (P\ (f\ a)) \Rightarrow (P\ (f\ (f\ a)))}\Rightarrow\text{-droite}}{\vdash (P\ a) \Rightarrow (P\ (f\ a)), (P\ (f\ a)) \Rightarrow (P\ (f\ (f\ a)))}\Rightarrow\text{-droite}}{\vdash (P\ a) \Rightarrow (P\ (f\ a)), \exists x\ ((P\ x) \Rightarrow (P\ (f\ x)))}\exists\text{-droite}}{\vdash \exists x\ ((P\ x) \Rightarrow (P\ (f\ x))), \exists x\ ((P\ x) \Rightarrow (P\ (f\ x)))}\exists\text{-droite}}{\vdash \exists x\ ((P\ x) \Rightarrow (P\ (f\ x)))}\text{ contraction-droite}$$

Dans cette démonstration la proposition $\exists x\ ((P\ x) \Rightarrow (P\ (f\ x)))$ est dupliquée par la règle *contraction* et donc utilisée deux fois. On dit que ce séquent, qui n'est pas démontrable sans utiliser la règle de contraction, a la multiplicité 2.

Quand on a une démonstration d'un séquent quelconque, en permutant les règles, on peut la transformer de manière à n'utiliser la règle de contraction que sur des propositions de la forme $\forall x\ A$ à gauche ou $\exists x\ A$ à droite. Toujours en permutant les règles, on peut encore transformer cette démonstration de manière à ce que pour chaque proposition de la forme $\forall x\ A$ à gauche ou $\exists x\ A$ à droite, on commence par effectuer des contractions de manière à obtenir un certain nombre de copies de cette proposition puis qu'on n'applique plus les règles de contractions aux propositions produites. En ajoutant des propositions inutiles dans les séquents on peut transformer la démonstration de manière à ce que chacune des propositions de cette forme soit dupliquée en un nombre identique de copies.

Si de plus, on a une démonstration qui n'utilise ni la règle droite du quantificateur universel "quel que soit" ni la règle gauche du quantificateur existentiel "il existe" (comme c'est le cas dans le fragment existentiel), on peut permuter des règles qui s'appliquent à des propositions différentes et on peut donc appliquer toutes les règles gauches du quantificateur universel "quel que soit" ou toutes les règles droites du quantificateur existentiel "il existe" juste après avoir dupliqué la proposition.

On peut alors être tenté de dire qu'un séquent est de multiplicité m s'il a une démonstration où chaque proposition est dupliquée en m copies au plus. Hélas, cette définition laisserait croire que le séquent $(P\ a) \vee (P\ b) \vdash \exists x\ (P\ x)$ a la multiplicité 1 car il peut se démontrer ainsi

$$\dfrac{\dfrac{\dfrac{}{(P\ a) \vdash (P\ a)}\text{ axiome}}{(P\ a) \vdash \exists x\ (P\ x)}\exists\text{-droite} \qquad \dfrac{\dfrac{}{(P\ b) \vdash (P\ b)}\text{ axiome}}{(P\ b) \vdash \exists x\ (P\ x)}\exists\text{-droite}}{(P\ a) \vee (P\ b) \vdash \exists x\ (P\ x)}\vee\text{-gauche}$$

Mais cette démonstration ne demande pas de contraction de la proposition $\exists x\ (P\ x)$ uniquement parce qu'elle commence par la règle gauche de la disjonction "ou". Si on permute cette règle avec la règle droite du

quantificateur existentiel “il existe”, on introduit une contraction

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(P\ a) \vdash (P\ a), (P\ b)}\ \text{axiome} \qquad \dfrac{}{(P\ b) \vdash (P\ a), (P\ b)}\ \text{axiome}}{(P\ a) \vee (P\ b) \vdash (P\ a), (P\ b)}\ \vee\text{-gauche}}{(P\ a) \vee (P\ b) \vdash (P\ a), \exists x\ (P\ x)}\ \exists\text{-droite}}{(P\ a) \vee (P\ b) \vdash \exists x\ (P\ x), \exists x\ (P\ x)}\ \exists\text{-droite}}{(P\ a) \vee (P\ b) \vdash \exists x\ (P\ x)}\ \text{contraction}$$

La recherche d'une démonstration sans contraction pour ce séquent échoue si elle a la malchance de commencer par traiter la proposition $\exists x\ (P\ x)$.

Un exemple similaire dans le fragment existentiel est obtenu en considérant la proposition $\exists x\ \exists y\ (((P\ x) \vee (P\ (f\ x)) \vee (P\ (f\ (f\ x)))) \Rightarrow (P\ y)$. Cette proposition a une démonstration ou chaque proposition est dupliquée en deux copies au plus

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(P\ a) \vdash (P\ a), (A\ a) \Rightarrow (P\ a)}\ \dots}{(P\ a) \vdash (P\ a), \exists y\ ((A\ a) \Rightarrow (P\ y))} \quad \dfrac{\dfrac{}{(P\ (f\ a)) \vdash (P\ a), (A\ a) \Rightarrow (P\ (f\ a))}\ \dots}{(P\ (f\ a)) \vdash (P\ a), \exists y\ ((A\ a) \Rightarrow (P\ y))}}{(P\ a) \vee (P\ (f\ a)) \vdash (P\ a), \exists y\ ((A\ a) \Rightarrow (P\ y))} \quad \dfrac{\dfrac{\dfrac{}{(P\ (f\ (f\ a))) \vdash (P\ a), (A\ a) \Rightarrow (P\ (f\ (f\ a)))}\ \dots}{(P\ (f\ (f\ a))) \vdash (P\ a), \exists y\ ((A\ a) \Rightarrow (P\ y))}}{(P\ a) \vee (P\ (f\ a)) \vee (P\ (f\ (f\ a))) \vdash (P\ a), \exists y\ ((A\ a) \Rightarrow (P\ y))}}{\vdash (A\ a) \Rightarrow (P\ a), \exists y\ ((A\ a) \Rightarrow (P\ y))}\ \Rightarrow\text{-droite}}{\vdash \exists y\ ((A\ a) \Rightarrow (P\ y)), \exists y\ ((A\ a) \Rightarrow (P\ y))}\ \exists\text{-droite}}{\vdash \exists x\ \exists y\ ((A\ x) \Rightarrow (P\ y)), \exists x\ \exists y\ ((A\ x) \Rightarrow (P\ y))}\ \exists\text{-droite (2 fois)}}{\vdash \exists x\ \exists y\ ((A\ x) \Rightarrow (P\ y))}\ \text{contraction}$$

où $(A\ x)$ est une abréviation pour $(P\ x) \vee (P\ (f\ x)) \vee (P\ (f\ (f\ x)))$.

