Réflexions sur l'enseignement initial de la pensée computationelle.

Une approche différente de la programmation

Damien Morard, Dimitri Racordon, Aurélien Coet,

Didier Buchs

REPUBLIQUE ET CANTON DE GENEVE

POST TENEBRAS LUX

UNIVERSITÉ DE GENÈVE

FACULTÉ DES SCIENCES

L’informatique au secondaire

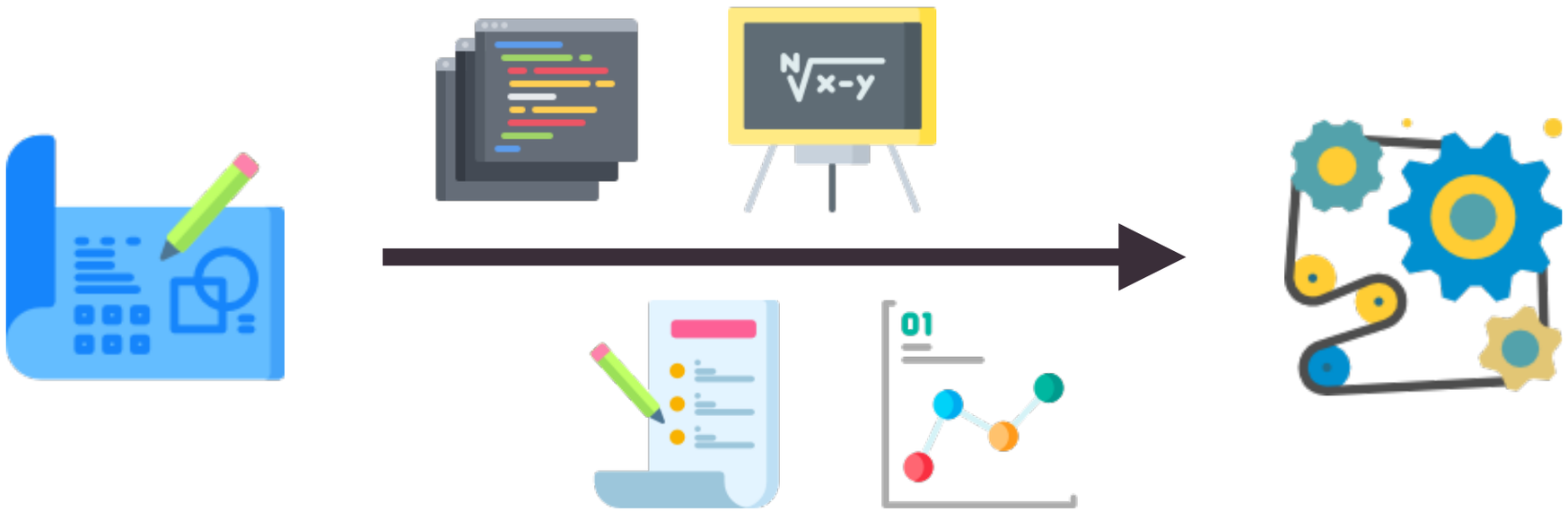
Plan d’étude:

- Pensée computationelle
  - Algorithmique
  - Programmation
- Sensibilisation aux domaines d’applications
  - Réseaux
  - Cryptographie
  - Bases de données
  - …
- Transdisciplinarité

La **pensée computationnelle** s'intéresse à la **résolution de problèmes**, à la conception de systèmes ou même à la compréhension des comportements humains en s'appuyant sur les concepts fondamentaux de l'informatique théorique. Wikipédia

La **pensée computationnelle** est le processus de pensée impliqué dans la **formulation** d'un problème et sa **solution** de manière à ce qu'un ordinateur — humain ou machine — puisse fonctionner efficacement. Jeannette Wing, Columbia University.

Programmation



4

Sondage : Qu'impriment les ligne 5 et 9 ?

