Théorie de la cognition centrale ACT-R 6

Séminaire « partage des savoirs »

Dominique Longin

25 avril 2012

Point de départ

- Une surprise peut s'avérer non surprenante
- Exemple : *j vient dire bonjour à i*

  i dit à *j* : « Hein !? Que fais-tu là ? Ah oui, c'est vrai que tu es de retour à Toulouse ! »

⇨ les logiques actuelles des états mentaux ne peuvent capturer cette séquence d'états épistémiques (toute information est accessible en permanence au raisonnement, qui est instantané)

⇨ Différence entre croyances explicites et implicites

Introduction

- Théorie de John Anderson :
  - Modélisation de la structure du cerveau à un niveau d'abstraction qui explique comment fonctionne la cognition centrale
  - Cognition centrale ≃ « microprocesseur » de l'être humain
  - Modélisation implémentée dans l'architecture cognitive ACT-R (v. 6)
- Exposé : présentation de la théorie en même temps que ACT-R

Pourquoi modéliser ?

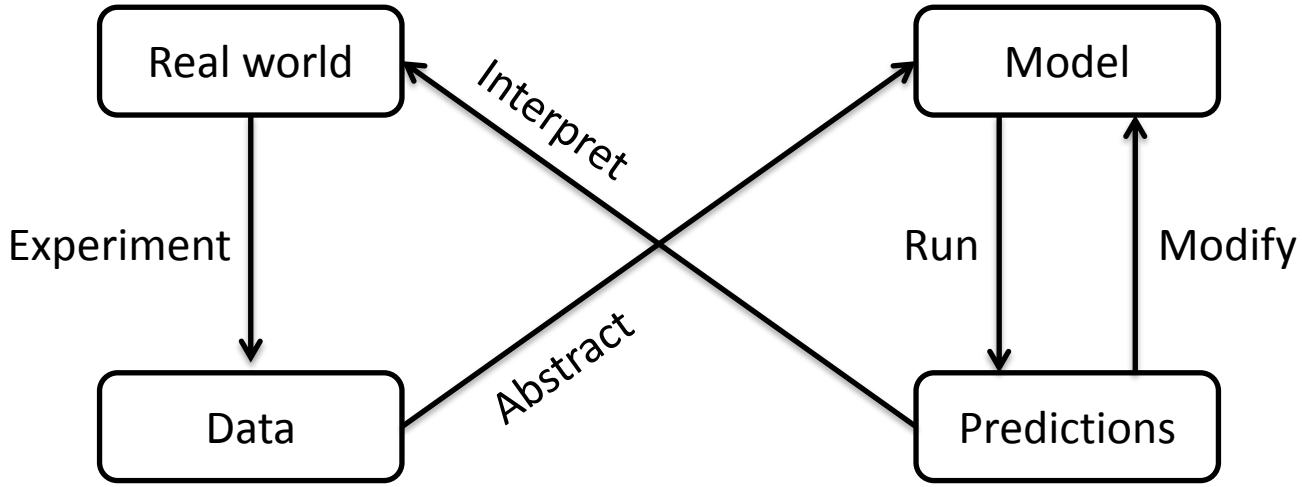


Schéma de Marvin L. Bittinger (2004)

ACT-R en bref

- Architecture cognitive à 2 niveaux :
  - symbolique
    - représentation des connaissances
    - raisonnement
  - Subsymbolique
    - Gestion dynamique de l'activation des connaissances
    - Gestion d'un temps simulé (ou temps cognitif) par opposition au temps réel (temps d'exécution du programme)
- Principe : la cognition fonctionne uniquement à partir de la connaissance suffisamment activée (ou : ayant un niveau d'activation suffisant)

Différences entre ACT-R et un Langage de Programmation Classique (LPC)

- LCP = généralement, langage de haut niveau à destination du microprocesseur bas niveau
- ACT-R = langage bas niveau à destination d'un « processeur » de haut niveau
- Exécution non séquentielle, possiblement après « tirage au sort » des règles s'appliquant
- Programme ACT-R représente la cognition : n'est pas forcément optimal du point de vue de la programmation informatique

Plan

1. Niveau symbolique
   - Représentation de la connaissance
   - Connaissance déclarative
   - Connaissance procédurale
   - Exemple
2. Niveau subsymbolique
   - Activation et mémoire déclarative
   - Utilité et mémoire procédurale
   - Apprentissage de nouvelles productions

