
# Le projet Kitty

Akim Demaille, Roland Levillain

Version 694

2009-05-12 09:03:47 +0200 (Tue, 12 May 2009)

## 1 Spécifications

### 1.1 Kitty 0 : Arithmétique pure

Cette tranche du langage ne supporte que les expressions arithmétiques selon la grammaire suivante.

```
exp -> exp '+' exp      -- Addition
     | exp '-' exp      -- Soustraction
     | exp '*' exp      -- Multiplication
     | exp '/' exp      -- Division
     | '-' exp          -- Moins unaire
     | '(' exp ')'      -- Parenthèses pour grouper
     | INT              -- Entier littéral
```

Tous les opérateurs binaires sont pris associatifs à gauche, avec les priorités suivantes dans l'ordre croissant :

1. `+ -`
2. `* /`
3. – unaire

### 1.2 Kitty 1 : Test

```
exp  →  'if' exp 'then' exp 'else' exp    -- Si-alors-sinon
     | exp '&&' exp                       -- ''Et'' logique paresseux
     | exp '||' exp                       -- ''Ou'' logique paresseux
```

La sémantique des opérateurs est la suivante.

**Booléens**

Kitty repose sur les entiers pour coder les valeurs de vérité. Seul `0` code le faux, tout autre entier (positif ou négatif) représente le vrai

**`if` $e_1$ `then` $e_2$ `else` $e_3$**

Si $e_1$ s'évalue en vrai, alors évaluer $e_2$, et renvoyer sa valeur, sinon évaluer $e_3$ et en renvoyer la valeur.

**$e_1$ `&&` $e_2$**

Si $e_1$ s'évalue en faux, renvoyer 0, sinon si $e_2$ est vrai, renvoyer 1, sinon 0.

**$e_1$ `||` $e_2$**

Si $e_1$ s'évalue en vrai, renvoyer 1, sinon si $e_2$ est vrai, renvoyer 1, sinon 0.

Les priorités sont les suivantes, croissantes :

1. `if`
2. `||` associatif à gauche
3. `&&` associatif à droite
4. arithmétique

### 1.3 Kitty 2 : Comparaisons

Les opérateurs suivants sont non-associatifs (i.e., par exemple `0 < 1 < 2` n'est pas une phrase Kitty valide). Les comparaisons s'évaluent en `0` pour faux, ou bien en `1` pour vrai.

```
exp → exp '=' exp    -- Test d'égalité
    | exp '<>' exp   -- Test de différence
    | exp '<' exp    -- Strictement plus petit
    | exp '<=' exp   -- Plus petit ou égal
    | exp '>' exp    -- Strictement plus grand
    | exp '>=' exp   -- Plus grand ou égal
```

Les priorités sont les suivantes, croissantes :

1. tests (`if`, `||`, `&&`)
2. `<`, `<=`, `>`, `>=`
3. `=`, `<>`
4. arithmétique

### 1.4 Kitty 3 : Chaînes littérales

Un deuxième type de données est supporté : les chaînes de caractères. Aucune opération nouvelle, mais la sémantique est différente :

- Toutes les comparaisons sont supportées entre deux chaînes de caractères.
- Des opérations "arithmétiques", seule l'addition fonctionne entre (deux) chaînes de caractères. Elle retourne alors la concaténation de ces chaînes.

Il est expressément demandé dans le cadre de ce travail d'implémenter le typage de l'arbre en plus de l'affichage de l'arbre et de son évaluation. Un arbre invalide d'après les règles de typage ne doit jamais être évalué.

## 2 Travail demandé

### 2.1 Interprète Kitty

Il s'agit d'implémenter un interprète pour l'un des langages Kitty. Chaque tranche supplémentaire du langage relève la note maximale que l'on peut obtenir. Bien entendu, un projet Kitty-1 qui marche parfaitement sera mieux noté qu'un Kitty-2 qui marche mal.

L'interprète Kitty est interactif : il attend une expression Kitty complète par ligne (une ligne se termine par le caractère retour à la ligne).