N° ligne Programme:

```
0   def f(x,y):
1       x += y
2
3   x = 2
4   f(x,x)
5   print(x)
6
7   x = [2]
8   f(x,x)
9   print(x)
```

Sondage : Qu'impriment les ligne 5 et 9 ?

N° ligne  Programme:

```
0
1
2
3   x = 2
4   f(x,x)
5   print(x)
6
7   x = [2]
8   f(x,x)
9   print(x)
```

```
def f(x,y):
    x += y
```

1)
```
Ligne 5: 2
Ligne 9: [2,2]
```

2)
```
Ligne 5: 2
Ligne 9: [2]
```

3)
```
Ligne 5: 4
Ligne 9: [4]
```

4)
```
Ligne 5: 4
Ligne 9: [2,2]
```

Sondage : Qu'impriment les ligne 5 et 9 ?

N° ligne  Programme:

```
0
1
2
3
4
5
6
7
8
9
```

```
def f(x,y):
    x += y


x = 2
f(x,x)
print(x)

x = [2]
f(x,x)
print(x)
```

1)
```
Ligne 5: 2
Ligne 9: [2,2]
```

2)
```
Ligne 5: 2
Ligne 9: [2]
```

3)
```
Ligne 5: 4
Ligne 9: [4]
```

4)
```
Ligne 5: 4
Ligne 9: [2,2]
```

Sondage : Qu'impriment les ligne 5 et 9 ?

N° ligne  Programme:

```
0
1
2
3
4
5
6
7
8
9
```

```
def f(x,y):
    x += y


x = 2
f(x,x)
print(x)

x = [2]
f(x,x)
print(x)
```

1)
```
Ligne 5: 2
Ligne 9: [2,2]
```

2)
```
Ligne 5: 2
Ligne 9: [2]
```

3)
```
Ligne 5: 4
Ligne 9: [4]
```

4)
```
Ligne 5: 4
Ligne 9: [2,2]
```

Sondage : Qu'impriment les ligne 5 et 9 ?

N° ligne  Programme:

```
0
1
2
3   x = 2
4   f(x,x)
5   print(x)
6
7   x = [2]
8   f(x,x)
9   print(x)
```

```
def f(x,y):
    x += y
```

1)
```
Ligne 5: 2
Ligne 9: [2,2]
```

2)
```
Ligne 5: 2
Ligne 9: [2]
```

3)
```
Ligne 5: 4
Ligne 9: [4]
```

4)
```
Ligne 5: 4
Ligne 9: [2,2]
```

Sondage : Qu'impriment les ligne 5 et 9 ?

N° ligne  Programme:

```
0
1
2
3   x = 2
4   f(x,x)
5   print(x)
6
7   x = [2]
8   f(x,x)
9   print(x)
```

```
def f(x,y):
    x += y
```

1)
```
Ligne 5: 2
Ligne 9: [2,2]
```

2)
```
Ligne 5: 2
Ligne 9: [2]
```

3)
```
Ligne 5: 4
Ligne 9: [4]
```

4)
```
Ligne 5: 4
Ligne 9: [2,2]
```

Sondage : Qu'impriment les ligne 5 et 9 ?

N° ligne Programme:

```
0
1
2
3
4
5
6
7
8
9
```

```
def f(x,y):
    x += y


x = 2
f(x,x)
print(x)

x = [2]
f(x,x)
print(x)
```

1)
```
Ligne 5: 2
Ligne 9: [2,2]
```

2)
```
Ligne 5: 2
Ligne 9: [2]
```

3)
```
Ligne 5: 4
Ligne 9: [4]
```

4)
```
Ligne 5: 4
Ligne 9: [2,2]
```

Sondage : Qu'impriment les ligne 5 et 9 ?

N° ligne Programme:

```
0
1
2
3
4
5
6
7
8
9
```

```
def f(x,y):
    x += y


x = 2
f(x,x)
print(x)

x = [2]
f(x,x)
print(x)
```

1)
```
Ligne 5: 2
Ligne 9: [2,2]
```

2)
```
Ligne 5: 2
Ligne 9: [2]
```

3)
```
Ligne 5: 4
Ligne 9: [4]
```

4)
```
Ligne 5: 4
Ligne 9: [2,2]
```

Sondage : Qu'impriment les ligne 5 et 9 ?

