

Langages et Compilation

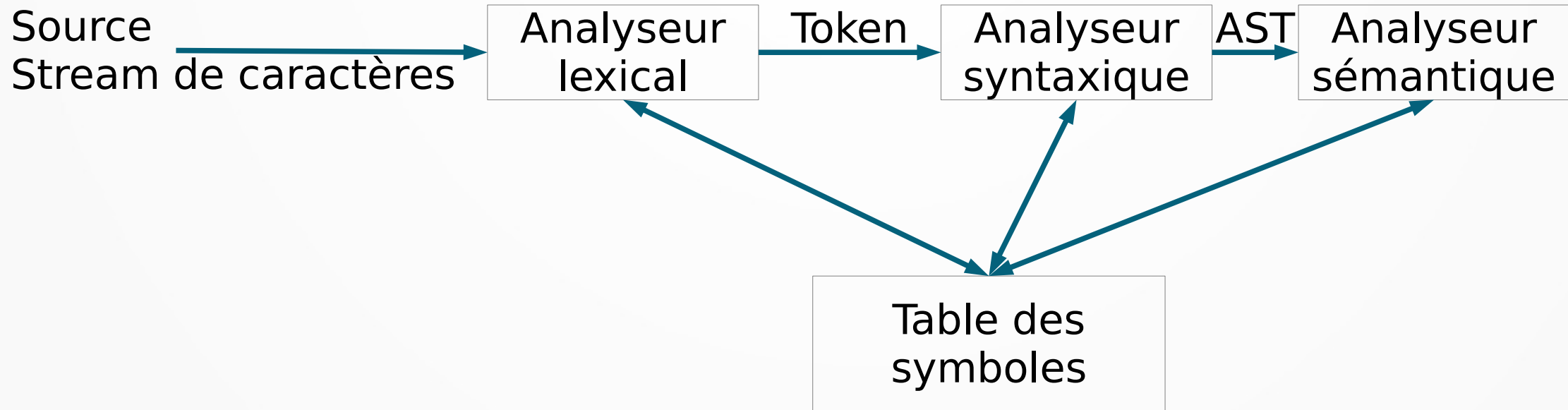
L5: Environnement

T. Goubier

L3A

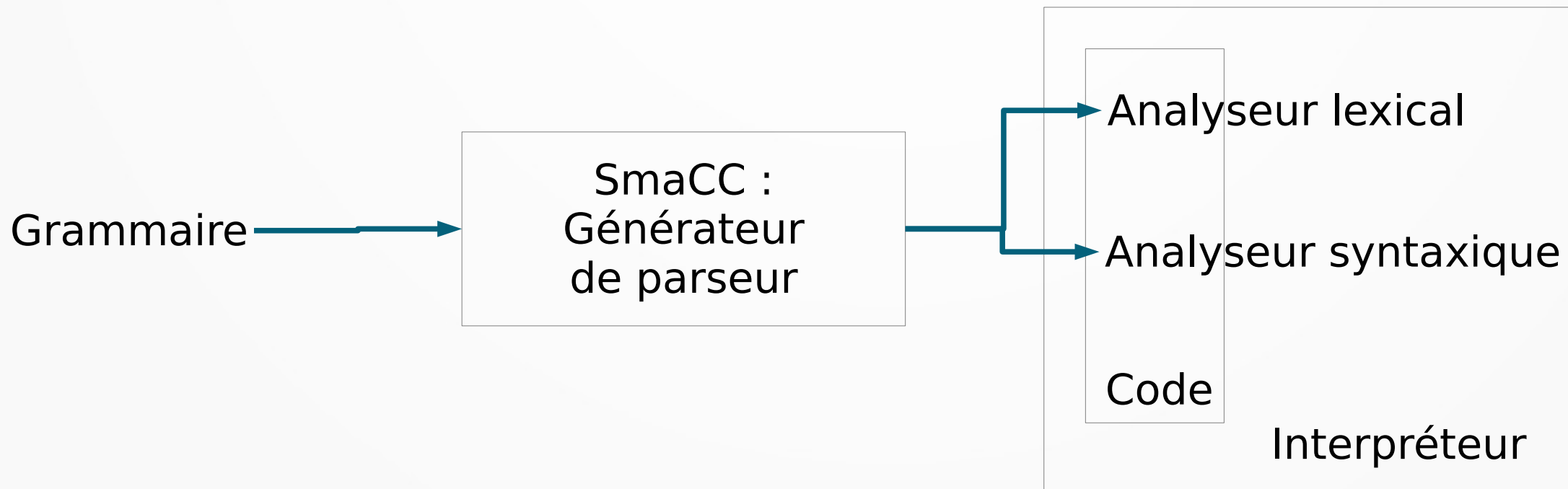
2019/2020

L5: Environnement



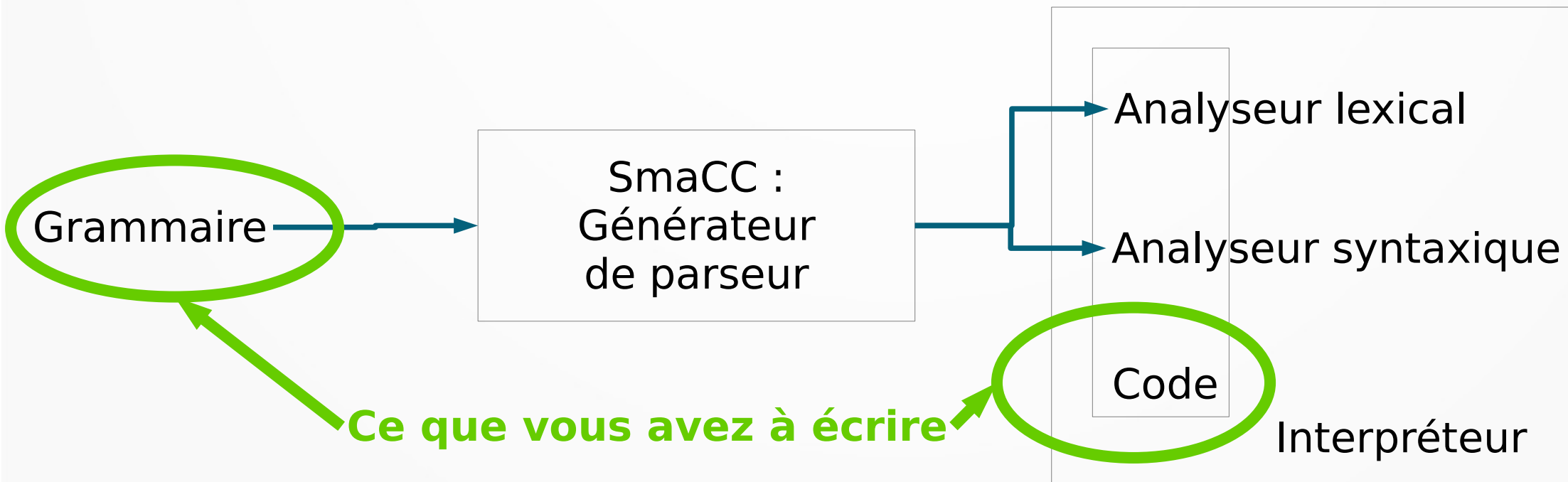
L5: Environnement

- Utilisation d'un générateur de parseurs pour écrire un interpréteur



L5: Environnement

- Utilisation d'un générateur de parseurs pour écrire un interpréteur



L5: Environnement

- SmaCC :
 - The Smalltalk Compiler Compiler
 - Un générateur de parseur
 - Avec des fonctions avancées de génération de code
 - En Smalltalk :
 - Langage purement objet
 - Aisément extensible
 - Modes de debug spéciaux

L5: Environnement

- Smalltalk:
 - Un langage purement objet, relativement ancien: 1980
 - Un langage simple
 - Objets, classes, messages, méthodes, blocks
 - Un langage uniforme
 - Tout est objet (y compris les entiers)
 - Chaque objet est instance d'une classe
 - L'exécution se fait en envoyant des messages à des objets
 - Les structures de contrôle (if, for) sont basées sur des blocks
 - Le langage possède un méta-modèle écrit en lui-même

L5: Environnement

- Smalltalk:
 - Une syntaxe minimaliste
 - 5 mots clés (true, false, self, super, nil)
 - Des littéraux
 - \$a : caractère
 - 'string' : une chaîne de caractères
 - #a : un symbole
 - #(1 1.0) : une array

```
exampleWithNumber: x
| y |
true & false not & (nil isNil)
  ifFalse: [self halt].
y := self size + super size.
#($a #a "a" 1 1.0)
  do: [ :each |
      Transcript show: (each class
name);
      show: ' ' ].
^x < y
```


L5: Environnement

- Smalltalk:

- Une syntaxe minimaliste

- Des variables locales: | y |

- Des blocks

- [self halt]

- [:each | ...]

- Un return : ^

- Une méthode complète:

- exampleWithNumber:

```
exampleWithNumber: x
    | y |
    true & false not & (nil isNil)
        ifFalse: [self halt].
    y := self size + super size.
    #($a #a "a" 1 1.0)
        do: [ :each |
            Transcript show: (each class
name);
            show: ' ' ].
    ^x < y
```


L5: Environnement

- Smalltalk:

- Des envois de messages:

halt

size

- Avec un paramètre:

show:

- Fin de ligne:

le point

- affecter une valeur:

:=

```
exampleWithNumber: x
| y |
true & false not & (nil isNil)
  ifFalse: [self halt].
y := self size + super size.
#($a #a "a" 1 1.0)
  do: [ :each |
      Transcript show: (each class
name);
      show: ' ' ].
^x < y
```

L5: Environnement

- Smalltalk:

- Une syntaxe minimaliste

- Une généralisation:

- Envoi de messages à

- true: ifTrue:, ifFalse:

- (c'est un if/else)

- un ensemble: do:

- (c'est une boucle)

```
exampleWithNumber: x
| y |
true & false not & (nil isNil)
  ifFalse: [self halt].
y := self size + super size.
#($a #a "a" 1 1.0)
  do: [ :each |
      Transcript show: (each class
name);
      show: ' ' ].
^x < y
```

L5: Environnement

- Smalltalk: une approche uniforme
 - L'environnement est écrit en lui-même
 - Le méta-modèle est écrit en lui-même
 - Une classe est un objet
 - Une méthode est un objet
 - Un block est un objet
 - Tout se fait par envoi de messages à des objets
 - Y compris le développement → l'IDE

L5: Environnement

- Créer une classe
 - Envoyer un message à la classe parent

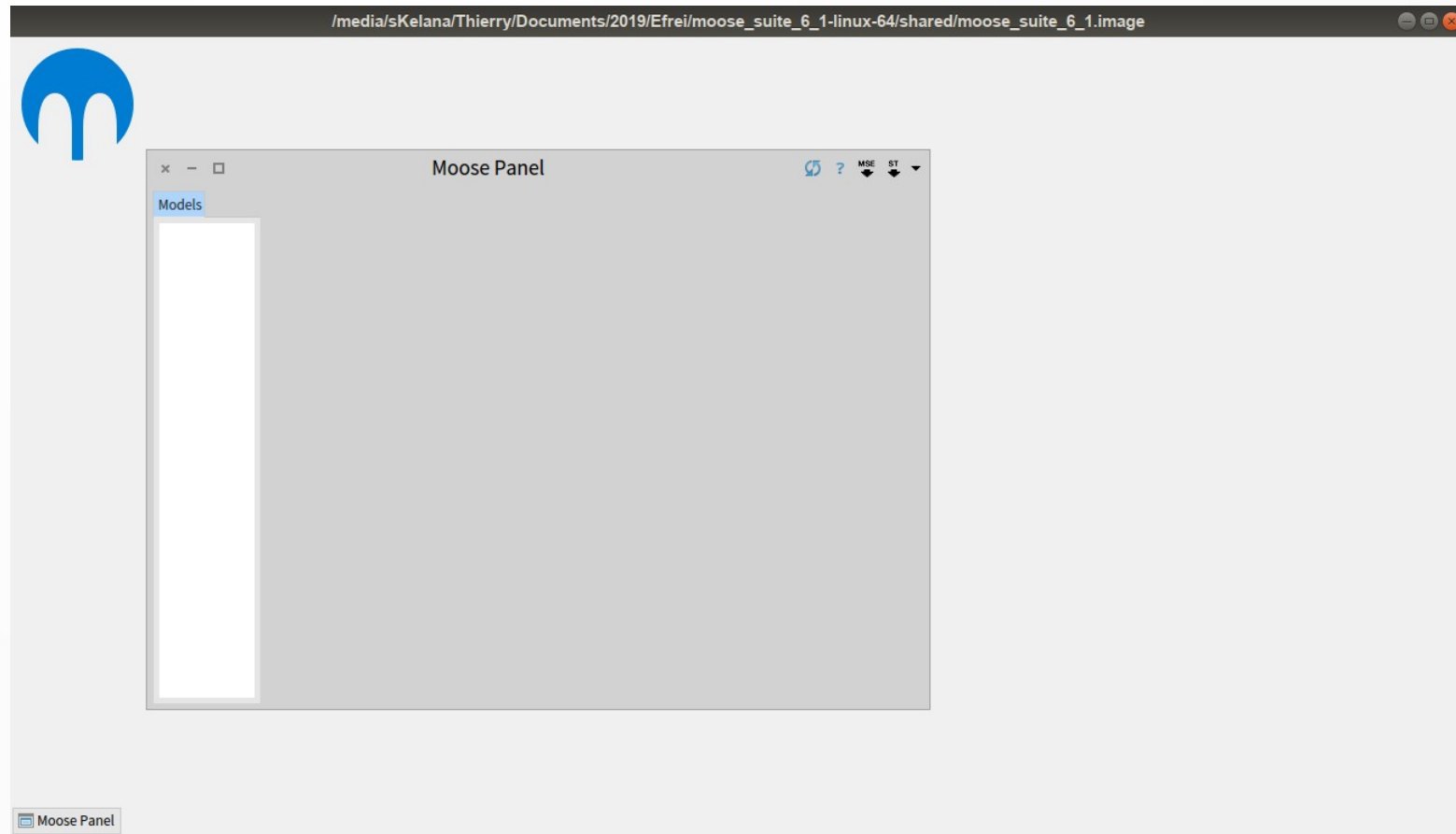
• subclass:	Object subclass: #MessagePublisher
instanceVariableNames:	instanceVariableNames: ''
classVariableNames:	classVariableNames: ''
poolDictionaries:	poolDictionaries: ''
category:	category: 'Smalltalk Examples'

L5: Environnement


- Créer un objet
 - Envoyer un message à la classe
 - new
 - Le message peut être surchargé `| a |`
`a := MessagePublisher new.`
 - new, new:, on: , etc.
 - new appelle initialize sur la nouvelle instance