Mais pour appliquer la règle droite du quantificateur existentiel “il existe” avant les règles gauches de la disjonction “ou”, il faut introduire des contractions. Sinon le schéma de démonstration

$$\dfrac{\dfrac{\vdash (A\ X) \Rightarrow (P\ Y), (A\ X') \Rightarrow (P\ Y), (A\ X) \Rightarrow (P\ Y'), (A\ X') \Rightarrow (P\ Y')}{\vdash \exists y\ ((A\ X) \Rightarrow (P\ y)), \exists y\ ((A\ X') \Rightarrow (P\ y))}}{\vdash \exists x\ \exists y\ ((A\ x) \Rightarrow (P\ y))}$$

aboutit à un séquent non démontrable.

On dit donc qu'un séquent a la multiplicité m s'il ne contient que des propositions atomiques et qu'il est démontrable par la règle *axiome* ou s'il contient une proposition non atomique et que quelle que soit la proposition non atomique de ce séquent on obtient des séquents de multiplicité m en faisant m copies de la proposition et en substituant la variable x par m termes dans ces m copies si la proposition choisie a la forme $\exists x\ A$ à droite ou $\forall x\ A$ à gauche, et en appliquant la règle correspondant au symbole principal et à la latéralité de la proposition sinon.

On montre d'abord que si un séquent a la multiplicité m alors il a aussi la multiplicité m' pour tout $m' > m$ et que tout séquent obtenu en lui ajoutant des propositions à gauche ou à droite a aussi la multiplicité m. On montre ensuite que pour tout séquent démontrable (sans règle droite du quantificateur universel “quel que soit” ou règle gauche du quantificateur existentiel “il existe”), il existe un entier m tel que ce séquent soit de multiplicité m. Pour chaque proposition de la forme $\forall x\ A$ à gauche ou $\exists x\ A$ à droite, on peut construire, comme nous l'avons vu, une démonstration qui commence par dupliquer cette proposition n_i fois puis applique la règle logique correspondante à ces copies. Le séquent obtenu est démontrable, et il contient des propositions plus petites, il a donc une multiplicité m_i. Pour chaque proposition qui n'est pas de cette forme, on peut construire une démonstration qui commence par appliquer la règle logique correspondant au connecteur principal et à la latéralité de la proposition en question. Le séquent obtenu est démontrable, et il contient des propositions plus petites, il a donc une multiplicité m_i. On prend pour m le maximum des n_i, m_i, le séquent de départ a la multiplicité m.

Quand on recherche une démonstration de multiplicité m, quand on traite une proposition de la forme $\exists x\ A$ il ne faut pas se contenter de substituer x par une métavariable X, mais il faut introduire m copies de A

en substituant x par m métavariables $X_1, ..., X_m$. Par exemple si on cherche une démonstration de multiplicité 2 de la proposition $\exists x\ ((P\ x) \Rightarrow (P\ (f\ x)))$ on se ramène au séquent $\vdash (P\ X_1) \Rightarrow (P\ (f\ X_1)), (P\ X_2) \Rightarrow (P\ (f\ X_2))$ puis au séquent $(P\ X_1), (P\ X_2) \vdash (P\ (f\ X_1)), (P\ (f\ X_2))$. En unifiant $(P\ X_2)$ avec $(P\ (f\ X_1))$ on obtient la substitution $X_2 \mapsto (f\ X_1)$ qui permet de conclure.

Avec une multiplicité 2, une proposition de la forme $\exists x\ \exists y\ P(x, y)$ donne naissance à deux propositions $\exists y\ P(X_1, y)$ et $\exists y\ P(X_2, y)$ puis à quatre propositions $P(X_1, Y_{1,1})$, $P(X_1, Y_{1,2})$, $P(X_2, Y_{2,1})$ et $P(X_2, Y_{2,2})$. Les noms des métavariables sont produits à partir de ceux des variables liées en leur ajoutant une liste d'indices. Lors de la dérivation, on associe à chaque proposition un *label* qui est la liste par laquelle il faudra indicer la prochaine métavariable.

Les différences avec le programme du chapitre précédent sont l'ajout d'une fonction `copie` et les modification des fonctions `der` et `derivable` pour tenir compte de cette multiplicité.

```
let rec copies m lab str a =
 if m = 0 then []
 else let lab' = lab ^ " " ^ (string_of_int m)
      in (lab',subst_prop [(Var str,Meta(str ^ lab'))] a)::(copies (m-1) lab str a);;

let rec permuteM l = permuteM_rec [] l
and permuteM_rec l'  l = match l with
    [] -> l'
  | ((lab,(Atomic (str,arg)))::l'') -> permuteM_rec ((lab,(Atomic(str,arg)))::l') l''
  | (p::l'') -> p::(l'@l'');;

let rec derM mult gamma delta = match (permuteM gamma,permuteM delta) with
  ((lab,Neg a)::gamma',delta')        -> derM mult gamma' ((lab,a)::delta')
| ((lab,Imp(a,b))::gamma',delta')     ->
    (derM mult gamma' ((lab,a)::delta'))@(derM mult ((lab,b)::gamma') delta')
| ((lab,Conj(a,b))::gamma',delta')    -> derM mult ((lab,a)::(lab,b)::gamma') delta'
| ((lab,Disj(a,b))::gamma',delta')    ->
             (derM mult ((lab,a)::gamma') delta')@(derM mult ((lab,b)::gamma') delta')
| ((_,Exist _)::_,_)                  -> raise Error
| ((_,Univ _)::_,_)                   -> raise Error
| (gamma',(lab,Neg a)::delta')        -> derM mult ((lab,a)::gamma') delta'
| (gamma',(lab,Imp(a,b))::delta')     -> derM mult ((lab,a)::gamma') ((lab,b)::delta')
| (gamma',(lab,Conj(a,b))::delta')    ->
              (derM mult gamma' ((lab,a)::delta'))@(derM mult gamma' ((lab,b)::delta'))
| (gamma',(lab,Disj(a,b))::delta')    -> derM mult gamma' ((lab,a)::(lab,b)::delta')
| (gamma',(lab,Exist(str,a))::delta') ->
            derM mult gamma' ((copies mult lab str a)@delta')
| (_,(_,Univ _)::_)                   -> raise Error
| (gamma',delta')                     ->
             [(List.map (fun (x,y) -> y) gamma', List.map (fun (x,y) -> y) delta')];;

let derivableM mult p = connect (derM mult [] [("",p)]);;
```

Quand on cherche ainsi une démonstration de multiplicité m, il se peut qu'on trouve une démonstration de multiplicité supérieure si on est dans un cas favorable. Par exemple, ils se peut qu'en cherchant une démonstration de multiplicité 1 pour le séquent $(P\ a) \vee (P\ b) \vdash \exists x\ (P\ x)$ on en trouve une. Mais on sait que s'il y a une démonstration de multiplicité inférieure à m alors la recherche aboutira.

