Habilitation à Diriger des Recherches

Étude et modélisation de la syntaxe des langues à l'aide de l'ordinateur

Analyse syntaxique automatique non combinatoire

Jacques Vergne

GREYC - UPRESA 6072 - Université de Caen

© Jacques Vergne - GREYC - Université de Caen

Plan de l'exposé

•1• **Axes de recherche** : la syntaxe des langues entre linguistique et informatique

•2• **Linguistique** informatique : recherches en syntaxe des langues à l'aide de l'ordinateur

•3• **Informatique** linguistique : recherches en analyse syntaxique automatique

•4• **Méthode** : l'ordinateur, instrument de recherche en syntaxe

•5• **Validation et valorisation des recherches**

•6• **Prospective** : orientations des recherches du "Groupe Syntaxe"

•1• Axes de recherche

la syntaxe des langues entre linguistique et informatique

Linguistique informatique

recherches en syntaxe des langues à l'aide de l'ordinateur

intégration des **processus** de production et de réception dans la syntaxe

Informatique linguistique

recherches en analyse syntaxique automatique

conception et réalisation d'analyseurs non combinatoires **≡ calculatoires**

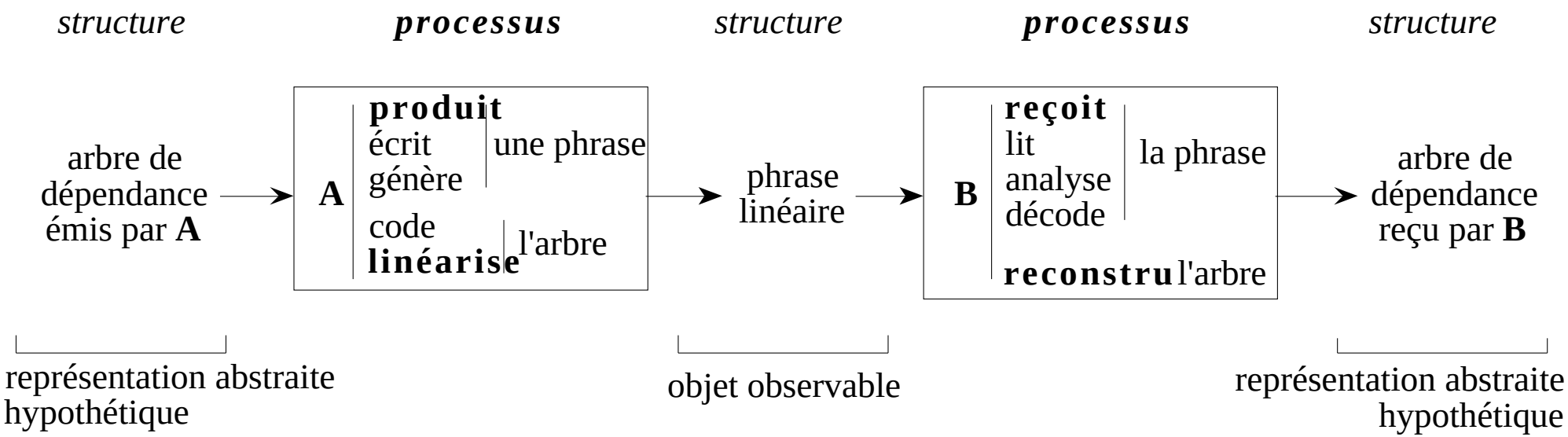
--- modèle linguistique du processus de réception --->

<--- concept du parcours d'arbre ---

les processus et la mémoire

•2• **Linguistique** informatique

modélisation des processus de transformation
entre arbre de dépendance et ordre linéaire



intégration des processus de production et de réception dans le champ de la syntaxe

•2• **Linguistique** informatique

modélisation des processus de transformation
entre arbre de dépendance et ordre linéaire

•2.1• en production, le processus de linéarisation de l'arbre :

hypothèse sur les contraintes de **mémoire** agissant sur le processus
de production : minimisation de l'effort de mémoire
hypothèse sur la contrainte de l'espace à une dimension
importation des concepts de parcours d'arbre, issu de l'informatique théorique

•2.2• en réception, le processus de reconstruction de l'arbre :

hypothèse sur un processus de mise en relation fondé sur la **mémoire**,
modélisée et généralisée dans le cadre d'un analyseur syntaxique

•2.1• observation du matériau
−− *induction* −−>
hypothèses sur les contraintes de mémoire sur le processus de production

• observation du matériau :

les unités reliées sont contiguës ou les plus proches possible

• hypothèse 1 :

les linéarisations observées minimisent la somme des distances entre unités reliées

définir une métrique des distances entre unités

• hypothèse 2 :

la minimisation de la somme des distances entre unités reliées :

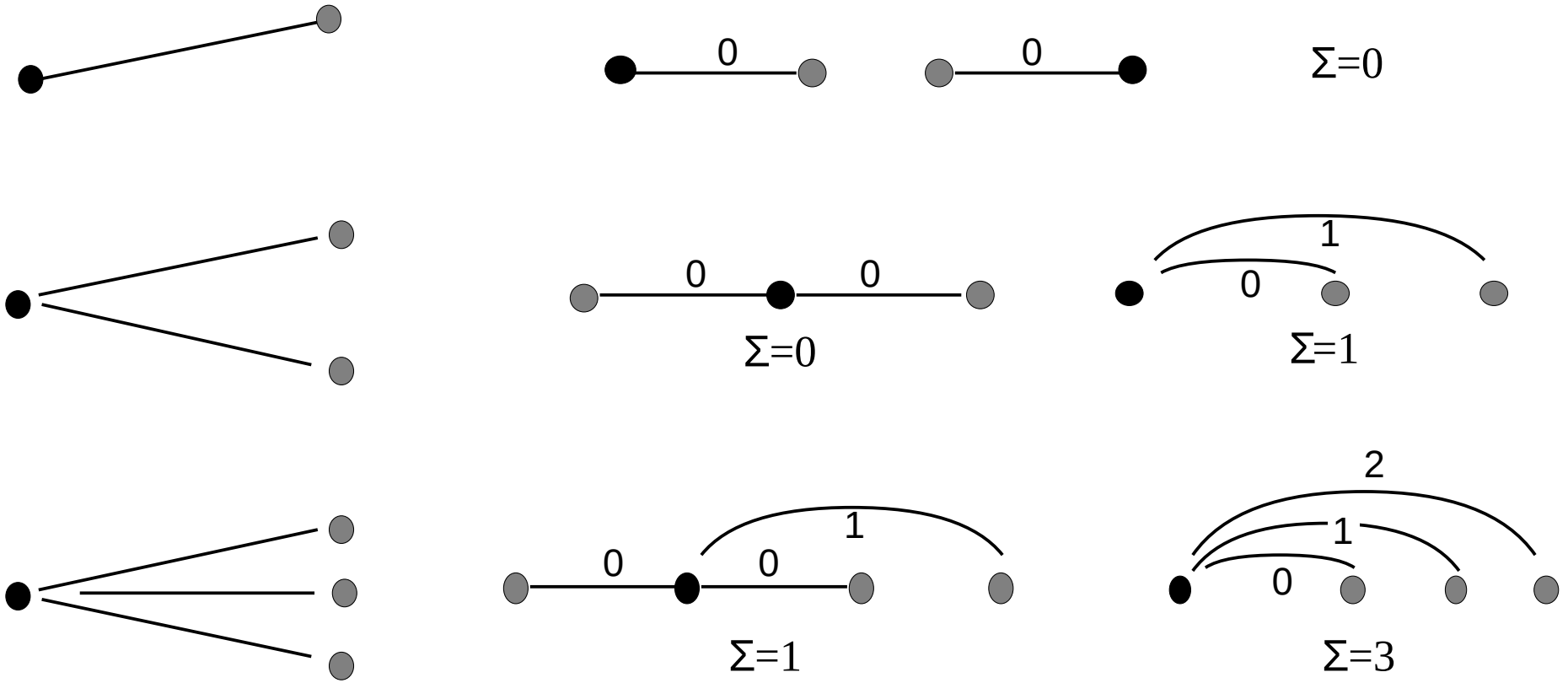
une trace de la minimisation de l'effort de mémoire dans le processus de production

