Synchronous Artificial Language for Transaction (SALT)

Un code se casse, une langue s'apprend

Lorsqu'en 1942 les Américains décident d'entrer en guerre suite à l'attaque de Pearl Harbour ils font appel à 200 indiens Navajos pour sécuriser leurs échanges radios. Plutôt qu'utiliser des codes secrets toujours cassables (Turing cassera Enigma quelques mois plus tard) et à usage limité (même en excluant les erreurs humaines, plus un code « cause » et plus il distribue de clés pour se faire décrypter), utiliser une langue offre de nombreux avantages.

La différence entre un *code* et une *langue* est qu'un code n'est qu'une méthode plus ou moins complexe de dissimulation par cryptage d'informations distribuées dans une langue donnée (tout code est compréhensible si on en connaît la règle), alors qu'une langue est un dispositif complet permettant non seulement de distribuer des informations, mais aussi de dialoguer, d'argumenter, de permettre aux interlocuteurs de prendre des décisions et donc de les conduire à entreprendre des actions de façon autonome. Un code n'est qu'un ordre *asynchrone* (faire ceci, dire cela, aller à tel endroit, rencontrer untel...) transmis par un émetteur à un récepteur, alors qu'une langue est un outillage complet et complexe permettant des *interactions synchrones* entre locuteurs : il n'y a pas émetteur d'un côté et récepteur de l'autre, mais interaction dynamique entre acteurs de l'échange.

L'ordre/code est *unidirectionnel* et *asynchrone* par définition. Pour construire un système complet de dialogues en code, il est nécessaire de permuter sans cesse l'ordre des émetteurs et des destinataires : X envoie l'ordre crypté à Y, Y le décrypte, prépare sa réponse, l'encrypte à son tour puis l'envoie à X, lequel la décrypte pour la comprendre, puis la renvoie à Z, etc. Chacune de ces opérations est interceptable par une machine hackeuse qui peut ainsi saisir peu à peu la structure du code.

Dans une langue en revanche, l'ordre des locuteurs n'importe pas ; quel que soit celui qui parle, c'est l'ensemble des interlocuteurs qui participe de façon *multidirectionnelle* et *synchrone* à la discussion.

Pour résumer, un code est utilisé dans l'*intention de masquer,* une langue dans l'*intention de communiquer*.

Dans **beAchain** la sécurisation des échanges entre machines (transactions, consensus, mises à jour, etc.) n'est pas basée sur des jeux de clés publiques/privées mais sur des codes et des langages :

– un *code public* commun à toutes les machines : toutes le parlent et toutes le comprennent (*lingua franca*).

– un *code privé* propre à chaque machine : c'est l'encodage/encryptage dans lequel elle écrit et stocke les informations qu'elle détient.

– une *langue de transactions* : c'est cette langue qui est utilisée par les machines qui auront à consensualiser les transactions en cours.

Les deux premiers sont des pseudo-langages ressemblant à des séries d'instructions cryptées ne nécessitant ni échanges ni prise de décisions de la part des machines, ils se rapprochent des systèmes de clés publiques/privées où le message asynchrone ne peut être décrypté que par les destinataires détenant la bonne clé.

La langue de transactions SALT en revanche permet des prises de décisions et de responsabilités : pour que des machines prennent de façon collégiale les *bonnes décisions par consensus* il est nécessaire que (1) elles se parlent, (2) personne ne puisse comprendre ce qu'elles se disent, ni une autre machine, ni évidemment un hacker ayant frauduleusement accès au dialogue. Même parvenant à intercepter leurs échanges, personne ne doit pouvoir comprendre la teneur de leurs échanges.

Dans un protocole complet de transaction **beAchain**, le schéma théorique est le suivant :

a – Alice envoie sur le réseau le message « *Je suis Alice et je veux verser 100 unités de crypto-monnaie à Bob* ». Elle émettra « *Je suis Alice* » en mode public (tout le monde comprend) et « *Je veux verser 100 unités de crypto-monnaie à Bob »* en code privé : ce qu'elle veut faire ne regarde publiquement personne à ce stade. Formellement ça ne fait aucune différence, les mêmes jeux de signes sont utilisés pour les deux parties mais la seconde partie du message reste incompréhensible. Comme tout message, il est envoyé à une adresse de distribution.

b – les « machines sociales » à l'écoute connaissent ou ne connaissent pas Alice (*sharding*). Ce qu'elles savent d'elle, elles le savent par ce qu'elles détiennent en interne comme informations écrites dans leur propre code privé (si leurs informations concernant Alice étaient écrites en mode public, une machine malveillante pourrait du coup dresser la liste de toutes machines connaissant Alice, les hacker et et recueillir l'intégralité des données concernant Alice). Elles sont donc capables de répondre publiquement en mode public « *Moi je connais Alice* ».

c – parmi toutes celles connaissant Alice, un protocole particulier va en choisir cinq qui vont se réunir pour créer d'abord une langue, tenter ensuite de comprendre ce que signifie « *Je veux verser 100 unités de crypto-monnaie à Bob* » et conduire enfin la transaction à son terme, à savoir ou l'accepter comme vraie ou la rejeter comme frauduleuse.

C'est de cette négociation basée sur une *langue artificielle* créée pour l'occasion et qui n'a de validité que le temps que le consensus se fasse (1 à 4 secondes) que dépendra le block où le résultat de la transaction sera encrypté.

Cet article traite de la création de cette ***langue artificielle synchrone de transactions*** (SALT).

Qu'est-ce qu'un langage ?

