# Expérience(s) du temps en informatique

Liesbeth de Mol

**Expérience(s) du temps en informatique**[1]

**Liesbeth De Mol**

Qu'est ce qu'est le temps en informatique? C'est une question qui m'a accompagné depuis quelques années. Elle est devenu une question pertinente pour moi depuis que j'ai recu un des commentaires sur ma thèse doctorale. Dans cette thèse en philosophie j'ai fait, parmi autres choses, une étude des systèmes de `tag''. Ce sont des systèmes formels dévéloppés par Emil Post et montrés equivalent à des machines de Turing par Minsky dans les années 60. J'étais interessée dans la relation entre le *comportement* imprévisible des petits systèmes particulier de tag et des questions de décidabilité et universalité. Une petite expérience que j'ai fait pendant cette étude était l'exploration du comportement d'une petite machine de Turing universelle construite par Minsky, avec des mots initiales arbitraires et non pas codés. Le résultat n'était pas très surprinant: quand on commence ce système avec tels mots, le comportement est prévisible car ces systèmes exige des mots codées pour atteindre l'universalité. Sur cette expérience, Maurice Margenstern, un des membres du jury a commenté:

"The tests which you performed on running small Turing machines on arbitrary finite configurations are very interesting. They do not surprise me and, for me, they indicate the threshold between mathematics and computer science. As you remark, your experiment indicate something which is mathematically uninteresting: of course! Mathematics and theoretical computer science do not share exactly the
same point of view on things. Computer Sciences look at objects as processes, as something which lives in time and which is oriented towards a goal. Mathematics is much more abstract."

C'étais à ce moment là que je m'ai réalisé, comme philosophe, la significance fondamental du temps en informatique, avec son intérêt dans des processus même du calcul. C'étais quelques chose qui étais toujours là dans mes recherches mais sans que je l'avais thématisé. La notion, je réalisais, me permet pas seulement de connecter des différents thèmes de mes recherches mais aussi de penser la nature du calcul du point de vue du temps.

[slide] Mais qu'est-ce qu'est le temps en informatique? C'est clair qu'on trouve du temps partout dans l'informatique. Je donne quelques exemple: sur le niveau du hardware on a besoin du temps pour le fonctionnement du mémoire. On peut se

1**Ce texte est préparé pour le colloque “Le calcul et le temps” organisé par J.-B. Joinet, 6-7 Novembre 2014, Université Jean Moulin, Lyon.**

souvenir quelques chose seulement quand on présuppose un processus qui évolue dans le temps. Quand il n'y a pas un avant il n'y a pas de mémoire. On trouve aussi du temps sur le niveau de la programmation p.e. par l'affectation et par des opérations de synchronisation et on trouve du temps dans l'informatique théorique et ces études des ressources d'espace et du temps des algorithmes. Dans ces exemples, c'est clair qu'une notion de temps est ancrée dans ce que Margenstern a remarqué: dans la mesure où l'informatique étude des processus, le temps y joue un rôle éminent. Mais ces processus même sont, dans l'informatique comme pratique, le résultat des interactions: entre des humains et des machines, entre des différentes couches de programmes, etc. Ainsi, les processus du calcul reflètent eux-mêmes les temps subjectifs différents des machines, des humains, etc. Inversement, les processus de calcul résultent dans différents type d'expérience de temps. Par conséquent, on est confronté avec une interaction complexe de différents temps. Comment est-ce que ces différents temps se manifestent dans la pratique de l'informatique? Quels types de processus en résultent? Est-ce qu'on peut penser une notion de calcul sans y intégrer aussi des temps subjectifs (de l'humain, du medium physique qui execute les processus, etc)? [slide] Ce sont des questions que je partage avec Maarten Bullynck et que je voudrais développer ici d'un point de vue des pratiques de calcul, qui sont ancrées dans trois exemples d'experiences du temps. Maarten les développe demain d'un point de vue de deux cas d'étude très concrets venant des pratiques d'informatiques des années 60. Au fond, nous supposons que des catégories fondamentales, comme celle du temps, sont là seulement dans la mesure qu'elles sont là. C'est à dire, elles ne sont pas presupposées, mais se développent en tant qu'elles se présentent dans des actions quotidiennes.

