Programmation Déclarative :

SNARK 2

(Symbolic Normalized Acquisition and Representation of Knowledge - Jean-Louis Laurière)

Bernard ESPINASSE

Professeur à l'Université d'Aix-Marseille

2004

- **Représentation des connaissances factuelles dans SNARK**
- **Représentation des connaissances opératoires dans SNARK**
- **Interprétation des connaissances dans SNARK**
- **La méta-connaissance dans SNARK**
- **Les règles démons**

SNARK

- **Système général** permettant la réalisation de **systèmes experts**
  - développé par le Prof. **J.L. Laurière** et son équipe à partir de 81 (Univ. PARIS VI et ELF aquitaine)
  - **logique 0,2; gestion des hypothèses**,... (M. Vialatte 84)
  - **commercialisé** par société **SINAPSE** Paris (**OpenSnark**)
- SNARK se compose:
  - **langage de représentation de la connaissance** (logique des prédicats d'ordre 1)
  - **moteur d'inférence** ou **interpréteur**
- SNARK peut **justifier** ses résultats
- SNARK permet de travailler en **logique des proposition (ordre 0) et des prédicats (ordre 1 et 2)**
- SNARK fonctionne en **chaînage avant** (proto chaînage arrière)
- SNARK permet la **non-monotonie**
- SNARK permet l'usage de **régime par tentatives** (gestion des hypothèses-Vialatte 84)
- SNARK permet l'usage de **méta-connaissance (méta-règles)**

Applications développées avec SNARK

- **archéologie** (Lagrange et Renaud 83-84)
  - aide à la modélisation de raisonnement; l'analyse du discours en archéologie: interprétation iconographique d'une stèle d'Anatolie du 13°siècle
- **bridge**(CHELEM: Popescu 83-85, >302 règles)
  - permet d'analyser la donne (compte des levées sures, des arrêts,...) et de trouver la ligne de jeu du déclarant
- **physique qualitative** (Dormoy 85)
  - conduite de centrale nucléaire et la détection d'incidents
- **plan de relevé EDF** (Vialatte 84, 44 règles)
  - regroupement de tournées contiguës dans des secteurs dont la taille est limitée par le nombre d'abonnés desservis
- **interpréteur Prolog** en SNARK (Laurière 82)
- **formation de textes** en français
- **modélisation de trafic automobile**
- **enseignement assisté** en physique
- ....

Représentation des connaissances dans SNARK

Les connaissances factuelles:

connaissances que le système a acquis sur le problème qu'il est en train de résoudre (base de faits)

Les connaissances opératoires:

expertise, le savoir-faire exprimé de façon déclarative sur le domaine abordé, notamment quelles actions accomplir dans telle ou telle situation (base de règles)

Représentation des connaissances dans SNARK:

logique des prédicats du premier ordre

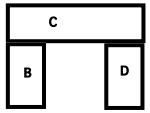
l'emploi de variables :

variables = entités quantifiées universellement et existentiellement et substituables par des objets de la base de faits.

Connaissances factuelles dans SNARK

un fait = triplet (objet, attribut, valeur)

soit à représenter l'arche A composée de trois briques B,C,D suivante:



dans SNARK:

arche	element	B
arche	element	C
arche	element	D
C	nature	brique
C	position	horizontale
C	posé_sur	B
C	posé_sur	D
B	nature	brique
B	position	verticale
D	nature	brique
D	non_accole_a	B
B	a_gauche_de	D

un fait en SNARK est une relation binaire

Connaissances factuelles dans SNARK

Ex1 : "Jean est plus grand que Paul"

dans SNARK:

Jean	taille	L1
Paul	taille	L2
L1	supérieur	L2

Ex2 :
"soit un ensemble A de cardinal 0"
"soit a élément de A; b élément de B"
"soit f une injection de A dans B telle que f(a)=b"

dans SNARK:

f	nature	application
f	propriété	injective
f	ens_départ	A
f	ens_arrivé	B
A	cardinal	10
A	élément	a
B	élément	b
a	f	b

Connaissances factuelles dans SNARK

" Jean donne un livre à Paul"

comporte trois (3) attributs:

- le donneur: <<Jean>>,
- le don: <<livre>>,
- le receveur: <<Paul>>

dans SNARK:

$d	Donneur	Jean
$d	Don	livre
$d	Receveur	Paul

("**$d**" est un objet appelé aussi : **quark**)

ainsi dans SNARK un objet :