On peut alors construire un semi-algorithme de recherche de démonstration en cherchant d'abord une démonstration de multiplicité 1, puis une démonstration de multiplicité 2, de multiplicité 3, etc. Ce programme mène à des calculs infinis quand la proposition n'est pas démontrable.

```
let rec recherche p = recherche_rec 1 p
and recherche_rec n p = (derivableM n p) or recherche_rec (n+1) p;;
```

Exemple

On peut maintenant chercher une démonstration de la proposition $\exists x\ ((P\ x) \Rightarrow (P\ (f\ x)))$.

```
# derivableM 1
   (Exist("x",Imp (Atomic("P",[Var "x"]),Atomic("P",[Funct("f",[Var "x"])]))));;
- : bool = false

# derivableM 2
   (Exist("x",Imp (Atomic("P",[Var "x"]),Atomic("P",[Funct("f",[Var "x"])]))));;
- : bool = true

# recherche
   (Exist("x",Imp (Atomic("P",[Var "x"]),Atomic("P",[Funct("f",[Var "x"])]))));;
- : bool = true
```

Chapitre 7

Calcul des prédicats

Il y a deux approches possibles pour étendre la méthode du chapitre précédent à l'ensemble du calcul des prédicats. La première consiste à montrer que pour chaque proposition (et plus généralement, pour chaque séquent) du calcul des prédicats, on peut construire une proposition existentielle, sa *forme prénexe skolémisée* qui est démontrable si et seulement si la proposition initiale l'est. Par exemple, la forme prénexe skolémisée de la proposition

$$\exists x\ \forall y\ ((P\ x) \Rightarrow (P\ y))$$

est la proposition

$$\exists x\ ((P\ x) \Rightarrow (P\ (f\ x)))$$

Une autre méthode consiste à chercher directement une démonstration en calcul des séquents de la proposition.

7.1 La condition de fraîcheur des variables

Contrairement au chapitre précédent, il se peut maintenant qu'on ait à utiliser la règle droite du quantificateur universel "quel que soit" ou la règle gauche du quantificateur existentiel "il existe" dans les démonstrations.

$$\frac{\Gamma \vdash A, \Delta}{\Gamma \vdash \forall x\ A, \Delta}\ \forall\text{-droite} \quad \text{si } x \text{ n'apparaît pas dans } \Gamma\Delta$$

$$\frac{\Gamma, A \vdash \Delta}{\Gamma, \exists x\ A \vdash \Delta}\ \exists\text{-gauche} \quad \text{si } x \text{ n'apparaît pas dans } \Gamma\Delta$$

Cela introduit plusieurs complications.

7.1.1 Contraintes d'occurrence

Quand on applique une telle règle à un séquent qui comprend des métavariables, par exemple $\vdash \forall y\ ((P\ X) \Rightarrow (P\ y))$ il ne suffit pas de vérifier que la variable x n'apparaît pas dans le séquent, il faut s'assurer qu'elle n'apparaîtra pas dans les termes substitués à la métavariable X après l'unification. Cela demande de garder la *contrainte d'occurrence* $y \notin Var(\sigma X)$ et d'échouer quand la substitution calculée par l'unification viole cette condition.

7.1.2 Gestion de la multiplicité

Comme au chapitre précédent, quand on a une démonstration, en permutant les règles, on peut la transformer de manière à n'utiliser la règle de contraction que sur des propositions de la forme $\forall x\ A$ à gauche ou $\exists x\ A$ à droite.

On peut aussi transformer cette démonstration de manière à ce que pour chaque proposition de la forme $\forall x\ A$ à gauche ou $\exists x\ A$ à droite, on commence par effectuer des contractions puis on n'applique plus la règle de contraction aux propositions produites.

En revanche, on ne peut plus toujours permuter des règles qui s'appliquent à des propositions différentes et donc on ne peut plus appliquer m règles gauches du quantificateur universel "quel que soit" ou m règles droites du quantificateur existentiel "il existe" juste après avoir dupliqué la proposition. Comme nous l'avons vu, les démonstrations ainsi obtenues pourraient violer les conditions de fraîcheur des variables.

Par exemple, pour démontrer la proposition $\exists x\ \forall y\ ((P\ x) \Rightarrow (P\ y))$ il faut appliquer la règle de contraction de manière à se ramener au séquent $\vdash \exists x\ \forall y\ ((P\ x) \Rightarrow (P\ y)), \exists x\ \forall y\ ((P\ x) \Rightarrow (P\ y))$ puis appliquer dans l'ordre les règles droites des quantificateurs existentiel, universel, existentiel puis universel

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdash (P\ x') \Rightarrow (P\ y'), (P\ y') \Rightarrow (P\ z')}{\vdash (P\ x') \Rightarrow (P\ y'), \forall y\ ((P\ y') \Rightarrow (P\ y))}\ \forall\text{-droite}}{\vdash (P\ x') \Rightarrow (P\ y'), \exists x\ \forall y\ ((P\ x) \Rightarrow (P\ y))}\ \exists\text{-droite}}{\vdash \forall y\ ((P\ x') \Rightarrow (P\ y)), \exists x\ \forall y\ ((P\ x) \Rightarrow (P\ y))}\ \forall\text{-droite}}{\vdash \exists x\ \forall y\ ((P\ x) \Rightarrow (P\ y)), \exists x\ \forall y\ ((P\ x) \Rightarrow (P\ y))}\ \exists\text{-droite}}{\vdash \exists x\ \forall y\ ((P\ x) \Rightarrow (P\ y))}\ \text{contraction}$$

le séquent $\vdash (P\ x') \Rightarrow (P\ y'), (P\ y') \Rightarrow (P\ z')$ étant alors facilement démontrable. Mais il n'est pas possible de permuter les règles de manière à appliquer deux fois la règle du quantificateur existentiel "il existe" avant d'appliquer la règle du quantificateur universel "quel que soit". En effet, le séquent $\vdash \forall y\ ((P\ x') \Rightarrow (P\ y)), \forall y\ ((P\ y') \Rightarrow (P\ y))$ n'est pas démontrable : si on applique la règle du quantificateur universel "quel que soit" on ne peut pas utiliser la variable y' qui apparaît dans le séquent.

De ce fait, quand on a une proposition de la forme $\forall x\ A$ à gauche ou $\exists x\ A$ à droite, on ne peut pas traiter cette proposition simplement en la remplaçant par m copies de la proposition A où la variable x est substituée par m métavariables. Si en cherchant une démonstration de multiplicité 2 du séquent $\vdash \exists x\ \forall y\ ((P\ x) \Rightarrow (P\ y))$, on le remplaçait par $\vdash \forall y\ ((P\ X_1) \Rightarrow (P\ y)), \forall y\ ((P\ X_2) \Rightarrow (P\ y))$, l'application de la règle droite du quantificateur universel "quel que soit" engendrerait des contraintes d'occurrence que les substitutions calculées par l'algorithme d'unification ne vérifierait pas et la recherche échouerait.