Représentation de la connaissance

- 2 types de connaissance :
  - Déclarative (pour les faits, les **chunks***)
  - Procédurale (pour les déductions, les **productions**)
- Ce sont les connaissances à long terme

(*) en **gras** les termes utilisés dans la théorie d'Anderson

Plan

1. Niveau symbolique
   - Représentation des connaissances
   - Connaissance déclarative
   - Connaissance procédurale
   - Exemple
2. Niveau subsymbolique
   - Activation et mémoire déclarative
   - Utilité et mémoire procédurale
   - Apprentissage de nouvelles productions

Connaissance déclarative

- Faits dont nous sommes conscients lorsqu'on raisonne
  - Ex. : Guillaume le Conquérant battit Harold à Hastings le 14 octobre 1066
- Proviennent de :
  - La perception
  - La déduction
- L'unité est le **chunks**
- (Module de) mémoire déclarative = ensemble des connaissances déclaratives

Les chunks

- Composés :
  - D'une catégorie (le **chunk-type**)
  - D'un ensemble d'attributs (**slots**)
- Et en plus, dans ACT-R seulement :
  - D'un nom*
- Exemples (syntaxe ACT-R) :

	Le chat poursuit la souris		**2+3=5**	
name*	Action023		Fact2+3	
chunk-type	isa	chase	isa	addition-fact
slot1/value	agent	cat	adden1	two
slot2/value	object	mouse	adden2	three
slot3/value			sum	five

(*) pour faciliter son référencement, n'est pas considéré comme partie du chunk

Syntaxe ACT-R des chunks*

- Création d'un chunk-type

```
(chunk-type count-order first second)
(chunk-type count-from start end count)
```

- Création d'un chunk de catégorie count-order

```
(add-dm
    (b isa count-order first 1 second 2)
    (c isa count-order first 2 second 3)
    (d isa count-order first 3 second 4)
    …
    (first-goal isa count-from start 2 end 4)
)
```

- Remarques :
  - Slot **isa** commun à tous les chunks, ne peut être modifié
  - Pas nécessaire d'initialiser les slots (valeur `nil` par défaut)

(*) couleurs, gras et soulignements ne font pas partie de la syntaxe

Plan

1. Niveau symbolique
   - Représentation de la connaissance
   - Connaissance déclarative
   - Connaissance procédurale
   - Exemple
2. Niveau subsymbolique
   - Activation et mémoire déclarative
   - Utilité et mémoire procédurale
   - Apprentissage de nouvelles productions

Connaissance procédurale

- Connaissance inconsciente mais servant à notre raisonnement (déduction, savoir-faire)
- Exemple : personne ne peut décrire les règles servant à parler, mais nous parlons.
- Représentation *via* des **productions**
- (Module de) mémoire procédurale =
  ensemble des productions

Les productions

- Représentent un contrôle du comportement
- De la forme *IF* <condition> *THEN* actions :
- Plusieurs productions peuvent s'appliquer (être « sélectionnées ») MAIS
  une seule peut être exécutée (tirée, **fired**)
  ⇨ hypothèse de séquentialité de la cognition interne
- Exemple :

  SI (le but est d'additionner 2 nombres n_1 et n_2 dans une colonne ET $n_1 + n_2 = n_3$)

  ALORS adopter le sous-but d'écrire n_3 dans la colonne
- Une règle utilise des **buffers** pour manipuler les chunks nécessaires

Les buffers

- Partie condition d'une **production** utilise des buffers indiquant un pattern de **chunk** devant être présent dans ces **buffers**
- Buffers = interfaces entre une production et un module donné (module de but, module de récupération d'un chunk, *etc.*) *via* des chunks
- Ils représentent la mémoire à court terme (ou mémoire de travail)
- Contiennent un chunk à un instant donné
- Leur contenu est affecté par les actions des productions

Syntaxe ACT-R des buffers

- Goal, retrieval, *etc.* sont des buffers (de but, récupération d'un chunk depuis la mém. décl., *etc.*)
- Le « = » indique une variable
  (ex. : **=goal** est la variable correspondant au buffer goal)
  - ⇨ le buffeur peut être lu (ds condition d'1 production)
  - ⇨ ou modifié (dans partie action d'une production)
- Le « + » indique (en général) la création d'un chunk dans le buffer (ex. : **+retrieval**)
- Le « - » indique l'effacement du buffer (ex. : **-visual**)