- **peut ne pas posséder d'existence propre,**
- **être qu'un intermédiaire pour représenter la connaissance.**
- **est défini par l'ensemble des relations binaires où il intervient,**

Connaissances factuelles dans SNARK

- de façon générale, dans SNARK les faits sont des triplets :

(objet1, objet2, objet3)

- dans lesquels:
  - **objet1 désigne une entité ou un individu du domaine étudié**
  - **objet2 désigne une relation**
  - **objet3 désigne une valeur**

- mais le même objet peut être:
  - une **entité** pour un fait considéré
  - une **valeur** pour un autre fait
  - une **relation** pour un troisième fait

Relations binaires et prédicats logiques (1)

• tout triplet est une relation binaire

• les relation binaires jouent le rôle de prédicats en logique

prédicats binaires

• utilisé tels quel,

le prédicat: **supérieur (a,b)**

devient:

```
a          superieur    b
```

prédicats unaires:

• doivent être transformés en prédicats binaires:

le prédicat: **homme (socrate)** devient:

```
socrate    homme        vrai
```

ou

```
socrate    est_un       homme
```


Relations binaires et prédicats logiques (2)

prédicats n-aires:

• doivent être **décomposés** en de **nouveaux prédicats binaires** portant tous sur une nouvelle même entité d'ancrage:

le prédicat: **R (n,q,t,cl)**

(une commande caractérisée par son numéro n; qté q; date t; client cl)

devient:

```
c          numéro       n
c          qte          q
c          date         t
c          client       cl
```

• cette décomposition est **toujours possible** :

- toute connaissance exprimée d'une façon générale sous la forme d'une relation de rang supérieur à deux, soit projetée en n relations binaires (on démontre que cette décomposition est toujours possible)
- représentation très fine
- lourdeur dans l'écriture

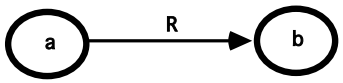
SNARK et les réseaux sémantiques

• on peut réduire un graphe à un ensemble de relations binaires, ce qui permet de traduire un réseau sémantique en SNARK et vice-versa

• chaque nœud du réseau étant alors un objet de SNARK

• une relation entre deux objet ou entre un objet et une valeur est représentée par un arc

le triplet (a, R , b) caractérisant un fait peut être décrit par:



richesse supplémentaire dans SNARK :

• une relation entre 2 objets peut aussi être considérée comme un 3° objet possédant ses propres propriétés:

```
f          nature       application
a          f             b
```


Syntaxe de représentation des connaissances factuelles

forme Naur-Backus

<fait> :: = <mot> <mot> <mot>

avec

<mot> :: = <caractère> (<caractère>)

• **mot** = suite de caractères non blanc (<50)

• caractère blanc = **séparateur**

• le mot peut être un nombre entier ou réel; la partie décimale d'un réel a une longueur maxi de 3

• les **mots-clés connus** du système ne peuvent être utilisés dans les faits

• les faits traduisent des **égalités**

• l'**ordre des faits** dans la base n'a **aucune importance**

Exemple de représentation de connaissances factuelles

Un problème de logique inspiré de Lewis Carroll:

"Alain, Eric, Patrick, Daniel et Jean-Marc vont cueillir des fleurs de montagne. Ils choisissent la gentiane bavaroise, l'arnica, le rhododendron, l'edelweiss et le chardon bleu et ils décident que chacun ne rapportera qu'une espèce de fleur.

Celui qui cueille l'arnica et celui qui cueille le rhododendron conseillent à Alain de mettre ses fleurs dans un sac en plastique, car elles fanent très vite.

Jean-Marc veut garder ses fleurs fraîches, tandis que le garçon qui cherche l'edelweiss et Eric ont l'intention de faire sécher les leurs.

Jean-Marc et celui qui cueille le rhododendron craignent d'avoir des difficultés, car les plantes qu'ils cherchent commencent à défleurir. Au contraire, le chardon bleu commence tout juste à se dorer.

Alain et Daniel recommandent à celui qui cherche l'edelweiss et à celui qui cherche le chardon bleu de ne pas trop cueillir de fleurs car ce sont des espèces rares.

Quelle fleur a choisie chacun de ces enfants ?"