N° ligne  Programme:

```
0
1
2
3   x = 2
4   f(x,x)
5   print(x)
6
7   x = [2]
8   f(x,x)
9   print(x)
```

```
def f(x,y):
    x += y
```

1)
```
Ligne 5: 2
Ligne 9: [2,2]
```

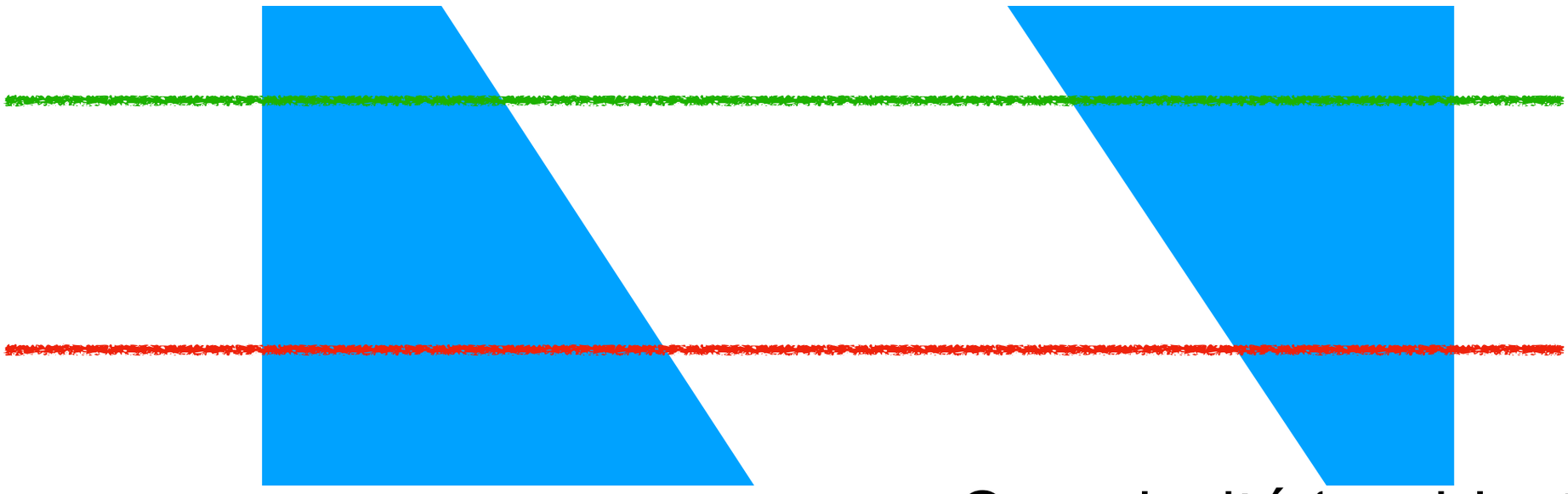
2)
```
Ligne 5: 2
Ligne 9: [2]
```

3)
```
Ligne 5: 4
Ligne 9: [4]
```

4)
```
Ligne 5: 4
Ligne 9: [2,2]
```

Résolution de Problèmes

Complexité intrinsèque



Complexité ‘accidentelle’

Expression de la **solution**
(Complexité)

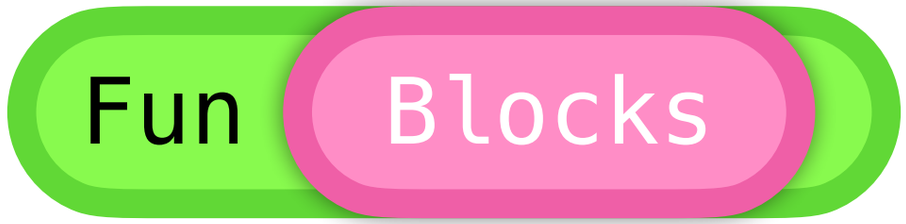
Expressivité du **langage**
(abstraction)

Fonctions abstraites calculables

15

FunctionalBlocks

16



- Outil éducatif pour l'enseignement
- Basé sur la programmation fonctionnelle
- Unique et simple mécanisme opérationnel
- Emphase sur les structures de données
- Traductions de graphique à textuelle, et inversement

Questions fondamentales

Qu'est-ce qu'une donnée?