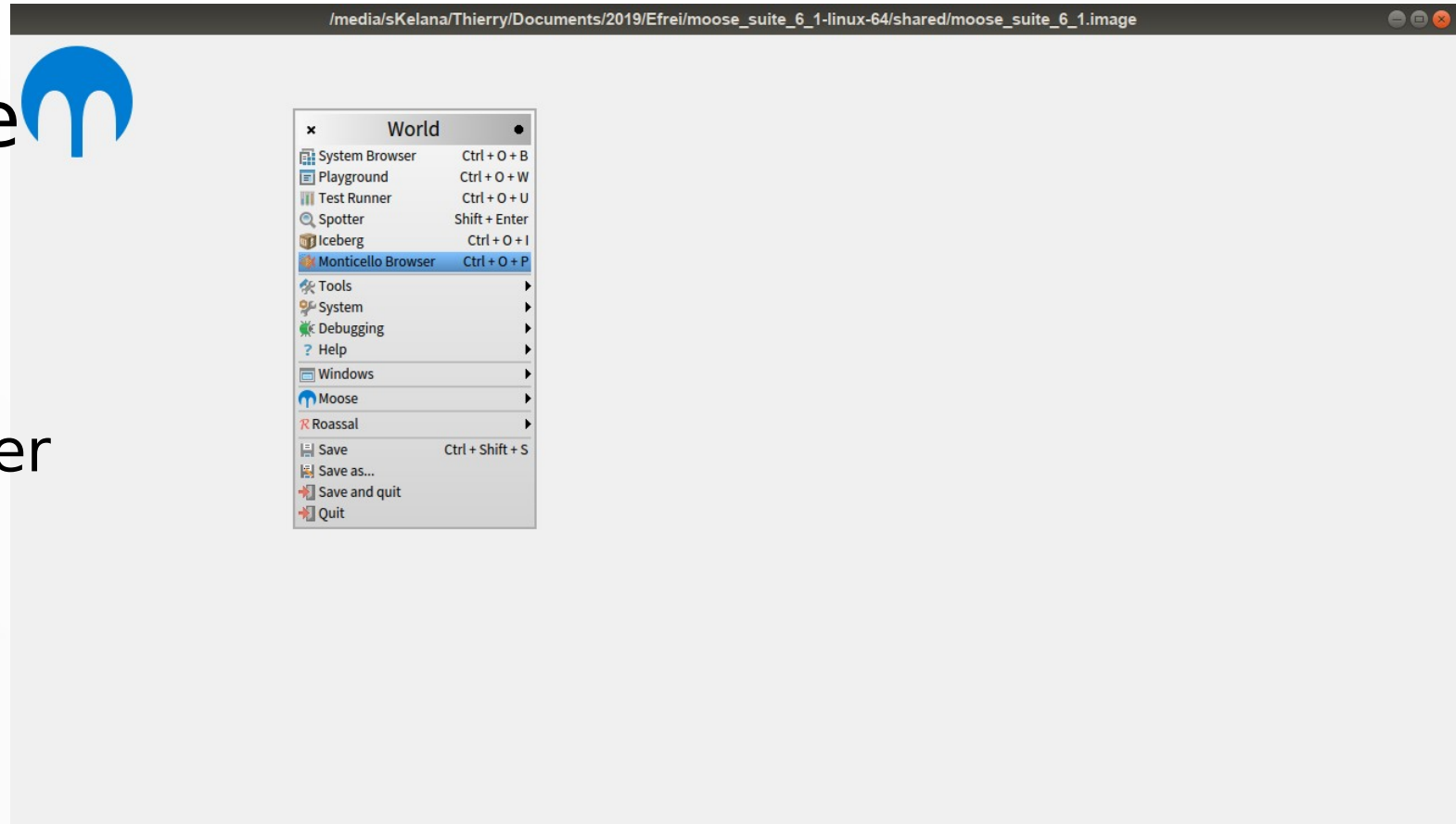
L5: Environnement

- Tutorial
 - Une petite calculatrice
 - avec juste + et ;
- Grammaire
$$S \rightarrow E ;$$
$$E \rightarrow E + T \mid T$$
$$T \rightarrow \langle \text{number} \rangle$$
- Télécharger Moose
 - Version 6.1
 - Pour votre système
 - <http://www.moosetechnology.org/#install>
 - Décompresser et Démarrer



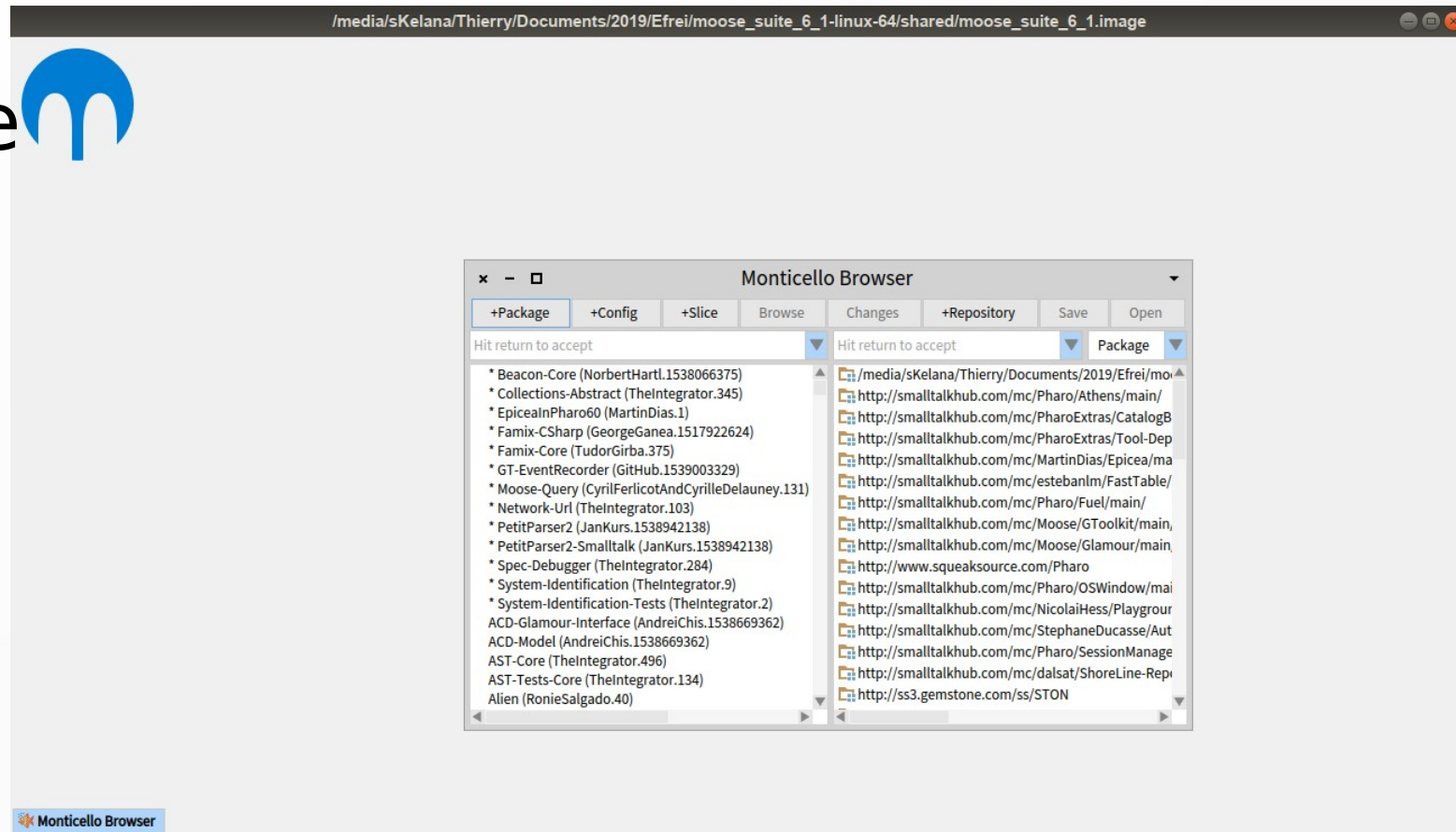
L5: Environnement

- Créer un package 
 - Ouvrir Monticello
 - World menu
 - Monticello Browser



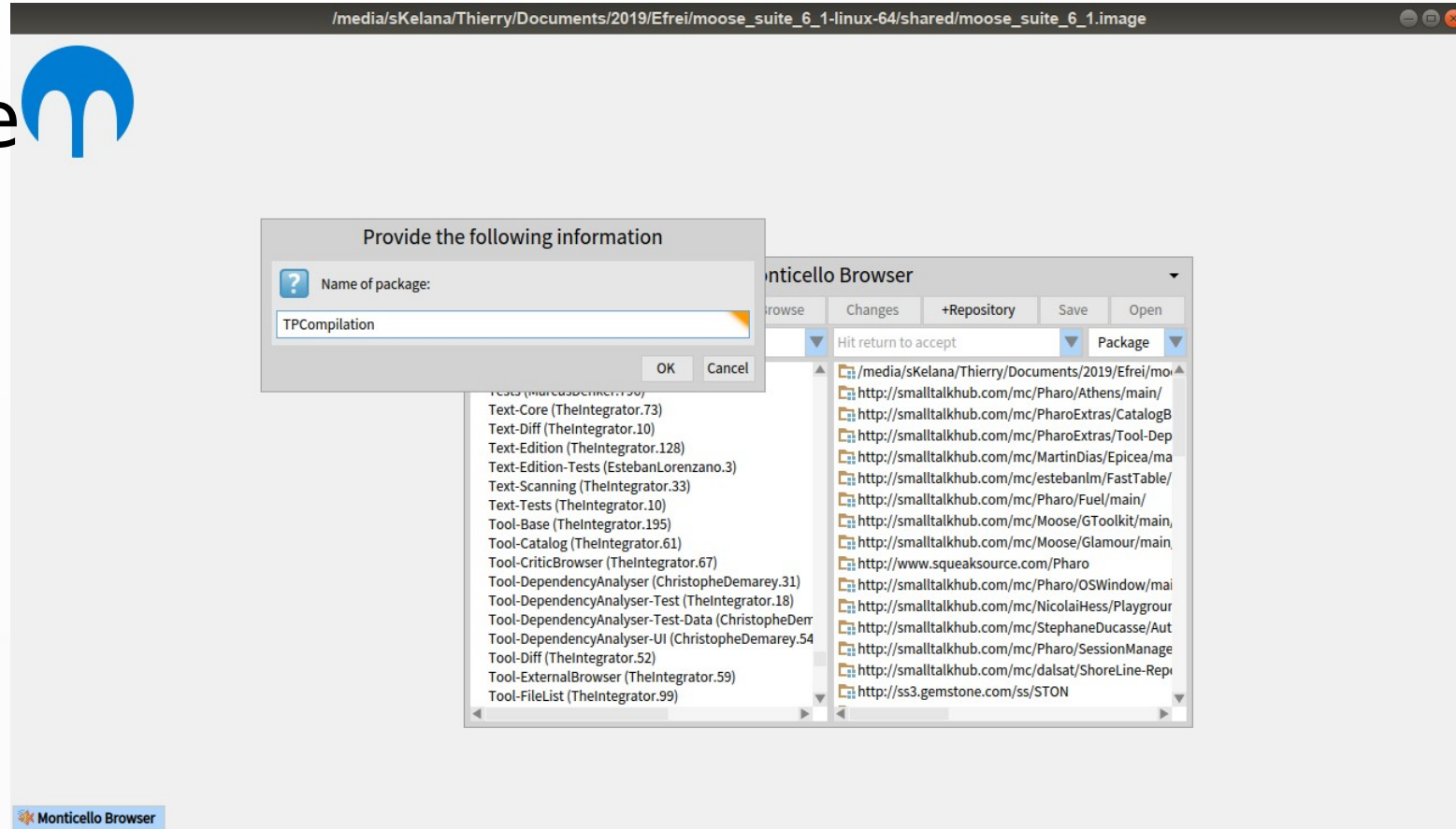
L5: Environnement

- Créer un package
 - Ouvrir Monticello



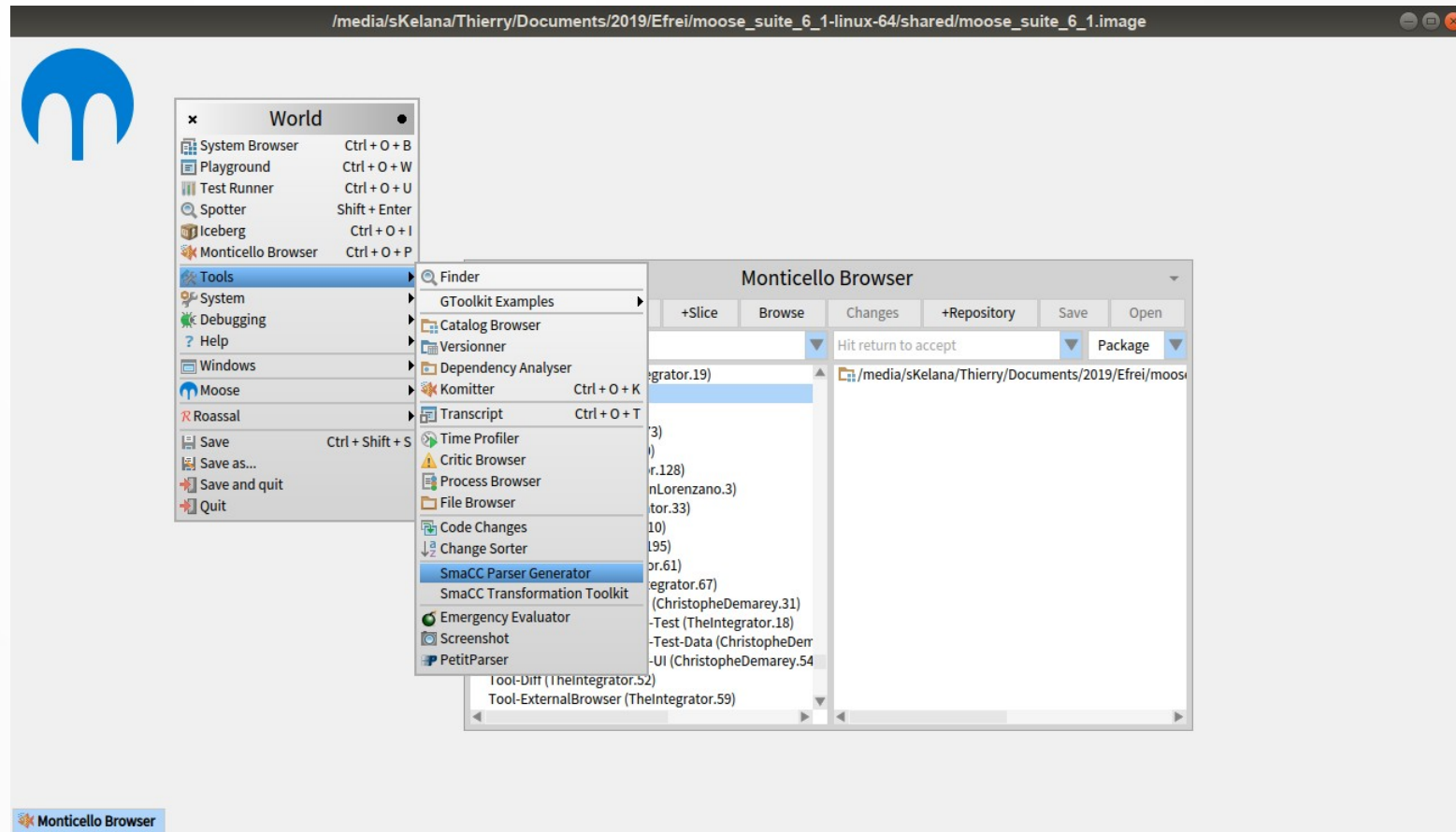
L5: Environnement

- Créer un package
 - Ouvrir Monticello
 - Appuyer sur
 - +Package
 - Remplir avec
 - TPCompilation