> « Le langage chez l'être humain est la faculté de mettre en œuvre un système de signes linguistiques (lequel constitue une langue) permettant **la communication et l'expression** de la pensée, ce qui est exclusif des humains, et des sentiments, ce qui est commun avec d'autres animaux. La linguistique est la discipline dont l'objet est l'étude scientifique du langage. […] Le mot “langage” désigne un ensemble d'activités mises en œuvre par un individu lorsqu'il parle, écoute, réfléchit, essaie de comprendre […] lit et écrit. »
> [https://fr.wikipedia.org/wiki/Langage_humain]

Le langage selon Chomsky

> « Dans *Structures syntaxiques,* en 1957, Noam Chomsky avance ses thèses principales. Il nomme "compétence linguistique" la capacité de langage chez un locuteur idéal et "performance linguistique" l'usage de cette compétence par les locuteurs réels. La compétence, ou savoir linguistique du locuteur, a trait à la potentialité d'utiliser une langue et la performance, ou réalisation concrète, se passe dans les actes de communication utilisant un langage verbal.
> Pour Chomsky, la grammaire d'une langue est la description de cette langue et de son fonctionnement, lexique, phonologie, morphologie compris, et **l'ensemble de la production des phrases** de cette langue. Il part, pour cette description, de syntagmes (constituants immédiats de la langue), qu'il présente selon un système génératif, ce qui veut dire un ensemble de règles de réécriture permettant de « produire » ou de « générer » les phrases.
> Les régularités formelles qui s'observent dans les règles de réécriture d'une langue ou d'une langue à l'autre sont la conséquence de la méthode utilisée. L'application de cette méthode montre des contraintes : **la syntaxe suit telle règle et pas une autre**. C'est bien plus qu'une grammaire au sens traditionnel c'est un modèle de la compétence d'un locuteur lorsqu'il utilise une langue.
> La linguistique est une science empirique dont le donné observable est constitué par l'ensemble des langues du monde. Les langues ou langages sont produits par la capacité linguistique, mais elles ont aussi une autonomie qui vient de qu'**elles sont véhiculées par une collectivité** et qu'elles ont des contraintes internes. Si, en tant que science du langage, la linguistique s'est donné pour tâche de définir ce qu'est le langage humain en général, au-delà des langues particulières, alors ne donne-t-elle pas accès à une capacité commune aux hommes qui elle-même a un support qui doit permettre cette généralité et cette communauté. »
> [http://www.philosciences.com/Pss/philosophie-et-humanite/psychologie-representation-cognition/95-noam-chomsky-autonomie-langage]

Introduction au problème

Le langage, en tant que *support culturel et technique d'échange*, est pour partie constitué d'un ensemble de règles formelles fixant une syntaxe, d'un corpus de termes constituant un vocabulaire et d'une mise en forme technique spécifique (vocalisation ou écriture) permettant à un émetteur d'être compris par un récepteur. Ou, au moins, d'être repéré comme émetteur par son ou ses destinataires : parler, d'un pur point de vue fonctionnel, c'est déjà pour commencer émettre un signal, lequel signal sera perçu comme devant être compris.

Au-delà des différences entre langues (vocalique ou consonantique, à tons ou atonale, à substantifs concrets ou abstraits, à prépositions/postpositions ou à déclinaisons, à système temporel ou aspectuel, ...) toute langue représente une pensée, ou plus exactement une façon de penser : si le français et l'anglais privilégie structurellement l'acteur-sujet [*(1)Alice (2)donne (3)ceci à (4)Bob*] dans un séquençage *sujet → verbe → complément,* d'autres langues privilégieront plutôt l'action [*(1)donne (2)ceci (3)Alice à (4)Bob*] là où d'autres privilégieront l'interaction entre sujets [*(1)Alice à (2)Bob (3)ceci (4)donne*]. On voit donc que des raisons entièrement indépendantes de l'action elle-même (le fait que Alice donne quelque chose à Bob) font que chaque langue construira ses propositions d'une façon qui lui est propre.

Les machines ne pensent pas. Elles n'ont pas à préférer une organisation particulière de la langue ni à privilégier une structure langagière plutôt qu'une autre. Elles n'articulent pas de concepts ni n'ont à se soucier de vocalisation. Ce qui leur importe n'est pas le *contexte de locution* mais le *contexte de signification* ; quand elles échangent des signes, ce ne sont pas des symboles qu'elles échangent (le sens d'un signe) mais la valeur sèche du signe : pour elles, le signe n'est pas autre chose que ce qu'il est : il n'est ni investi de symboles – quand je dis *j'aimerais être sur une île* ce n'est pas d'île en tant que « morceau de terre physiquement émergée » dont il est question mais mais du mode de vie symbolique associé – ni représentant d'un concept plus large : un signe est un signe où ce qui signifie est exactement ce qui est signifié : A vaut A et rien d'autre. Elles ne « disent » pas, elles causent. Elles partagent du signe sans partager du sens.

Pour les mêmes raisons la *langue écrite* des machines n'est ni hiéroglyphique (un signe représente l'objet de discours qu'il porte) parce que les machines n'ont pas de système de représentation, ni consonantique (un signe vaut un son) parce qu'elles n'ont rien à vocaliser. SALT manipule des signes-codes dont le sens ne fait sens que parce qu'il est communément partagé entre des machines qui ont momentanément à se comprendre pour prendre une décision et consensualiser/approuver une transaction ou la rejeter.