Qu'est-ce qu'un calcul?

18

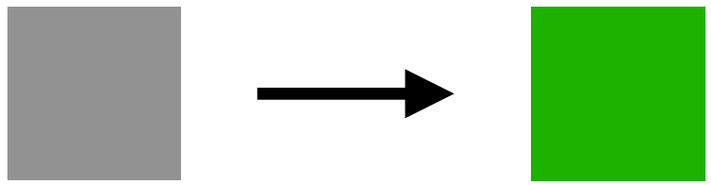
Réécriture de terme

Carré gris devient vert

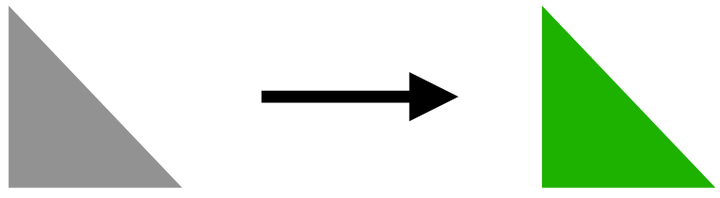
19

Réécriture de terme

Carré gris devient vert

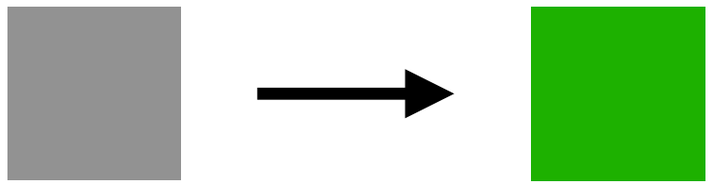


Triangle gris devient vert