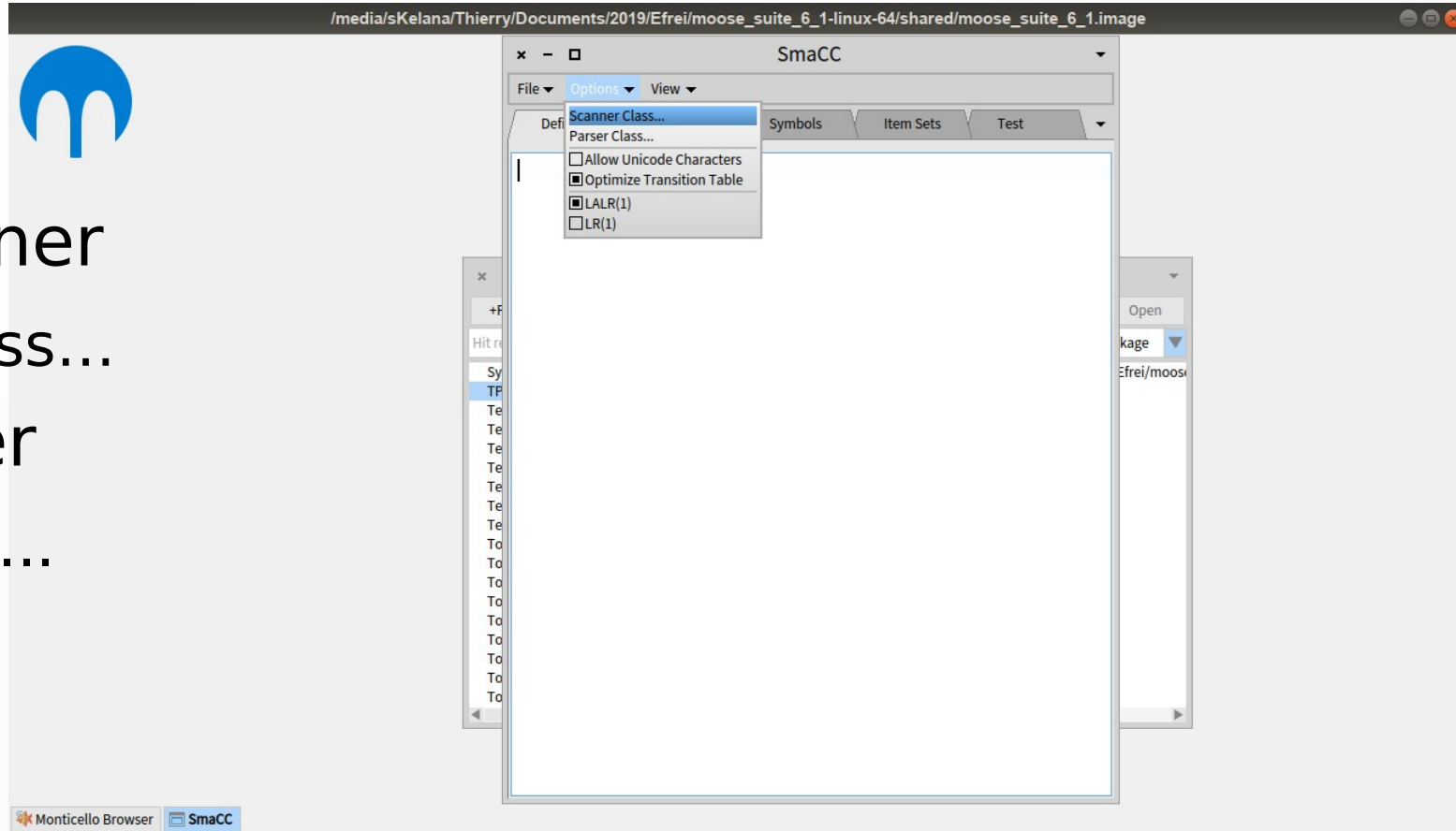
L5: Environnement

- Ouvrir SmaCC
 - World menu
 - tools submenu
 - SmaCC parser generator



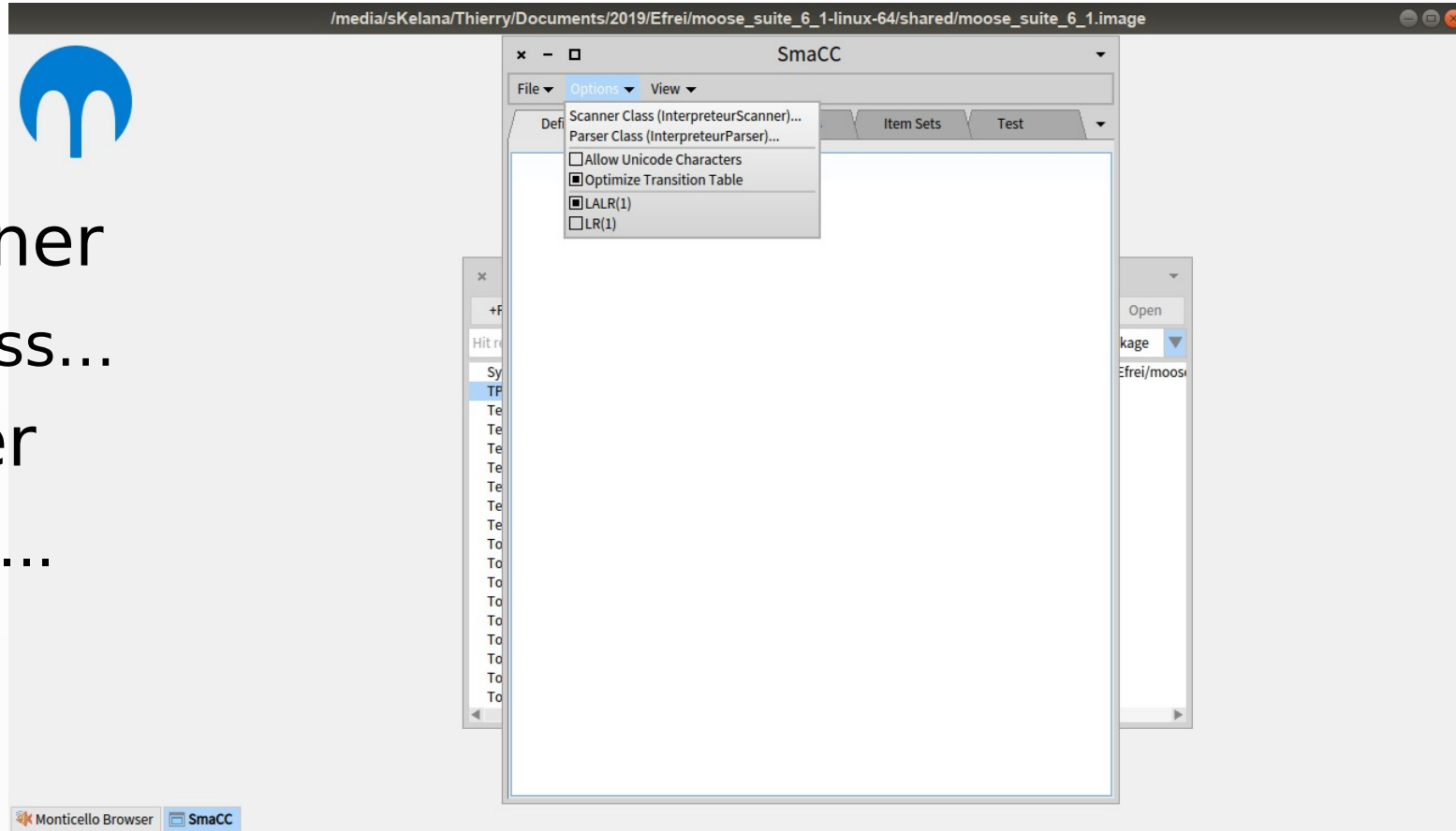
L5: Environnement

- Ecrire
 - InterpreteurScanner
 - Pour Scanner Class...
 - InterpreteurParser
 - Pour Parser Class...
 - Menu Options



L5: Environnement

- Ecrire
 - InterpreterScanner
 - Pour Scanner Class...
 - InterpreterParser
 - Pour Parser Class...
 - Menu Options



L5: Environnement

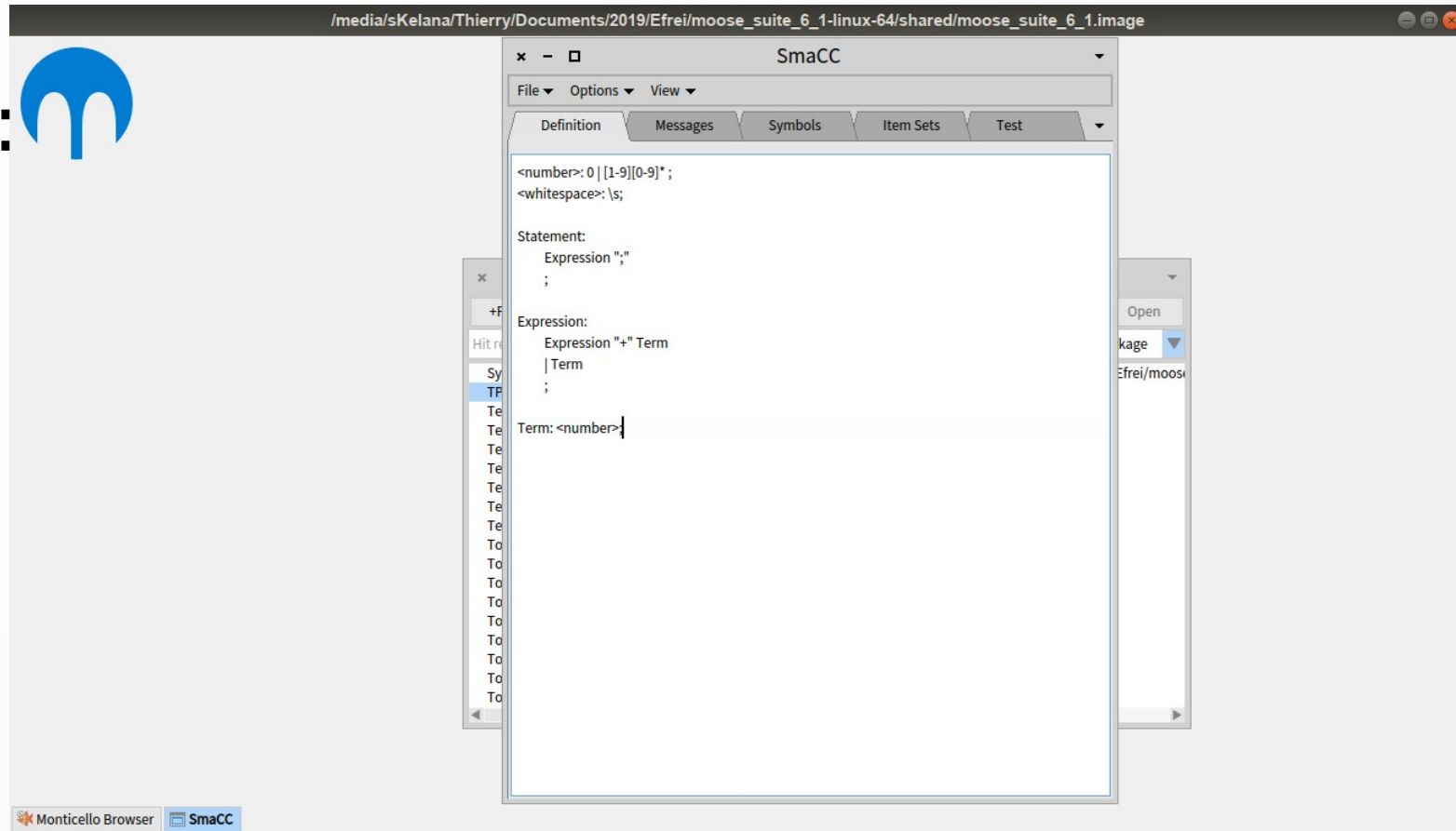
- Saisir la grammaire:

```
<number>: 0 | [1-9][0-9]* ;  
<whitespace>: \s;
```

```
Statement:  
    Expression ";"  
    ;
```

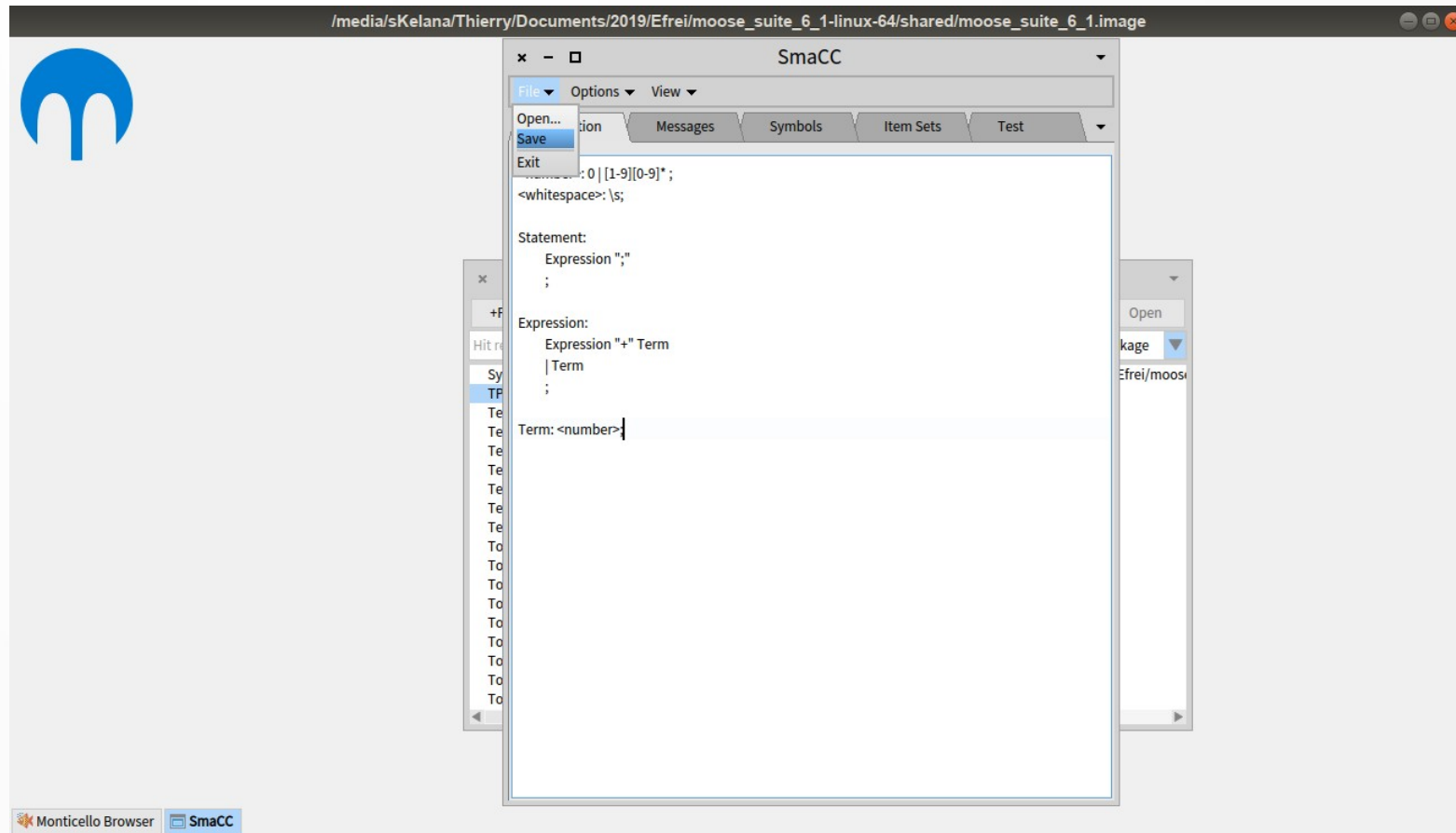
```
Expression:  
    Expression "+" Term  
    | Term  
    ;
```

```
Term: <number>;
```