Syntaxe ACT-R des productions

```
(p increment
   =goal>
      isa      count-from
      count    =num1
    - end      =num1
   =retrieval>
      isa      count-order
      first    =num1
      second   =num2
==>
   =goal>
      count    =num2
   +retrieval>
      isa      count-order
      first    =num2
)
```

SI

Le but est de compter à partir de `count` (dont la valeur courante est `=num1`) où `count` ≠ `end`

ET

On a récupéré en mémoire le fait que `=num2` est le successeur de `=num1`

ALORS

Le but est maintenant de compter à partir de `=num2`

ET

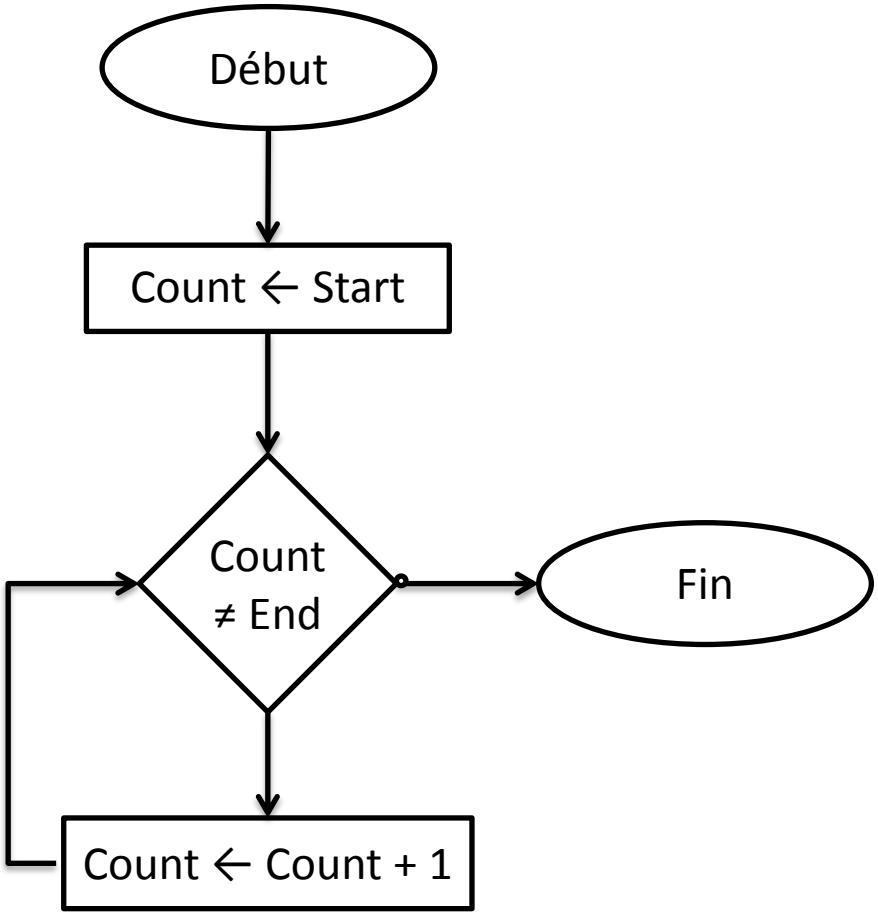
De récupérer en mémoire le successeur de `=num2`

Plan

1. Niveau symbolique
   - Représentation de la connaissance
   - Connaissance déclarative
   - Connaissance procédurale
   - Exemple
2. Niveau subsymbolique
   - Activation et mémoire déclarative
   - Utilité et mémoire procédurale
   - Apprentissage de nouvelles productions