Comme toute langue, SALT est constituée :

a – **d'une syntaxe** : une certaine façon d'organiser les termes de la transaction/négociation.

b – **d'une levée des ambiguïtés** (rôle des déclinaisons ou pré/postpositions en langages humains).

c – **d'un vocabulaire** constitué d'*unités lexicales sémantiques* (ULS).

d – **d'une procédure** pour préciser si la phrase est interrogative (appelle une réponse), négative, réflexive, etc.

e – **d'une conjugaison** : un système interne de déclinaison permettant de fixer le présent du locuteur pour étager les structures temporelles conséquentes (dans le cas d'un smart-contract à effet différé par exemple) ou encore de lever des ambiguïtés langagières.

f – **d'une écriture** pour transmettre tous les objets de langages produits d'une machine à une autre.

A – La syntaxe

N'étant pas contraintes par un mode de penser les choses qui leur imposerait une type de formulation plutôt qu'un autre (privilégier l'acteur-sujet, l'action elle-même, l'interaction, etc.) les machines ont la capacité de confondre « compétence linguistique » et « performance linguistique » (Chomsky) en une seule « compétence linguistique performante ». Utilisant toutes les configurations possibles sans en privilégier aucune, elles multiplient à l'infini les combinaisons syntaxiques possibles pour une même proposition :

ADCB = *Alice Donne (Ceci) à Bob*
ADBC = *Alice Donne à Bob (Ceci)*
ACDB = *Alice (Ceci) Donne à Bob*
ACBD = *Alice (Ceci) à Bob Donne*
BACD = *à Bob Alice (Ceci) Donne*
BADC = *à Bob Alice Donne (Ceci)*
ou encore
CBAB = *(Ceci) est-Donné à Bob par Alice*
ou encore
DCBA = *est-Donné (Ceci) à Bob par Alice*
Soit plus de 20 formulations différentes pour une seule et même action.

Si on rajoute un acteur supplémentaire, les formulations possibles passent alors à plus 100 :
ABCDE = [*Alice à Bob et à Chris Donne x Euros*]
pourra s'exprimer aussi bien sous la forme AEDBC [*Alice xEuros Donne à Bob et à Chris*] que CBEDA [*Chris et Bob ont xEuros Donnés par Alice*] ou encore EDABC [*xEuros Donnés par Alice à Chris et Bob*]...

Une proposition à 6 objets génèrera plus de 700 possibles et près de 5000 pour 7 objets. On a donc dans ce cas, et au cas un hacker malicieux parviendrait à « observer » le dialogue entre machines, 5000 *formulations syntaxiques différentes possibles* pour la même proposition. Sachant que la langue sera « oubliée » trois ou quatre secondes plus tard, notre hacker aura donc quatre secondes maximum pour explorer toutes les variantes potentielles jusqu'à découvrir la bonne, celle que les machines utilisent actuellement. Soit l'analyse d'*une syntaxe possible par millième de seconde*.

B – un langage sans ambiguïtés

La formulation CBEDA [*Chris Bob xEuros Donnés Alice*] peut simultanément signifier un certain nombre fini de choses contradictoires :
– que Alice donne 100 euros à Bob et 100 à Chris,
– que Alice donne 100 euros à Bob+Chris (50 à chacun)
– que Chris+Bob [x Euros] Donnent à Alice, mais sans qu'on sache si Bob et Chris donnent chacun 100 à Alice (qui détient désormais 200 euros) ou lui donnent 100 au total, soit 50 euros chacun.

La formulation syntaxique structurée CBEDA est donc polysémique ; elle recouvre plusieurs sens possibles. Qui dit polysémie dit ambiguïtés.

Pour lever ces ambiguïtés SALT est construite comme une langue à déclinaisons.

Quand elle dit :
$A(D_1)xEBC$ = elle dit [*Alice Donne [en 1 fois] 100 Euros à Bob+Chris*] donc par exemple 50 chacun, et
$A(D_2)xEBC$ = dira donc [*Alice donne [en 2 fois] 100 Euros à Bob et à Chris*] soit 200 euros au total,
ou D_1 et D_2 sont des déclinaisons du radical D.
Plutôt que d'insérer des prépositions comme le fait le français pour lever toute ambiguïté (comparez
[*Alice Donne Ceci* **À** *Bob*] et
[**À** *Alice est Donné Ceci* **PAR** *Bob*] → forme ADCB dans les deux cas,
elle utilise des variations/déclinaisons des ULS (unités lexicales sémantiques).

C – un vocabulaire sémantique

La structuration ABCDE n'est qu'une structuration. Pour qu'elle ait un sens, il faut que A, B, C, D et E signifient quelque chose.

J'avais travaillé sur une question similaire il y a trois ou quatre ans sur un projet de *Serious Game* en intelligence artificielle destiné au monde médical où l'enseignant, dans une situation donnée (par exemple poser un certain nombre de questions à un patient présentant tel ou tel symptômes ambigus, c'est-à-dire nécessitant un entretien pour lever les points d'ambiguïtés) devait conduire un entretien complet avec le patient pour pouvoir poser son diagnostic. L'étudiant confronté au même scénario de départ devait à sont tour conduire cet entretien pour poser à son tour son diagnostic. Pour que l'épreuve soit validée, l'étudiant devait donc poser les mêmes questions et dans le même ordre (arbre de diagnostic) à son patient. Comment juger que les deux questions, celle du médecin-enseignant et celle de l'étudiant, étaient bien les mêmes, et ce quels que soient le vocabulaire, la syntaxe et/ou l'organisation structurelle discursive utilisée ? Il fallait que les questions similaires « *avez-vous mal au ventre la nuit pendant votre sommeil ?* » et « *ressentez-vous des douleurs gastriques nocturnes ?* », l'une posée par l'enseignant et l'autre par l'étudiant, puissent être comparées pour être évaluées. Se ressemblaient-elles ou ne se ressemblaient-elles pas ?