•2.1• contrainte de l'espace à une dimension

métrique : distance entre 2 nœuds = nombre de nœuds linéarisés entre ces 2 nœuds

arbres

linéarisations



•2.1• méthodes de linéarisation : projection ou parcours

- la linéarisation par **projection** de l'arbre dépend du dessin de l'arbre (Hays)

  ordre linéaire = résultat d'un processus sur le **dessin** de l'arbre

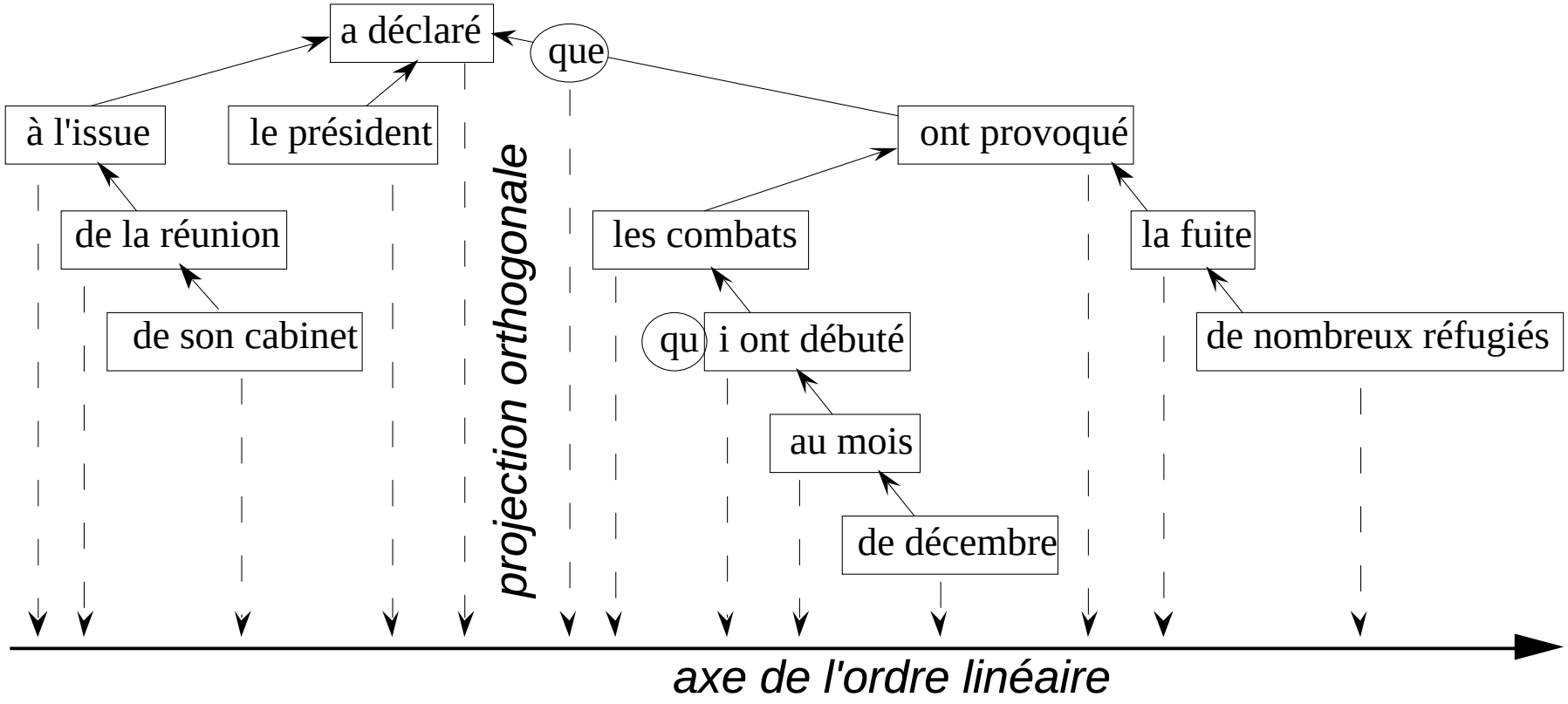
  tentative de coder *simultanément* dans un même schéma
  à la fois l'ordre structural et l'ordre linéaire (comme l'arbre syntagmatique)