Qu'est qu'est le temps en l'informatique? Comme j'ai déjà expliqué, je pense le temps dans des processus de calcul et les sujets impliquées dans ces processus avec leurs différents temps subjectifs. Pour clarifier la différence entre ces temps et la significance pour l'informatique de ces différences, je voudrais d'abord visiter une expérience particulière du temps dans des processus de calcul sans machines, celle d'Emil Post.

[slide] En 1920 Emil Post a fini sa thèse intitulé "Introduction to a general Theory of Elementary propositions", publié en 1921. Dans ce texte Post a montré pas seulement la complétude de la logique propositionelle, mais aussi sa décidabilité et sa consistence. Il y développe aussi l'idée des logiques à valeurs multiples. Plus important ici c'est la méthode générale que Post y annonce pour généraliser ces résultats "to the complete system of `Principia' [de Russel et Whitehead] and so ultimately of mathematics." [slide] Pour cette méthode, Post fait référence au chapitre

6 de "Survey of Symbolic Logic" de Clarence I. Lewis dans lequel Lewis propose la vue suivante sur les mathematiques:

"A mathematical system is any set of strings of recognizable marks in which some of the strings are taken initially and the remainder derived from these operations performed according to rules which are independent of any meaning assigned to the marks. [The] distinctive feature of this definition lies in the fact that it regards mathematics as dealing, not with certain denoted things -- numbers, triangles, etc -- nor with certain symbolized ``concepts'' or ``meanings'', but solely with recognizable marks, and dealing with them in such a wise that it is wholly independent of any question as to what the marks represent. This might be called the ``external view of mathematics'' or ``mathematics without meaning''. [W]hatever the mathematician \textit{has in his mind} when he develops a system, what he \textit{does} is to set down certain marks and proceed to manipulate them […]"

C'etait exactement ce point de vue que Post voudrait appliquer pour attaquer des problèmes qui concerne *toutes* les assertions des mathematiques, plus particulièrement le problème de complétitude et le problème de décidabilité. Ce dernier il appèle le problème de finitude, ou "finiteness problem" en anglais. C'est à dire, il voudrait dévélopper des formes de la logique dirigé par:

"the method of combinatory iteration [which] completely neglects meaning, and considers the entire system purely from the symbolic standpoint as one in which both the enunciations and assertions are groups of symbols or symbol-complexes […] and where these symbol assertions are obtained by starting with certain initial assertions and repeatedly applying certain rules for obtaining new symbol-assertions from old. [slide]

Dans sa thèse un résultat de cette méthode etait la soi-disante forme canonique A résultant d'une généralisation d'une formulation axiomatique de la logique propositionelle. Post la propose comme forme générale pour étudier des systèmes de la logique comme systèmes d'inferences obtenu par des manipulations symboliques finis. En effet, ce qu'il voudrait en fait c'est de développer une forme générale pour saisir les processus de la logique symbolique.

Après sa thèse Post commencait un PostDoc à Princeton pour un an. Son but étais très ambitieux: il voudrait montrer la décidabilité de la logique des prédicats et, à la fin, des mathématiques en générales. Convainque que des formes de la logique auprès le modèle de Lewis "allow for a greater freedom of method and technique" et, ainsi, rendre plus facile des problèmes comme celui de la décidabilité, son plan

d'attaque initiale étais de (1) montrer la logique des prédicats réducible à la forme A et (2) de montrer la décidabilité de la forme A. Il succédait a réduire la logique des prédicats à la forme A par l'introduction d'une forme intermédiare (la forme canonique B) mais, n'était pas capable de montrer la décidabilité de la forme A. Ainsi, pour attaquer ce problème, il dévéloppait des nouvelles formes qui, dans l'esprit de Lewis, sont encore plus dénué de sens que la forme A. Ces formes sont maintenant connues comme des systèmes de tag [slide]

Qu'est ce qu'est un système de tag? Je l'explique avec un exemple donné par Post. On commence avec un nombre entier $v$ qu'on appèle le nombre d'effacement, par exemple 3, un alphabet finis, par exemple 0 et 1, un ensemble de mots sur l'alphabet, chaque mot correspondant avec un des lettre de l'alphabet, et un mot intial $A_{0}$ sur l'alphabet. Dépendant du premier lettre d'$A_{0}$, on ajoute le mot correspondant avec ce lettre et supprime les 3 premier lettres, résultant dans un nouveau mot $A_{1}$. On ajoute le mot correspondant avec son premier lettre et supprime les 3 premier lettres, etc.