Une théorie de la linguistique appelée *réductionnisme* pose que toute formulation discursive peut se réduire à un ensemble d'unités radicales interchangeables : même si, sémantiquement parlant, dire « *j'ai une voiture* » ou « *je possède une automobile* » ou encore « *la voiture est à moi* » ou « *ce véhicule m'appartient* » sont différenciables, tous sont réductibles à [MOI] [AVOIR] [AUTO], qu'on le dise dans ce sens ou dans le sens [AUTO] [AVOIR] [MOI]. Ne pouvant pas en tant qu'être humain appartenir à une voiture, c'est nécessairement l'inverse qui est énoncé.

Les machines étant identifiées dans **beAchain** par leurs IDs respectives, la question de l'identification des objets parlants ne pose pas de problème. Ce qui en pose un, c'est « *de quoi parlent-elles* ? » et « *que veulent-elles faire* ? ». La séquence [*ID:Alice + Donne + Ceci + ID:Bob*] ne laisse strictement aucun doute quant à Alice et à Bob, mais qu'est-ce que « *Donner* » et qu'est-ce que « *Ceci* » ?

Pour les actions verbales (les verbes, qu'ils soient d'état ou d'action) l'ensemble des besoins est relativement restreint et la collection est relativement simple à sérier. BeAchain s'appuie sur une cinquantaine d'items couvrant la presque totalité des besoins courants. Quelque exemples :

[GET] avoir, détenir, posséder, ...

[GIVE] donner, remettre, transmettre, offrir, …

[SHARE] partager, échanger, mutualiser, co-utiliser...

[USE] utiliser, se servir de, actionner, mettre en œuvre...

[SAY] dire, parler, causer, proposer, demander...

[MOVE] déplacer, bouger, transporter, changer de place...

[CREATE] fabriquer, concevoir, inventer, générer...

[DO] faire, produire, matérialiser, rendre concret...

[SEEM] sembler, ressembler, paraître...

[EGAL] être égal, équivalent, pareil, identique...

[LESS] être moins que, inférieur à, diminuer...

[MORE] être plus que, supérieur à, augmenter...

[CHANGE] changer, modifier, transformer, déformer...

[ACT] agir, exécuter, actionner...

[STAY] être en un lieu, rester, ne pas changer de place...

...

L'ensemble du vocabulaire ULS (unités lexicales sémantiques) verbal permet assez simplement d'apprendre aux machines de nouveaux mots ou de nouvelles catégories en mixant plusieurs ULS existantes, par exemple :

[BREAK] casser, scinder, diviser... (BREAK existe déjà, ce n'est ici que pour l'exemple)

peut leur être enseigné comme étant [DO]+[LESS] « *faire être-plus-petit* »

D+E – un système grammatical

On se retrouve ainsi avec une langue synthétique artificielle qui certes n'a pas toute la richesse sémantique d'une langue humaine mais ce n'est pas non plus notre besoin : nos machines n'ont ni états d'âme ni un grand sens de la nuance poétique. Nous détenons en revanche tout ce qu'il nous faut pour conduire un smart-contract :

(si) [ID:drone_Alice] [STAY] → lat/long [ID:Bob]

(alors) ID:Bob [GIVE] + 100 cryptos → [ID:Alice]

Dans cet exemple Alice vend et distribue ses produits et son drone livre Bob à domicile. Dès que la livraison est effectuée (drone d'Alice géolocké chez Bob) le transfert en crypto-monnaie est instantané. Contrat qui aurait pu aussi s'écrire ainsi :

(si) lat/long [ID:drone_Alice] [EGAL] → lat/long [ID:Bob]

(alors) ...

C'est la seconde clause (Bob paye Alice) qui fera dès lors objet de consensualisation. Les cinq machines approbatrices devront dialoguer entre elles pour assurer l'écriture finale du block :

(si) wallet [ID:Bob] [EGAL]/[MORE] 100 cryptos (alors) …

→ (si) Bob détient au minimum 100 unités de crypto-monnaie pour payer Alice (alors) ...

Un certain nombre d'objets logiques (*si, alors, ou, et, donc, sauf, sinon, tous, aucun, excepté...*) complètent les formulations conditionnelles des smart-contracts :

→ [IF][THEN][ELSE][AND][OR][ALL][ANY][NONE][EXCEPT][WHEN][CLOSE]...

Plus quelques formulations spécifiques :

La réflexivité :

[IF] ID:drone_Alice [MOVE] ID:drone_Alice → *si drone d'Alice déplace drone d'Alice* ...n'a pas de raison de créer une source d'ambiguïté potentielle. La formulation s'écrit donc en utilisant une forme réflexive de déclinaison :

→ [IF] ID:drone_Alice [MOVE$_0$] → *si le drone d'Alice SE déplace*

La négation :

[IF] ID:Alice [!GET] 100 [CRYPTOS] → *Si Alice ne possède pas le crédit suffisant*

[IF] ID:Alice [!STAY] lat/long[ID:HOME] → *Si Alice n'est pas géolocalisée chez elle*

L'interrogation :

ID:Alice [?GET] 100 [CRYPTOS] → *Alice possède-t'elle le crédit suffisant ?*

ID:Alice [?STAY] lat/long[ID:HOME] → *Alice est-elle géolocalisée chez elle ?*

Reste enfin à définir tout un vocabulaire technique ou descriptif d'objets : WALLET , HOME , DRONE … qui sont simplement entrés dans des tableaux appelés. L'étendue du vocabulaire augmente avec les entrées utilisateurs successives.