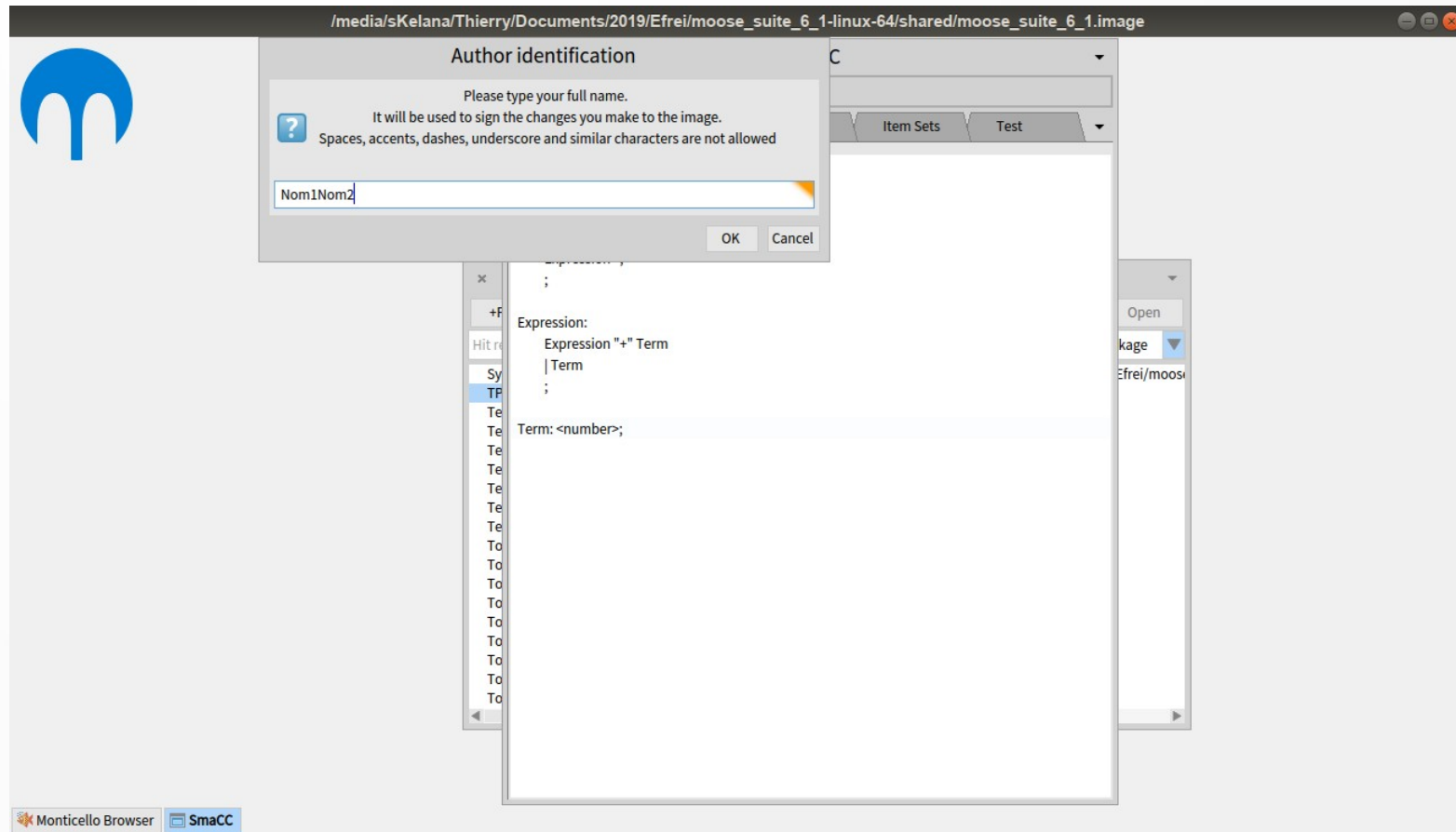
L5: Environnement

- Menu File
 - Save



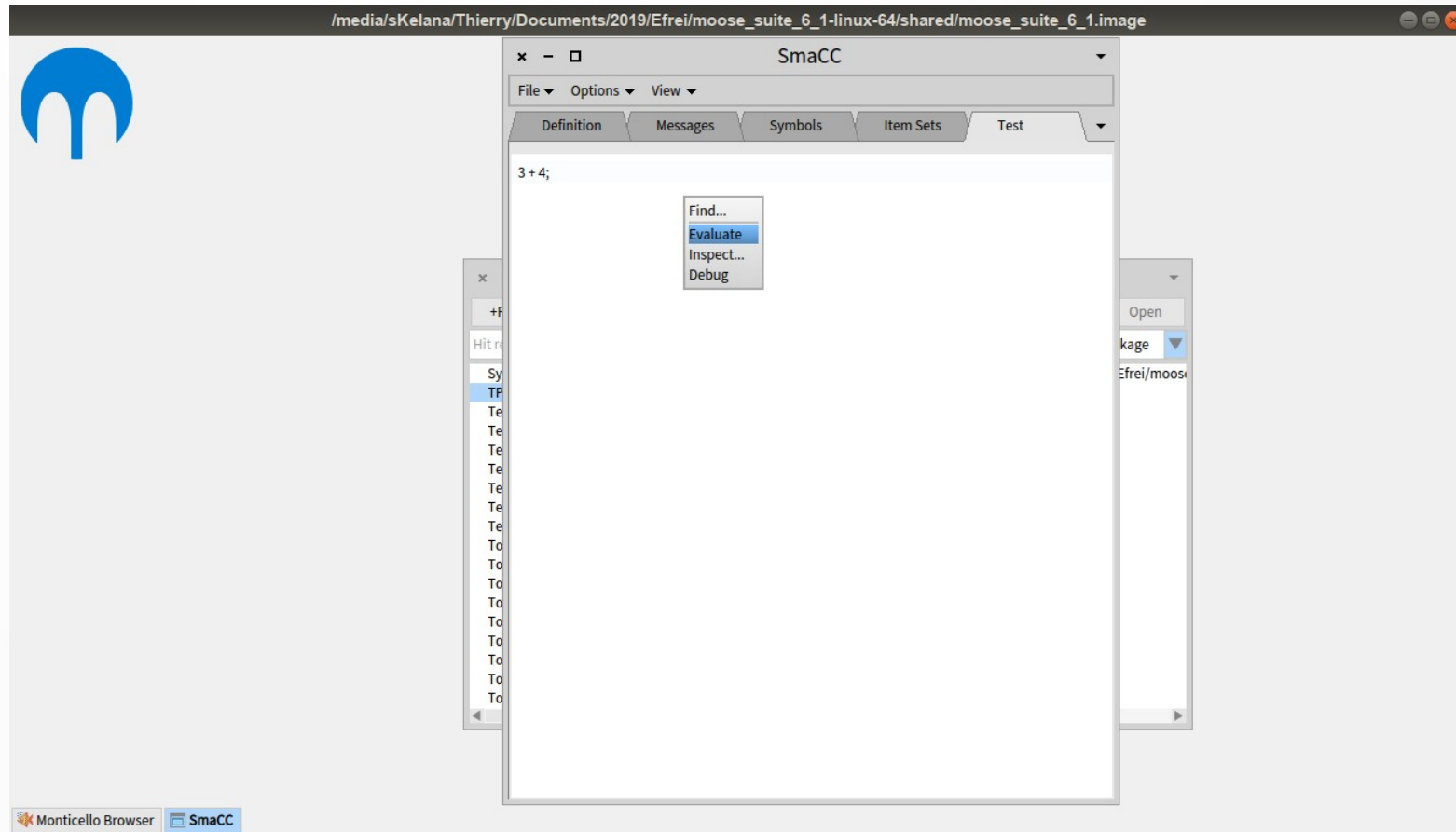
L5: Environnement

- Saisir les noms du Binome
 - Nom1Nom2



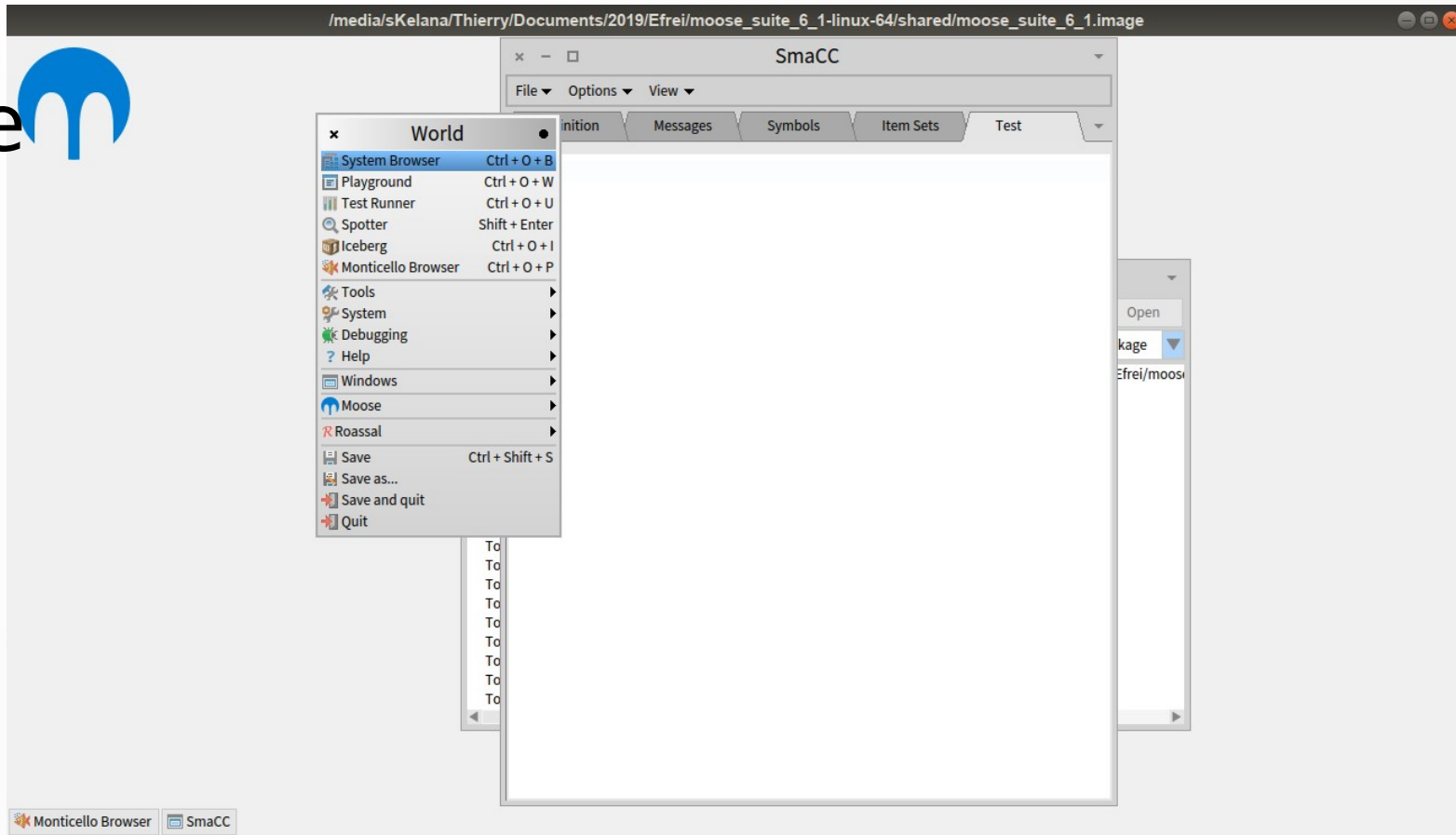
L5: Environnement

- Tester
 - Tab test, taper
 $3 + 4;$
 - Menu click droit
 - Evaluate
 - Parses without errors