Connaissances contenues dans l'énoncé

• **6 propriétés/relations :**

CUEILLE: cueille (i) = (j) indique que le garçon i cueille la fleur j.

NONCUEILLE: noncueille (i) donne l'ensemble des fleurs que i ne peut cueillir.

IMCUEILLE: imcueille (i) donne l'ensemble des fleurs que i peut encore cueillir.

CARDIMCUEILLE: cardimcueille (i) donne le nombre de fleurs que i peut encore cueillir.

NOM: nom (i) donne le nom alphanumérique que possède le quark i. Rappelons que cette propriété est déjà connue du système contrairement aux autres qui prennent leur signification par l'ensemble des relations où elles interviennent dans la base de faits.

NATURE: nature (i) indique la nature de i soit: garçon. Cette propriété est utile afin d'éviter des instanciations qui n'auraient aucun sens comme par exemple, qu'une fleur puisse cueillir une autre fleur.

Connaissances factuelles

• la phrase de l'énoncé suivante:

"Eric est un garçon, il peut cueillir toutes les fleurs (soit 4 fleurs) excepté l'Edelweiss,"

• sera représentée dans le langage SNARK par:

```
ERIC      NATURE          GARCON;
ERIC      IMCUEILLE       GENTIANE;
ERIC      IMCUEILLE       ARNICA;
ERIC      IMCUEILLE       RHODODENDRON;
ERIC      IMCUEILLE       CHARDON-BLEU;
ERIC      CARD-IMCUEILLE  4.0;
ERIC      NON-CUEILLE     EDELWEISS;
```


Base de faits de l'exemple

```
ALAIN      NATURE          GARCON;
ALAIN      IMCUEILLE       GENTIANE;
ALAIN      CARD-IMCUEILLE  1.0;
ALAIN      NON-CUEILLE     ARNICA;
ALAIN      NON-CUEILLE     RHODODENDRON;
ALAIN      NON-CUEILLE     EDELWEISS;
ALAIN      NON-CUEILLE     CHARDON-BLEU;

ERIC       NATURE          GARCON;
ERIC       IMCUEILLE       GENTIANE;
ERIC       IMCUEILLE       ARNICA;
ERIC       IMCUEILLE       RHODODENDRON;
ERIC       IMCUEILLE       CHARDON-BLEU;
ERIC       CARD-IMCUEILLE  4.0;
ERIC       NON-CUEILLE     EDELWEISS;

PATRICK    NATURE          GARCON;
PATRICK    IMCUEILLE       GENTIANE;
PATRICK    IMCUEILLE       ARNICA;
PATRICK    IMCUEILLE       RHODODENDRON;
PATRICK    IMCUEILLE       EDELWEISS;
PATRICK    IMCUEILLE       CHARDON-BLEU;
PATRICK    CARD-IMCUEILLE  5;
```

```
DANIEL     NATURE          GARCON;
DANIEL     IMCUEILLE       GENTIANE;
DANIEL     IMCUEILLE       ARNICA;
DANIEL     IMCUEILLE
           RHODODENDRON;
DANIEL     CARD-IMCUEILLE  3;
DANIEL     NON-CUEILLE     EDELWEISS;
DANIEL     NON-CUEILLE     CHARDON-
BLEU;

JEAN-MARC  NATURE          GARCON;
JEAN-MARC  IMCUEILLE       GENTIANE;
JEAN-MARC  IMCUEILLE       ARNICA;
JEAN-MARC  CARD-IMCUEILLE  2;
JEAN-MARC  NON-CUEILLE
           RHODODENDRON;
JEAN-MARC  NON-CUEILLE     EDELWEISS;
JEAN-MARC  NON-CUEILLE     CHARDON-
BLEU;
```


Représentation des connaissances opératoires dans SNARK

• Règle de production:

```
<règle>::= REGLE : <nom>
           SI  <antécédent>
           [<antécédent>]
           ALORS <conséquent>

           FR
```

  • **REGLE, SI, ALORS, FR** sont des **mots-clés** prédéfinis du langage SNARK

• les **antécédents** (ou prémisses) et les **conséquents**:

  • adoptent une **forme relationnelle binaire** comme les faits

  • **peuvent contenir des variables** qui seront instanciées par des objets de la base de faits

  • ces variables sont appelées des **DJINNs**

Les Djinns

```
REGLE :  0001
 SI NON-CUEILLE (I) = (J)
     IMCUEILLE   (I) = (J)
 ALORS
     TUERFAIT IMCUEILLE (I)  (J)
     CARD-IMCUEILLE (I) <-- - CARD-IMCUEILLE (I)  1
 FR
```