Nous avons maintenant un système syntaxique, lexical et grammatical minimal mais suffisant. Nous reste à construire une écriture. Mais avant ça, revenons à notre hacker qui aurait réussi à « comprendre » l'organisation syntaxique de la phrase. Les tableaux lexicaux (ULS) sont eux-mêmes des tableaux de tableaux, ce qui fait que chaque transaction utilise certains tableaux et pas d'autres, et dans un certain ordre et pas un autre. Nous passons ainsi de quelques centaines ou milliers de possibilités à plusieurs millions. Nous verrons plus tard comment les machines apprennent lesquelles utiliser.

E – l'écriture du langage

Dans **beAchain** tous les échanges sont construits sur des encodages particuliers qui n'ont pas pour but premier de rendre l'information illisible par les humains mais de réduire au maximum le volume d'informations à computer. Moins il y a d'octets à traiter, plus ils sont rapides à computer et plus les machines « faibles » (capteurs, prises connectées) sont capables de participer de plein droit et à part entière aux échanges. L'illisibilité relative du cryptage n'est qu'un avantage par effet de bord.

En langage public commun, l'ID d'une machine « Alice » sera par exemple : [*eb4C_f8[N1œéPa*] et celle de « Bob » : [*ö3sSu-!7ZmL)0z*].

Donc [*eb4C_f8[N1œéPa* [GIVE] *ö3sSu-!7ZmL)0z*] signifie littéralement « *Alice donne à Bob* ».

Si « Donner » [GIVE] est constitué du radical *pN9çÄ* auquel on ajoute des déclinaisons utiles selon le

contexte (réflexivité, levées d'ambiguïtés, etc) notre phrase devient : [*eb4C_f8[N1œéPapN9çÄ3bö3sSu-!7ZmL)0z*].

Toute machine recevant cette proposition en langage commun comprendrait qu'Alice remet quelque chose à Bob. Sauf qu'elle n'est jamais distribuée en langage commun. Elle va suivre et dépendre du protocole de création du langage.

Quand la phrase « *Alice veut…* » sous sa forme commune publique apparaît sur le réseau comme initiation d'une nouvelle requête, un algorithme va décider quelles machines vont être appelées pour traiter sa requête. Cette requête est menée au sein du groupe des machines connectées pour trouver celles parmi elles qui connaissent Alice. Une fois ces machines réunies, en comparant et additionnant ce qui est su d'Alice (le *sharding horizontal* empêche une machine de tout savoir à elle seule sur Alice) on obtient un portrait complet de la machine Alice. En même temps qu'elle va poster sa requête au groupe restreint (le monde entier n'a pas à savoir ce qu'Alice veut faire) elle va poster un certain nombre de données identificatoires issues de son ID permettant de calculer sa PoI, sa preuve d'ipséité. Les cinq premières machines à résoudre l'équation PoI et à confirmer que Alice est réellement qui elle dit être s'isolent alors sur un canal de chat privé. C'est là, sur ce canal privé, qu'elles vont créer la langue qui sera utilisée le temps de la négociation/validation.

On a donc au minimum cinq machines.

1 – la première va fixer l'ordre syntaxique en envoyant ABCDEF ou BACDFE ou DCFEBA ou n'importe quel autre modèle structurel possible sous une forme *langage privé* (le second langage interne de **beAchain**)

2 – la seconde machine peut alors fixer le vocabulaire utilisé en déduisant des valeurs computées à partir du code-modèle initial passé par 1. Cette valeur est également passée en *langage privé.*

3 – la troisième fixe les 3 règles d'écritures sous la forme [3][8][45] (série de 4 chiffres).

Règle 1 : [3] signifie qu'on permutera tous les 3 signes : *abc+def+ghi* deviendra *cbafedihg*

Règle 2 : [8] signifie qu'on permutera par paquets de 8 signes. *cbafedih+glkjonmr* deviendra *glkjonmrcbafedih*

Règle 3 : [45] signifie qu'on replacera chaque signe par un autre selon tableau de permutations n°45 : *glkjonmrcbafedih* deviendra *bG4npTDesmv9*

Une fois les règles d'écriture fixées, la langue est créée. Elle ne sera compréhensible que par ces 5 machines. Pour nous assurer qu'il s'agit bien de nos 5 machines et pas d'une tentative de hack, leurs PoI respectives sont computées par chacune des machines et cinq retours valides lancent l'utilisation de SALT comme outil de travail.

4 – la quatrième machine enverra un test aléatoire basique pour que toutes les machines ont bien compris les règles de la langue de travail.

5 – la cinquième lancera alors le protocole de consensus en SALT à partir d'une clé commune.

Les millions de possibilités présentées par les règles syntaxiques et lexicales sont multipliées par les millions de possibles liées aux règles d'écritures, ce qui représente au final une presque infinité de langues différentes possibles. Une même phrase aussi simple que « *Alice verse 100 cryptos à Bob* » peut se présenter sous une quasi infinité de formes différentes, selon que la structure syntaxique générale de la phrase sera ABCD ou DABC ou BADC, selon que [GIVE] sera *_üB8f* ou *Pn!Ks* ou *Om-7i* et selon que les règles d'écritures seront [3][8][45], [9][2][77] ou [6][8][22].

Ces règles n'étant pas écrites par avance mais créées dynamiquement par les machines concernées, on ne peut donc pas créer par avance des jeux de patterns à tenter de matcher sur des éléments pris au hasard dans les propositions échangées. Même si une machine frauduleuse parvenait à « comprendre » quelles sont les règles à partir desquelles le SALT est construit (point 1 à 3), comme elle n'est pas l'une des cinq machines référencées elle échouera au test de

reconnaissance d'identité puisque qu'elle n'enverra jamais de quoi créer le PoI attendu, et donc ne le validera pas. Les dialogues de consensus de transaction durant 1 à 4 secondes, cette machine hackeuse n'aurait donc que ce laps de temps pour by-passer le test PoI, tester toutes les clés produites par la machine 5 et accéder « en clair » aux dialogues transactionnels. Explorer des milliards de possibles en moins de 4 secondes et en n'ayant que l'ID de Alice comme référence publique relève de l'exploit.