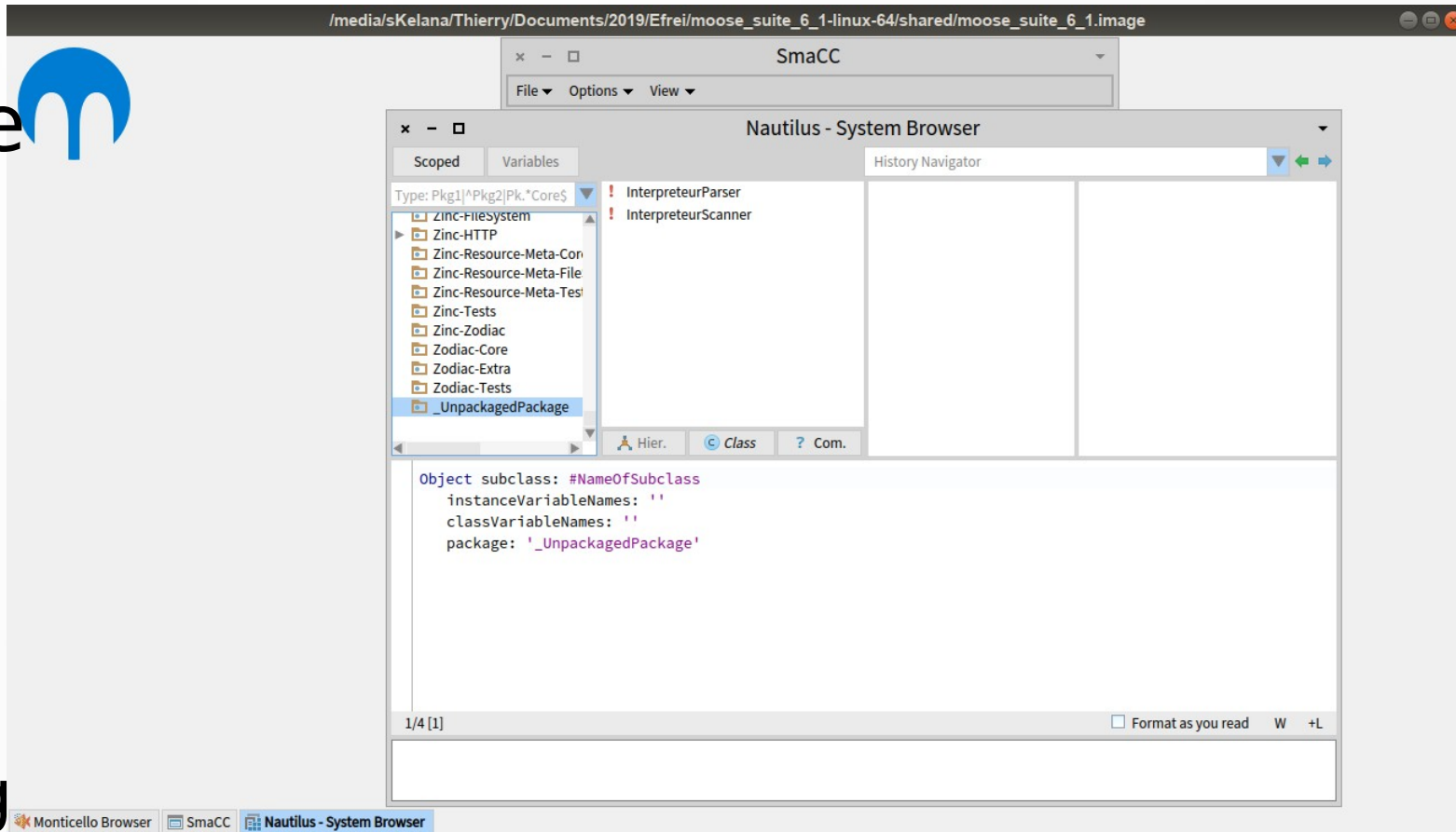
L5: Environnement

- Trouver le code et le déplacer dans le bon package
- World menu
 - System Browser



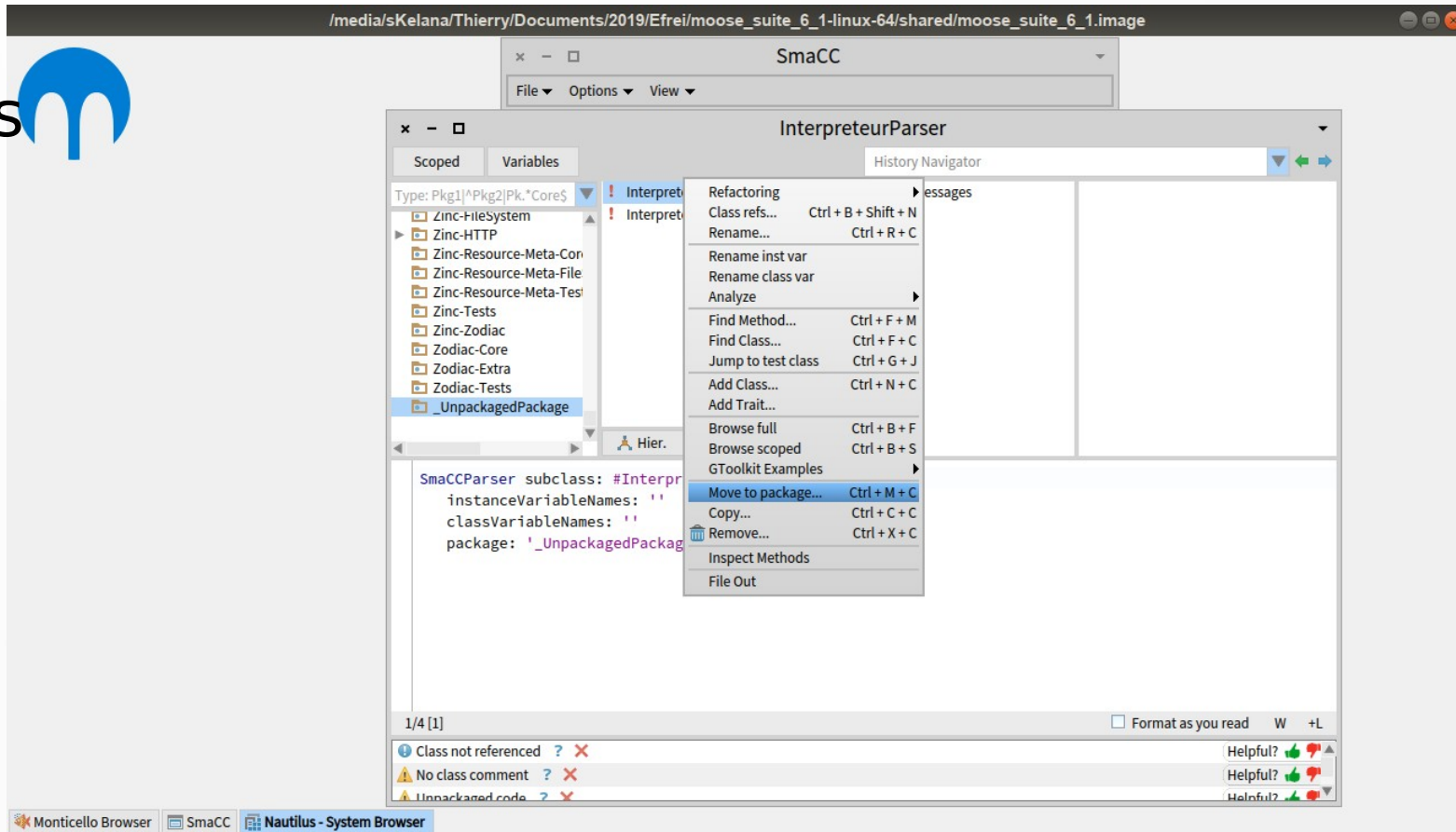
L5: Environnement

- Trouver le code et le déplacer dans le bon package
- World menu
 - System Browser
- Rechercher
 - `_UnpackagedPackage`



L5: Environnement

- Déplacer les deux classes
 - InterpreteurParser
 - InterpreteurScanner
- Vers le package
TPCompilation
- Sélectionner la classe
 - Puis click droit,
Move to package...
 - Saisir
TPCompilation



L5: Environnement

- Revenir à SmaCC
- Ajouter des actions
- File > Save

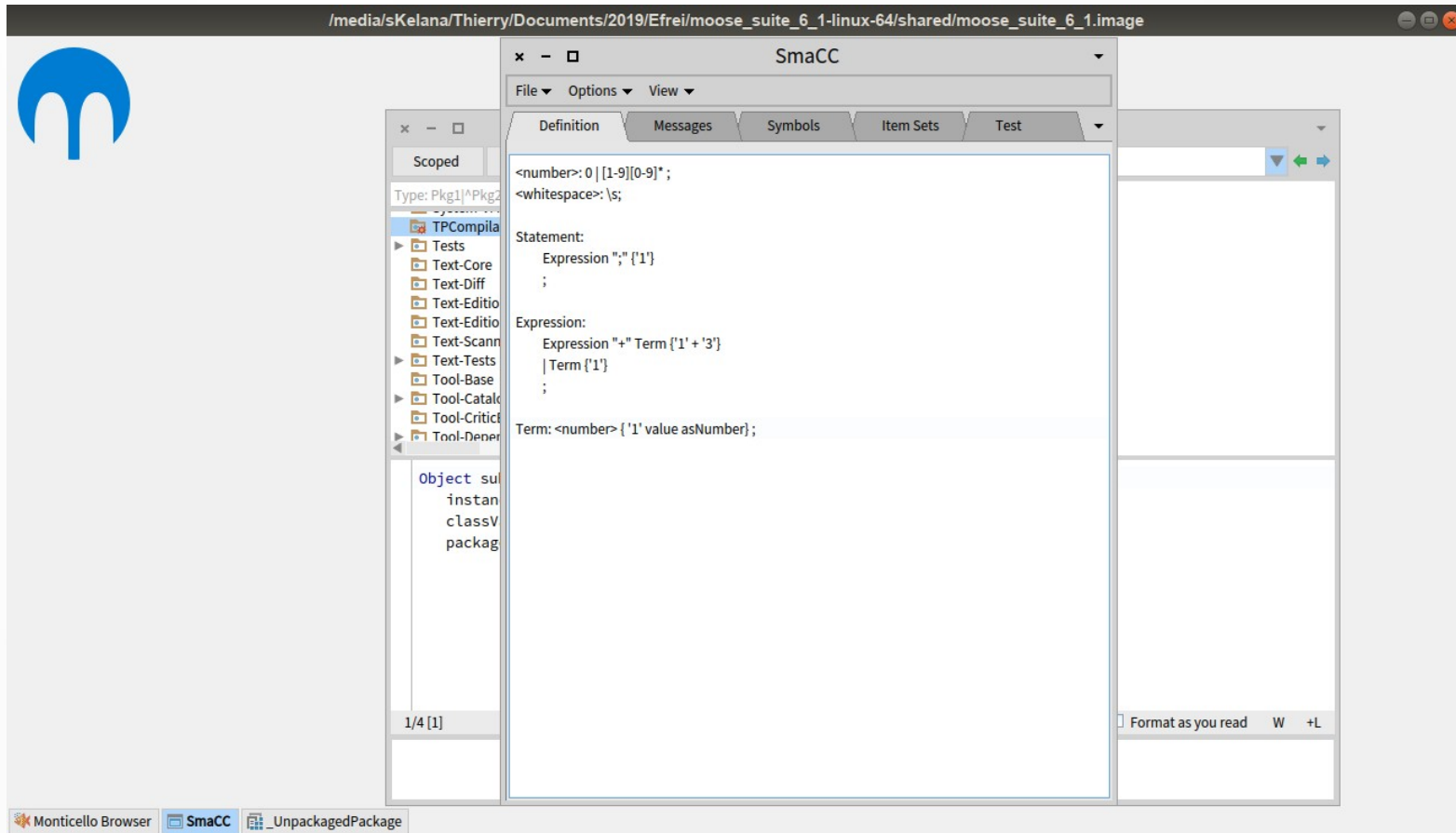
Statement:

```
Expression ";" {'1'}  
;
```

Expression:

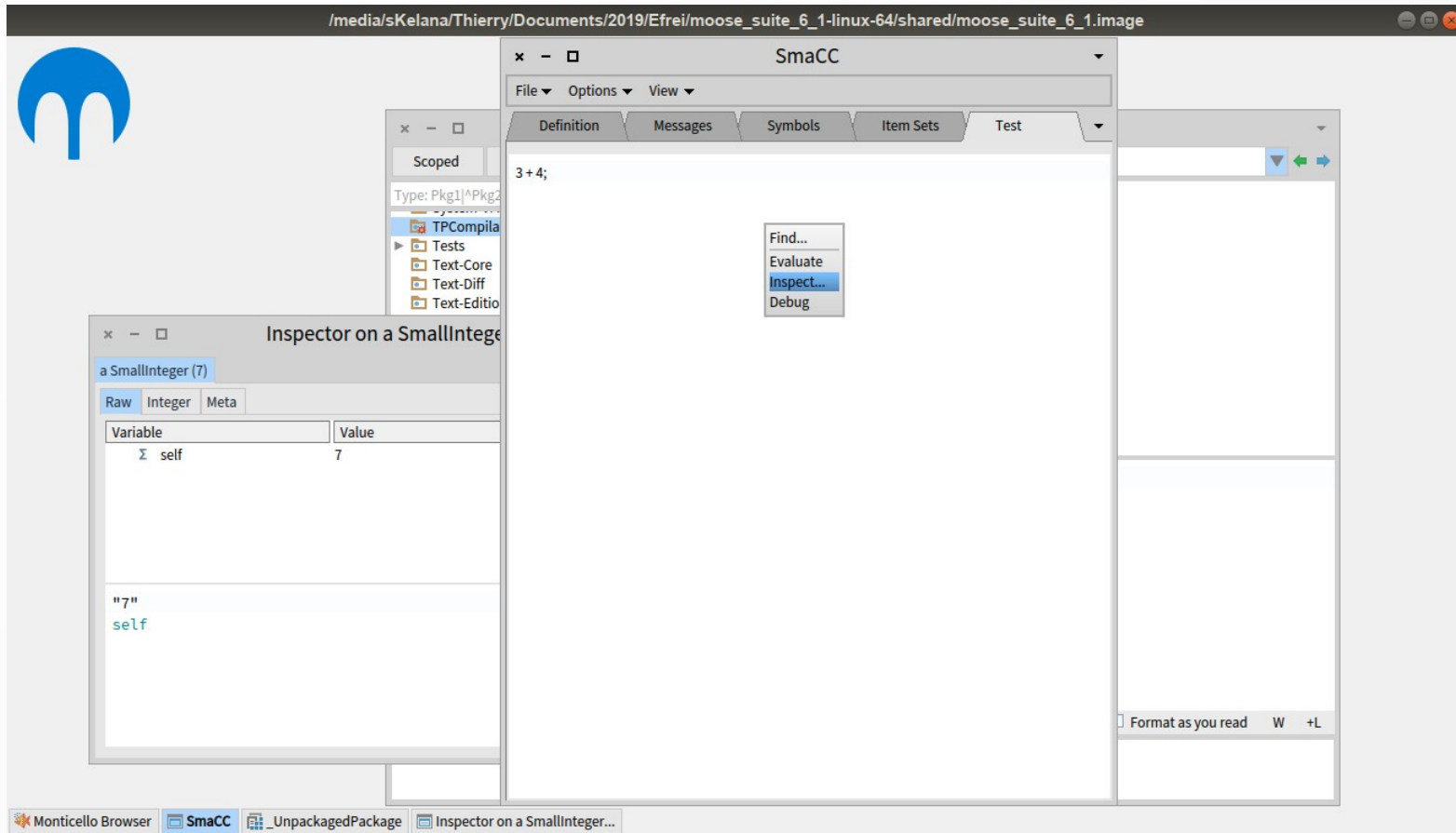
```
Expression "+" Term {'1' + '3'}  
| Term {'1'}  
;
```

Term: <number> { '1' value
asNumber } ;