C'est clair que ses systèmes sont dans l'esprit de la methode de Lewis: hormis la production des chaînes finis de caractère sans sens il n'y a apparement rien d'autre. Le résultat est une forme extrèmement simple et ce n'est pas surprenant que Post identifiait ces systèmes comme "the most primitive form of mathematics". Mais, étant donné la nature de ces systèmes de productions, comment est-ce qu'on peut les étudier? La réponse est évidente: on étude les processus des productions même et c'est exactement ca que Post a fait, commencant avec les systèmes les plus simple avec un nombre d'effacement de 1 et 1 symbole. [slide] Pendant 9 mois, Post étudiais le comportement de ces systèmes. Le succès majeure auprès Post étais la démonstration que des systèmes de tag avec 2 symboles et un nombre d'effacement de deux sont décidable. En ajoutant un peu plus de complexité: avec un lettre ou par augmenter le nombre d'effacement avec 1, Post découvrait un bazar de systèmes de tag d'une "bewildering complexity". Il étudiait des différents cas, comme l'exemple que j'ai donné, explorait le comportement des productions produit par des mots initiales différents mais était incapable de "controler" les systèmes de tag. Il était incapable de trouver une méthode mathématique pour prévoir le comportement de ces systèmes, et, concludait après 9 mois que: "*the solution of the general problem of ``tag'' appeared hopeless, and with it our entire program of the solution of finiteness problems.*" Ainsi, Post n'était plus convaincu que tous les problèmes des mathématiques se laisse réduire à une forme décidable.  Au contraire, après ses expériences avec les systèmes de tag [slide] il dévéloppait une nouvelle forme similaire aux systèmes de tag -- la forme normale -- et l'utilisait dans la formulation d'une thèse qui, comme nous savons aujourd'hui, est équivalente à celle de Curch et

Turing. La thèse énonce que tous les /processus/ pour générer des sequences finis se laisse saisir pour la forme normale. Sur le base de cette thèse Post montrait l'indécidabilité ou l'incalculabilité des systèmes normales, et, en supposant sa thèse, concluait que cette une indécidabilité absolue: il n'existe aucune méthode fini qui nous permettra de rendre des systemes normales décidable.

Ainsi, commencant avec l'idée qu'il serait possible de montrer la décidabilité des mathématiques, Post finissait avec l'idée que c'est en fait impossible, utilisant la notion d'un ensemble généré ("generated set" en anglais) par des processus finis. Cette conclusion est le résultat de la méthode de Post qui avait pour but, contrairement à Church et Turing, de saisir dans une forme la logique symbolique compris dans le sens de Lewis. C'est à dire, comme la méthode d'itérations combinatoires qui néglige totalement la signification. [slide] Une conséquence primordiale de ce point de vue est que l'attention se déplace à des processus qui produisent des séquences de caractère qui se développe en temps:

"Where in these formulations the informal basic idea is that of effective calculability, our own is that of a {\it generated set}. This derives from the idea of a symbolic logic rather than that of an algorithm, and may be described by saying that each member of the set is at some time generated [Produced, created -- in practice, written down.] by the continued application of a given method"

Ainsi, plutôt de développer une théorie de l'algorithme, le but de Post, comme il l'explique 15 ans après dans un lettre à Church, était de développer une théorie des processus finis. Cette attention sur des processus de la logique n'est pas seulement une conséquence de la méthode de Post, mais est ancrées dans un point de vue philosophique qui comprend la logique et les mathématiques comme étant essentiellement des activités humains. C'était pour cette raison que la logique est compris comme une activité qui se développe en espace et temps. Et, c'était aussi pour cette raison que Post ajoute l'adjective "finis", comme il l'explique:

"Where we say "symbolic logic" the tendency now is to say "finitary symbolic logic." However, it seems to the writer that logic should be considered essentially a human enterprise, and that when this is departed from, it is *then* incumbent on such a writer to add a qualifying "non-finitary."