Les fréquences SALT

Il existe un classement international des langues basé sur un rapport signe/temps. Les capacités cognitives des interlocuteurs à distinguer des sons émis (reconnaître le sens du son entendu) détermine des vitesses d'élocution. La langue la plus « lente » du monde est le chinois mandarin parce qu'un seul son (*wan* par exemple) peut avoir plus de 20 significations différentes (courber, achever, s'amuser, attirer, golfe, bol, soir, poignet, le nombre 10 000, etc...) selon l'intonation tonale donnée. Pour être compris de ses interlocuteurs, l'émetteur doit donc parler de façon suffisamment articulée pour que les destinataires soient en capacité de distinguer le sens exact de chaque nuance de ce qui est dit. A l'opposé, l'espagnol est la langue la plus « rapide » au monde : les levées d'ambiguïtés y sont si rares que dans un laps de temps très court l'émetteur peut faire passer énormément d'informations sans risque de déperdition ou de mésinterprétation.

SALT est une langue « rapide ». Elle doit rendre les participants au dialogue transactionnel capables de capturer le sens monosémique (sans doutes ni interprétation ni ambiguïté) dans l'instantanéité de l'émission : sitôt qu'une machine parle elle doit parler vite et court. Le volume d'informations transmises doit impérativement être densifié à l'extrême. C'est le rôle de son écriture cryptée : faire passer le plus d'informations possibles dans le moins de signes possibles et le plus vite possible. Chaque octet est un poids à porter ; moins il y en a, plus le déplacement des significations est rapide et efficace.

Un exemple pratique

Soit la phrase « *Alice Donne 100 Cryptos à Bob* » : A, B, C et D + E (100).

En structure syntaxique symbolique on a donc plus d'une centaine de possibles :

[A B C D E] → *Alice à Bob des Cryptos Donne à hauteur de 100*

[B E D C A] → *à Bob 100 unités sont Données en Cryptos par Alice*

[C E B A D] → *des Cryptos à hauteur de 100 à Bob par Alice sont Donnés*

etc.

Prenons le second. La machine 1 envoie la structure [B E D C A] ; à ce stade personne ne sait à quoi correspondent les signes A, B, C, D et E. La machine 2 va choisir un registre lexical tel que [GIVE] soit *üfKç0*. Comme on a des centaines de registres différents, la gamme des langages possibles passe alors à plusieurs milliers.

[BEDCA] devient par exemple :

Pno_3bm8 (Bob) + *sÇup* (E=100) + *üfKç0* (Donne) + *Ör3-v2* (Cryptos) + *ëb4n!CoR* (Alice)

soit : *Pno_3bm8sÇupüfKç0Ör3-v2ëb4n!CoR*

La machine 3 fixe à son tour la règle d'écriture [5 - 9 - 21] (on passe alors à des millions de langues possibles).

La première règle [5] demande de permuter les signes par paquets de 5 :

Pno_3 + bm8sÇ + upüfK + ç0Ör3 + -v2ëb + 4n!Co + R

3_onP + Çs8mb + Kfüpu + 3rÖ0ç + bë2v- + oC!n4 + R

La seconde règle[9] demande de permuter la proposition entière par paquets de 9 :

3_onPÇs8m + bKfüpu3rÖ + 0çbë2v-oC + !n4R → 0çbë2v-oCbKfüpu3rÖ3_onPÇs8m!n4R

La troisième règle [21] demande d'utiliser la 21ème table de conversion « signe à signe » :

0çbë2v-oCbKfüpu3rÖ3_onPÇs8m!n4R → uNj?àG5f-Cnp63b1pù_8wZœSbm#2Rz

Pour une machine, qu'on dise [*Pno_3bm8sÇupüfKç0Ör3-v2ëb4n!CoR*] ou qu'on dise [*Nj?àG5f-Cnp63b1pù_8wZœSbm#2Rz*] n'a aucune importance, il ne s'agit pas de couches de complications rajoutées les unes aux autres mais de règles langagières. Quand la règle finale est connue, peu importe comment la phrase est écrite.

A ce stade on sait que « *quelque chose va être donné par quelqu'un à quelqu'un d'autre* » et qu'Alice est concernée, mais ne sait toujours pas exactement quoi, ni par qui ni à qui. La machine 4 va lever cette ambiguïté en envoyant une demande à laquelle les autres machines devront répondre pour confirmer leur apprentissage du langage.

Une machine hackée introduite frauduleusement dans la boucle serait théoriquement capable de comprendre correctement le process langagier en cours ; c'est la raison pour laquelle le test PoI n'enverra le code explicitant l'intégralité de la transaction en cours qu'aux machines « véridiques » ayant satisfait à PoI (preuve d'ipsétité).

Ce code précise à la fois les identités des machines concernées (Alice et Bob), la chose échangée (100 cryptos) et qui les donne à qui (Alice à Bob).

On a vu plus haut comment les schèmes sont construits :

[*Pno_3bm8* (**Bob**) *sÇup* (**E=100**) *üfKç0* (**Donne**) *Ör3-v2* (**Cryptos**) *ëb4n!CoR* (**Alice**)] → [BEDCA]

Sans cette clé-code, la phrase est comprise comme :

[*Pno_3bm8* (**X1**) *sÇup* (**Y**) *üfKç0* (**Donne**) *Ör3-v2* (**Z**) *ëb4n!CoR* (**X2**)] → [?? D ??]

où ni les acteurs **X1** et **X2**, ni les valeurs de **Y** et **Z** ne sont précisées.