Pour chercher des démonstrations de multiplicité m donnée, il faudrait ajouter une marque aux quantificateur en distinguant $\forall$ qui n'est pas encore dupliqué de $\forall'$ qui l'a déjà été et ne peut plus l'être.

Quand on a une proposition de la forme $\forall x\ A$ à gauche ou $\exists x\ A$ à droite, on la remplace par m propositions de la forme $\forall' x\ A$ ou $\exists' x\ A$. Ensuite quand on a une proposition de la forme $\forall' x\ A$ à gauche ou $\exists' x\ A$ à droite, on remplace cette proposition par A où x est substituée par une nouvelle métavariable.

7.1.3 Arborescence du choix de la proposition

Mais, comme nous l'avons vu, cette substitution ne peut pas toujours se faire tout de suite après la duplication. Le choix de la proposition à traiter devient donc arborescent. Par exemple, si on cherche une démonstration de multiplicité 2 du séquent $\vdash \exists x\ \forall y\ ((P\ x) \Rightarrow (P\ y))$, on le remplace par $\vdash \exists' x\ \forall y\ ((P\ x) \Rightarrow (P\ y)), \exists' x\ \forall y\ ((P\ x) \Rightarrow (P\ y))$, on remplace ensuite la première proposition par $\forall y\ ((P\ X_1) \Rightarrow (P\ y))$, ce qui donne $\vdash \forall y\ ((P\ X_1) \Rightarrow (P\ y)), \exists' x\ \forall y\ ((P\ x) \Rightarrow (P\ y))$, ici si on traite la seconde proposition on se ramène, comme ci-dessus, à un séquent non démontrable : il faut commencer par la première.

7.2 Graphes d'occurrence

Si on cherche toutes les démonstrations possibles de multiplicité 2 de la proposition $\exists x\ \forall y\ ((P\ x) \Rightarrow (P\ y))$, on commence par dupliquer la proposition, puis on applique les quatre règles des quantificateurs, en introduisant deux métavariables X_1 et X_2 et deux variables y_1 et y_2. Dans tous les cas on aboutit au séquent $(P\ X_1), (P\ X_2) \vdash (P\ y_1), (P\ y_2)$, mais les contraintes d'occurrence sont différentes. Si on opère dans l'ordre

X_1, X_2, y_1, y_2 on introduit les contraintes

$$y_1 \notin Var(\sigma X_1)$$

$$y_1 \notin Var(\sigma X_2)$$

$$y_2 \notin Var(\sigma X_1)$$

$$y_2 \notin Var(\sigma X_2)$$

Si on opère dans l'ordre X_1, y_1, X_2, y_2 on introduit seulement les contraintes

$$y_1 \notin Var(\sigma X_1)$$

$$y_2 \notin Var(\sigma X_1)$$

$$y_2 \notin Var(\sigma X_2)$$

L'ordre y_1, X_1, X_2, y_2 est quant à lui impossible car le quantificateur introduisant y_1 est dans le champ de celui introduisant X_1 donc X_1 doit être introduit avant.

À chacune de ces tentatives de démonstration, on peut associer une relation d'ordre strict sur l'ensemble des variables et métavariables $a < b$ si a est introduite dans la démonstration en dessous de b. (Les variables qui ne sont pas introduites par la règle droite du quantificateur universel "quel que soit" ou par la règle gauche du quantificateur existentiel "il existe" sont inférieures à toutes les autres.)

Cet ordre contient toujours *l'ordre des champs* $\{X_1 < y_1, X_2 < y_2\}$, puisque le quantificateur introduisant y_1 étant dans le champ de celui introduisant X_1, X_1 doit être introduite en dessous de y_1.

L'idée qui va nous permettre d'éviter de construire toutes ces tentatives de démonstrations qui ne diffèrent que par l'ordre d'application des règles consiste à décomposer la démonstration en trois informations :

- un schéma de démonstration ne se distingue d'une démonstration par le fait que
  - la règle gauche du quantificateur universel "quel que soit" et la règle droite du quantificateur existentiel "il existe" introduisent une métavariable et non un terme,
  - l'arbre peut contenir des feuilles qui ne sont pas démontrées par la règle *axiome*,
  - la règle droite du quantificateur universel "quel que soit" et la règle gauche du quantificateur existentiel "il existe" n'ont pas de condition de fraîcheur des variables,
  - les variables introduites par la règle droite du quantificateur universel "quel que soit" et la règle gauche du quantificateur existentiel "il existe", n'apparaissent pas au dessous de cette règle et que les variables introduites dans deux copies d'une même proposition introduites par une règle binaire (et donc dans deux branches différentes) portent le même nom.
- une substitution σ qui associe un terme à chaque métavariable du schéma de démonstration, et qui appliquée à chaque feuille du schéma de démonstration donne un séquent qui a une proposition commune à gauche et à droite,
- une relation d'ordre sur les variables et métavariables introduites par les règles des quantificateurs qui contient l'ordre des champs du schéma de démonstration (c'est-à-dire que si b est introduit par un quantificateur qui se situe dans le champ de celui qui introduit a alors $a < b$) et le *graphe d'occurrence* de la substitution (c'est-à-dire que si $y \in Var(\sigma X)$ alors $y < X$).

Le fait que les variables introduites par la règle droite du quantificateur universel "quel que soit" et la règle gauche du quantificateur existentiel "il existe" introduites dans deux copies d'une même proposition introduites par une règle binaire différentes portent le même nom est important pour pouvoir permutter les règles portant sur des propositions différentes. Par exemple dans le schéma

$$\dfrac{\dfrac{A \vdash P(x)}{A \vdash \forall x\ (P(x))}\,\forall\text{-droite} \quad \dfrac{B \vdash P(y)}{B \vdash \forall x\ (P(x))}\,\forall\text{-droite}}{A \vee B \vdash \forall x\ (P(x))}\,\vee\text{-gauche}$$

on ne peut pas permutter les deux règles. C'est en revanche possible dans le schéma

$$\dfrac{\dfrac{A \vdash P(x)}{A \vdash \forall x\ (P(x))}\,\forall\text{-droite} \quad \dfrac{B \vdash P(x)}{B \vdash \forall x\ (P(x))}\,\forall\text{-droite}}{A \vee B \vdash \forall x\ (P(x))}\,\vee\text{-gauche}$$

pour obtenir le schéma

$$\dfrac{\dfrac{A \vdash P(x) \quad B \vdash P(x)}{A \vee B \vdash P(x)}}{A \vee B \vdash \forall x\ (P(x))}\,\vee\text{-gauche}$$

Si un séquent a une démonstration, alors on peut construire un tel triplet : on commence par renommer les variables introduites par la règle droite du quantificateur universel "quel que soit" et la règle gauche du quantificateur existentiel "il existe" comme ci-dessus. Ensuite, on remplace chaque terme introduit par la règle gauche du quantificateur universel "quel que soit" ou la règle droite du quantificateur existentiel "il existe" par une métavariable, on construit la substitution qui associe à chaque métavariable le terme qu'elle remplace. Enfin la relation d'ordre est définie par : $a < b$ si a est introduit dans la démonstration en dessous de b. Cette relation d'ordre vérifie les deux conditions ci-dessus.