- la linéarisation par **parcours** d'arbre ne dépend que de la structure de l'arbre

  ordre linéaire = résultat d'un processus sur la **structure** de l'arbre :
  la linéarisation optimisée par parcours en profondeur d'abord

  dessin de l'arbre *dissocié* de l'ordre linéaire → toute liberté de dessin

•2.1• exemple d'arbre projetable à la manière de Hays



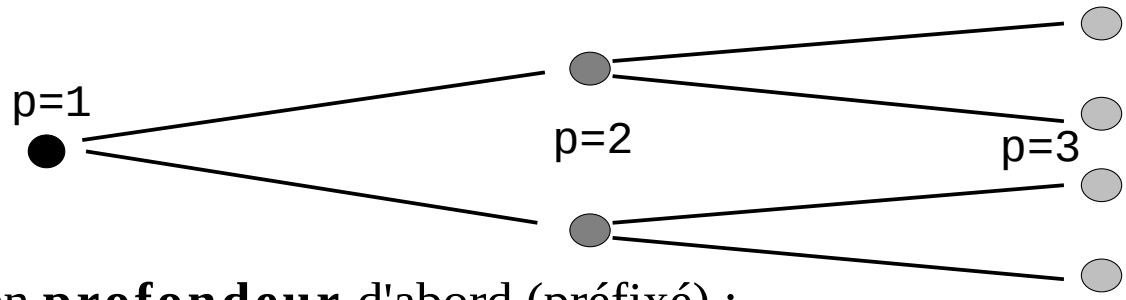
ordre structural et ordre linéaire codés *simultanément* dans un même schéma

•2.1• linéariser l'arbre par parcours en profondeur d'abord

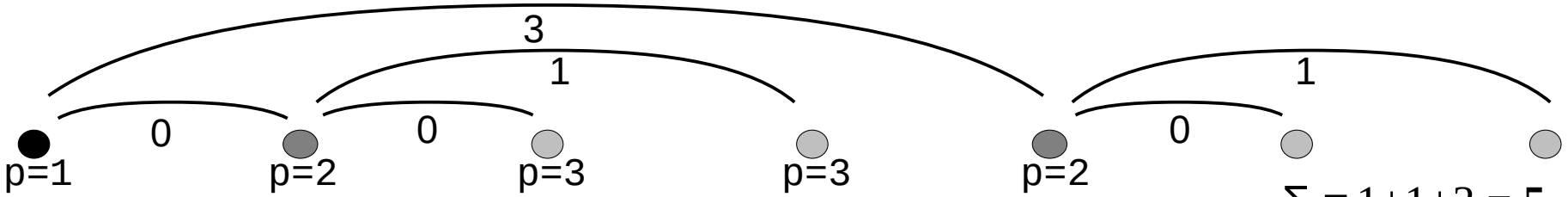
le parcours en profondeur d'abord minimise les distances entre nœuds reliés

métrique : distance entre 2 nœuds = nombre de nœuds linéarisés entre ces 2 nœuds

arbre non linéarisé

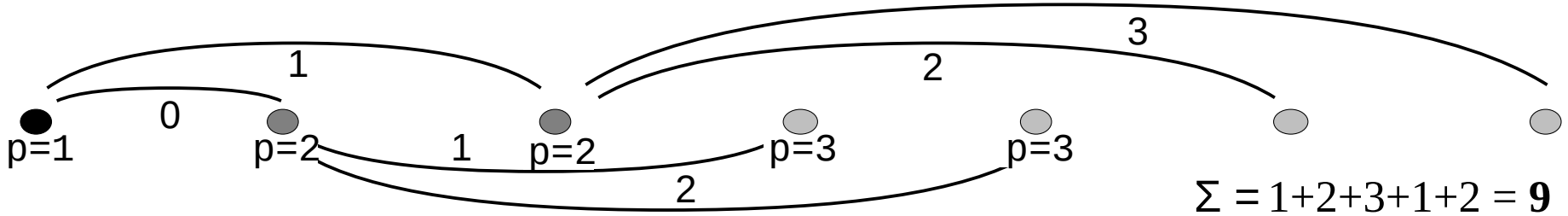


linéarisation par parcours en **profondeur** d'abord (préfixé) :



Σ = 1+1+3 = **5**

linéarisation par parcours en **largeur** d'abord (préfixé) :



Σ = 1+2+3+1+2 = **9**

•2.1• calcul du choix de la linéarisation optimisée

• on a un critère de comparaison entre des linéarisations différentes :

pour chaque linéarisation : la somme des distances entre unités reliées

• à partir de l'hypothèse du **moindre effort de mémoire**

→ définition géométrique du critère d'optimisation de la linéarisation :

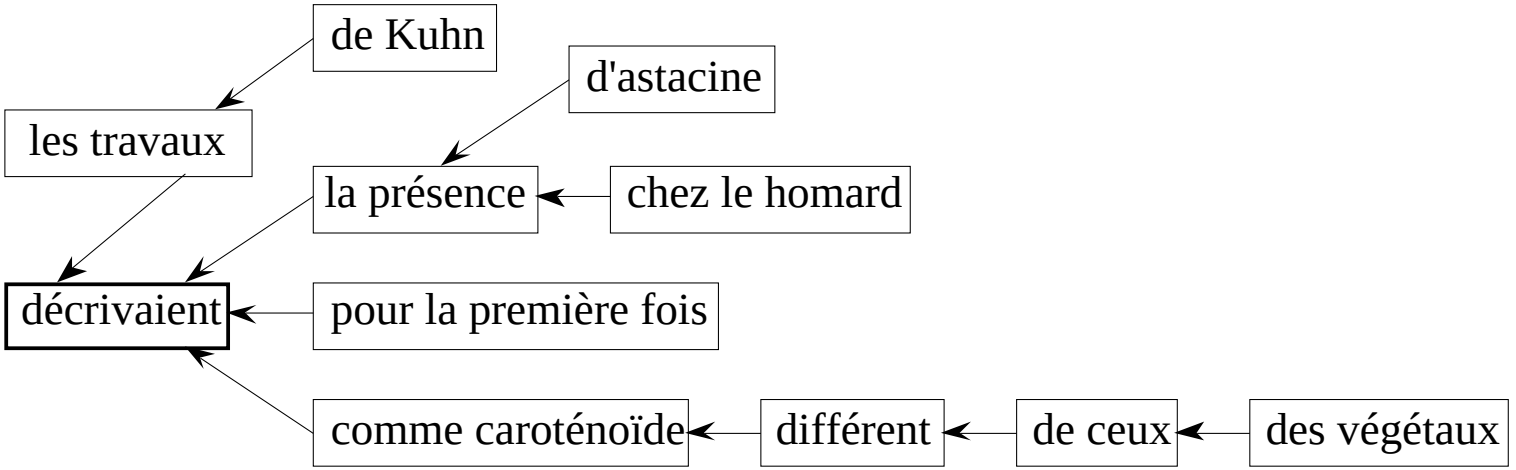
linéarisation optimisée = linéarisation qui minimise la somme des distances entre unités reliées