Ce code-clé associe dans un tableau X1 à Bob, X2 à Alice, Y à 100 et Z à Cryptos. Il décline également le verbe « Donner » de façon à ce qu'on sache que c'est Alice qui donne à Bob et pas l'inverse ; on obtient ainsi une série complète de flexions du verbe : génitif, datif, accusatif...

La séquence initiale [*uNj?àG5f-Cnp63b1pù_8wZœSbm#2Rz*]

devient alors [*uNj?àG5f-Cnp63b1pù_8wZœSbm#2Rz+b8ck*] (clé-code)

La langue de transaction SALT créée pour cette transaction étant maintenant complète, le process de négociation peut alors commencer. Chaque machine ira chercher dans ses savoirs si des données concernant Alice et Bob sont disponibles et conformes en horodatage et en validations/approbations antérieures (garantie collective de fiabilité), si leurs wallets respectifs sont cohérents en valeurs détenues, si Alice détient au moins le montant de cryptos qu'elle souhaite verser à Bob, etc. Une fois que le dialogue entre les machines est terminé elles prennent alors la décision consensuelle d'accepter ou de rejeter la demande d'Alice. Si la transaction est approuvée le block est écrit en *langage public* puis distribué sur le réseau, la SALT est immédiatement effacée et tous les tableaux sont vidés. Les machines redevenues disponibles peuvent dès lors participer à d'autres transactions menées selon d'autres SALT.

La question temporelle

Les langues humaines ont des façons très différentes de placer les événements dans un continuum temporel ; si certaines conjuguent au passé, présent et futur (langues temporelles, le français notamment), d'autres langues, appelées

aspectuelles, ne connaissent ni passé ni présent ni futur, mais des formes de flexions indiquant si une action est déjà réalisée et achevée ou si elle ne l'est pas encore : un événement terminé ou une action achevée sont généralement situés dans le passé, une action inachevée ou un événement non-réalisé sont plus souvent situés dans le futur.

SALT n'a pas directement de système temporel : elle n'a ni passé ni présent ni futur. C'est une langue de type aspectuel qui sérialise les événements selon qu'ils sont réalisés ou pas en utilisant des marqueurs logiques : [IF] [THEN][ELSE] etc.

Imaginons qu'Alice a un véhicule autonome à qui elle demande de venir se garer devant chez elle à 10h. On a donc par exemple la séquence :

[IF] OBJ:Heure [EGAL] 10 [THEN] ID:voiture_Alice [MOVE$_0$] lat/long [ID:home_Alice]

signifiant « *à 10 heures du matin la voiture d'Alice se met en marche et vient se garer devant chez elle*. »

Quel que soit le moment où le contrat est évalué, il y a toujours un rapport temporel d'antériorité/postériorité impliqué par les conditions IF et THEN : la condition décrite par IF (il est 10h pile) est nécessairement antérieure à la conséquence THEN qui lui est postérieure. A 9h du matin, les deux propositions relèvent encore d'un futur irréalisé ; à 10h et quelques minutes, IF relève déjà du passé et THEN encore du futur proche (l'action « se garer » n'est pas encore achevée) ; à 11h, c'est le process complet IF-THEN qui est rejeté tout entier dans le passé. Il est donc possible de sérier chronologiquement des séquences d'événements en imbriquant des conditions IF-THEN-ELSE les unes dans les autres en tenant compte du marquage conditionnel « réalisé/irréalisé ».

L'art du dialogue

Une fois que tout ceci est fait, la dernière chose complexe restant à réaliser est de faire comprendre aux machines qu'elles doivent discuter entre elles, et pas seulement écouter ce qui se dit autour d'elles. Elles doivent participer à la construction de propositions d'actions que toutes doivent pouvoir comprendre et appliquer. L'objectif final est d'obtenir un accord consensuel non-tacite concernant la transaction en cours – non-tacite signifie que chaque machine doit donner son avis et que tant que tous les avis ne sont pas donnés, le consensus ne peut avoir lieu : le proverbe « *qui ne dit mot consent* » ne s'applique pas dans SALT.

Pour ce faire, on va les contraindre via des algorithmes dédiés à aller chercher dans leur mémoire toutes les données utiles dont elles auront besoin. Ainsi la phrase : *ID:voiture_commune [MOVE$_0$] lat/long[ID:Alice]* représente à la fois l'affirmation suivante « *la voiture partagée se déplace jusque là où Alice se trouve* » mais appelle aussi des réponses aux questions suivantes :

1 – *que sait-on de la voiture ?* → *c*haque machine recherche ce qu'elle sait de la voiture commune partagée : existe-t-elle ?, est-elle actuellement libre ou occupée ?, est-elle en état de se déplacer ?, où se trouve-t-elle actuellement ?, etc.

2 – *que sait-on d'Alice ?* → Alice fait-elle partie du groupe des usagers ?, est-elle en droit d'utiliser la voiture ?

3 – s*ait-on où se trouve Alice* ? → Chaque machine renvoie les données de position lat/long actuelle du phone d'Alice qui a lancé la requête.

On obtient ainsi à force de questions-réponses séquencées un *accord général consensuel* garantissant qu'à la fois Alice, sa position et le véhicule sont des objets fiables et connus et que la transaction VOITURE → ALICE est acceptée et donc réalisée. Nul ne pourra démarrer la voiture ni l'envoyer là où est Alice sans ce consensus.