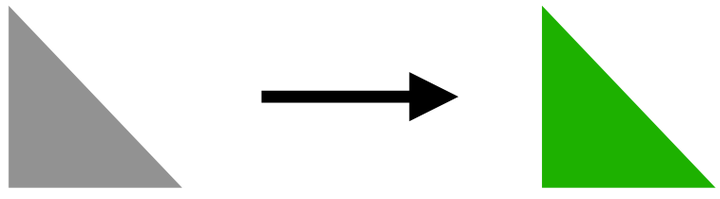


20

Réécriture de terme

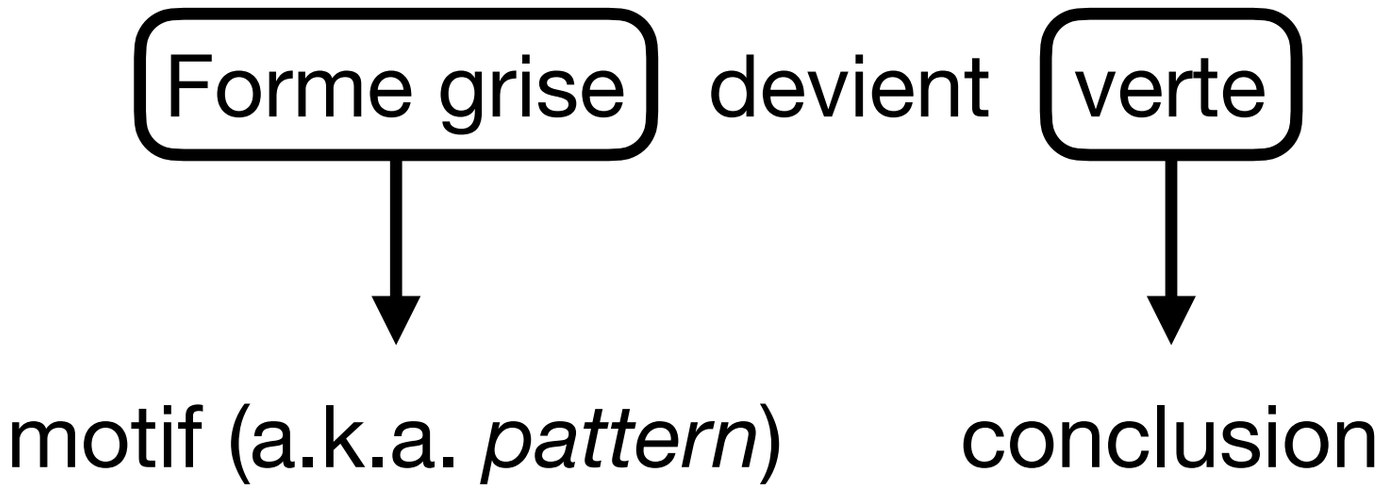


Forme grise devient verte



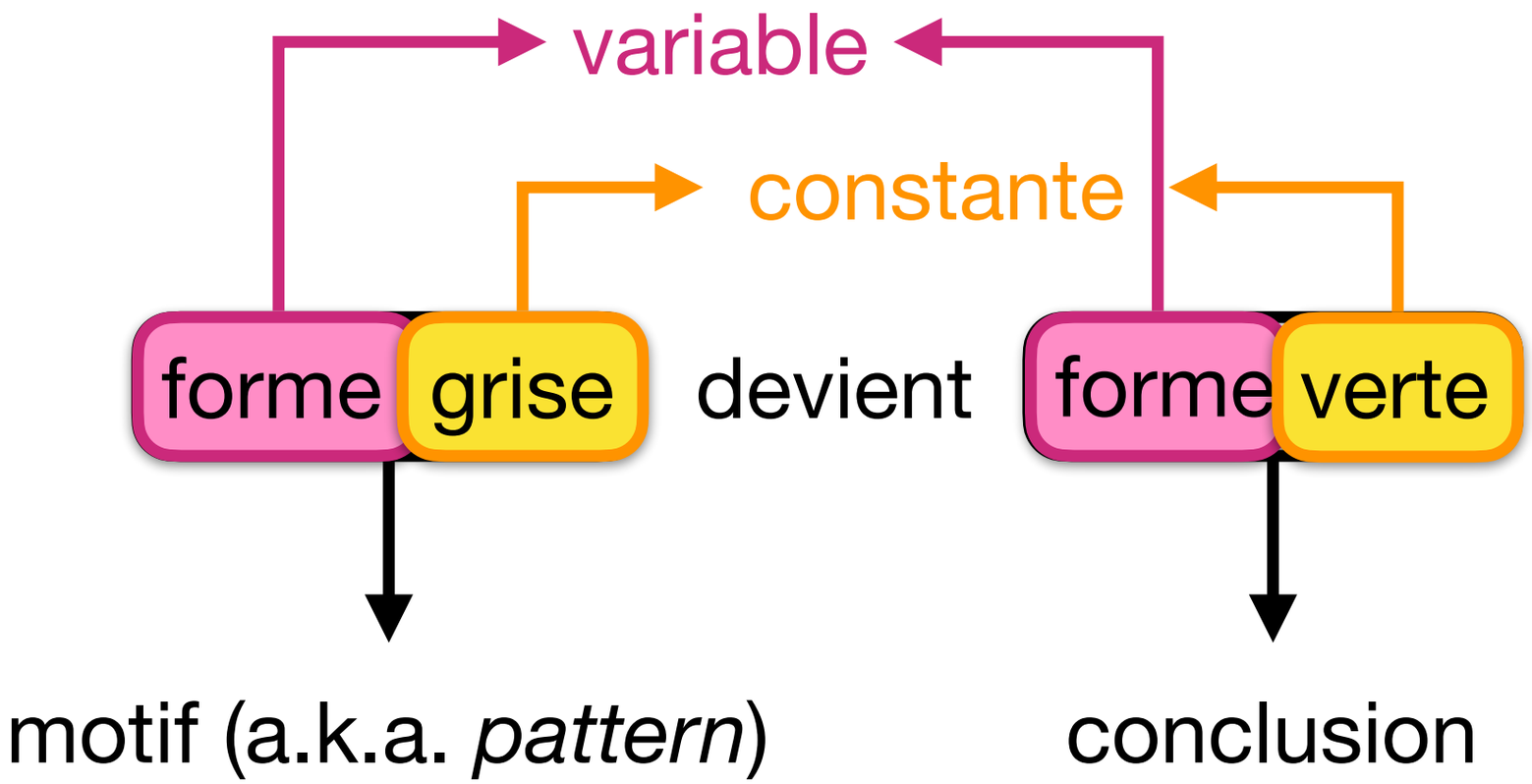
21

Réécriture de terme

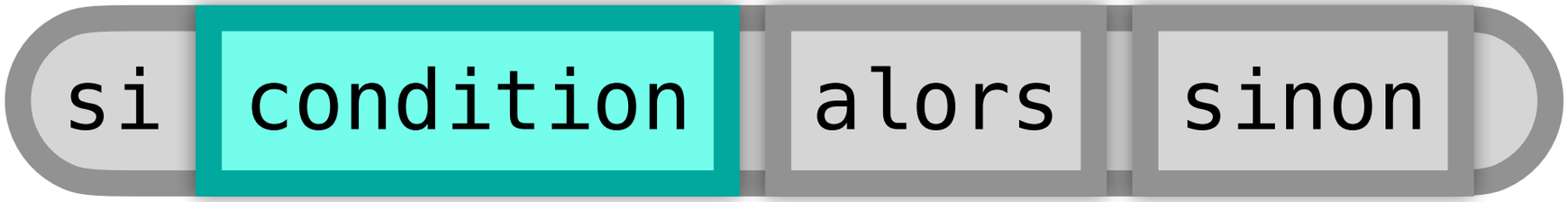


22

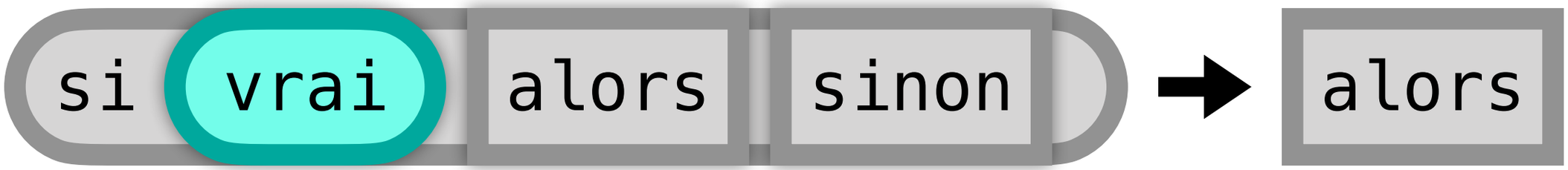