Ainsi, les limitations de l'activité logique n'est pas une limitation des processus finis de la logique symbolique mais une limitation de l'humanité même qui les produit:

"if symbolic logic has failed to give wings to mathematicians this study of symbolic

logic opens up a new field concerned with the fundamental limitations of mathematics, more precisely the mathematics of Homo Sapiens."

Cet insistance sur la dimension humaine peut au moins être partiellement expliqué par les expériences particulières de Post avec les formes et leurs processus qu'il a étudié et plus particulierement la forme de "tag" qui résultait dans l'inversion de son programme pour montrer la décidabilité des mathématiques. Comme j'ai expliqué, pendant qu'il étudiait cette forme il devenait de plus en plus frustrer. En fait, il a communiqué à Martin Davis que plus jamais il voudrait travailler sur "tag". Dans sa correspondence avec Church il le décrit comme suit: "[The] problem [of tag] is a special case of my unsolveable problem […] and should it too prove unsolveable I will be supplied with the perfect alibi\st{e} for a year of frustration." En plus, il avoue dans la correspondence que c'était en travaillant sur ces systèmes de "tag" qu'il a experiencé son premier attaque maniaco-dépressif, la maladie qui lui poursuivera pour le reste de sa vie et résultera ultérieurement dans sa mort à cause du traitement électrochoc:
"So I told [my wife], really for the first time, the exact history of my mental ups and downs and worse from its first inception in trying to solve the probably unsolvable tag-problem in Princeton and how at 50 experience and lessen[[e]]d personal importance of failure or success with at best 70-50 $<$ 20 years to go -- but I see my 50 years of experience may still not be enough -- God help me"

[slide] Ainsi, la confrontation même avec les processus des systèmes de tag et l'incapacité de Post de les controler rendre plus compréhensible le point de vue philosophique de Post sur des problèmes indécidables -- le fait que l'absoluté de la nonrésolubilité est, pour Post, défini relatif aux humains est peut-être ancrés dans l'expérience de Post avec ces systèmes de "tag". Cette expérience resultait d'un conflit entre le temps subjectif du logicien qui ne peut que suivre des productions de "tag" dans un temps déterminé par la vitesse de la machinerie qui effecterait l'écriture de ces productions et ce qui est en dehors le temps fini -- la réponse à la question si un système de "tag" particulier produisera un mot vide ou pas. Ces systèmes de "tag" devenaient une frustration parce que du point de vue humain ils prennent et requièrent du temps pour développer leurs comportements et qu'on ne peut pas dépasser au delà d'un certain horizon de temps. Peut-être qu'on peut accélérer le processus mais on ne peut pas le sauter. Alors, la seule manière pour Post pour prendre connaissance d'un système particulier etait de tracer le processus même du système dans son temps subjectif.

Même si on trouve déjà des traces dans l'oeuvre de Post avant 1921 d'un point de vue qui comprend les mathématiques et la logique comme un processus, après

1921 Post a presque une obsession avec le temps et des processus. P.e., dans le texte décrivant ses travaux de 1921, la notion de processus est utilisé plus de 60 fois. Et comme a remarquée Grattan-Guinness, dans ces carnets, ils notent souvent scrupuleusement le temps, p.e.:
``Post shows a remarkable degree of concern with the passage of time -- and not only in the logical texts. For example, on one of the manuscripts on Laplace transform theory [...] Post noted that he was writing the paper on 19 september 1923, starting at 10:08 a.m. and ending at 12.28 p.m. and he gave three running times within the text''

Ainsi, l'expérience du temps de point de vue de Post étais double: primo, le developpement des formes qui permet de saisir des processus finis de la logique symbolique résultant dans une thèse equivalente aux celles de Turing et Church et ainsi dans une "définition" du calcul qui accentuait son caractère processuel; deuxièmmement, l'expérience même de l'incapacité d'aller au-delà de ces processus. C'étais ce premier côté, l'idée de saisir des processus finis de la logique dans une forme, qui se transpose dans les années 1940 et 50 dans une nouvelle situation au moment que l'ordinateur électronique et programmable etait introduit. [slide]

En 1946 un des premier ordinateur électronique et programmable, l'ENIAC, est présenté au public. Cette machine était utilisée pour une variété de calculs par une variété de chercheurs: des logiciens, des physiciens et des mathématiciens. Ce que rendrait la machine si attractive et spéciale pour cette variété de gens était la combinaison de programmabilité -- qui implique l'utilisation de la machine pour plusieurs applications -- avec une vitesse électronique.