Des protocoles de type Byzantine Fault Tolerant précisent sous quelles conditions ces décisions collectives sont recevables et fiables. Le résultat du consensus est ensuite transcrit en langage commun et réparti sur le réseau.

Un embryon d'intelligence

L'analyse lexico-sémantique réductionniste en ULS permet aux machines de « comprendre » ce qu'un humain comprendrait sans peine dans une phrase : le travail cognitif consistant à computer nos connaissances lexicales pour comprendre que [AUTO] et [VOITURE] sont synonymes ne va pas de soi pour une machine qui pense en 1 et en 0.

Autant apprendre à fabriquer et à parler une nouvelle langue en quelques fractions de secondes ne leur pose pas de problème particulier, autant réduire une proposition en ULS : [A] [SAY_1] [B] [HELLO] à partir de la formulation humaine « *Alice dit bonjour à Bob* » est un peu plus complexe, quoique assez simple ; en revanche réduire la phrase « *Si Alice détient plus de 100 unités de crypto-monnaie elle en donnera 70 à Bob et 30 à Chris si Bob en possède moins de 50 ou en donnera 30 à Bob et 70 à Chris si Bob en possède 50 ou plus* » est beaucoup plus sportif.

La méthode utilisée consiste à détecter puis isoler les acteurs en commençant par l'auteur-sujet, Alice.

Si [ALICE] *détient plus de 100 unités de crypto-monnaie* [ALICE] *donnera 70 à Bob et 30 à Chris si Bob en possède moins de 50 ou* [ALICE] *donnera 30 à Bob et 70 à Chris si Bob en possède 50 ou plus.*

Ce qui donne la structure générale suivante :

[IF] [ALICE] -----
    [THEN] [ALICE] ----
    [OR] [ALICE] ----

On recherche ensuite les acteurs-objets et les articulations verbales :

[IF] [ALICE] [GET] -----
    [THEN] [ALICE] [GIVE] [BOB] [CHRIS] ----
    [OR] [ALICE] [GIVE] [BOB] [CHRIS] ----

On explore enfin les sous-conditions internes à chaque proposition :

[IF] [ALICE] [GET] -----
    [THEN] [ALICE] [GIVE] [BOB] [CHRIS]
        [IF] ----
    [OR] [ALICE] [GIVE] [BOB] [CHRIS]
        [IF] ----

La machine commence à comprendre que des conditions s'imbriquent les unes dans les autres et que des jeux de décisions [IF] [THEN] [ELSE] sont nécessaires :

[IF] [ALICE] [GET] -----
    [THEN]
        [IF] [BOB] …. **[THEN]** → [ALICE] [GIVE] [BOB] [CHRIS] ....
        [ELSE] [IF] [BOB] …. **[THEN]** → [ALICE] [GIVE] [BOB] [CHRIS] ....
[ELSE] → rien, fin du process

A ce stade il est est nécessaire de comprendre comment est organisée la sous-condition discriminante [IF] [BOB] qui structure le propos conditionnel *si ceci sinon cela...*

[IF] ID:wallet[BOB] [EGAL] (50-) [THEN] ….

[IF] ID:wallet[BOB] [EGAL] (50) [OR] (50+) [THEN] ….

Qui revient à :

[IF] ID:wallet[BOB] [EGAL] (50-) [THEN] ….

[ELSE] → [THEN] ….

La résolution conditionnelle étant levée, on peut passer au résultat de chaque option :

[IF] ID:wallet[BOB] [EGAL] (50-) [THEN]

[ALICE] [GIVE] ID:wallet[BOB] (70)

[ALICE] [GIVE] ID:wallet[CHRIS] (30)

[ELSE] [THEN]

[ALICE] [GIVE] ID:wallet[BOB] (30)

[ALICE] [GIVE] ID:wallet[CHRIS] (70)

Il ne reste plus qu'à ré-inclure ces sous-conditions à la condition initiale IF ALICE :

[IF] ID:wallet[ALICE] [EGAL] (100+) **[THEN]**

[IF] ID:wallet[BOB] [EGAL] (50-) **[THEN]**

[ALICE] [GIVE] ID:wallet[BOB] (70)

[ALICE] [GIVE] ID:wallet[CHRIS] (30)

[ELSE] [THEN]

[ALICE] [GIVE] ID:wallet[BOB] (30)

[ALICE] [GIVE] ID:wallet[CHRIS] (70)

[ELSE] [CLOSE]

L'apprentissage des décompositions/réductions en ULS de propositions complexes est toujours améliorable. Les étapes suivantes consisteront à modéliser un très grand nombre de patterns auxquels comparer les phrases écrites par les utilisateurs humains pour parvenir au plus vite à générer ce type de structures syntaxiques réductionnistes.

En conclusion

La création d'une langue SALT est à la fois destinée à créer un idiome commun de négociation entre machines n'ayant pas la même langue/clé privée(s) et à empêcher toute machine non-autorisée à participer au dialogue de consensus ou à comprendre le sens des échanges en cours.

Dans **beAchain** *la confiance repose entièrement sur la confiance que les machines s'octroient les unes aux autres.* Comme dans le mode social des hommes, c'est en se parlant qu'elles apprennent à se comprendre et à se faire confiance les unes les autres. Si une seule machine parvenait à tromper cette confiance, des données litigieuses pourraient peu à peu se distribuer de machine à machine et finir par obérer la confiance totale du système tout entier ; c'est la raison pour laquelle divers protocoles sont mis en œuvre à chaque stade du process transactionnel. SALT en fait partie.

Alain Brégy • ab@bechain.com • 02-2017