Réécriture de terme

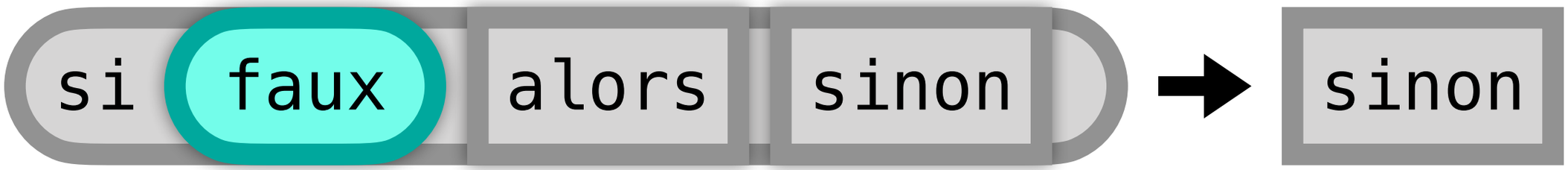


23



24





25

```
case max($x,$y) => si(>($x, $y), $x , $y)
```

```
case max($x,$y) => si(>($x, $y), $x , $y)
```

```
max(2,4) =>

si(2 > 4 , 2 , 4) =>

si(false, 2 , 4) =>

4
```

Question:

Comment représenter une liste?