L5: Environnement

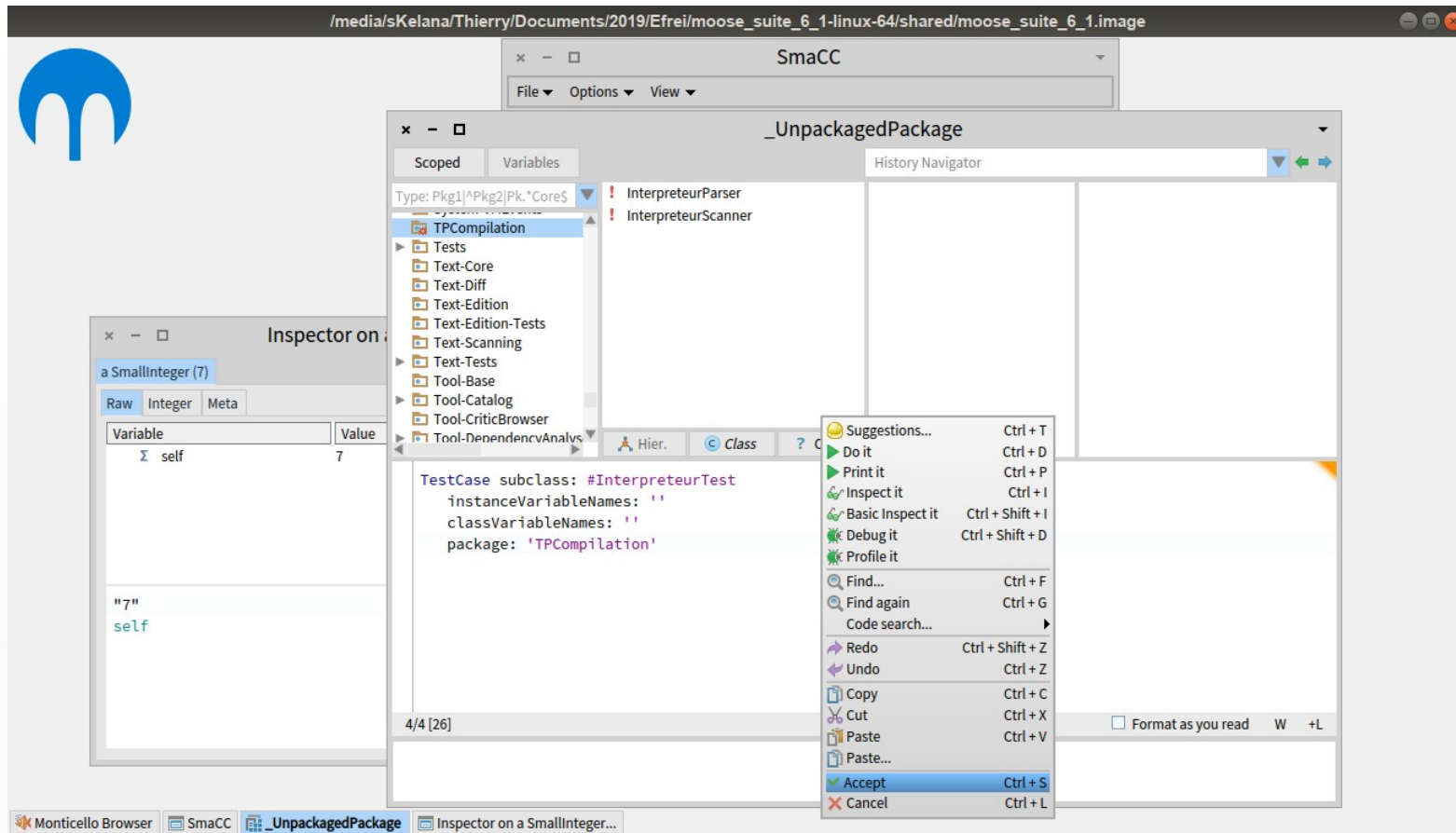
- Tester `3 + 4`; avec
Parse and inspect
- Observer la valeur:
7



L5: Environnement

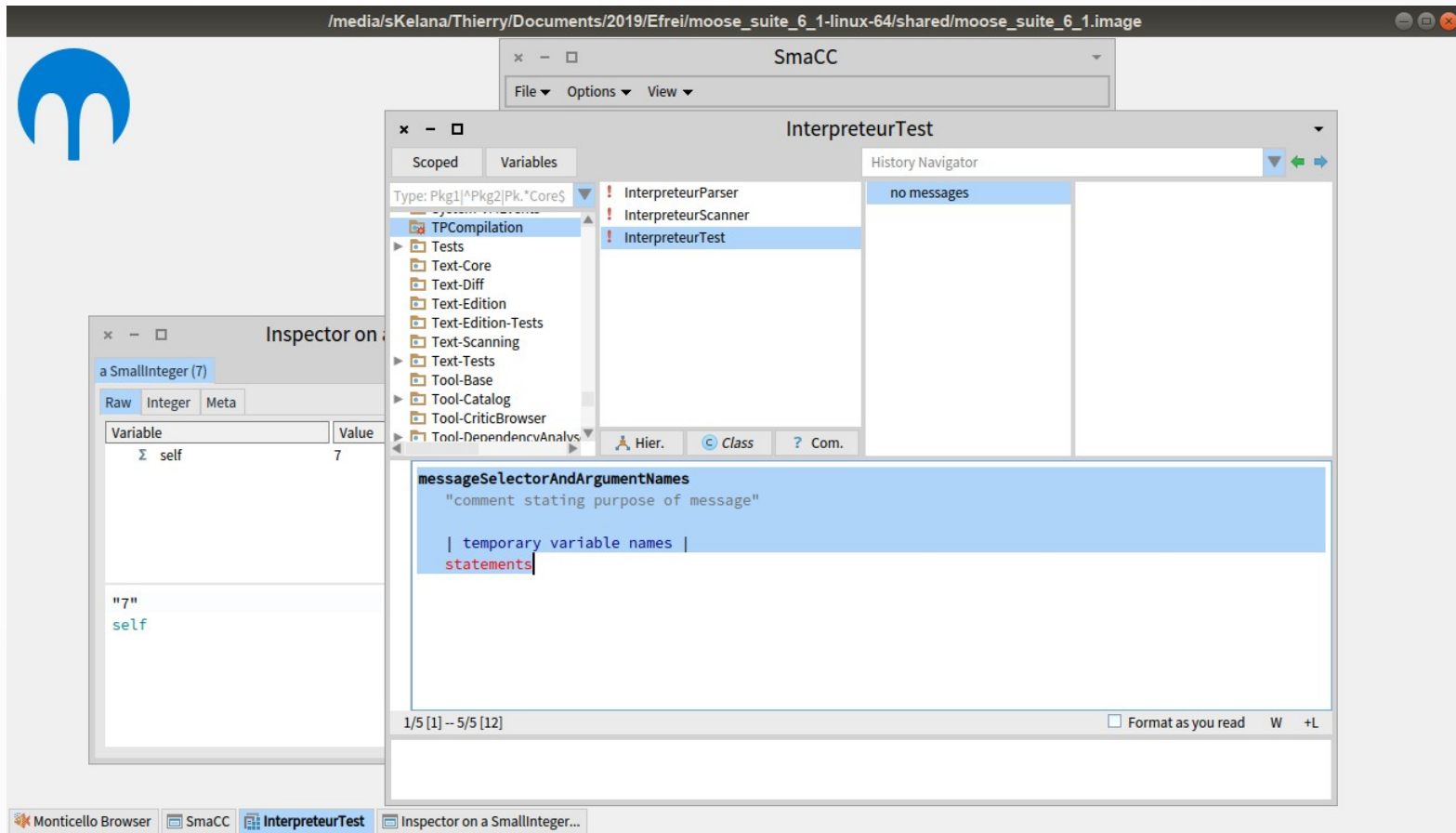
- Ecrire des tests
 - Tab New Class
 - Ecrire une sous-classe de TestCase
 - Et faire Click droit
> accept

```
TestCase subclass:  
#InterpreteurTest  
instanceVariableNames: ''  
classVariableNames: ''  
package: 'TPCompilation'
```



L5: Environnement

- Ecrire le test 1
 - Sélectionner
no message
 - Remplacer le
template de
méthode avec le
code du test.



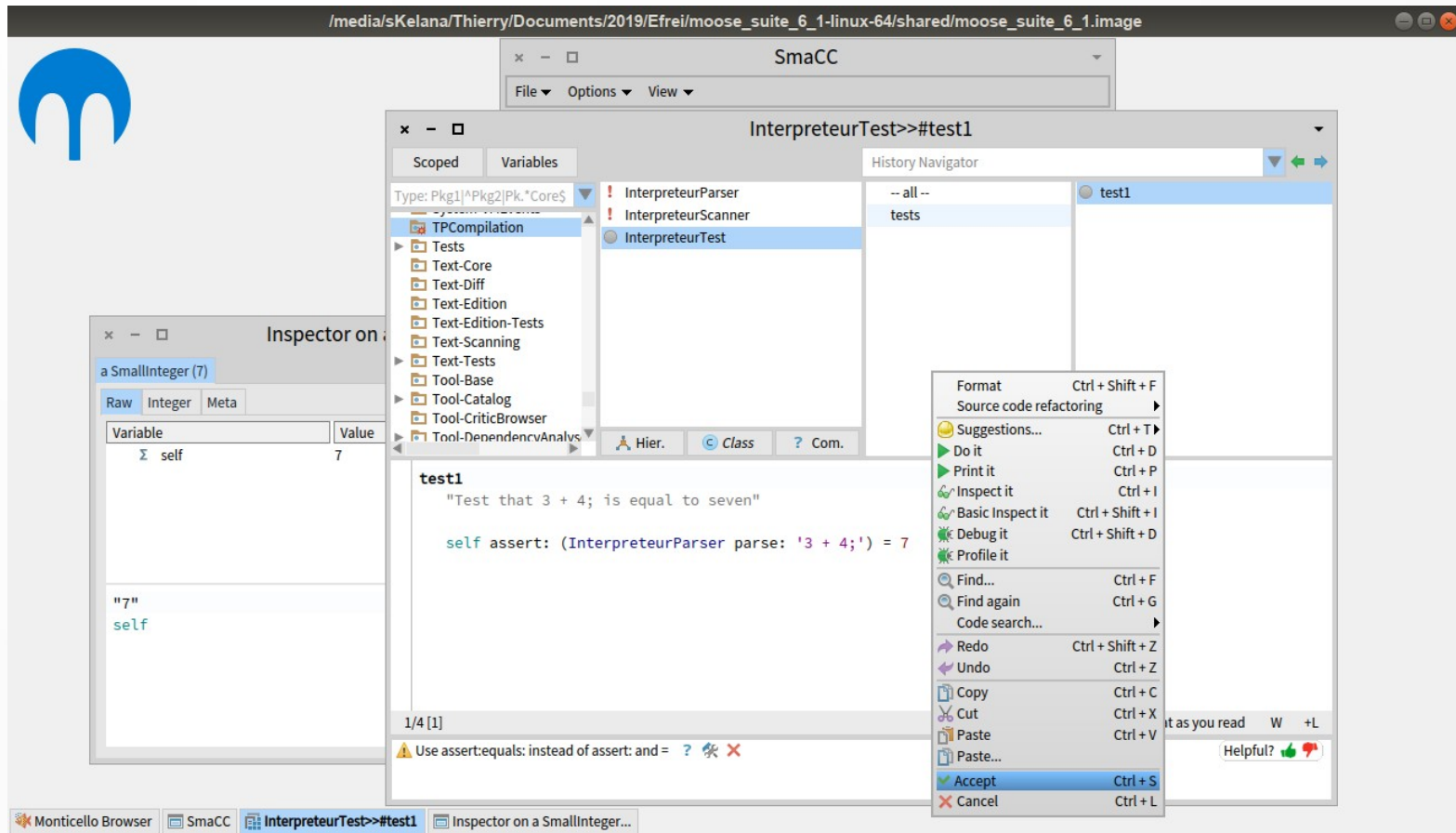
```
test1
```

```
"Test that 3 + 4; is equal to seven"
```

```
self assert: (InterpreteurParser parse: '3  
+ 4;') = 7
```

L5: Environnement

- Ecrire le test 1
 - Sélectionner
no message
 - Remplacer le
template de
méthode avec le
code du test.

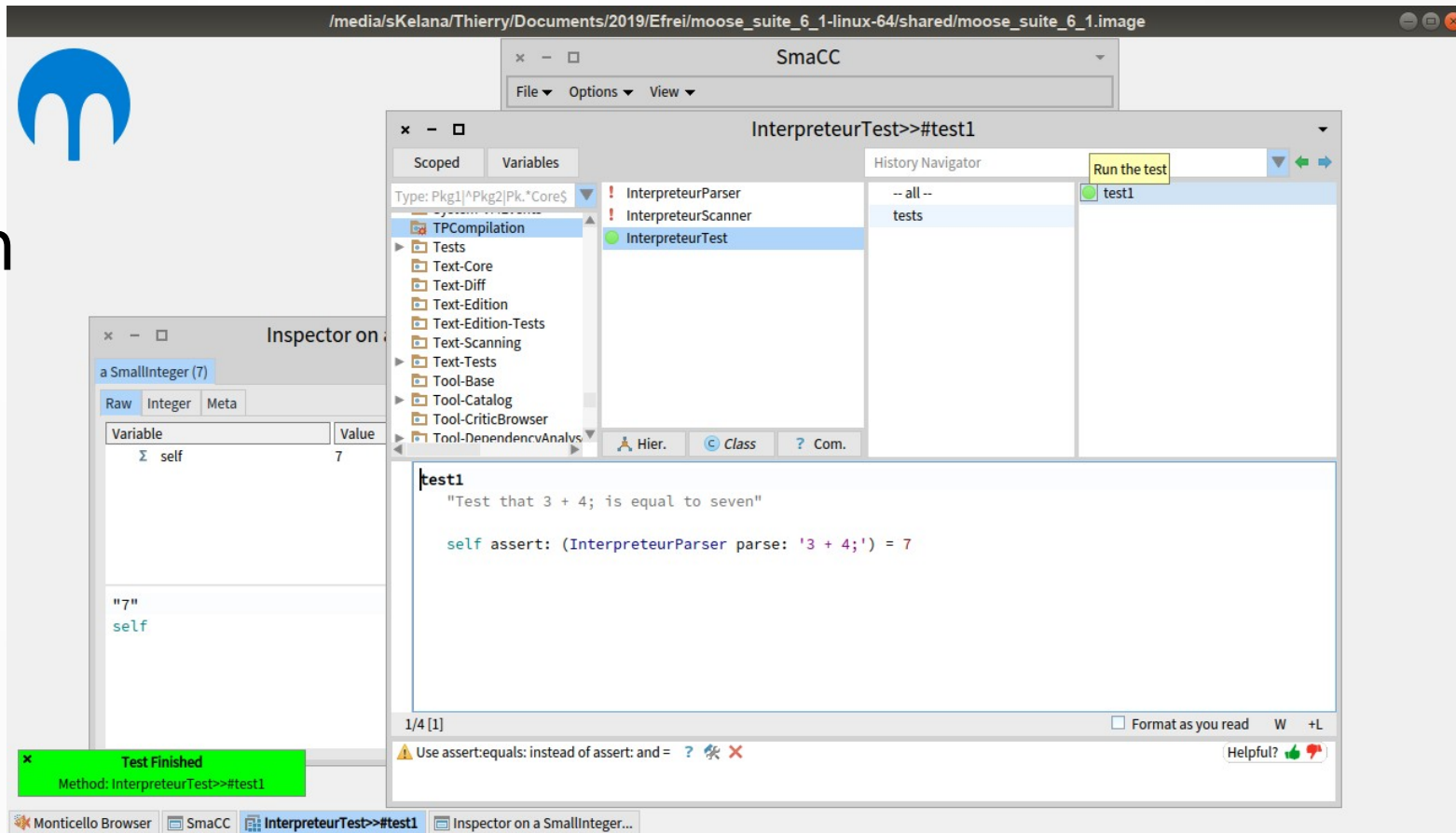


```
test1
    "Test that 3 + 4; is equal to seven"

    self assert: (InterpreteurParser parse: '3
+ 4;') = 7
```