Réciproquement, si un séquent admet un tel triplet, alors on commence par renommer les variables introduites par la règle droite du quantificateur universel "quel que soit" et la règle gauche du quantificateur existentiel "il existe" comme ci-dessus. En permutant les règles du schéma de démonstration s'appliquant à des propositions différentes, on peut construire un schéma de démonstration tel que si $a < b$ alors a n'est pas introduit au dessus de b. Considerons un cas où $a < b$ et a se trouve au dessus de b dans le schéma de démonstration Comme l'ordre contient l'ordre des champs, a n'est pas dans le champ de b. Le symbole b n'est pas dans le champ de a car a serait alors en dessous de b dans le schéma de démonstration. Donc a et b appartiennent à des propositions différentes. Dans les schémas de propositions, comme la condition de fraîcheur des variables n'a plus à être vérifiée, et que si une règle binaire est suivie d'une règle droite du quantificateur universel "quel que soit" ou d'une la règle gauche du quantificateur existentiel "il existe", les variables introduites dans les deux copies de la proposition sont identiques, on peut toujours permuter deux règles qui s'appliquent à des propositions différentes. On peut permuter toutes les règles situées entre a et b concernant la proposition de a avec toutes les règles situées entre a et b et ne concernant pas cette proposition jusqu'à ce que a soit en dessous de b. Quand il n'y a plus de paire $a < b$ telle que a se trouve au dessus de b dans la démonstration, on sait que si $y \in Var(\sigma X)$ alors $y < X$ donc y ne se trouve pas au dessus de X, X ne se trouve pas en dessous de y.

En appliquant la substitution à ce schéma de démonstration, on obtient une démonstration. En effet, on montre que chacun des séquents sur le chemin qui va de la conclusion à la règle d'introduction de y ne contient pas d'occurrence de y, c'est en particulier vrai pour le séquent conclusion de l'introduction de la règle de y et donc la condition de fraîcheur des variables de cette règle est respectée.

Par exemple, la proposition $\exists x\ \forall y\ ((P\ x) \Rightarrow (P\ y))$ a le schéma de démonstration suivant

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(P\ X_1), (P\ X_2) \vdash (P\ y_1), (P\ y_2)}{(P\ X_1) \vdash (P\ y_1), (P\ X_2) \Rightarrow (P\ y_2)}\,\Rightarrow\text{-droite}}{\vdash (P\ X_1) \Rightarrow (P\ y_1), (P\ X_2) \Rightarrow (P\ y_2)}\,\Rightarrow\text{-droite}}{\vdash (P\ X_1) \Rightarrow (P\ y_1), \forall y\ ((P\ X_2) \Rightarrow (P\ y))}\,\forall\text{-droite}}{\vdash \forall y\ ((P\ X_1) \Rightarrow (P\ y)), \forall y\ ((P\ X_2) \Rightarrow (P\ y))}\,\forall\text{-droite}}{\vdash \forall y\ ((P\ X_1) \Rightarrow (P\ y)), \exists x\ \forall y\ ((P\ x) \Rightarrow (P\ y))}\,\exists\text{-droite}}{\vdash \exists x\ \forall y\ ((P\ x) \Rightarrow (P\ y)), \exists x\ \forall y\ ((P\ x) \Rightarrow (P\ y))}\,\exists\text{-droite}}{\vdash \exists x\ \forall y\ ((P\ x) \Rightarrow (P\ y))}\,\text{contraction}$$

avec la substitution $\{X_2 \mapsto y_1\}$ et l'ordre $X_1 < y_1 < X_2 < y_2$. On permute donc l'introduction de la

métavariable X_2 et de la variable y_1

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(P\ X_1), (P\ X_2) \vdash (P\ y_1), (P\ y_2)}{(P\ X_1) \vdash (P\ y_1), (P\ X_2) \Rightarrow (P\ y_2)}\Rightarrow\text{-droite}}{\vdash (P\ X_1) \Rightarrow (P\ y_1), (P\ X_2) \Rightarrow (P\ y_2)}\Rightarrow\text{-droite}}{\vdash (P\ X_1) \Rightarrow (P\ y_1), \forall y\ ((P\ X_2) \Rightarrow (P\ y))}\forall\text{-droite}}{\vdash ((P\ X_1) \Rightarrow (P\ y_1)), \exists x\ \forall y\ ((P\ x) \Rightarrow (P\ y))}\exists\text{-droite}}{\vdash \forall y\ ((P\ X_1) \Rightarrow (P\ y)), \exists x\ \forall y\ ((P\ x) \Rightarrow (P\ y))}\forall\text{-droite}}{\vdash \exists x\ \forall y\ ((P\ x) \Rightarrow (P\ y)), \exists x\ \forall y\ ((P\ x) \Rightarrow (P\ y))}\exists\text{-droite}}{\vdash \exists x\ \forall y\ ((P\ x) \Rightarrow (P\ y))}\text{contraction}$$

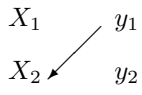
On peut à présent appliquer la substitution en respectant la condition de fraîcheur des variables (on substitue également la métavariable X_1 par une variable quelconque) et on obtient une démonstration

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(P\ x), (P\ y_1) \vdash (P\ y_1), (P\ y_2)}{(P\ x) \vdash (P\ y_1), (P\ y_1) \Rightarrow (P\ y_2)}\Rightarrow\text{-droite}}{\vdash (P\ x) \Rightarrow (P\ y_1), (P\ y_1) \Rightarrow (P\ y_2)}\Rightarrow\text{-droite}}{\vdash (P\ x) \Rightarrow (P\ y_1), \forall y\ ((P\ y_1) \Rightarrow (P\ y))}\forall\text{-droite}}{\vdash ((P\ x) \Rightarrow (P\ y_1)), \exists x\ \forall y\ ((P\ x) \Rightarrow (P\ y))}\exists\text{-droite}}{\vdash \forall y\ ((P\ x) \Rightarrow (P\ y)), \exists x\ \forall y\ ((P\ x) \Rightarrow (P\ y))}\forall\text{-droite}}{\vdash \exists x\ \forall y\ ((P\ x) \Rightarrow (P\ y)), \exists x\ \forall y\ ((P\ x) \Rightarrow (P\ y))}\exists\text{-droite}}{\vdash \exists x\ \forall y\ ((P\ x) \Rightarrow (P\ y))}\text{contraction}$$