Structures récursives

```
type List :: vide | cons nat (List)
```


29

Question:

Comment transformer une liste?

30

```
rule size :: List  -> nat
```

outil principal: récursion

31

Calcul récursif difficile à comprendre ?

```
type List :: vide | cons nat (List)
```


32

récursion guidée par la structure !

```
case size(vide) => 0
case size(cons($y, $z)) => succ(size($z))
```

```
rule count  :: nat , List -> List

case count($x, vide) => 0
case if $x=$y then count($x,cons($y,$z)) => succ(count($x,$z))
case if not$x=$y then count($x,cons($y,$z)) => count($x,$z)
```

FunctionalBlocks

Environnement interactif pour l'apprentissage de la programmation

Dimitri Racordon, Emmanouela Stachtiari, Didier Buchs

UNIVERSITÉ DE GENÈVE
FACULTÉ DES SCIENCES

REPUBLIQUE ET CANTON DE GENEVE
POST TENEBRAS LUX

References: logiciels et documents

https://github.com/kyouko-taiga/FunBlocks

https://blissful-bhaskara-4b1032.netlify.app

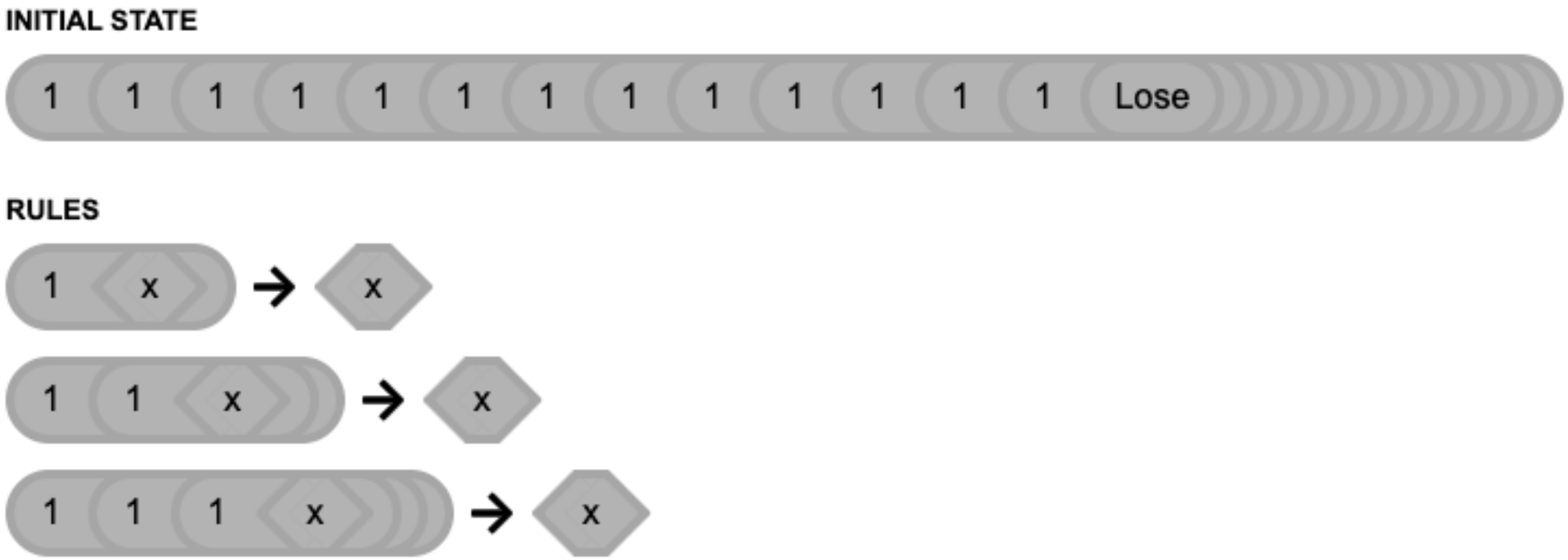
https://archive-ouverte.unige.ch/unige:150868