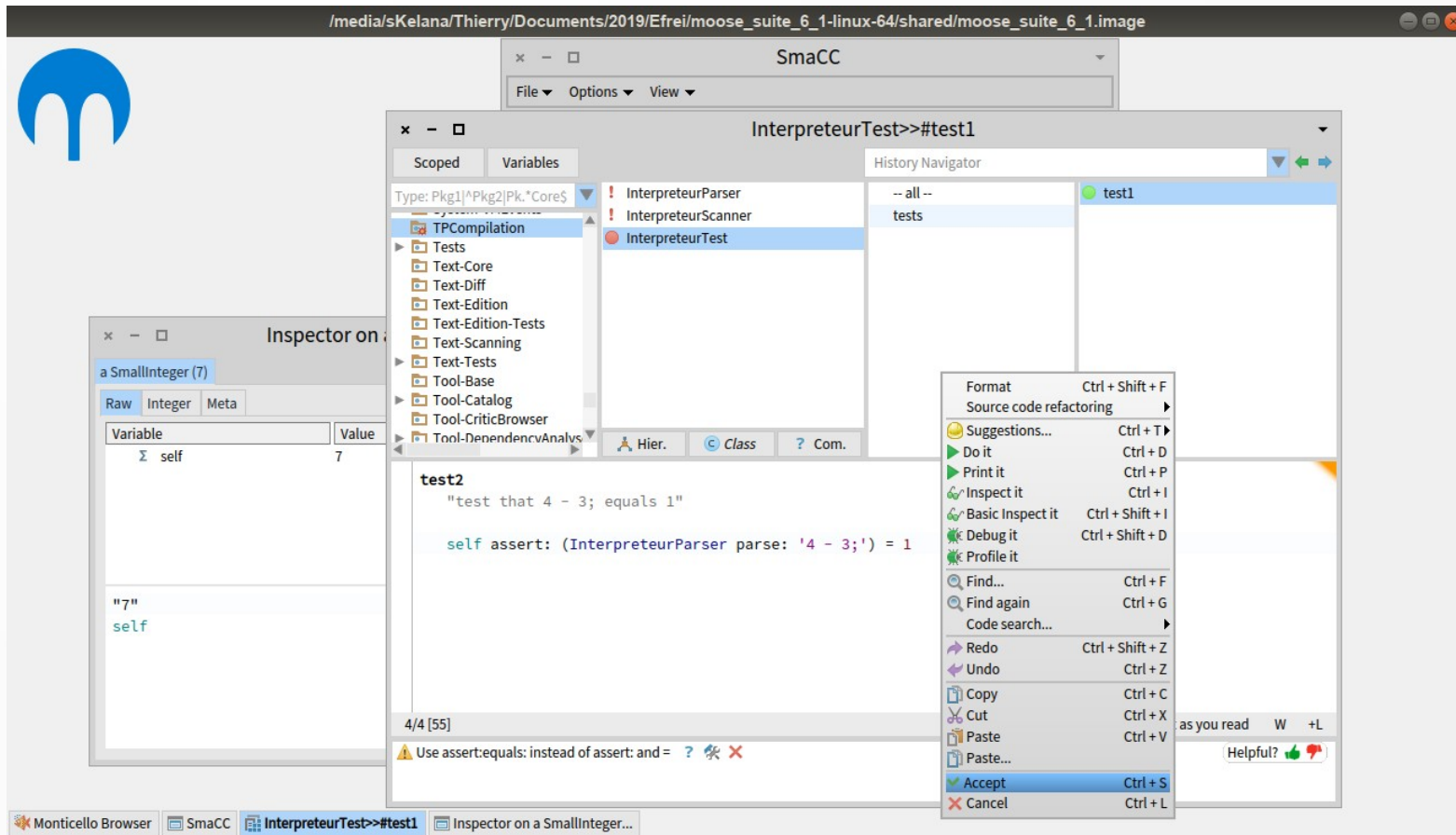
L5: Environnement

- Exécuter le test 1
 - Clicker sur le bouton test1
 - Noter qu'il passe au vert
 - Et qu'un message s'affiche...



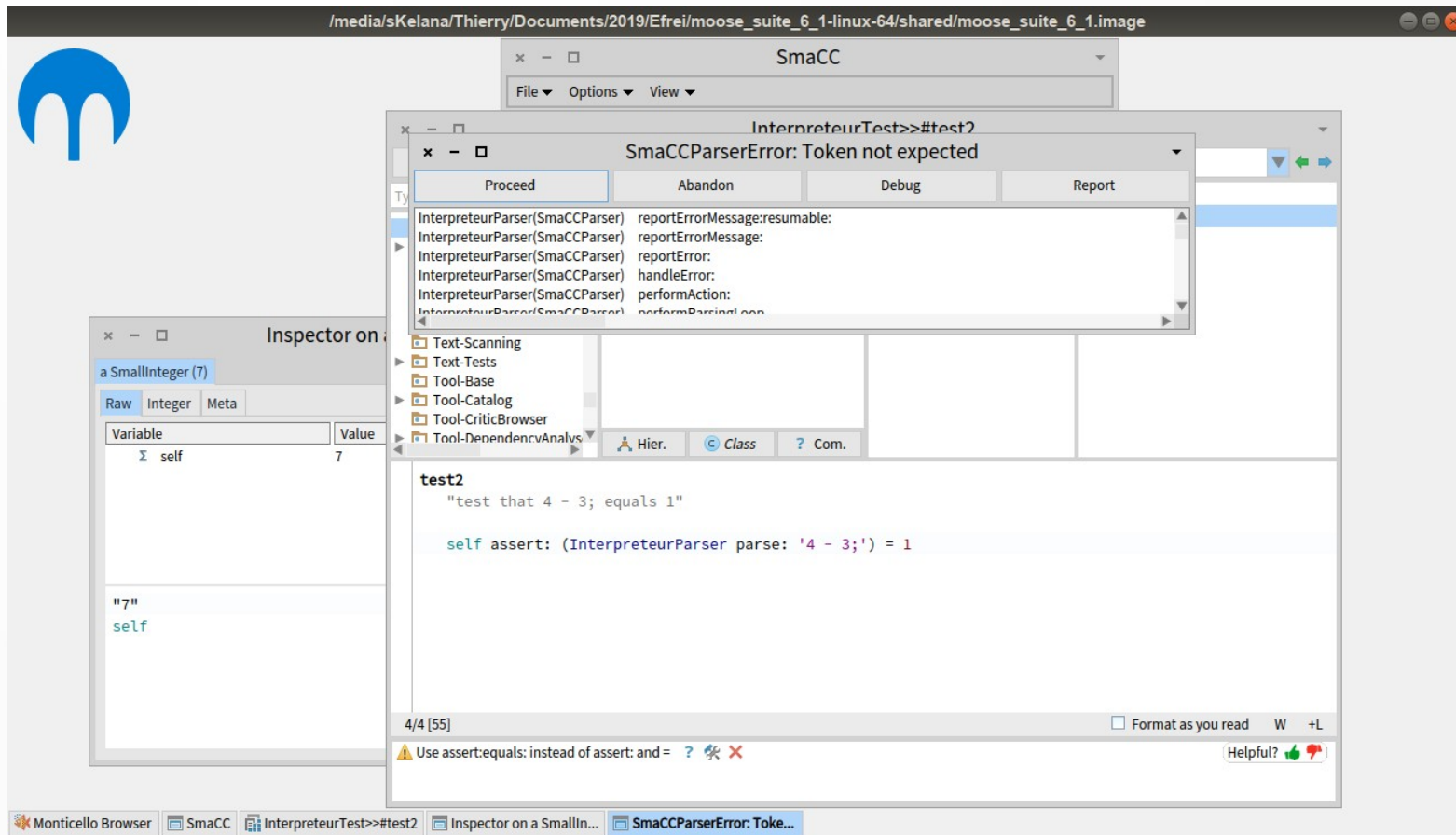
L5: Environnement

- Ecrire le test 2
 - selectionner test1
 - Remplacer et accepter
 - Copier depuis le sujet de TP
 - N'oubliez pas le 'accept'
 - Aussi Ctrl+S



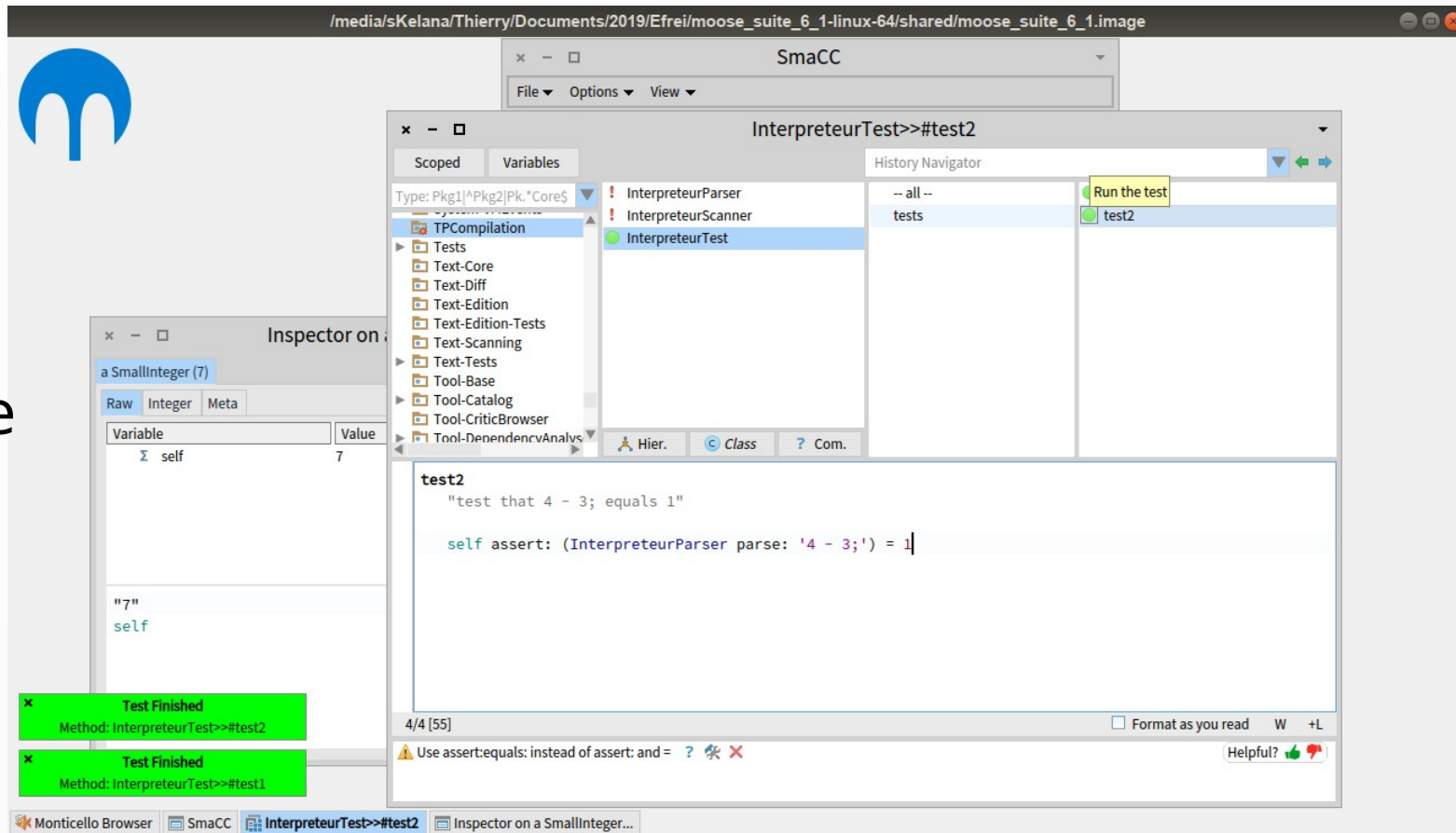
L5: Environnement

- Tester le test 2
 - Appuyer sur le bouton du test
 - Ouverture d'un debugger
 - Avec une erreur de syntaxe
 - Notez le test en rouge



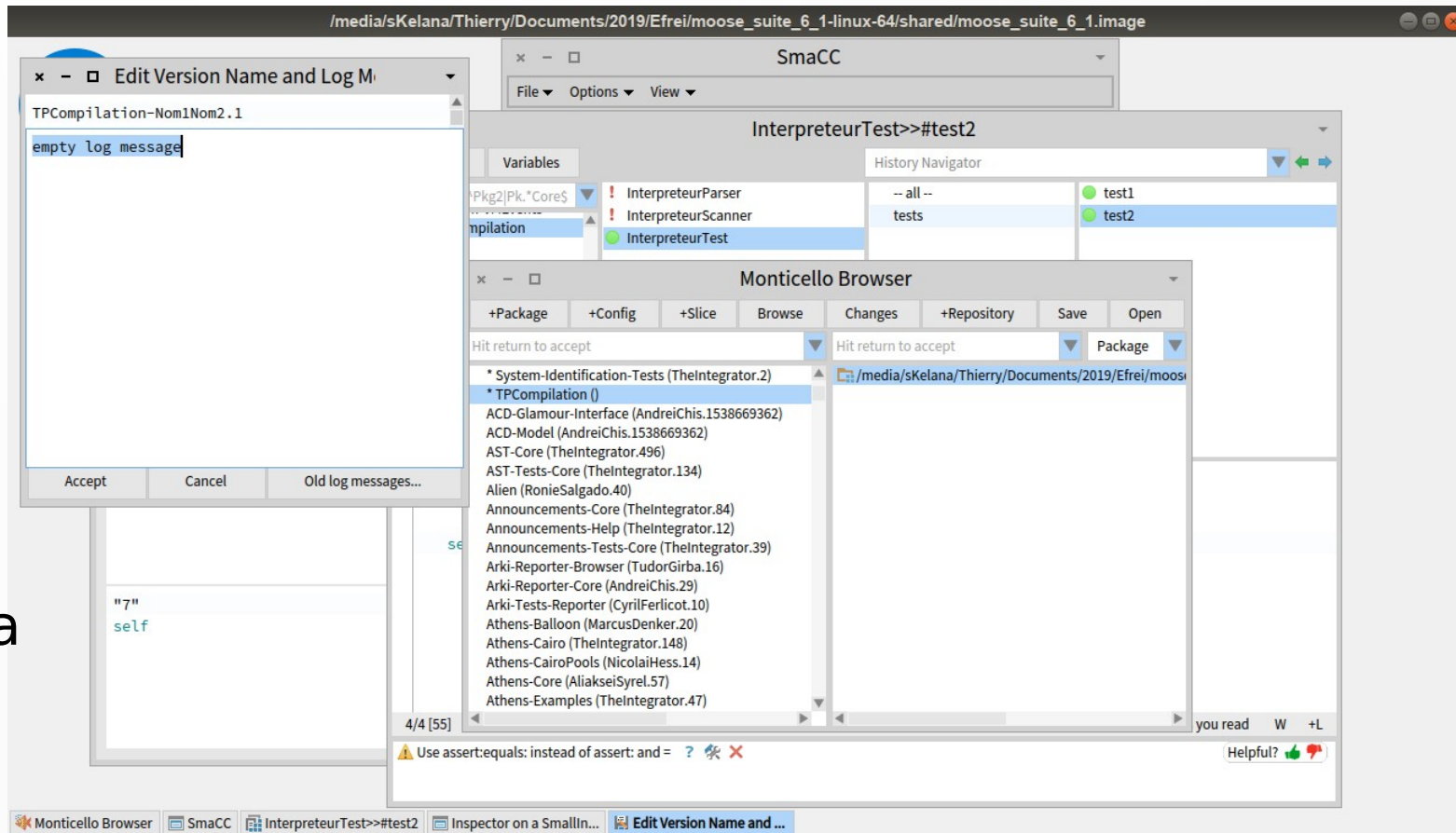
L5: Environnement

- Tester le test 2
 - Enlever le debugger
 - Noter l'échec.
 - Compléter la grammaire
 - Puis refaire
 - File > Save
 - Retester
 - TDD
 - Test-driven development