• I, J sont des **djinns** = **variables** de la **base de règles**,

• les djinns seront **instanciés** par des objets de la base de fait

• chaque djinn **est local à la règle** où il est utilisé

• obligatoirement **entouré de parenthèses** (non confondu avec une valeur alphanumérique)

• **2 djinns différents** dans la partie prémisse d'une même règle entraînera l'instanciation avec **2 objets différents**

• pour avoir **2 djinns identiques**, utiliser le **même identificateur**

Déclenchement d'une règle

```
REGLE :  0001
 SI NON-CUEILLE (I) = (J)
     IMCUEILLE   (I) = (J)
 ALORS
     TUERFAIT IMCUEILLE (I)  (J)
     CARD-IMCUEILLE (I) <-- - CARD-IMCUEILLE (I)  1
 FR
```

Si l'ensemble des relations de la partie prémisse d'une règle peut être mis en correspondance avec les relations binaires associées de la base de faits:

  • **instanciation des prémisses** : substitution des djinns des prémisses par des objets de la base de fait

ALORS la règle sera déclenchée:

  • **instanciation des conséquents** : substitution des djinns des conséquents par des objets de la base de fait

  • **actions de modification ou création d'objet dans la base de faits**

  • **actions sur la base de règles**

Logique d'ordre 1 (prédicats ordre1)

• utilisation de **djinns**

• **djinn** = variable dans les prédicats quantifiée universellement et existanciellement

• substituables par des **objets-entites** ou **objets-valeur** de la base de faits

```
SI     homme(x)  =  vrai
ALORS  mortel(x)  <-- vrai
```

si on a déjà `Socrate  homme  vrai`

on en déduit `Socrate  mortel  vrai`

Logique d'ordre 0 (log. propositions)

• utilisation de **propositions (pas de djinns)**

• proposition = entité parfaitement **définies**

```
SI     evolution    niv_pression =  négative
ALORS  etat         chaufferette <- en_service
       etat         aspirateur   <-- fermé
```

sera déclenché si on a déjà

```
niv_pression    evolution    négative
```


Logique d'ordre 2 (prédicats ordre2):

• utilisation de **djinns**

• **djinns** = variables prédicats **quantifiées universellement** et **existanciellement**

• substituables par des **objets-relation** de la base de faits :

```
SI     propriete  (R)  =  symétrique
       (R)        (x)  =    (y)
ALORS  (R)        (y)  <=    (x)
```

par ex déclenché par :

```
rel    propriété    symétrique
obj1   rel          obj2
```


Les antécédents ou prémisses (1)

• expriment les **conditions d'applicabilité de la règle** :

```
<antécédent >::=   <antécédent positif>/
                   <antécédent négatif>/
                   <antécédent condition>
```

• **antécédent positif**
  • les djinns qu'il contient sont instanciés par des objets de la base de faits
  • ces instanciations sont propagées dans les autres antécédents de la règle

• **antécédent condition**

```
(x)    >     5
(x)    <>    pression
```

  • djinns x et y déjà présents dans d'autres antécédents (antécédents positifs)
  • les antécédents positifs examinés la condition est évaluée

• **antécédent négatif**

```
NON (etat  (x)  =  en_panne)
```

  • satisfait s'il n'est pas possible de satisfaire:

```
etat    (x)    =  en_panne
```

  • c.a.d. : il n'existe pas d'objet OBJ / OBJ etat en_panne

Antécédents positifs

```
<antécédent positif> :: =
        <objet><arg1><opérateur><arg2>
        <objet>:: = <mot> / [<mot>]
        <arg1> :: = <objet> /  <objet> <arg1>
        <opérateur> :: =   = / <> / < / > / <= / >=
        <arg2> :: = <arg1> / CONNU
```

exemples:

```
état      circuit  =   en_panne
valeur    (x)      >   9
(p)       (x)      <=  (y)
```

- **remarques:**
  - les opérateurs <, >,<=,>= ne sont associés qu'à des objets numériques
  - les opérateurs = et <> à des objets quelconques

Antécédents conditions

```
<antécédent condition> :: =<objet><opérateur><objet>
<objet> :: = <mot> / [<mot>]
<opérateur> :: =   = / <> / < / > / <= / >=
```