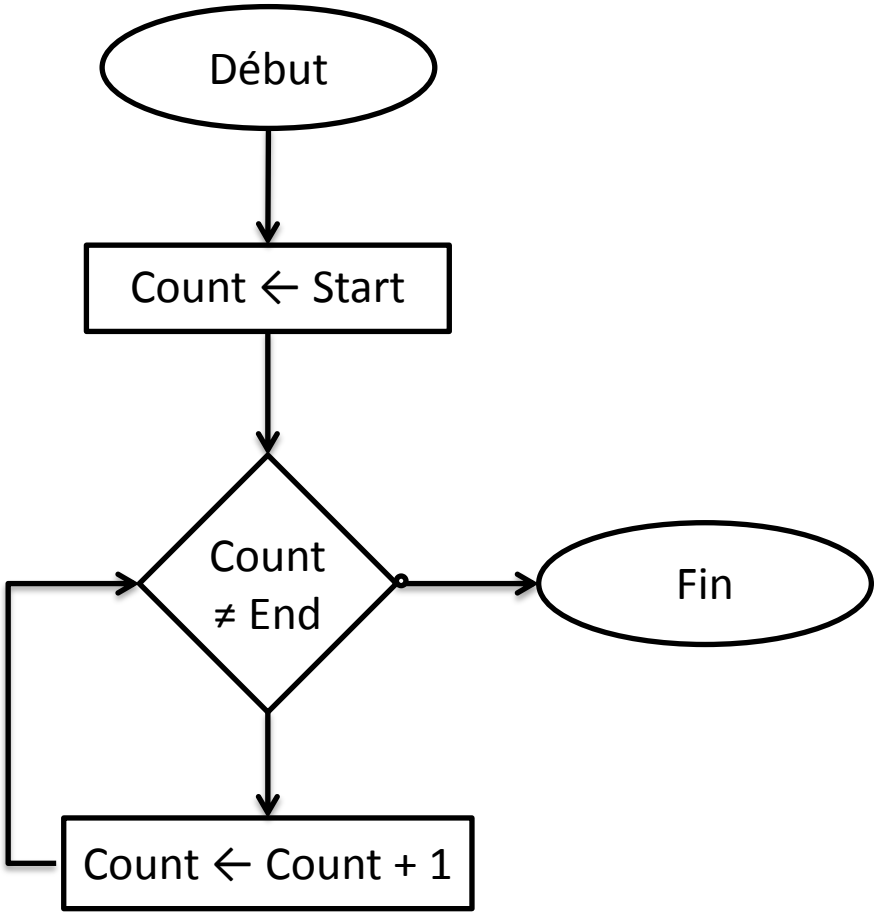
Exemple : compter de Start à End

Algorithme classique

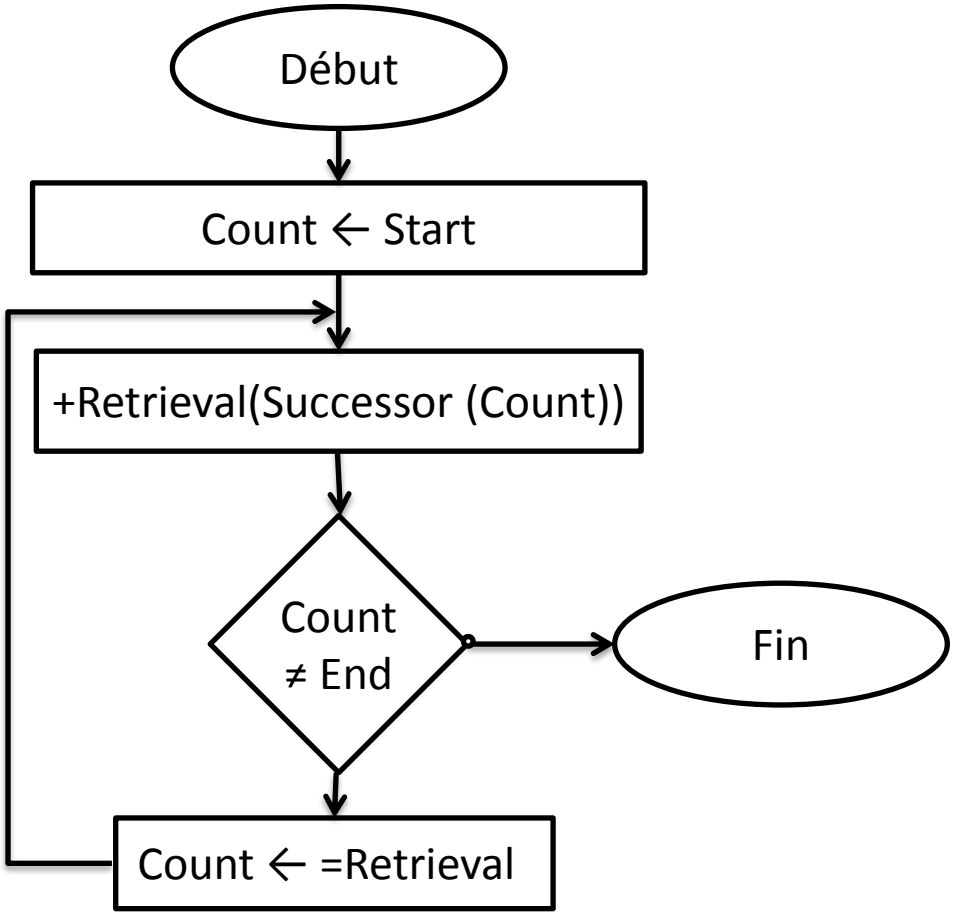


Exemple : compter de Start à End

Algorithme classique



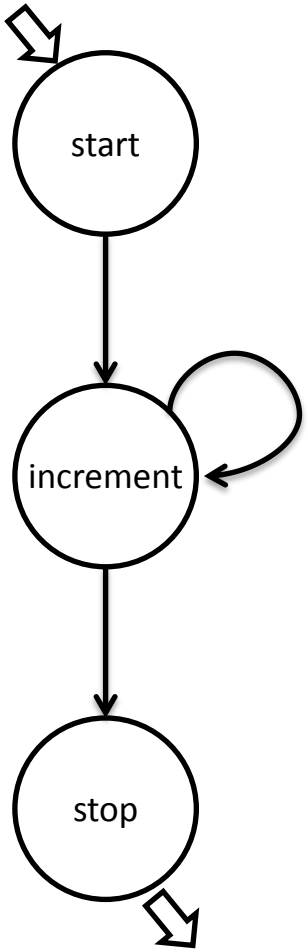
Algorithme cognitif



Programme ACT-R (compter)

```
(clear-all)

(define-model count

(sgp :esc t :lf .05 :trace-detail high)

(chunk-type count-order first second)
(chunk-type count-from start end count)

(add-dm
 (b ISA count-order first 1 second 2)
 (c ISA count-order first 2 second 3)
 (d ISA count-order first 3 second 4)
 (e ISA count-order first 4 second 5)
 (f ISA count-order first 5 second 6)
 (first-goal ISA count-from start 2 end 4)
 )

(p start
   =goal>
      ISA         count-from
      start       =num1
      count       nil
 ==>
   =goal>
      count       =num1
   +retrieval>
      ISA         count-order
      first       =num1
)
```

```
(P increment
   =goal>
      ISA         count-from
      count       =num1
    - end         =num1
   =retrieval>
      ISA         count-order
      first       =num1
      second      =num2
 ==>
   =goal>
      count       =num2
   +retrieval>
      ISA         count-order
      first       =num2
   !output!       (=num1)
)

(P stop
   =goal>
      ISA         count-from
      count       =num
      end         =num
 ==>
   -goal>
   !output!       (=num)
)

(goal-focus first-goal)
)
```



Programme ACT-R (trace)

```
0.000   GOAL                SET-BUFFER-CHUNK GOAL FIRST-
                             GOAL REQUESTED NIL

0.000   PROCEDURAL          CONFLICT-RESOLUTION
0.000   PROCEDURAL          PRODUCTION-SELECTED START
0.000   PROCEDURAL          BUFFER-READ-ACTION GOAL
0.050   PROCEDURAL          PRODUCTION-FIRED START
0.050   PROCEDURAL          MOD-BUFFER-CHUNK GOAL
0.050   PROCEDURAL          MODULE-REQUEST RETRIEVAL