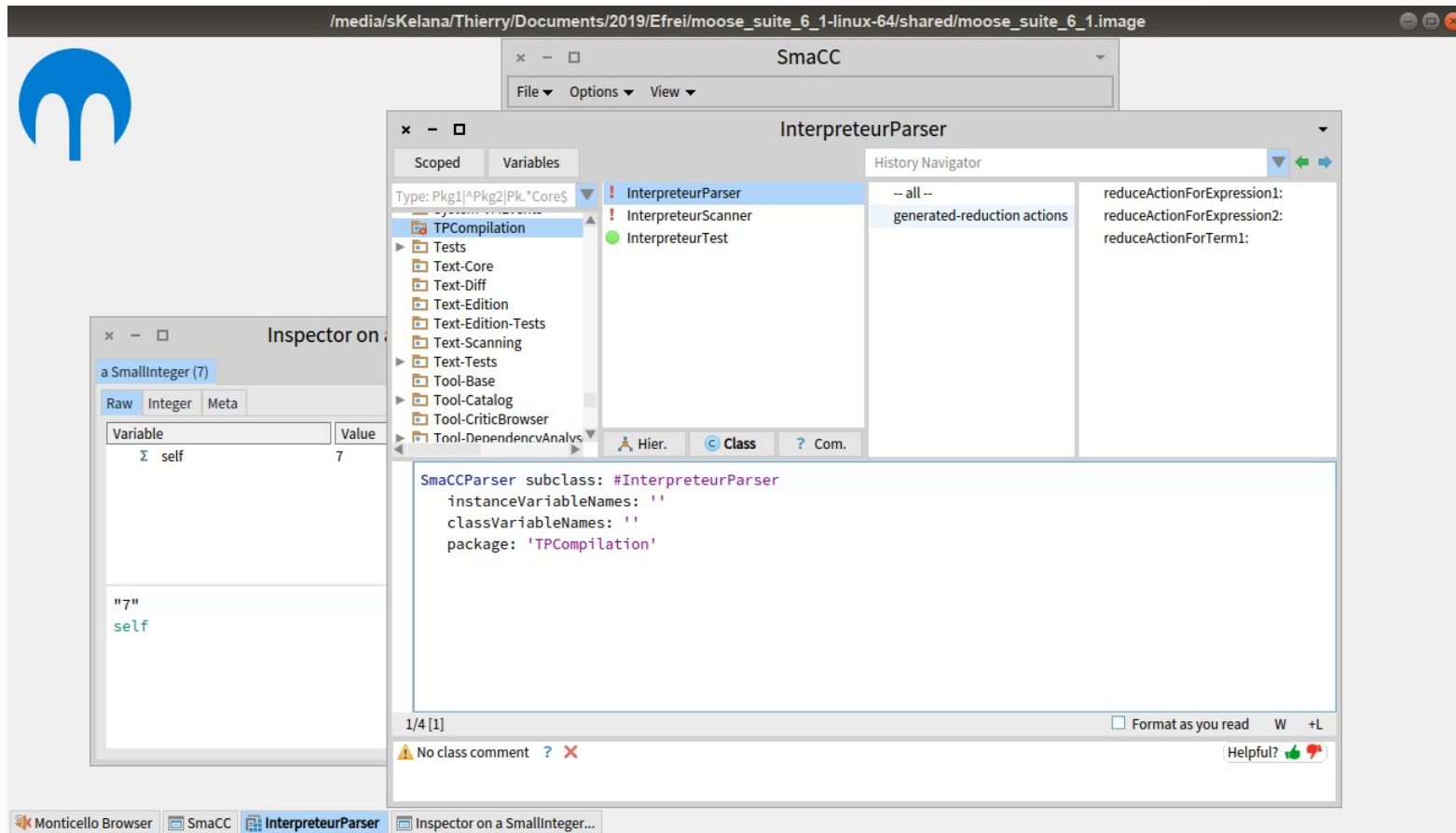
L5: Environnement

- Sauvegarder le TP
 - Ouvrir Monticello
 - Choisir TPCompilation
 - Choisir le repository
 - package-cache
 - Faire Save
 - Vérifier qu'un fichier .mcz a bien été créé.
 - Sauvegarder le .mcz
 - Le soumettre pour être noté



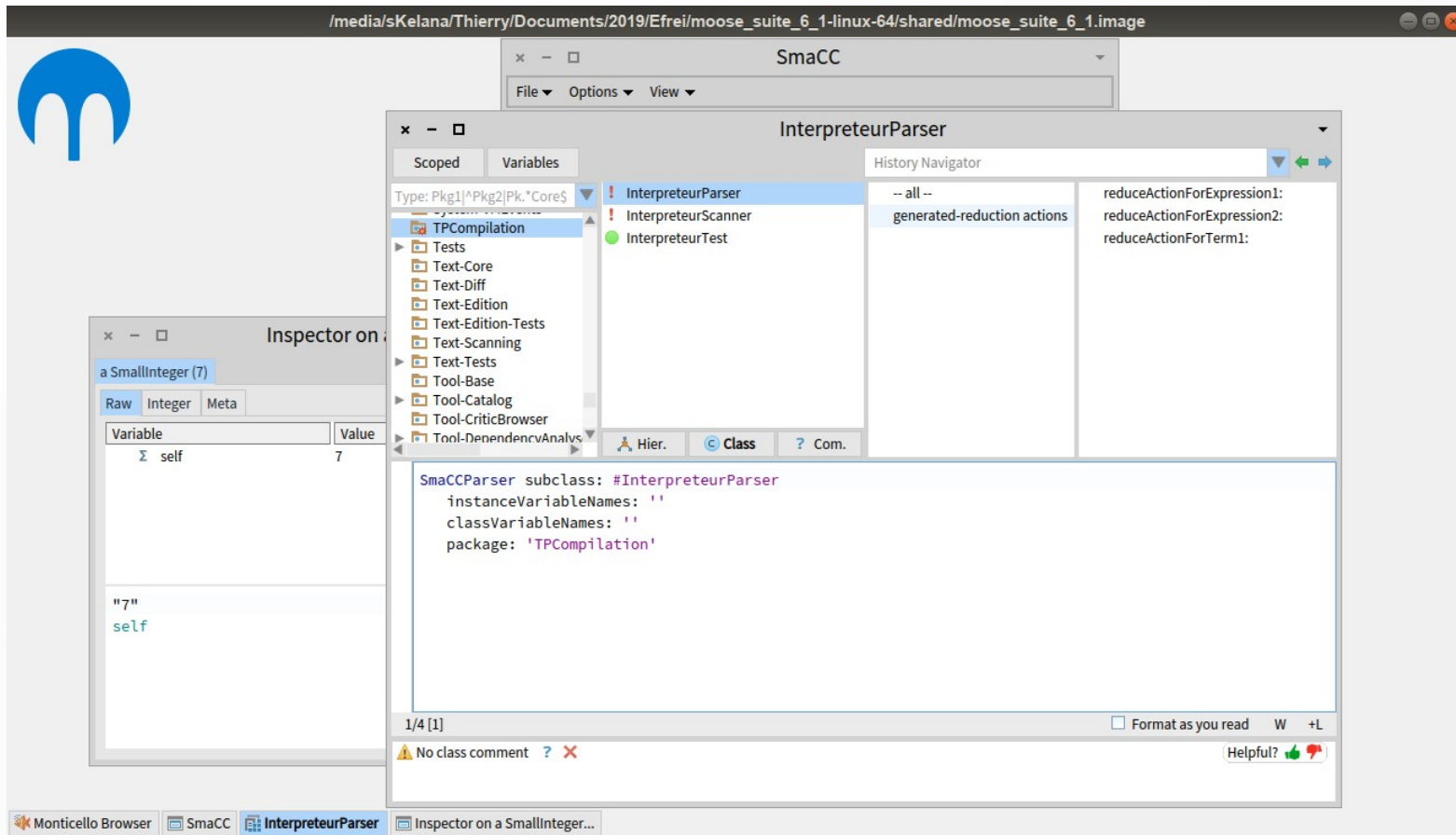
L5: Environnement

- Etendre le parseur
 - A partir du system browser
 - Choisir la classe `InterpreteurParser`
 - La modifier
 - Les modifications seront valides pour toutes les instances du parseur



L5: Environnement

- Etendre le parseur
 - Les actions s'exécutent comme des méthodes de InterpreterParser
 - Elles ont accès à ses variables d'instances
 - Et ses autres méthodes



L5: Environnement

- Deux approches
 - Pour faire l'interpréteur
 - Directe
 - Interprétation au fur et à mesure du parse
 - AST
 - Construire un AST du code pendant le parse
 - Ecrire un évaluateur (visiteur) pour interpréter
- Important
 - Le If/else
 - Les Scopes
 - Les fonctions
- Supposent que vous maîtrisiez l'empilement des contextes d'exécution.

L5: Environnement

- Exemple avec génération d'AST:

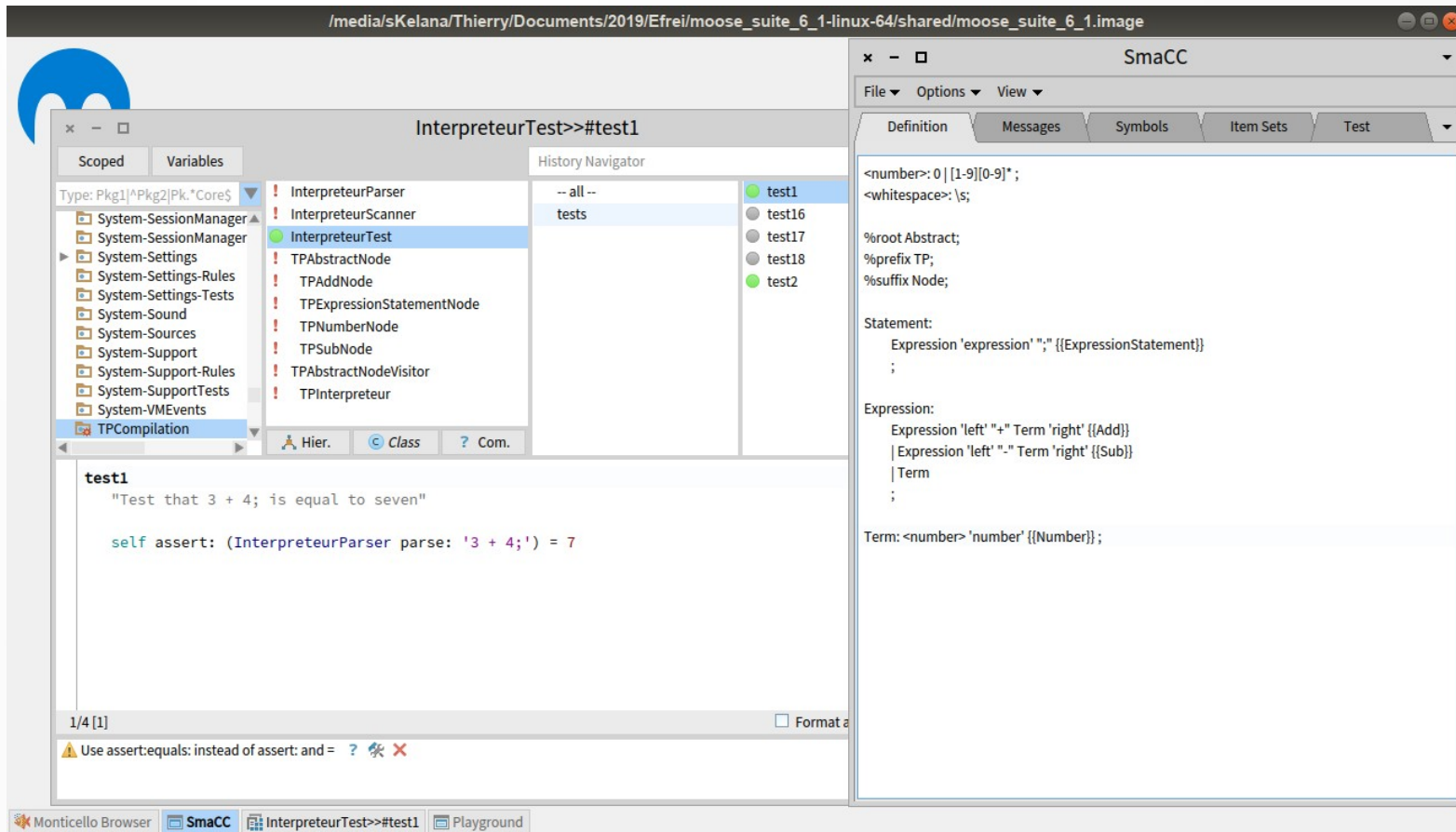
```
<number>: 0 | [1-9][0-9]* ;  
<whitespace>: \s;
```

```
%root Abstract;  
%prefix TP;  
%suffix Node;
```

```
Statement:  
  Expression 'expression' ";"  
  {{ExpressionStatement}}  
  ;
```

```
Expression:  
  Expression 'left' "+" Term 'right'  
  {{Add}}  
  | Expression 'left' "-" Term 'right'  
  {{Sub}}  
  | Term  
  ;
```

```
Term: <number> 'number'  
{{Number}} ;
```



L5: Environnement

- More about SmaCC
 - <https://github.com/SquareBracketAssociates/Booklet-SmaCC>
- More about Pharo / Smalltalk
 - <http://pharo.org/>
 - Look for Documentation >> Pharo by Example