Exemples:

```
(x)       >       35
(x)       <>      (y)
2.5       >=      (valeur)
```

Antécédents négatifs

```
<antécédent négatif> ::= NON(<objet><objet><opérateur><objet>
      <objet>:: = <mot> / [<mot>]
      <opérateur> :: =   = / <> / < / > / <= / >=
```

Exemples:

```
NON  (état circuit =   en_panne)
NON  (valeur  (x)  >   9)
NON  (position (x) =   a_droite)
```


Les conséquents

- Non restreint aux clauses de HORN, SNARK accepte **plusieurs conséquents** pour une **même règle**
- Ce sont des **actions** qui seront effectuées si la règle est déclenchée et qui portent sur les connaissances **factuelles** et **opératoires courantes**
- **Ordre d'écriture** des conséquents pris en considération lors de leurs exécutions

```
<conséquent> :: = <option d'affectation>/
                  <création>/
                  <suppression>/
                  <retour en arrière>/
                  <impression>/
                  <autodestruction>/
                  <arrêt>/
```

- **Actions sur les connaissances factuelles :**
  - Ajout ou modification de connaissance
  - Suppression de connaissance
  - Retour en arrière (back-track)
- **Actions sur les connaissances opératoires :**
  - Désactivation d'une règle (auto-destruction)
  - Arrêt de l'interpréteur
  - Regroupement de règles
- **Actions sur l'environnement :**
  - Écriture à l'écran....

Les conséquents: Actions sur les connaissances factuelles

Ajout ou modification de connaissance

- **2 affectations :**

<== : opérateur ajoutant un nouveau triplet à la base de faits

exemple : `élément (A) <==  (i)`

<-- : opérateur ajoutant un nouveau triplet à la base de faits en supprimant simultanément toute autre triplet de valeur différente

Exemple :

```
couleur (A) <--   (c)
couleur (A) <--   bleu
valeur (x) <-- +1  valeur  (x)
valeur (x) <-- LOG ABS + 1*2  EXP  *3(y)
```

- **création d'objet-entité**

```
<création> :: = CREER (<mot>)
```

- permet l'ajout d'un nouvel objet-entité dans la base de faits
- le djinn indiqué ne doit pas se retrouver dans la règle en amont de sa création

Les conséquents: Actions sur les connaissances factuelles

Suppression de connaissance

```
<suppression> ::=
TUER <objet>/
TUERFAITS <objet-rel> <objet>/
TUERFAIT <objet> <objet-rel> <objet>
<objet>::= <mot> / [<mot>]
```

- **TUER** supprime de la base de faits tous les faits mentionnant le quark instanciant le djinn

```
TUER   (d)
```

- **TUERFAITS** supprime de la base de faits tous les faits correspondant à l'objet-relation désigné pour un objet donné

```
TUERFAITS   noncueille    (J)
```

- **TUERFAIT** supprime simplement un triplet donné

```
TUERFAIT   inconnue    (pb)     D2
```


Les conséquents: Actions sur les connaissances factuelles

Opération de retour en arrière (back-track)

```
<retour en arrière > ::=
REVENIR-SUR <objet> <objet> <objet>
<objet>::= <mot> / [<mot>]
```

- **REVENIR-SUR** à pour effet de remplacer les informations de la base de faits dans l'état où elles seraient si le fait associé <objet> <objet> <objet> n'avait pas été inféré

exemple :

```
SI        etat  (pb) =  bloqué
          dernière_hyp  (pb) =  (x)
          valeur_choisie  (x) =    (v)
ALORS
          REVENIR-SUR valeur_choisie (x)(v)
          nouveau_choix (x)   <-- a_faire
```

- le fait précisé et tous les faits auxquels il a conduit sont effacés de la base de faits
- le djinn précisé doit être présent en prémisse

Les conséquents: Actions sur les connaissances opératoires

Désactivation d'une règle (auto-destruction)

```
<autodestruction> ::= HARAKIRI
```

- après avoir complètement terminé l'évaluation de la règle, la règle associée est définitivement désactivée

ex:

```
SI        etat  (pb)  = debut
          inconnue (pb) = (i)
ALORS
          valeur_initiale (i) <-- 0
          HARAKIRI
```

Arrêt de l'interpréteur

```
<arrêt> ::= STOP
```

- l'utilisateur arrête l'interprétation après le déclenchement d'une règle

ex:

```
SI        nb_solution_trouvée (pb)  = 5
ALORS
          STOP
```


Les conséquents: Actions sur les connaissances opératoires

Regroupement de règles

Les étapes :

- permet de regrouper les règles afin de forcer l'interpréteur à les traiter dans un ordre séquentiel préétabli (un traitement procédural dans un traitement déclaratif) :

```
FR FIN ETAPE
```

à la fin du groupe de règles.