https://cui-unige.github.io/team-smv/projects/Travail_de_Bachelor_Matthieu_Vos.pdf

https://swish.swi-prolog.org/p/FunBlockWithStick.swinb

Jean-Françis Ravoux de de Saussure (Genève) prépare des séquences d’enseignement sur FunBlocks

Exemple: jeux des bâtons




```
case 1($x) => $x
case 1(1($x)) => $x
case 1(1(1($x))) => $x
```

Compétences attendues

- vue transformationnelle
- séquence d’étapes
- traitement par catégories et par cas
- maitrise des abstractions

Que peut-on observer sur un programme FunBlocks

- Syntaxe très simple et uniforme
- Sémantique statique du typage fort.
- Sémantique dynamique simple mais complète:
  - déterministe ou non-déterministe
  - terminaison
  - complétude

Aprentissage par paradigmes ?

- Les paradigmes décrivent les mêmes fonctionnalités ! (imperative, object-oriented, functional)
- Ces paradigmes sont orthogonaux.
- Les langages réels sont multi-paradigmes.
- Est-ce utile de d’enseigner ces fonctionnalités par les paradigmes?

Research works

- Armoni, M., Meerbaum-Salant, O., & Ben-Ari, M. (2015). **From Scratch to “real” programming**. Transactions on Computing Education (TOCE), 14(4), 25:1–25:15.

- David Statter and Michal Armoni. 2020. Teaching Abstraction in Computer Science to 7th Grade Students. ACM Trans. Comput. Educ. 20, 1, Article 8 (January 2020), 37 pages. DOI:https://doi.org/10.1145/3372143du Boulay, B. (1986). **Some difficulties of learning to program.** Journal of Educational Computing Research, 2(1), 57–73.

- Grover, S., & Basu, S. (2017). **Measuring student learning in introductory block-based programming: Examining misconceptions of loops, variables, and Boolean logic.** In Proceedings of the ACM Symposium on Computer Science Education (SIGCSE) (pp. 267–272). New York: ACM.

- Lewis, C. M. (2010). **How programming environment shapes perception, learning and goals: Logo vs. Scratch.** In Proceedings of the ACM Symposium on Computer Science Education (SIGCSE) (pp. 346–350). New York: ACM.

- Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2011). **Habits of programming in Scratch.** In Proceedings of the SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE) (pp. 168–172). New York: ACM.

- Powers, K., Ecott, S., & Hirshfield, L. M. (2007). **Through the looking glass: Teaching CS0 with Alice.** SIGCSE Bulletin, 39(1), 213–217.

- Stefik, A., & Gellenbeck, E. (2011). **Empirical studies on programming language stimuli.** Software Quality Journal, 19(1), 65–99.

- Weintrop, D., & Wilensky, U. (2015b). **Using commutative assessments to compare conceptual understanding in blocks-based and text-based programs.** In Proceedings of the Conference on International Computing Education Research (ICER). New York: ACM.

- Weintrop, D., & Wilensky, U. (2017a). **Comparing blocks-based and text-based programming in high school computer science classrooms.** Transactions on Computing Education (TOCE), 18(1), 3:1–3:25.

- Weintrop, D., & Wilensky, U. (2017b). **Between a block and a typeface: Designing and evaluating hybrid programming environments.** In Proceedings of the International Conference on Interaction Design and Children (pp. 183–192). New York, NY: ACM.