Avant l'invention des machines comme l'ENIAC, les calculs -- essentiellement des calculs de tables numériques -- étaient le résultat des calculs humains assistés par des calculatrices de bureau ou des machines analogues comme l'analyseur differentiels. Le problème avec ces méthodes n'étaient pas seulement qu'ils étaient susceptible de résulter dans des erreurs mais aussi qu'ils étaient trop lent. C'était surtout le cas pour des tables d'artillerie fin des années 30, début des années 40 -- étant donnée la situation politique -- et c'etait dans telle contexte que la proposition de construire l'ENIAC est acceptée par l'Armée des Etats Unis. Comme le recompte Polachek:

"The computations required for the preparation of these firing tables were so time-consuming that they overwhelmed the facilities available at the laboratory. In spite of the extensive arrangements the laboratory made with the University of Pennsylvania

for assistance, the backlog continued to grow. […] It was to relieve this bottleneck that John W. Mauchly and J. Presper Eckert, two members of the faculty of the Moore School of Engineering, the University of Pennsylvania, proposed the construction of the ENIAC."

Ainsi, dans le cas de l'ENIAC c'était la vitesse de la machine qui consituait la raison d'être de la machine. Mais, pour prendre avantage de ce vitesse c'est nécessaire d'avoir des technique internes et externes qui permettent d'exploiter cette vitesse. C'est à dire, le temps subjectif de la machine, qui ce développe dans plusieurs ordre de magnitude plus grand que celui de l'humain ou des autres agents, comme la mémoire externe méchanique qui travaillait avec la machine, nécessitait des nouvelles méthodes qui sont adaptés à l'introduction de ce temps de la machine. L'introduction des générateurs pseudo-aléatoires est un bon exemple de telles techniques: c'est parce que la mémoire de l'ENIAC n'était pas bien adapté au temps de la machine que c'était désirable que la machine elle-même générait ces nombres quand ils sont demandés par le programme. Mais aussi la technique de la programmation même, comme pratique intermédiaire entre l'humain et la machine, devait être penser et repenser: pour exploiter la vitesse de la machine, c'était nécessaire que les processus de calcul ne soient pas interrompu. Par conséquence, la contrôle du calcul par la machine même devenait pas seulement souhaitable mais inévitable. [slide] Alors, l'idée des programmes enregistrés comme technique d'ingénierie et non pas des mathématiques, est une conséquence naturelle de ce problème d'adaptation des différents temps et c'est ainsi que von Neumann le proposait dans un exposé de 1946:

"It is evident that in order to be efficient, you will have to treat the problem of logical instructions on the same level on which you treat other memory problems. In a complicated process, you must not only remember some intermediate result [...] but you must also remember what it is you are supposed to do. [I]t has all the earmarks of a memory problem because while you might put your instructions there in some permanent form, you must be able to sense them very rapidly. So, by and large, it is probably true that in a fast electronic machine you will have to produce a memory which cannot only hold hundreds of numbers, but also a few hundred logical instructions."

[slide] Mais le problème de la programmation ne se pose pas seulement et purement comme problème d'ingénierie, c'est à dire, l'inadaption de la vitesse de la mémoire externe à la vitesse de la machine, mais aussi comme problème entre les temps subjectifs de la machine et de l'humain.  Parce que l'humain ne peux plus "vivre" les processus du calcul -- ils sont trop vite -- elle devait anticipé ce qui se peux passer

dans les processus du calcul qu'elle veut implementer dans la machine alors que la machine même pouvait réagir quand, p.e., un certain valeur est surpassé. C'est dans telle confrontation avec les limites du temps subjectif humain en face de la machine, que des programmes enrégistrés deviennent nécessaire car ce n'est plus l'humain qui fait les décisions *pendant* le processus du calcul mais la machine. L'humain, elle devient l'organisateur des séquences qui coordonnaient ces processus. Comme l'expliquait Franz Alt:

"I think the thing that we learned with this high speed was that ...you had to have a way to program ahead of time -- long sequence[s] of operations. ...On slow computers you can add manually, you can request manually the performance of each arithmetic operation. But a fast computer is useless unless you have some way to program it …. The idea of a stored program […] couldn't have come up before we had the ENIAC because it would have been useless. But with ENIAC it was mandatory to have something like that"

Tel "prévoyance" tacite dans la programmation, demandaient des méthodes qui essaient à mettre en harmonie les perspectives de l'humain et de la machine. La machine devait comprendre ce que veut dire le programmeur et le programmeur devait comprendre ce que la machine peut faire. C'était à ce point si que la logique commence à jouer un rôle fondamental dans la programmation du point de vue humaine. Comme l'explique von Neumann:

"[It is] necessary to consider carefully the ability of the computing mechanism to take our intention correctly. And the person controlling the machine must foresee where it can go astray, and prescribe in advance for all contingencies. To appreciate this, contemplate the prospect of locking twenty people for two years during which they would be steadily performing computations. And you must give them such explicit instructions at the time of incarceration that at the end of two years you could return and obtain the correct result for your lengthy problem! This dramatizes the necessity for high planning, foresight, and consideration of the logical nature of computation. This integration of logic in the problem is a consequence of the high speed."

[slide] Ainsi, la programmation, et je cite à nouveau von Neumann: "is not a static process of translation, but rather the technique of providing a dynamic background to control the automatic evolution of a meaning[.] [As such] it has to be viewed as a logical problem and one that represents a new branch of formal logics"

C'était exactement pour cette raison que von Neumann développait avec Goldstein la notation des "diagramme de flux": ils sont interprété comme des structures logique

d'un programme qui rendre plus facile la programmation par la séparation stricte entre des opérations logique, qui controle le dynamique d'un processus et des opérations arithmétiques. Avec ces diagrammes, primitives qu'ils soient, le premier pas pour développer un interface avec une structure qui permet de maitriser des problèmes de programmation était pris. Ainsi, nous voyons que l'introduction de la logique dans l'ordinateur -- c-a-d des programmes enrégistrés -- et sa programmation est ancrées dans des problèmes pratiques qui se présente dans le décalage entre le temps subjectif des humains et de la machine électronique.

Mais cette utilisation de la logique n'est pas éventuelle: c'est la machine même qui nécessite l'introduction des formalismes. C.a.d., le formalisme comme une manipulation aveugle de marques sans sens, et, par conséquence, sans différence entre p.e. opérateur et opérande, c'est ca que la machine fait. Dans un tel contexte, le formalisme devient une pratique qui dirige la machine. Comme l'expliquait Dijkstra plus tard:

"The manipulation of uninterpreted formulae is [...] a most familiar operation for the computing scientist: it is the one and only operation computers are very good at. [...] The manipulation of uninterpreted formulae requires unambiguous formalisms [...] Our disappointing experiences with formality should not be interpreted as something being wrong with formality; the experiences were disappointing because we were incompetent amateurs. But this has been changed by our exposure to computing"

[slide] Au moment qu'on se réalise que l'ordinateur est en fait une manipulateur des marques sans sens, la séparation dans la programmation entre ce que l'humain fait -- programmer -- et ce que la machine fait -- executer une séquence d'opération -- disparaîtait: si la machine est capable de "comprendre" le formalisme, elle aussi peut programmer. Comme l'expliquait Lieutenant Grace Hopper [slide]:

"Of course, at that time the Establishment promptly told us [...] that a computer could not write a program; it was totally impossible; that all that computers could do was arithmetic, and that it couldn't write programs; that it had none of the imagination and dexterity of a human being. I kept trying to explain that we were wrapping up the human being's dexterity in the program that he wrote [...] and that of course we could make a computer do these things so long as they were completely defined."
Cette réalisation a résulté dans une extension de ce qu'on peut faire avec les calculs numériques: il ne sont plus limités à des calculs d'arithmetique mais compris tous qu'on peut traduire dans des transformation des marques dans des autres marques en suivant des règles finis et explicite. Ainsi, ou la logique a été introduit au début comme moyen de contrôle du décalage temporel entre l'humain et la machine, elle

devenait aussi un moyen pour donner plus de contrôle à la machine, résultant dans une machine plus complexe: l'intériorisation de plus en plus de différents type de calculs n'exige pas seulement une adaptation entre les temps subjectif humain et machinale mais aussi entre les différent temps subjectifs des processus de ce différent types de calcul. Cette situation n'a pas résulté dans une résolution des problèmes de programmation mais a créé des nouveaux problèmes et a résulté dans une communication plutôt non-linéare [slide]: aujourd'hui, la programmation implique normalement un complexe d'interactions avec des différents interfaces qui se developpent dans le temps en parallèle avec à l'arrière-plan des processus non-visible. Au moment que j'écrivais ce texte, 13 programme visible était ouvert et 86 processus perceptible était en train d'être executer! Tous ces processus ont leur propre temps et c'est presque incroyable que c'est encore possible d'avoir une communication éloquente qui suppose essentiellement une synchronization de ces processus et le temps subjectif de leurs interlocuteurs humain ou non-humain. [slide] De cette manière, la logique comme moyen de contrôle resultait pas nécessairement dans une résolution de l'imprévisibilité des programmes et les problèmes qui en résultent: en donnant plus de contrôle à la machine on contrôle en fait moins.

Jusqu'ici j'ai partagé avec vous deux expérience du temps du point de vue humain et qui sont ancrées dans l'incapacité de l'humain d'être plus vite que lui même. Mais ce n'est pas seulement l'humain qui est limité par son temps. Je voudrais finir cet exposé avec une dernière expérience, plus particulièrement, une petite expérience à moi que se situe dans le contexte des mathématiques assistées par l'ordinateur. Elle montrera une expérience du temps qui est ancrées dans l'interaction même entre l'humain, l'ordinateur et leurs limites [slide].

Le résultat de cette expérience était ma première publication dans la deuxième année de mon doctorat. A ce moment là j'étais interessé à l'utilisation des visualisation pour mieux comprendre des propriétés des sequences de caractères comme celles produit par des systèmes de "tag", utilisant des fractales. Une technique pour visualiser des fractales "normales" comme le triangle de Sierpinski est le jeu de chaos [slide]. Comment est-ce que ca marche? Commencons avec les trois transformation du triangle de Sierpinski, en chosissant trois point arbitraires dans l'espace à deux dimensions et un point initial. Nous choisissons d'une manière aléatoires 1, 2 ou 3 et appliquons la transformation correspondante au point initial. Nous répétons cette procédure au point résultant, etc. Le résultat sera une visualisation assez vite du triangle de Sierpinski. Pour mon expérience je commencais avec une des séquences générer par des systèmes de "tag". J'appliquais le jeu de chaos, utilisant ce séquence fixe au lieu d'un générateur pseudo-aléatoire, pour chaque point dans une région limité dans l'espace autour de

l'attracteur, calculant, pour chaque point, la distance requis alors que ce point atteignait l'attracteur. La visualisation était une visualisation de ces distances. Le résultat était une vraie surprise [slide]: l'espace de ces point se structure dans une manière autosimilaire: le classe des point qui prend q'une itération pour atteindre l'attracteur forme le même fractale que l'attracteur mais plus grand [slide], le classe des points qui prend deux itérations pour atteindre l'attracteur, aussi forme le même fractale mais encore plus grande que l'attracteur originale [side], etc. [slide] Cette expérience, qui m'a rien appris sur des séquences de caractères, était pour moi une expérience profonde montrant la dissonance entre mon temps subjectif et celle de la machine -- j'étais simplement pas capable de prévoir le comportement du calcul que j'avais programmer -- et, par cette dissonance, la significance du temps dans des processus du calcul: c'est parce que je suis un être fini qui vit son temps que je ne peux pas sursauter des processus de l'ordinateur qui prends leur temps  et qu'ainsi, j'ai l'expérience de leurs temps.