Quantifications universelle et existentielle

- Les quantificateurs sont implicites,
- Implicites à la structure de la représentation des connaissances opératoires.

Quantificateur universel

quelque soit x, homme(x) => mortel(x)

en SNARK :

```
SI          homme(x) = vrai
ALORS       mortel(x) = vrai
```

Quantification existentielle

quel que soit A,B, (il existe e) tel que e élément de A et e non-élément de B , alors A non-inclus B

en SNARK :

```
SI          élément (A) = e
            non-élément(B) = e
ALORS       non-inclus(A) <= (B)
```


Conjonctions dans les prémisses et les conséquents

A1 et A2 et A3 => B1 et B2 et B3

en SNARK :

```
SI          A1 = vrai
            A2 = vrai
            A3 = vrai
ALORS       B1= vrai
            B2= vrai
            B3= vrai
```

Disjonctions entre prémisses

A1 ou A2 ou A3 => B1 et B2

est équivalent à l'écriture de plusieurs règles:

```
SI          A1
ALORS       B1B2
SI          A2
ALORS       B1B2
SI          A3
ALORS       B1B2
```

Disjonction de conséquents ne peut être traduite dans SNARK.

Exemple de représentation de connaissances opératoires

Problème des enfants et des fleurs

Règle R01:

```
REGLE :  0001
* MISE A JOUR DE L'ENSEMBLE IMCUEILLE
 SI NON-CUEILLE (I) = (J)
     IMCUEILLE   (I) = (J)
 ALORS
     TUERFAIT IMCUEILLE (I)  (J)
     CARD-IMCUEILLE (I) <-- - CARD-IMCUEILLE (I)  1
 FR
```

- met à jour l'ensemble des fleurs que i peut encore cueillir en tenant compte des informations lui indiquant que la fleur j ne peut être cueillie par i (NONCUEILLE(i) = (j)),
- met à jour CARDIMCUEILLE(i).

Règle R02:

```
REGLE :  0002
          * SURJECTIVITE DE LA FONCTION CUEILLE
 SI CARD-IMCUEILLE (I) = 1
     IMCUEILLE   (I) = (K)
 ALORS
     CUEILLE (I) <-- (K)
 FR
```

- lorsque l'on sait qu'il ne reste plus qu'un type de fleur, que i peut cueillir (CARDIMCUEILLE(i) = 1):
- permet d'assigner le type de fleur que i cueille à l'aide de CUEILLE (i) <-- (k).

Exemple de représentation de connaissances opératoires

Problème des enfants et des fleurs (suite)

Règle R03:

```
REGLE :  0003
          * INJECTIVITE DE LA FONCTION CUEILLE
SI CUEILLE (I) =  (J)
    NATURE (K) =  GARCON
ALORS
    NON-CUEILLE (K) <== (J)
FR
```

si l'on sait qu'une fleur a déjà été cueillie par un enfant:

- permet de l'ajouter parmi celles que ne peuvent cueillir les autres
- présence du FIN ETAPE qui permet de regrouper les trois premières règles

Exemple de représentation de connaissances opératoires

La base de règles:

```
OPTION2
REGLE :  0001
* MISE A JOUR DE L'ENSEMBLE IMCUEILLE
 SI NON-CUEILLE (I) = (J)
     IMCUEILLE   (I) = (J)
 ALORS
     TUERFAIT IMCUEILLE (I)  (J)
     CARD-IMCUEILLE (I) <-- - CARD-IMCUEILLE (I)  1
 FR

 REGLE :  0002
           * SURJECTIVITE DE LA FONCTION CUEILLE
 SI CARD-IMCUEILLE (I) = 1
     IMCUEILLE   (I) = (K)
 ALORS
     CUEILLE (I) <-- (K)
 FR

 REGLE :  0003
           * INJECTIVITE DE LA FONCTION CUEILLE
 SI CUEILLE (I) =  (J)
     NATURE (K) =  GARCON
 ALORS
     NON-CUEILLE (K) <== (J)
 FR
```