0.050   PROCEDURAL          CLEAR-BUFFER RETRIEVAL
0.050   DECLARATIVE         START-RETRIEVAL
0.050   PROCEDURAL          CONFLICT-RESOLUTION
0.100   DECLARATIVE         RETRIEVED-CHUNK C
0.100   DECLARATIVE         SET-BUFFER-CHUNK RETRIEVAL C
0.100   PROCEDURAL          CONFLICT-RESOLUTION
0.100   PROCEDURAL          PRODUCTION-SELECTED INCREMENT
0.100   PROCEDURAL          BUFFER-READ-ACTION GOAL
0.100   PROCEDURAL          BUFFER-READ-ACTION RETRIEVAL
0.150   PROCEDURAL          PRODUCTION-FIRED INCREMENT
2
0.150   PROCEDURAL          MOD-BUFFER-CHUNK GOAL
0.150   PROCEDURAL          MODULE-REQUEST RETRIEVAL
0.150   PROCEDURAL          CLEAR-BUFFER RETRIEVAL
0.150   DECLARATIVE         START-RETRIEVAL
0.150   PROCEDURAL          CONFLICT-RESOLUTION
0.200   DECLARATIVE         RETRIEVED-CHUNK D
0.200   DECLARATIVE         SET-BUFFER-CHUNK RETRIEVAL D
0.200   PROCEDURAL          CONFLICT-RESOLUTION
0.200   PROCEDURAL          PRODUCTION-SELECTED INCREMENT
0.200   PROCEDURAL          BUFFER-READ-ACTION GOAL
0.200   PROCEDURAL          BUFFER-READ-ACTION RETRIEVAL
0.250   PROCEDURAL          PRODUCTION-FIRED INCREMENT
3
0.250   PROCEDURAL          MOD-BUFFER-CHUNK GOAL
0.250   PROCEDURAL          MODULE-REQUEST RETRIEVAL
0.250   PROCEDURAL          CLEAR-BUFFER RETRIEVAL
0.250   DECLARATIVE         START-RETRIEVAL
0.250   PROCEDURAL          CONFLICT-RESOLUTION
0.250   PROCEDURAL          PRODUCTION-SELECTED STOP
0.250   PROCEDURAL          BUFFER-READ-ACTION GOAL
0.300   DECLARATIVE         RETRIEVED-CHUNK E
0.300   DECLARATIVE         SET-BUFFER-CHUNK RETRIEVAL E
0.300   PROCEDURAL          PRODUCTION-FIRED STOP
4
0.300   PROCEDURAL          CLEAR-BUFFER GOAL
0.300   PROCEDURAL          CONFLICT-RESOLUTION
0.300   ------              Stopped because no events left to process
```