• hypothèse **corroborée** sur corpus : les linéarisations observées sont optimisées

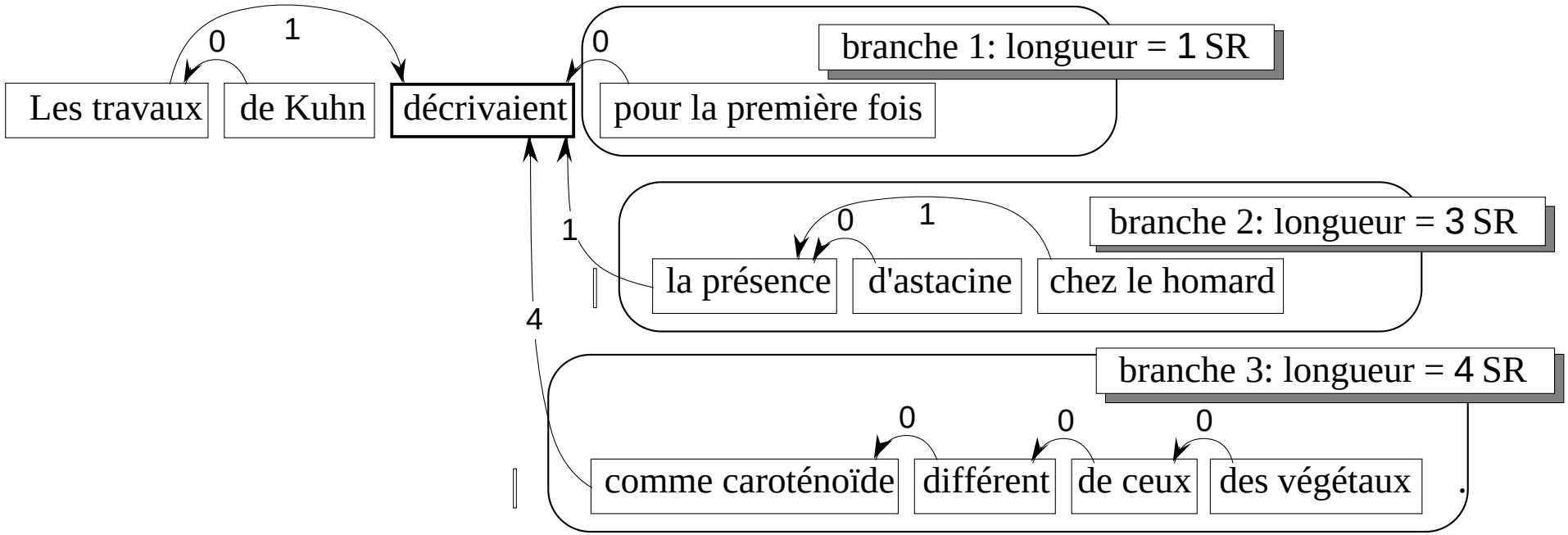
•2.1• exemple d'arbre de dépendance à transmettre

nœud = Syntagme Non Récursif (SNR) = Groupe Accentuel (GA)

relation = relation de dépendance



•2.1• exemple de calcul de la linéarisation optimisée



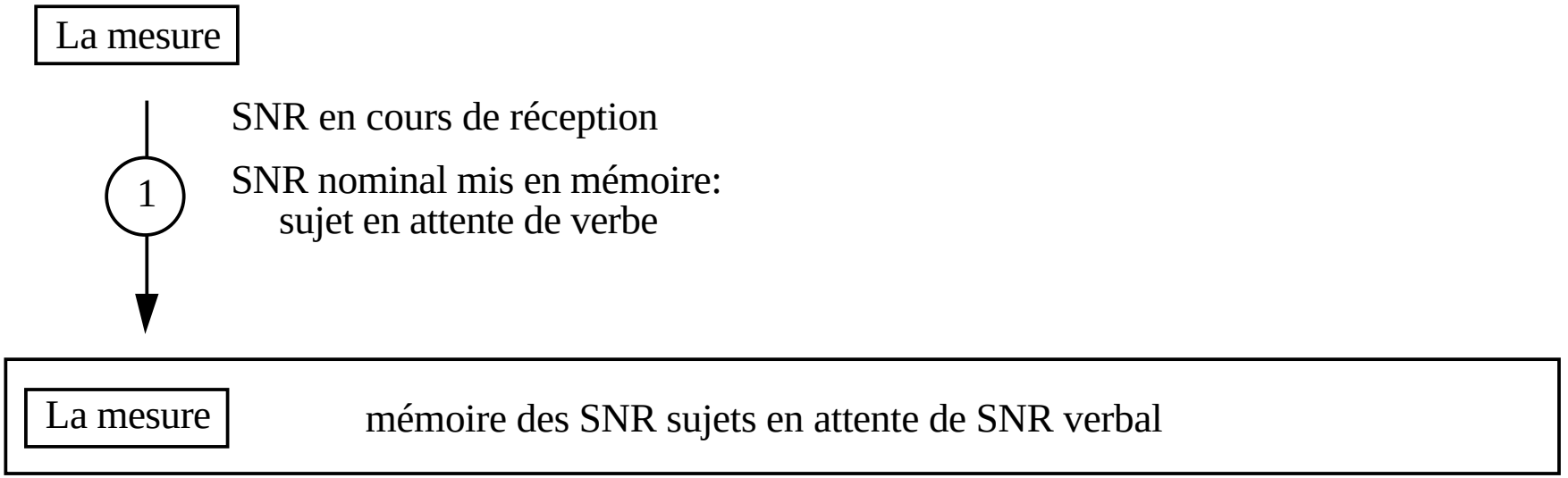
les branches dépendantes du verbe sont linéarisées

dans l'ordre des longueurs croissantes : 1, 3, 4 (démontré)