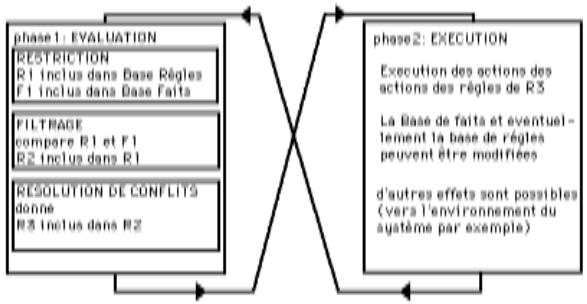

Interprétation des connaissances dans SNARK

- SNARK: interpréteur de connaissances
- SNARK fonctionne en **chaînage avant**,
  - il déclenche toutes les règles dont les prémisses sont satisfaites
  - ajoute les nouveaux faits à la base de faits
  - recommence jusqu'à ce qu'il n'y ait plus de nouvelles déductions possibles
- peut **justifier** ses résultats
- permet la **non-monotonie**
- possibilité de **régime par tentatives** (gestion des hypothèses VIALATTE 84)
- permet l'usage de **méta-connaissance (méta règles)**
- représentation **binaire** des connaissances
  - risque plus élevé d'**explosion combinatoire**
  - importance d'un **filtrage des règles**, afin d'éviter des essais inutiles

Cycle de base de l'interpréteur SNARK

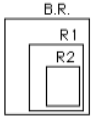
Cycle d'EVALUATION-EXECUTION

(H.FARRENY):

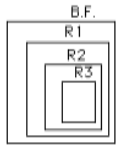


La phase d'EVALUATION

- **Étape de SÉLECTION**
  - dans SNARK, il n'en existe pas, il existe cependant la notion d'étape...
- **Étape de FILTRAGE**
  - SNARK sélectionne l'ensemble des règles candidates c'est-à-dire celles dont tous les mots-clés ont au moins une occurrence dans la base de faits (R2):



- **Étape de RÉSOLUTION DE CONFLITS**
  - dans SNARK, R3 contient au maximum une règle, celle qui possède le rapport (nb d'actions/nb de variables) le plus élevé
  - si aucune règle ne peut être déclenchée:
    - SNARK vérifie s'il existe une autre étape qu'il pourrait alors tenter d'évaluer
    - si aucune autre étape n'existe, Snark se termine.



Phase d'EXÉCUTION (1)

L'instanciation des prémisses coûte très cher (combinatoire)
=> nécessaire d'avoir un heuristique, dans SNARK on a l'heuristique suivant :

1) Ordonner les prémisses de cette règle en attribuant à chacune une note déterminée par le type (fonction du nombre de variables) de la prémisse, soit:

				note
1:	prop (x)	cop	val ou quark	0
2:	prop (x)	cop	connu	1000
3:	prop quark	cop	(y)	0
4:	prop (x)	cop	(y)	4000

- la note est augmentée :
  - du nb d'occurrences de la propriété (ou de la valeur) dans la base de faits
  - de 4000 points si l'opérateur est différent de l'égalité
- c'est la prémisse qui possède le moins de variables qui a le plus de chances d'être choisie

Phase d'EXÉCUTION (2)

La prémise étant choisie :

2) Instancier la prémisse choisie en cherchant dans la base de faits la première occurrence de la propriété et ainsi assigner à chacune de ses variables (djinn) un objet de la base de faits

3) Propager l'instanciation dans les autres prémisses de la règle traitée et les actions de la règle choisie:

4) Aller en 1, cad choisir une autre prémisse parmi celles non encore évaluées.