Plan

1. Niveau symbolique
   - Représentation de la connaissance
   - Connaissance déclarative
   - Connaissance procédurale
   - Exemple
2. Niveau subsymbolique
   - Activation et mémoire déclarative
   - Utilité et mémoire procédurale
   - Apprentissage de nouvelles productions

Niveau subsymbolique : pourquoi ?

- Supposons que la requête

```
+retrieval>
   Isa       square
   Integer   4
```

  soit lancée
- Que se passe-t-il si deux chunks sont candidats ? (lequel sera récupéré depuis DM)
- Même question pour les productions : que se passe-t-il si plusieurs productions peuvent être tirées ?

⇨ La résolution des conflits se gère au niveau subsymbolique

Plan

1. Niveau symbolique
   - Représentation de la connaissance
   - Connaissance déclarative
   - Connaissance procédurale
   - Exemple
2. Niveau subsymbolique
   - Activation et mémoire déclarative
   - Utilité et mémoire procédurale
   - Apprentissage de nouvelles productions

Activation et mémoire déclarative (1)

- Valeur numérique associée à tout chunk
- C'est le degré de pertinence d'un chunk à un instant donné par rapport à l'expérience passée et le contexte actuel
- Règle de sélection d'un chunk : celui correspondant à la requête Retrieval (*matching* symbolique) dont l'activation est la plus forte
- Contrainte : l'activation du chunk doit être > à un seuil τ (**retrieval threshold**) sinon erreur (rien n'est récupéré).
- Remarques :
  - Configuration couche subsymbolique *via* fonction (**spg)**
  - Utilisation couche = paramètre **:esc** à la valeur **t**
  - Valeur de τ fixée par paramètre **:rt**

Activation et mémoire déclarative (2)

- Activation A_i d'un chunk $i = B_i + C_i + \varepsilon$
- Où
  - B_i = niveau de base (dernière date et fréquence de **présentation** du chunk *i*)
  - C_i = niveau d'activation contextuelle
  - ε = composante de bruit (bruit permanent + bruit instantané au moment du **+retrieval**)

Présentation d'un chunk

- 2 types de présentation d'un chunk
  - Première entrée en DM
    - Soit de manière explicite (commande **add-dm**)
    - Soit lors de l'effacement d'un buffer (**-goal** par exemple)
  - Nouvelle entrée en DM
    - Lors de l'effacement d'un buffer quand le chunk ainsi présenté à la DM existe déjà
      (attributs et valeurs correspondantes identiques 2 à 2)
      ⇨ processus appelé **chunk merging**

Activation de base d'un chunk

- $B_i = \ln(\sum_{j=1}^{n} t_j^{-d})$
- n = nombre de présentation du chunk i
- t_j = temps depuis la j^{ième} présentation
- d = paramètre de délabrement de l'activation (fixe pour un modèle, **:bll**)