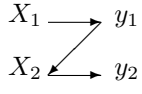
Quand on a un schéma de démonstration et une substitution, on peut calculer l'ordre des champs du schéma de démonstration, dans cet exemple

$$X_1 \longrightarrow y_1$$
$$X_2 \longrightarrow y_2$$

et le graphe d'occurrence de la substitution défini par $y < X$ si $y \in Var(\sigma X)$



Il existe une relation d'ordre contenant ces deux graphes si et seulement si la réunion de ces deux graphes est acyclique, dans cet exemple



Il n'est donc jamais nécessaire de calculer l'ordre, il suffit de vérifier que le graphe est acyclique.

7.3 Multiplicité

La notion de multiplicité se définit comme au chapitre précédent. Un schéma de démonstration a la multiplicité m si

— ses feuilles ne contiennent que des propositions atomiques et il existe une substitution σ et un ordre $<$ tels que en appliquant σ à chacune de ces feuilles on obtienne un séquent démontrable par la règle *axiome* et $<$ contient l'ordre des champs du schéma de démonstration et le graphe d'occurrence de σ.

— ou si l'une de ses feuilles contient une proposition non atomique et que quelle que soit la proposition non atomique d'une des feuilles, on obtient un schéma de démonstration de multiplicité m en faisant m copies de la proposition et en substituant la variable x par m métavariables dans ces m copies si la proposition choisie a la forme $\exists x\ A$ à droite ou $\forall x\ A$ à gauche, et en appliquant la règle correspondant au symbole principal et à la latéralité de la proposition sinon.

On montre d'abord que si un schéma de démonstration a la multiplicité m alors il a aussi la multiplicité m' pour tout $m' > m$ et que tout schéma de démonstration obtenu en lui ajoutant des propositions à gauche ou à droite dans l'une de ses feuilles a aussi la multiplicité m. On montre ensuite, en utilisant les propriétés de permutation des règles, que pour tout schéma de démonstration D qui peut se prolonger en un schéma de démonstration D' tel qu'il existe une substitution σ et un ordre $<$ tels que σ appliquée à chaque feuille de D' donne un séquent démontrable par la règle *axiome*, $<$ contient l'ordre des champs de D' et le graphe d'occurrence de σ, il existe un entier m tel que D soit de multiplicité m. On en déduit que pour tout séquent démontrable il existe un entier m tel que ce séquent soit de multiplicité m.

Lorsqu'on applique les règles logiques, on garde pour chaque proposition son *chemin*, c'est-à-dire la liste des variables et métavariables qui ont été introduites par un quantificateur dans le champ duquel la proposition en question apparaissait. À chaque fois qu'en traitant une proposition, on introduit une nouvelle variable ou métavariable, on introduit un arc dans le graphe des champs qui va de chaque élément du chemin de la proposition à cette variable.

En plus de la liste de séquents cette fonction retourne le graphe des champs du schéma de démonstration.

```
let rec copiesG m ch lab str a =
 if m = 0 then ([],[])
          else let lab' = lab ^ " " ^ (string_of_int m)
            in let str' = str ^ lab'
            in let a' = subst_prop [(Var str,Meta str')] a
            in let (g,l) = (copiesG (m-1) ch lab str a)
            in ((List.map (fun x -> (x,Meta str')) ch)@g, ((Meta str')::ch,lab',a')::l);;

let rec permuteG l = permuteG_rec [] l
and permuteG_rec l' l = match l with
    [] -> l'
  | ((ch,lab,Atomic (str,arg))::l'') ->
                                      permuteG_rec ((ch,lab,Atomic (str,arg))::l') l''
  | (p::l'') -> p::(l'@l'');;

let rec derG mult graphe gamma delta = match (permuteG gamma, permuteG delta) with
  ((ch,lab,Neg a)::gamma',delta')    -> derG mult graphe gamma' ((ch,lab,a)::delta')
| ((ch,lab,Imp(a,b))::gamma',delta') ->
   let (graphe1,feuil1) = derG mult graphe gamma' ((ch,lab,a)::delta')
   in let (graphe2,feuil2) = derG mult graphe1 ((ch,lab,b)::gamma') delta'
   in (graphe2, feuil1@feuil2)
| ((ch,lab,Conj(a,b))::gamma',delta') ->
                              derG mult graphe ((ch,lab,a)::(ch,lab,b)::gamma') delta'
| ((ch,lab,Disj(a,b))::gamma',delta') ->
   let (graphe1,feuil1) = derG mult graphe ((ch,lab,a)::gamma') delta'
   in let (graphe2,feuil2) = derG mult graphe1 ((ch,lab,b)::gamma') delta'
   in (graphe2,feuil1@feuil2)
| ((ch,lab,Exist(str,a))::gamma',delta') ->
   let str' = str ^ lab
   in let a' = subst_prop [(Var str,Var str')] a
   in let g = List.map (fun x -> (x,Var str')) ch
```

```
   in derG mult (g@graphe) (((Var str')::ch,lab,a')::gamma') delta'
| ((ch,lab,Univ(str,a))::gamma',delta') ->
   let (g,gamma1) = copiesG mult ch lab str a
   in derG mult (g@graphe) (gamma1@gamma') delta'
| (gamma',(ch,lab,Neg a)::delta')    -> derG mult graphe ((ch,lab,a)::gamma') delta'
| (gamma',(ch,lab,Imp(a,b))::delta') -> derG mult graphe ((ch,lab,a)::gamma') ((ch,lab,b)::delta')
| (gamma',(ch,lab,Conj(a,b))::delta') ->
   let (graphe1,feuil1) = derG mult graphe gamma' ((ch,lab,a)::delta')
   in let (graphe2,feuil2) = derG mult graphe1 gamma' ((ch,lab,b)::delta')
   in (graphe2,feuil1@feuil2)
| (gamma',(ch,lab,Disj(a,b))::delta') ->
                              derG mult graphe gamma' ((ch,lab,a)::(ch,lab,b)::delta')
| (gamma',(ch,lab,Exist(str,a))::delta') ->
   let (g,delta1) = copiesG mult ch lab str a
   in derG mult (g@graphe)  gamma' (delta1@delta')
| (gamma',(ch,lab,Univ(str,a))::delta') ->
   let str' = str ^ lab
   in let a' = subst_prop [(Var str,Var str')] a
   in let g = List.map (fun x -> (x,Var str')) ch
   in derG mult (g@graphe) gamma' (((Var str')::ch,lab,a')::delta')
| (gamma',delta') -> (graphe,
                      [(List.map (fun (x,y,z) -> z) gamma', List.map (fun (x,y,z) -> z) delta')]);;
```

Maintenant, on cherche une substitution σ qui unifie les séquents de la liste et telle que la réunion du graphe des champs et du graphe d'occurrence soit acyclique. S'il y a une telle substitution alors la solution principale du problème d'unification vérifie cette propriété.