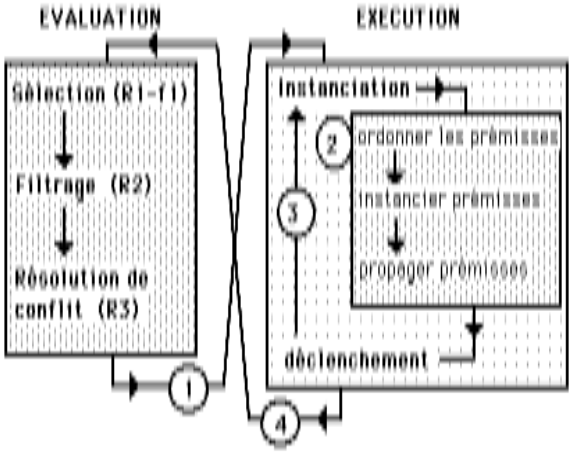
Phase d'EXÉCUTION (3)

- **Retours arrière dans l'instanciation des prémisses**
  - Il y aura retour arrière dans la pile des choix de objets-entité chaque fois :
    1) qu'une prémisse ne peut être satisfaite compte tenu des choix effectués antérieurement lors de l'instanciation de prémisses précédentes.
    2) qu'une règle a été évaluée et déclenchée. Il faut alors rechercher les autres instanciations possibles
    3) que toutes les instanciations possibles d'un djinn ont été faites.
  - dès qu'un djinn ne peut plus être instancié par un nouveau quark, l'évaluation de la règle se termine et cette dernière est retirée momentanément des règles candidates (R2) le temps d'un cycle afin d'éviter de boucler sur la même règle.
  - lorsque tous les djinns de la règle de R3 ont été instanciés par des quarks, nous passons au déclenchement de la règle.

Phase d'EXÉCUTION (4)

- **Déclenchement de la règle :**
  - exécuter les actions définies dans la règle
  - une fois la règle déclenchée, on recherche s'il existe d'autres instanciations possibles pour cette règle.
  - Si une nouvelle instanciation est possible, on passe immédiatement au déclenchement (1-2-3-2-3...),
  - sinon un nouveau cycle recommence (1-2-3-4-1...).

Cycle d'interprétation dans SNARK



La méta-connaissance dans SNARK

- **méta-connaissance sur l'interprétation**
  - dans le cycle d'interprétation il faut effectuer les taches suivantes :
    - déterminer les règles candidates (filtrage)
    - ordonner ces règles candidates pour examen
    - choisir une règle (résolution de conflits)
  - l'interpréteur possède ses propres heuristiques
- dans SNARK l'utilisateur peut piloter en charge une partie de ces taches:
  - piloter l'ordre d'examen des règles (priorité sur les règles):

```
EXAMINER  (R)
PRIORITE (R)   2
```

  - pour certaines règles pour lesquelles certaines conditions sont vérifiées les supprimer momentanément ou définitivement ou les réactiver:

```
INHIBER  (R) / ACTIVER  (R)
ANTECEDENT  (R): ....
/CONSEQUENT  (R): ....
```


Examen prioritaire de certaines règles par rapport à d'autres

- permet la gestion d'un **agenda de règles**
- usage de **méta-règles** permettant :
  - de repérer une ou plusieurs règles selon :
    - ses **antécédents**
    - ses **conséquent**
  - de **placer** cette règle ou ces règles dans un paquet prioritaire
  - d'établir une **priorité relative** entre les différentes règles référencées dans la méta-règle

exemple:

```
REGLE : META
SI
ALORS
EXAMINER   (R)
ANTECEDENT  (R) : etat (P)  =  debut
PRIORITE  (R)  1
EXAMINER  (S)
ANTECEDENT  (S) : initialisation  (P)  =  oui
CONSEQUENT  (S) : etat (P)  <--  inconnu
PRIORITE  (S)  1
EXAMINER  (T)
ANTECEDENT  (T) : etat (P)  = en_cours
ANTECEDENT  (T) : evaluation  (P)  =  oui
PRIORITE  (T)  2
```


Examen prioritaire de certaines règles par rapport à d'autres

Remarques:

- une règle sélectionnée n'en est pas pour autant déclenchable
- les règles à examiner sont repérées par leur contenu (et non leur nom) dans la base de fait car:
  - règles indépendantes les unes des autres
  - règles fournies en vrac (pas d'ordre)
  - non-monotonie...
- les priorités des règles à examiner sont des priorités locales, traduction de d'abord....puis...puis...enfin...

Les règles démons

- permet la **gestion de l'agenda de règles**
- règles dont la priorité d'examen est absolue par rapport aux autres règles non-démon

```
<règle démon> ::= REGLE : <nom>
                  DES QUE  <antécédent>
                  [<antécédent>]
                  ALORS    <conséquent>
                  [<conséquent>]
```

exemple :

```
DES QUE  nb_solution   (pb) = 5
ALORS
     ECRIRE  "les 5 solutions sont trouvées"
```