cet ordre minimise la somme des distances entre unités reliées

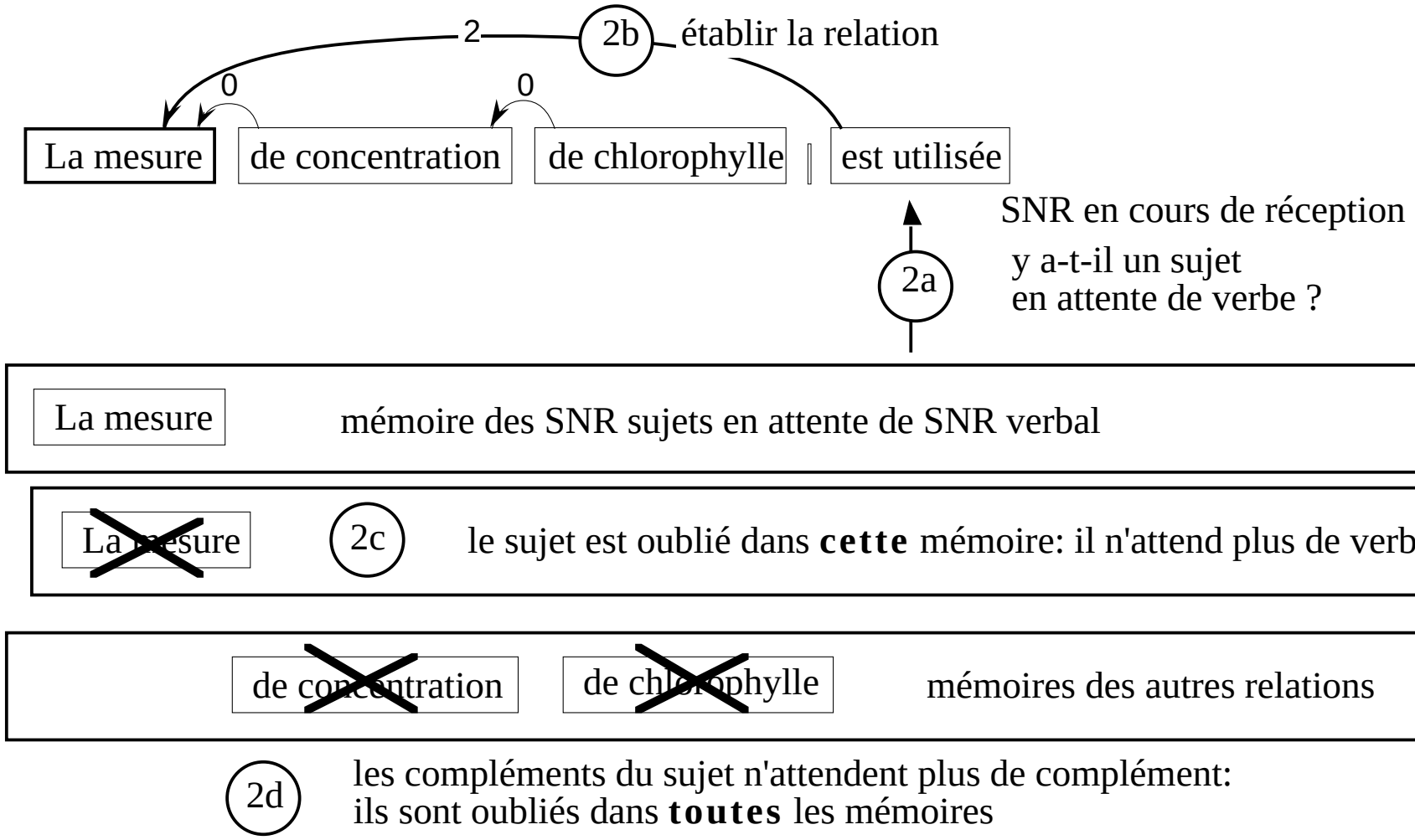
•2.2• processus de mise en relation exemple : sujet - verbe

temps **1**



•2.2• processus de mise en relation

temps **2**



•2.2• définition de la mise en relation comme processus de calcul

par le récepteur, par l'intermédiaire de sa mémoire, en 2 temps

temps 1 : • en recevant une unité de type X :

le récepteur la mémorise comme étant en attente d'une unité de type Y

temps 2 : • en recevant une unité de type Y :

a) le récepteur se souvient d'une unité de type X en attente d'une unité de type Y,
b) il relie les 2 unités,
c) il oublie l'unité de type X comme n'attendant plus d'unité de type Y,
d) puis il oublie toutes les unités situées entre les deux unités reliées
⇔ les unités oubliées n'attendent plus rien

temps 2 = calcul sur la représentation présente d'un événement passé (le temps 1)
(= calcul sur un état présent de la mémoire)

•2.2• **généralisation** du processus de mise en relation

• une mémoire spécialisée pour chaque type de relation

• chaque mise en relation **interagit** avec les autres mises en relation en cours
en provoquant l'**oubli** dans toutes les mémoires
de toutes les unités en attente entre les 2 unités reliées
(⇔ une branche close l'est définitivement)

• aucun attendu
sur les structures syntaxiques situées entre les deux unités reliées
sur la distance qui sépare ces deux unités

• **simulation** et **validation** du processus de mise en relation
en le transposant dans l'analyseur syntaxique automatique

•3• **Informatique** linguistique

analyse syntaxique automatique | non combinatoire
| calculatoire

•3.1• résolution **combinatoire** de l'analyse syntaxique automatique

•3.2• résolution **calculatoire** de l'analyse syntaxique automatique

•3.1• résolution combinatoire d'un problème

- "combinatoire" caractérise non pas à un **problème**,
  mais une **résolution** d'un problème (processus, algorithme)

- quel problème ? attribuer une valeur à des attributs d'un ensemble d'unités

- "processus combinatoire" :
  processus qui énumère toutes les combinaisons possibles
  des valeurs des attributs d'un ensemble d'unités
  (déterminant, pronom) *(catégorie)* *(token)*

- un tel processus est implémentable en un parcours d'arbre en profondeur d'abord
  retours en arrière (backtracking)
  (profondeur de l'arbre = nombre d'unités)

- sa complexité théorique en temps est exponentielle selon le nombre d'unités

•3.1• pourquoi une résolution combinatoire à un problème ?