On construit donc une fonction qui indique si un graphe est acyclique ou non.

```
let rec mem a l = match l with
   [] -> false
 | (b::l') -> a = b or mem a l';;

let rec fils a l = match l with
    [] -> []
  | ((x,y)::l') -> if x = a then y::(fils a l') else fils a l';;

let rec descendants a vus graphe = enumere vus graphe (fils a graphe)
and enumere vus graphe l = match l with
    [] -> vus
  | (x::l') -> let vus' = if mem x vus then vus else descendants x (x::vus) graphe
               in enumere vus' graphe l';;

let ajoute a l = if mem a l then l else a::l;;

let rec noeuds l = match l with
    [] -> []
  | ((a,b)::g) -> ajoute b (ajoute a (noeuds g));;

let rec acyclique graphe = acy graphe (noeuds graphe)
and acy graphe l = match l with
    [] -> true
  | (a::l) -> (not (mem a (descendants a [] graphe))) & (acy graphe l);;
```

Ensuite à chaque unification, on calcule le graphe d'occurrence lié à la solution, et on vérifie que le graphe obtenu en réunissant le graphe des champs et le graphe d'occurrence de la substitution est acyclique. Sinon on échoue.

```
let rec flat l = match l with
   [] -> []
 | (a::l) -> a@(flat l);;

let rec vars t = match t with
    (Var y) -> [(Var y)]
  | (Meta y) -> []
  | (Funct (_,l)) -> flat (List.map vars l);;

let rec graphe_sub l = match l with
    [] -> []
  | ((x,t)::l) -> (List.map (fun u -> (u,x)) (vars t))@(graphe_sub l);;

let rec connectG sub graphe l = match l with
    [] -> acyclique (graphe@(graphe_sub sub))
  | ((gamma,delta)::l') -> connectG' sub graphe gamma delta l'
and connectG' sub graphe gamma delta l = match gamma with
    [] -> false
  | (a::gamma') -> (connectG'' sub graphe a delta l) or (connectG' sub graphe gamma' delta l)
and connectG'' sub graphe a delta l = match delta with
    [] -> false
  | (b::delta') ->
        (try let s = unif_atomic_prop a b
          in let sub' = (List.map (fun (x,y) -> (x,subst s y)) sub)@s
          in connectG sub' graphe (List.map (subst_seq s) l)
         with Unif -> false)
        or  (connectG'' sub graphe a delta' l);;
```

Enfin on conclut.

```
let derivableG mult p = let (graphe,c) = derG mult [] [] [([],"",p)]
                        in connectG [] graphe c;;

let rec rechercheG p = rechercheG_rec 1 p
and rechercheG_rec n p = (derivableG n p) or rechercheG_rec (n+1) p;;
```

Exemples

$\exists x\ \forall y\ ((P\ x) \Rightarrow (P\ y))$

```
# derivableG 1
  (Exist ("x",Imp (Atomic("P",[Var "x"]),Univ ("y",Atomic("P",[Var "y"])))));;
- : bool = false

# derivableG 2
  (Exist ("x",Imp (Atomic("P",[Var "x"]),Univ ("y",Atomic("P",[Var "y"])))));;
- : bool = true
```

```
# rechercheG
  (Exist ("x",Imp (Atomic("P",[Var "x"]),Univ ("y",Atomic("P",[Var "y"])))));;
- : bool = true
```

$$(\forall x\ ((P\ x) \Rightarrow (P\ (f\ x))) \wedge (P\ a)) \Rightarrow (P\ (f\ a))$$

```
# derivableG 1
(Imp
  (Conj
    (Univ ("x",Imp (Atomic("P",[Var "x"]),Atomic("P",[Funct("f",[Var "x"])]))),
    (Atomic("P",[Funct ("a",[])]))),
  Atomic("P",[Funct ("f",[Funct ("a",[])])])));;
- : bool = true
```

$$(\forall x\ ((P\ x) \Rightarrow (P\ (f\ x))) \wedge (P\ a)) \Rightarrow (P\ (f\ (f\ a)))$$

```
# derivableG 1
(Imp
  (Conj
    (Univ ("x",Imp (Atomic("P",[Var "x"]),Atomic("P",[Funct("f",[Var "x"])]))),
    (Atomic("P",[Funct ("a",[])]))),
  Atomic("P",[Funct ("f",[Funct ("f",[Funct ("a",[])])])])));;
- : bool = false

# derivableG 2
(Imp
  (Conj
    (Univ ("x",Imp (Atomic("P",[Var "x"]),Atomic("P",[Funct("f",[Var "x"])]))),
    (Atomic("P",[Funct ("a",[])]))),
  Atomic("P",[Funct ("f",[Funct ("f",[Funct ("a",[])])])])));;
- : bool = true
```

$$\forall x\ (P\ x) \Rightarrow \exists y\ (P\ y)$$

```
# derivableG 1
  (Imp (Univ ("x",Atomic ("P",[Var "x"])),Exist ("y",Atomic("P",[Var "y"]))));;
- : bool = true
```

$$\exists x\ (P\ x) \Rightarrow \forall y\ (P\ y)$$

```
# derivableG 1
 (Exist ("x",Imp (Atomic("P",[Var "x"]),Univ ("y",Atomic ("P",[Var "y"])))));;
- : bool = false
```

$$\exists x\ \forall y\ (P\ x\ y) \Rightarrow \forall v\ \exists u\ (P\ u\ v)$$

```
# derivableG 1
  (Imp (Exist("x", Univ ("y",Atomic ("P",[Var "x"; Var "y"]))),
        Univ("v", Exist ("u",Atomic("P",[Var "u"; Var "v"])))));;
- : bool = true
```

$$\forall x\ \exists y\ (P\ x\ y) \Rightarrow \exists u\ \forall v\ (P\ u\ v)$$

```
# derivableG 1
  (Imp (Univ("x", Exist ("y",Atomic("P",[Var "x"; Var "y"]))),
        Exist("v", Univ ("u",Atomic("P",[Var "u"; Var "v"])))));;
- : bool = false
```