⇨ B_i ↗ avec t_j petit (chunk utilisé il y a très peu de temps) ET n grand (chunk utilisé fréquemment)

Activation contextuelle d'un chunk (1)

- Les chunks dans les buffers de la production courante créent un contexte
- Lors de **+retrieval**, ces chunks vont diffuser de l'activation dans les chunks de la DM qui leur sont liés
- C'est le processus de diffusion de l'activation (**spreading activation**)

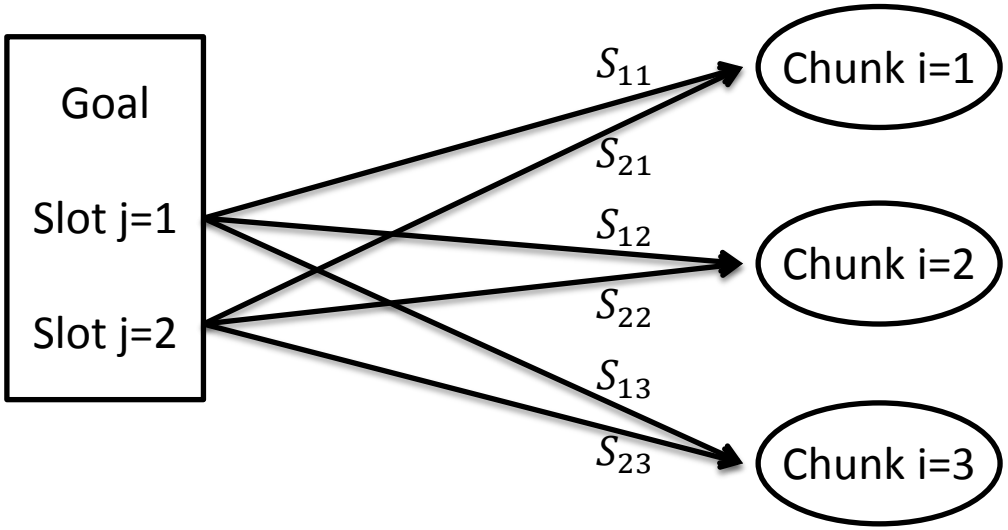
Activation contextuelle d'un chunk (2)

- $C_i = \sum_k \sum_j W_{kj} S_{ji}$
- Où :
  - k appartient à l'ensemble des buffers
  - j appartient à l'ensemble des slots du buffer k
  - W_{kj} est l'activation du slot j parmi le (chunk contenu dans le) buffer k
    (en général, si W_k est l'activation du chunk contenu dans le buffer k alors W_{kj} = W_k/nb_chunks_in(k))
  - S_{ji} est la force qui lie le chunk contenu dans le slot j au chunk i dont on souhaite calculer l'activation
    - Si $j \neq i$ ET $j \notin i$ alors $S_{ji} = 0$
    - Sinon $S_{ji} = S - \ln(fan_j)$
      (fan_j= nombre de chunks en DM dont j est un slot + 1 (pour j))

Exemple d'activation contextuelle (1)

- On suppose que seul le buffer de goal diffuse de l'activation ($k = 1$)
- De plus, on suppose que le chunk contenu dans le buffer de goal a une activation A_{goal}
- Donc :
  - Activation W_1 du slot 1 du goal est $A_{goal}/2$
  - Et $W_2 = W_1$
- On suppose enfin que 3 chunks de DM sont liés au buffer de goal : le slot 1 avec les 2 premier, et le slot 2 avec les deux derniers

Exemple d'activation contextuelle (2)



- Finalement, pour ces 3 chunks qui matchent avec le goal :
  - $A_1 = B_1 + W_1 S_{11} + W_2 S_{21} + \varepsilon$
  - $A_2 = B_2 + W_1 S_{12} + W_2 S_{22} + \varepsilon$
  - $A_1 = B_3 + W_1 S_{13} + W_2 S_{23} + \varepsilon$
- Les autres chunks x ayant un slot en commun avec le goal reçoivent également de l'activation (S_{jx} est non nul)

Composante bruit de l'activation d'un chunk

- Sert à introduire une composante aléatoire dans le degré d'activation d'un chunk
- 2 types de bruit :
  - Bruit permanent (**:pas**)
  - Bruit instantané (**:ans**), calculé à chaque **+retrieval**
- En principe, le bruit permanent est désactivé, seul le bruit instantané est utilisé