- on a seulement trouvé un critère d'évaluation d'une combinaison **complète**

- mais on n'a pas (encore) trouvé de critère pour calculer les valeurs des attributs
  au fur et à mesure de l'arrivée des unités

- une résolution combinatoire est en somme une "proto-résolution"

- donc, pour construire une résolution non combinatoire,
  il faut avoir une meilleure connaissance des propriétés des unités traitées
  et mieux exploiter cette connaissance

•3.1• résolution combinatoire de l'analyse syntaxique

le problème de l'analyse syntaxique automatique est traditionnellement **résolu**

sur le modèle de la compilation, analyse d'un code connu exhaustivement :

• la combinatoire des valeurs d'attributs est énumérée "exhaustivement" dans le *lexique*

(catégorie, genre, nombre, personne, ... des graphies des mots)

• la combinatoire des structures est énumérée "exhaustivement" dans la *grammaire*

(structures des phrases et des syntagmes)

• à partir de ces données, le processus combinatoire affecte une valeur

aux attributs des unités traitées : les mots et les syntagmes d'une phrase entrante

• on a seulement trouvé un critère d'évaluation d'une combinaison **complète** :

une combinaison est évaluée de manière booléenne sur la grammaticalité de la phrase

•3.2• résolution | calculatoire de l'analyse syntaxique
| ≡ non combinatoire

• prise en compte explicite de l'**ouverture** de la langue :

une langue n'est pas caractérisable exhaustivement, contrairement à un langage de programmation

→ ressources lexicales partielles, et même minimales

→ pas de grammaire formelle ≡ pas d'inventaire (exhaustif) des structures attendues

et pas de test de grammaticalité

• processus de **calcul** :

- passer des *règles* "conditions actions" une fois sur chaque unité :
  caractères, tokens, syntagmes, propositions, phrases, (paragraphes, ... , textes entiers)
- les *conditions* portent sur les attributs d'unités et sur les relations entre unités
- les *actions* affectent des valeurs aux attributs, établissent des relations,
  et génèrent les unités du niveau supérieur
- traitement en flux à débit constant,
  sans découper **a priori** une unité à traiter dans sa totalité telle que la phrase
- un élément du flux est traité complètement, une fois pour toutes, en couche unique,
  avant de passer à l'élément suivant

•3.2• complexité du processus calculatoire

- **complexité pratique** en temps linéaire, démontrée par chronométrage sur gros corpus

- quelques éléments sur la **complexité théorique** en temps :

  - pour chaque unité, un nombre constant de règles est testé une fois

  - complexité des *conditions* de chaque règle ?

    chaque règle comporte un nombre constant d'évaluations d'expressions booléennes
    sur les attributs et sur les relations entre unités
    (pas de backtrack dans les opérateurs booléens)

  - complexité des *actions* de chaque règle ?

    chaque règle comporte un nombre constant d'affectations de valeur à des attributs,
    de mises en relations d'unités et de générations d'unités supérieures

•3.2• les règles : à la fois déclaratives et impératives

- ressource déclarative :
  des règles dites "déclaratives" sont interprétées par un moteur générique
- des règles "conditions actions" sont ainsi **déclaratives**,
  mais exécutent aussi des actions : elles sont aussi **impératives**
- impératives, car ensemble elles décrivent explicitement
  un PROCESSUS sur des attributs d'unités linguistiques,
  et non pas des STRUCTURES de ces unités
- ainsi, des règles impératives sont formulées et exécutées de manière déclarative
- ces règles constituent un *nouveau langage de programmation impératif*, composé uniquement d'**alternatives** "si condition, alors action" sur l'unité courante
- les **répétitions** sont gérées par le moteur, dans deux dimensions :

```
pour chaque unité
    pour chaque règle
        passer cette règle sur cette unité
```

•3.2• trois phases d'une résolution calculatoire

à partir d'un problème posé, énoncé :

un fils a le quart de l'âge de son père, et leur différence d'âge est de 30 ans

1) écrire des équations, ou des règles = traduire, **coder** perdre le sens

(phase humaine)

`x = y / 4` `y - x = 30`

2) résoudre, **calculer**, déduire = opérer sur des formes sens caché temporairement

(phase calculatoire automatisable // analyseur automatique)