Mais cette expérience me confrontait pas seulement avec les limites de mon temps subjectifs. Elles me montrais aussi la finitude de la machine: dans la programmation de telles expériences on doit se rend compte qu'en étudiant des objets et processus potentiellement infinis, on les étude avec une machine qui est elle-même fini et, ainsi, pas capable de sursauter l'infinis. Mais, parce qu'on s'oriente déjà envers la machine à cause de notre limites -- comme l'a remarqué déjà Turing, "if one were to predict everything a computer was going to do one might just as well do without it." -- pratiquement ca veut dire qu'on doit imposer des limites dans les programmes qui donnera aux résultats souvent un caractère heuristique. Dans l'exemple avec le jeu de chaos des choses comme "être sur l'attracteur" ou "tous les points dans la région autour de l'attracteur" sont en réalité "être dans la proximité d'un approximation d'un point de l'attracteur" ou "être un des points discret sur l'écran". Ces techniques resultent dans des questions profonde comme: Est-ce que les visualisation utilisant le jeu de chaos sont "vraies" ou seulement un effet de la finitude de la machine ou de l'humain qui décide quand "être dans la proximité" sera interpreté par le programme comme "être sur l'attracteur". Quelles conclusions est-ce que je peux tirer sur le base de l'information finis sur des objets ou processus (potentiellement) infinis? Ainsi, la finitude de l'humain et de la machine exige des techniques de programmation locale qui ne sont pas capable de sursauter les processus qu'on étude, mais trouvera néanmoins une balance précaire entre le temps fini de l'humain et le temps (et, dans le cas de l'exemple, espace) fini de la machine alors qu'on peut attaquer des nouveaux problèmes.

Dans cet exposé je vous ai donné trois expérience du temps dans le calcul. Est-ce que je peux maintenant vous donner une réponse provisoire sur la question du

temps en calcul? Est-ce qu'il y a du temps dans le calcul et, si oui, qu'est-ce qu'est ce temps? Prenons la définition de la forme de "tag". Est-ce qu'il y a du temps dans cette définition? Non. Le temps est introduit qu'au moment que le processus d'un système de "tag" est initié. Plus généralement, est-ce qu'il y a du temps dans un algorithme? Non. Au mieux, on peut y trouve des traces du temps si c'est p.e. un algorithme de synchronisation mais pas plus. Le temps est introduit au moment que l'algorithme est transformé dans un processus de calcul. Mais telle transformation est seulement possible s'il y a quelqu'un qui s'en occupe. Mais ce quelqu'un est nécessairement quelqu'un qui vis un temps en s'occupant du processus, car sans temps il n'y a pas de processus. C'est ce quelqu'un qui détermine le temps du processus et c'est le processus qui détermine l'expérience du temps de ce quelqu'un. Ainsi, dans mon perspective, le temps en calcul est seulement là comme pratique, comme quelque chose qui se passe et existe seulement dans des actions réelles. De la même facon, le temps est seulement une propriété essentiel du calcul dans la mesure qu'on comprend le calcul comme processus qui se ne laisse pas réduire aux algorithmes.

En informatique, telles processus réels sont toujours le résultat d'une interaction entre, au moins, deux "êtres" finis: l'ordinateur et l'humain qui ont tous les deux leurs temps subjectifs différents, pas seulement avec référence au vitesse mais aussi avec référence à l'expérience du temps dans des processus du calcul: quand j'utilise mon ordinateur pour executer une simulation d'un système de tag, mon ordinateur a une relation dynamique avec ce processus qui est très différent de la mienne résultant dans une structuration des temps subjectifs impliqués très différents, exigeant des moments de synchronisation dans l'interface. C'est la confrontation de ces différents temps subjectifs et, plus particulièrement, les dissonances entre ces différents temps qui montre la significance du temps en informatique pas seulement comme problème pratique mais aussi comme condition philosophique. C'était en fait cette condition qui a résulté presque dans une obsession avec le temps dans les travaux de Post qui n'était pas capable de sursauter son temps propre. Ainsi, plutôt de rémédier et/ou de cacher ces dissonances, peut-être la vraie tâche philosophique est de penser des situation qui invoquent ces dissonances. De cette manière, penser le temps dans l'informatique nous confrontera avec notre condition humaine et, plus généralement, la condition universelle de la temporalité des choses.