Temps (simulé) de récupération RT d'un chunk

- Si un chunk *i* est sélectionné $RT_i = Fe^{-A_i}$
- Sinon (aucun chunk sélectionné) $RT_i = Fe^{-\tau}$
- Où :
  - F est le facteur de latence (**:lf**)
  - A_i est l'activation du chunk *i*
  - τ *le seuil de récupération des chunk*

Exemple

- Tâche : un individu visualise une liste de 20 paires <nom,chiffre>
- Résultats expérimentaux :
- ACT-R, *via* ses couches symbolique
  et subsymbolique permet de faire une simulation générant ces résultats

Trial	Accuracy	Latency
1	0,000	0,000
2	0,526	2,156
3	0,667	1,967
4	0,798	1,762
5	0,887	1,680
6	0,924	1,552
7	0,958	1,467
8	0,954	1,402

Plan

1. Niveau symbolique
   - Représentation de la connaissance
   - Connaissance déclarative
   - Connaissance procédurale
   - Exemple
2. Niveau subsymbolique
   - Activation et mémoire déclarative
   - Utilité et mémoire procédurale
   - Apprentissage de nouvelles productions

Utilité et mémoire procédurale

- Chaque production est associée à une utilité brute
- Indique quelle production sera « tirée » parmi celles sélectionnées (ce sera la + utile)
- Remarque :
  - Configuration niveau subsymbolique des production *via* commande **(spp)**
  - Utilité brute (via **:u**) Valeur par défaut : 0
  - Comme l'activation d'un chunk, l'utilité est « bruitée »
  - Utilité réelle = utilité brute ± bruit

Utilité de l'utilité

- Favoriser une règle par rapport à une autre
- Exemple :
  - données expérimentales indiquent qu'un raisonnement est fait majoritairement par rapport à un autre
  - On privilégie la ou les règles du premier raisonnement par une utilité plus fort
  - Il en découle que cette règle sera plus souvent utilisée, mais pas nécessairement tout le temps (facteur bruit)

Utilité et apprentissage

- L'utilité peut devenir une valeur dynamique, variant en fonction d'une récompense
- $U_i(n) = U_i(n\text{-}1) + k[R_i(n) - U_i(n\text{-}1)]$ où
  - $U_i(n)$: **utilité** de la règle *i* après *n* applications
  - $R_i(n)$: **récompense reçue** par la règle *i* après *n* applications
  - k : **taux d'apprentissage** (modifiable via **:alpha**)
- Rem. :
  - récompense reçue = **récompense attribuée** – temps écoulé depuis que cette règle a tiré
  - Toutes les règles utilisées depuis la dernière attribution d'une récompense sont récompensées (moins fortement pour les plus anciennement utilisées)

Plan

1. Niveau symbolique
   - Représentation de la connaissance
   - Connaissance déclarative
   - Connaissance procédurale
   - Exemple
2. Niveau subsymbolique
   - Activation et mémoire déclarative
   - Utilité et mémoire procédurale
   - Apprentissage de nouvelles productions

Apprentissage de production

- Deux règles qui s'appliquent successivement peuvent être réduite en une nouvelle règle (**production compilation**)
- Cela induit une réduction du temps de calcul
- Permet de simuler typiquement des tâches répétitives où l'humain réagit de + en + rapidement
- Exemple : pour chaque nombre présenté à l'écran, appuyer sur la touche 'p' ou 'i' selon que le nombre est pair ou impair
  (⇨ apprentissage au niveau de DM et de PM)

Et plus loin…

- Gestion du buffer visuel (tâche de lecture à l'écran)
- Gestion du buffer moteur (simulation du déplacement de la souris ou de l'appui sur une touche)
- Gestion du buffer de vocalisation
- Gestion du buffer de l'audition

Conclusion

- Dichotomie connaissance déclarative vs procédurale
- Notion d'activation liée aux connaissances (influence sur les connaissances disponibles pour le raisonnement)
- Notion d'utilité bruitée pour les règles (influence sur les règles utilisées)
- L'utilité et l'activation sont dynamiques
- Des productions peuvent être crées spontanément par apprentissage
- Gère le temps simulé de la cognition (tirer une production ou exécuter une action prend du temps)
- Gère des patterns d'erreur (on peut ne pas récupérer l'information attendue)