`x = y / 4` et `y - x = 30` `4x - x = 3x = 30`

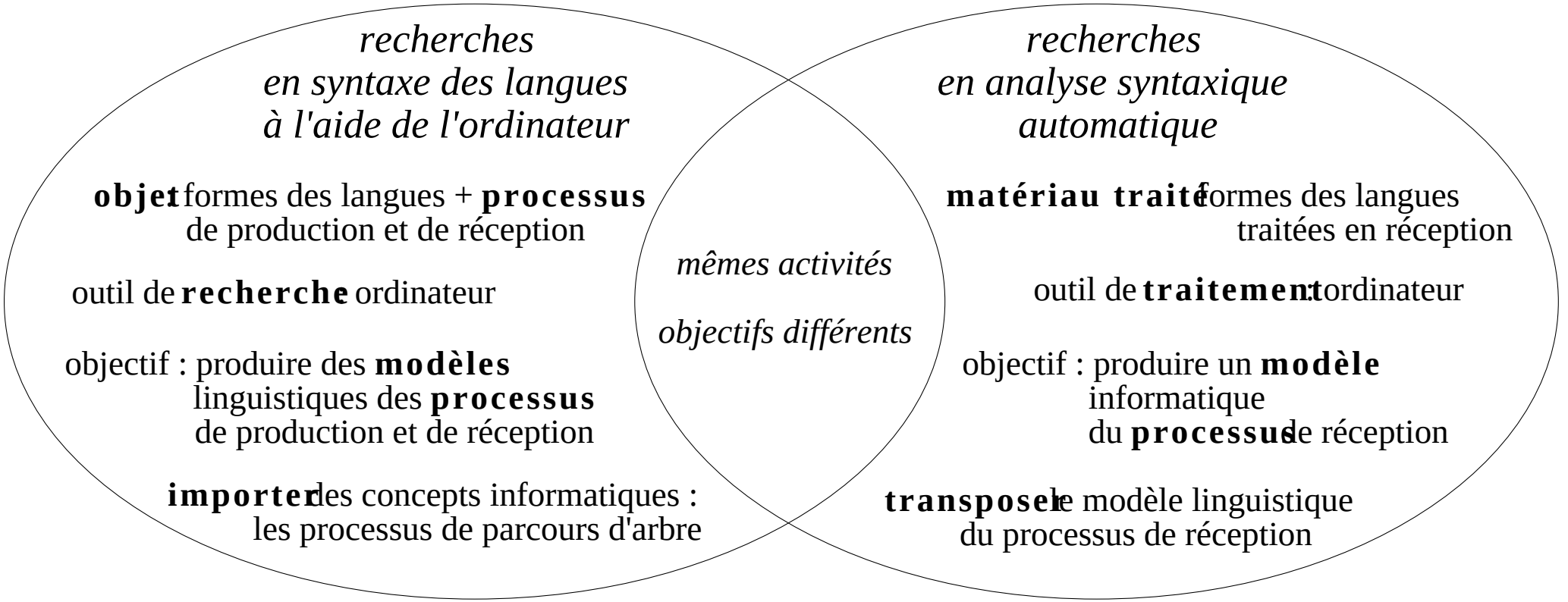
`x = 10` `y = 4x = 40`

3) interpréter, traduire, **décoder** les résultats = redonner un sens

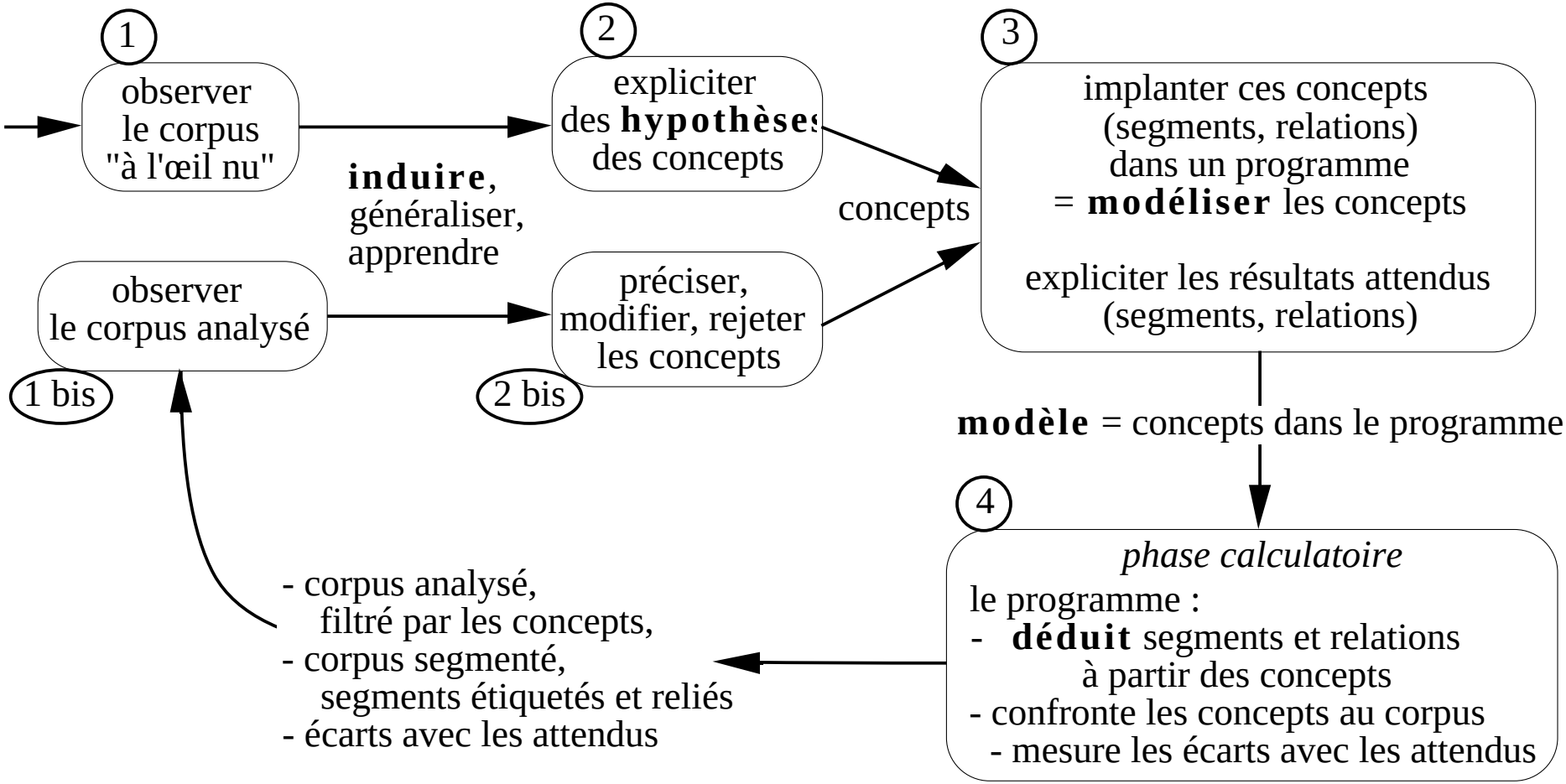
(phase humaine)

le fils a 10 ans, et son père a 40 ans

synthèse des deux axes de recherche



•4• Méthode de recherche



•4• Méthode de recherche

- utilisation explicite de l'**induction** expérimentale

  (inhérente à la cyclicité de la méthode)

- formulation de concepts, d'**hypothèses**,

  et **déduction** automatisée à partir de ces concepts :

  intégration de la démarche hypothético-déductive

- rôle de l'ordinateur comme instrument de déduction,

  observation, confrontation, et expérimentation

- structure itérative, cyclique de la démarche

•5• Validation et valorisation des recherches

•5.1• **validation opératoire** :

première place à l'action d'évaluation GRACE

•5.2• **validation de concepts sur la prosodie** :

Projet de Synthèse Vocale "Kali"

•5.3• **valorisation industrielle** :

Projet de Filtrage de Flux Textuels "Linguix"

•5.1• validation opératoire :

évaluation comparative "GRACE" des étiqueteurs du français

"**tagging**" : processus calculatoire par exploitation du contexte, propriété du matériau

• ressources explicitées : règles de déduction locale sur le contexte,
et non pas les structures attendues

• le contexte ne peut être exploité qu'à l'intérieur des SNR (composés de plusieurs tokens)

• quelques % des tokens (SNR composés d'un token unique) ne peuvent être étiquetés
que par la mise en relation (= par l'analyse syntaxique complète)

• notre analyseur s'est montré le plus proche de l'étiquetage humain (à 94,5%)

• "tagging" : palliatif provisoire à l'aspect combinatoire des analyseurs actuels