$$(\forall x\ ((A\ x) \vee (B\ x))) \Rightarrow (\forall x\ ((B\ x) \vee (A\ x)))$$

```
# derivableG 1
  (Imp(Univ("x",Disj(Atomic("A",[Var"x"]),Atomic("B",[Var"x"]))),
       Univ("x",Disj(Atomic("B",[Var"x"]),Atomic("A",[Var"x"])))));;
- : bool = true
```

$$\neg((\forall x\ \exists y\ \forall z\ ((P\ x\ y\ z) \Rightarrow (Q\ x\ y\ z))) \wedge (\exists x\ \forall y\ \exists z\ ((P\ x\ y\ z) \wedge \neg(Q\ x\ y\ z))))$$

```
# derivableG 1
  (Neg(Conj(Univ("x",Exist("y",Univ("z",
             Imp(Atomic("P",[Var"x"; Var"y"; Var"z"]),
                 Atomic("Q",[Var"x"; Var"y"; Var"z"])))),
            Exist("x",Univ("y",Exist("z",
             Conj(Atomic("P",[Var"x"; Var"y"; Var"z"]),
                  Neg(Atomic("Q",[Var"x"; Var"y"; Var"z"])))))))));;
- : bool = true
```

7.4 Recherche de termes

Si on cherche une démonstration d'une proposition contenant des métavariables, ces métavariables peuvent être instanciées au cours de la recherche. On cherche donc, en fait, une démonstration d'une instance de la proposition. On peut modifier le programme précédent de manière à afficher les termes substitués aux métavariables lors de la démonstration.

```
let rec connectT sub graphe l = match l with
    [] -> if acyclique (graphe@(graphe_sub sub)) then sub else raise Not_found
  | ((gamma,delta)::l') -> connectT' sub graphe gamma delta l'
and connectT' sub graphe gamma delta l = match gamma with
    [] -> raise Not_found
  | (a::gamma') -> (try  connectT'' sub graphe a delta l
                    with Not_found -> connectT' sub graphe gamma' delta l)
and connectT'' sub graphe a delta l = match delta with
    [] -> raise Not_found
  | (b::delta') ->
        (try (try let s = unif_atomic_prop a b
          in let sub' = (List.map (fun (x,y) -> (x,subst s y)) sub)@s
             in connectT sub' graphe (List.map (subst_seq s) l)
        with Unif -> raise Not_found)
        with Not_found ->  (connectT'' sub graphe a delta' l));;

let derivableT mult p = let (graphe,c) = derG mult [] [] [([],"",p)]
                        in connectT [] graphe c;;

let rec meta t = match t with
    (Var _) -> []
  | (Meta y) -> [y]
```

```
  | (Funct (_,l)) -> flat (List.map meta l);;

let rec meta_prop p = match p with
  (Atomic (_,l)) -> flat (List.map meta l)
| (Neg a) -> (meta_prop a)
| (Imp (a,b)) -> (meta_prop a)@(meta_prop b)
| (Conj (a,b)) -> (meta_prop a)@(meta_prop b)
| (Disj (a,b)) -> (meta_prop a)@(meta_prop b)
| (Exist (str,b)) -> (meta_prop b)
| (Univ (str,b)) -> (meta_prop b);;

let rec rechercheT p = rechercheT_rec 1 p
and rechercheT_rec n p = try let s = derivableT n p
                         in List.map (fun x -> (x, subst s (Meta x))) (meta_prop p)
                         with Not_found -> rechercheT_rec (n+1) p;;
```

On peut alors chercher qui est le grand-père de Charlemagne. On commence par définir les propositions

$$\forall x\ \forall y\ \forall z\ ((\text{père}(x,y) \wedge \text{père}(y,z)) \Rightarrow \text{grand père}(x,z))$$

$$\text{père}(\text{Charles Martel}, \text{Pépin le bref})$$

$$\text{père}(\text{Pépin le bref}, \text{Charlemagne})$$

```
let p = Univ ("x",Univ ("y",Univ ("z",
          Imp(Conj(Atomic("pere",[Var "x"; Var "y"]),
                   Atomic("pere",[Var "y"; Var "z"])),
              Atomic("grand pere",[Var "x"; Var "z"])))));;

let q =
      Atomic("pere",[Funct ("Charles Martel",[]); Funct ("Pepin le bref",[])]);;

let r = Atomic("pere",[Funct ("Pepin le bref",[]); Funct ("Charlemagne",[])]);;
```

Puis on recherche une démonstration de la proposition

$$\text{grand père}(X, \text{Charlemagne})$$

```
let s = Atomic("grand pere",[Meta "X"; Funct ("Charlemagne",[])]);;

# rechercheT (Imp(Conj(Conj (p,q),r),s));;
- : (string * term) list = ["X", Funct ("Charles Martel", [])]
```

On peut aussi chercher le double de 2. On commence par définir les propositions

$$\forall x\ \forall y\ (\text{double}(x,y) \Rightarrow \text{double}(S(x), S(S(y))))$$

$$\text{double}(0,0)$$

```
let p = Univ ("x", Univ("y",
   Imp(Atomic("double", [Var "x"; Var "y"]),
       Atomic("double", [Funct ("S", [Var "x"]);
                         Funct("S",[Funct("S",[Var "y"])])]))));;

let q = Univ ("x", Atomic("double", [Funct("0",[]); Funct("0",[])]));;
```

Puis on cherche une démonstration de la proposition

$$\text{double}(S(S(0)), X)$$

```
let r =
   Atomic("double",[Funct("S", [Funct ("S", [Funct("0",[])])]); Meta "X"]);;

# rechercheT (Imp(Conj(p,q),r));;
- : (string * term) list =
["X",
 Funct ("S", [Funct ("S", [Funct ("S", [Funct ("S", [Funct ("0", [])])])])])]
```

Un des avantages de la programmation déclarative est sa réversibilité, ainsi on peut calculer la moitié de 4 avec le même programme en cherchant une démonstration de la proposition

$$\text{double}(X, S(S(S(S(0)))))$$

```
let s =
   Atomic("double",[Meta "X"; Funct ("S", [Funct("S",
                              [Funct("S", [Funct ("S", [Funct("0",[])])])])])]);;

# rechercheT (Imp(Conj(p,q),s));;
- : (string * term) list =
["X", Funct ("S", [Funct ("S", [Funct ("0", [])])])]
```

Bibliographie

[1] R. Cori, D. Lascar, Logique mathématique, *Masson* (1993).

[2] J. Gallier, Logic for computer science : Foundations of automated theorem proving, *Harper and Row* (1986).

[3] J.Y. Girard, Y. Lafont, P. Taylor, Proofs and types, *Cambridge University Press* (1989).

[4] L. A. Wallen, Automated deduction in nonclassical logics, *MIT Press* (1990).