Cette ligne doit être parsée pour produire un arbre de syntaxe abstraite. Cette expression *exp* doit être évaluée en une valeur *val*, puis l'interpréteur doit afficher '*exp*' `: integer =>` *val*' ou *exp*' est une expression qui peut-être différente de *exp*, mais qui doit lui être équivalente. La partie ' `: integer`' est bien sûr sans intérêt pour les premiers niveaux de Kitty, mais comme elle sert à partir de Kitty-3, elle est obligatoire.

Les erreurs de syntaxe ne doivent pas conduire à la terminaison de l'interpréteur.

### 2.2 Outils

Les outils suggérés/utilisés sont les suivants :

**JFlex** Générateur de scanners, `http://jflex.de/`.

**Bison** Générateur de parseurs,
`http://www.lrde.epita.fr/~akim/download/bison-2.4.266-4ff3b.tar.bz2`.

**GNU Make** Une implémentation de Make, `http://www.gnu.org/software/make/`.

Toute ou partie de ces outils est disponible sur vos machines de travail.

Vous êtes libres d'utiliser d'autres outils, mais vous serez moins encadrés. L'installation de ces différents outils devrait être très simple. Pour installer Bison sur votre compte :

```
> wget http://www.lrde.epita.fr/~akim/download/bison-2.4.266-4ff3b.tar.bz2
> tar jvf bison-2.4.266-4ff3b.tar.bz2
> cd bison-2.4.266-4ff3b
> ./configure --prefix=$HOME/local
> make
> make install
```

Si vous êtes administrateur sur votre machine, alors vous pouvez utiliser la séquence suivante :

```
> wget http://www.lrde.epita.fr/~akim/download/bison-2.4.266-4ff3b.tar.bz2
> tar jvf bison-2.4.266-4ff3b.tar.bz2
> cd bison-2.4.266-4ff3b
> ./configure
> make
> sudo make install
```

Pour JFlex :

```
> wget http://jflex.de/jflex-1.4.3.tar.gz
> sudo tar -C /usr/local/share xvzf jflex-1.4.3.tar.gz
> sudo ln -s /usr/local/share/jflex/bin/jflex /usr/local/bin/jflex
```

## 2.3 Rendu

Tout rendu qui n'est pas fait avec '`make distcheck`' a toutes les chances d'avoir 0/20. Je veux pouvoir faire '`make check`' dans votre projet et voir que vous avez écrit de nombreux tests. Il s'agit d'ajouter des fichiers (vous pouvez modifier celui qui est donné en exemple) dans le répertoire '`tests/`'. N'oubliez pas de mettre à jour la variable Make *`TESTS`* dans le '`Makefile`'.

## 2.4 Implémentation

Une architecture de base vous est fournie pour vous donner un point de départ. Vous avez la charge de compléter les éléments suivants :

**scanner** Toutes les règles nécessaires ne sont pas fournies.

**parseur** Idem.

**désucrage** Réfléchissez bien : toutes les règles de syntaxe/tous les opérateurs ne nécessitent pas d'exister en tant que tel dans l'AST. Par exemple, nul besoin d'implémenter dans l'ast de support pour le moins unaire.

**ast** Certaines classes sont esquissées, certaines fonctionalités sont déjà implémentées, d'autres n'ont même pas de signature.

# 3 Exemple de session

Dans l'exemple qui suit, les lignes qui commencent par > sont les entrées de l'utilisateur. Il n'est pas exigé d'afficher le caractère d'invitation (>). Voir également les fichiers de tests.

## 3.1 Kitty 2

```
> 1+2
(1+2) : integer => 3
> -23
(0-23) : integer => -23
> 1++2
syntax error, unexpected '+', expecting number or '-' or '('
> 2+2=2*2
((2+2)=(2*2)) : integer => 1
```

## 3.2 Kitty 3

```
> "1"+"2"
("1"+"2") : string => "12"
> 1+"2"
invalid use of "+"
```