• "tagging" : un renouveau de l'analyse syntaxique automatique

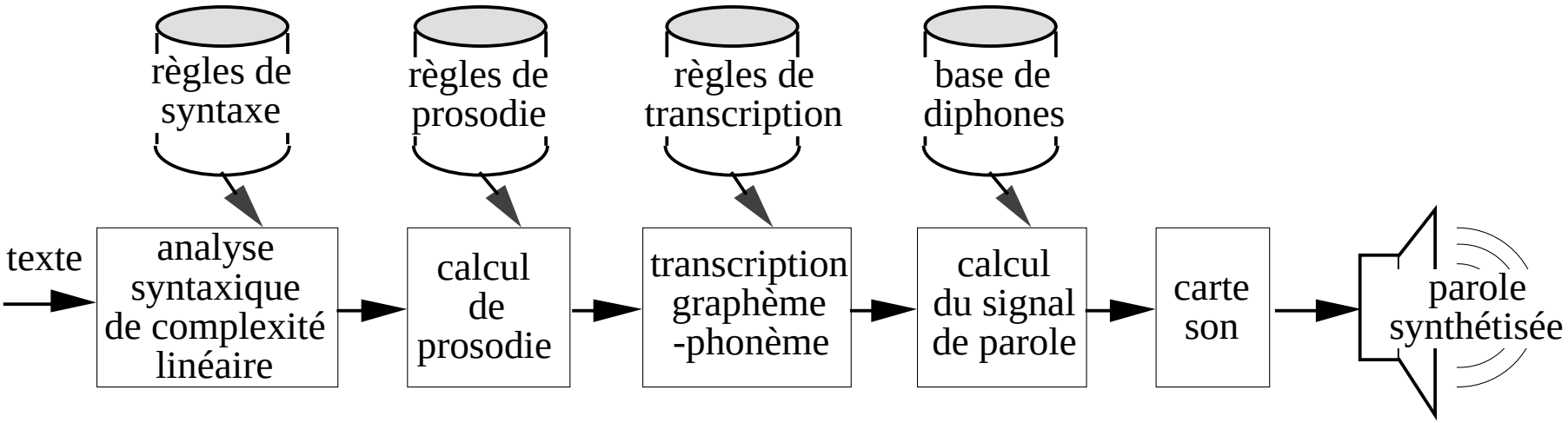
→ les analyseurs calculatoires

→ l'abandon des grammaires formelles

comme outil de modélisation des structures syntaxiques des langues

•5.2• validation de concepts sur la prosodie :

Projet de Synthèse Vocale "Kali" (Electrel, Club MicroSon)



réception - traduction - production

- écrit et oral : deux instances du même matériau linguistique,
  à inclure ensemble dans le champ de la syntaxe
- non pas "quelles relations entre **la** prosodie et **la** syntaxe ?"

  mais créer **un** modèle adéquat aux deux instances, l'écrit et l'oral

hypothèse pour **un** modèle de syntaxe adéquat à l'écrit et à l'oral

• fonction de la prosodie (hauteur, durée, intensité des phonèmes émis par le locuteur)
= permettre à l'auditeur de :

1) segmenter en Groupes Accentuels : 1 seul accent primaire / GA (GA = SNR)

2) reconstruire les relations entre Groupes Accentuels :
- pas de pause entre 2 GA contigus GA reliés
- pause entre 2 GA contigus GA non reliés,

relation sur la pause, de longueur proportionnelle à la durée de la pause
avec la métrique vue précédemment

• groupe prosodique = suite de GA contigus dits sans pause (donc contigus reliés)

• calcul de la prosodie de la synthèse vocale Kali :
- conçu sur ces principes,
- **corroboré** par les tests perceptifs des utilisateurs
sur des textes quelconques : articles de presse, romans, ...

•5.3• **valorisation industrielle des recherches :**

Projet de Filtrage de Flux Textuels "Linguix" (Datops)

• travail collectif du "Groupe Syntaxe" (développement : 2 années-hommes sur 9 mois)

• fonctions réalisées et intégrées :

- diagnostic de langue,
- analyse syntaxique du français et de l'anglais,
- repérage et évaluation des métaphores,
- segmentation des textes en parties différenciées puis reliées

• implémentation de la **triple généricité** conçue par Emmanuel Giguet dans sa thèse :

un **processus unique** paramétrable :
- selon la **fonction** réalisée,
- selon le **niveau** linguistique traité,
- selon la **langue** traitée

• un **moteur générique unique** (écrit en Java) assure l'ensemble de ces fonctions,
avec des ressources déclaratives
- propres à une fonction,
- propres à un couple de 2 niveaux linguistiques contigus,
- propres à une langue

• certaines **ressources** sont **bilingues** : **mêmes règles** pour l'anglais et le français
pour délimiter les propositions et relier les syntagmes dans les propositions

• unité linguistique modélisée et implémentée aussi de manière générique,
en une **classe unique**, indépendamment de son niveau réel :
mot, syntagme, proposition, phrase, paragraphe, section, ... , texte entier

• processus de complexité pratique linéaire en temps
en fonction du nombre d'unités traitées
(pour chaque unité, un nombre constant de règles est testé une fois).

• moteur paramétrable en fonction

- des tâches à réaliser,
- de l'enchaînement de ces tâches,
- des ressources déclaratives nécessaires à chaque tâche,
- de la tâche fournissant le flux de sortie,
- et des répertoires du flux entrant et du flux XML sortant.

• règles déclaratives : toutes du même formalisme "conditions actions"

portant
- sur les valeurs des attributs des unités linguistiques
- et sur les relations entre unités linguistiques

• priorité de la position / morphologie :

la morphologie entre dans le calcul en association avec la position et non seule

• lexique : retour à l'option minimale de ma thèse :

mots grammaticaux + marques morphologiques,

par des règles dans le formalisme commun

• déductions entre 2 niveaux linguistiques contigus dans les 2 sens :

niveau inférieur niveau supérieur niveau supérieur niveau inférieur

•6• Prospective :
orientations des recherches du "Groupe Syntaxe"

• étudier et traiter les formes des langues, du caractère au texte entier

• prendre une langue comme un ensemble ouvert, évolutif,
qu'on ne peut caractériser que partiellement (→ ne pas chercher à tout stocker)

• chercher à expliciter des processus, plutôt que des structures statiques

• travail expérimental sur corpus, à l'aide de l'ordinateur comme outil d'observation, de modélisation et de validation, avec les deux orientations théorique et opératoire

• corpus : textes attestés entiers, langues variées, formes écrite et orale

- langues différentes = instances différentes du même matériau
- formes écrite et orale = deux instances du même matériau,
deux types de formes dont les recherches s'enrichissent mutuellement

• approche multilingue sous l'angle du génie logiciel :
nécessité industrielle, et méthode de production de logiciels indépendants
de la langue traitée, avec des ressources monolingues ou multilingues extérieures

• exploiter des propriétés linguistiques générales
traiter le matériau avec peu de ressources très générales → généricité et robustesse

• approche calculatoire :

- l'ordinateur utilisé comme machine à calculer des résultats à partir de données,
et non pas comme machine à stocker et à choisir parmi des valeurs stockées

- sensibilité à l'algorithmique et à la complexité des algorithmes
pour concevoir des solutions et les évaluer

- traiter en flux, à débit constant : algorithmes de complexité pratique linéaire

●