1.1.4 Le test

Le *test* est une construction qui permet de former une instruction à partir d'une expression booléenne `b` et de deux instructions p_1 et p_2. En Java, cette instruction s'écrit `if (b)` p_1 `else` p_2.

Pour exécuter l'instruction `if (b)` p_1 `else` p_2 dans un état `s`, on commence par calculer la valeur de l'expression `b` dans l'état `s`, puis selon que cette valeur est `true` ou `false`, on exécute p_1 ou p_2 dans l'état `s`.

En Caml, cette instruction s'écrit `if b then` p_1 `else` p_2. *En C, elle s'écrit comme en Java.*

1.1.5 La boucle

La *boucle* est une construction qui permet de former une instruction à partir d'une expression booléenne `b` et d'une instruction `p`. En Java, cette instruction s'écrit `while (b) p`.

Pour exécuter l'instruction `while (b) p` dans un état `s`, on commence par calculer la valeur de `b` dans l'état `s`. Si cette valeur est `false`, on a terminé. Sinon, on exécute `p`, puis on recalcule `b` dans le nouvel état. Si cette valeur est `false`, on a terminé. Sinon, on exécute `p`, puis on recalcule `b` dans le nouvel état. Si cette valeur est `false`, on a terminé. Sinon, on exécute `p`, puis on recalcule `b` dans le nouvel état, ... Et ce processus se poursuit jusqu'à ce que la valeur de `b` soit `false`.

Avec cette construction apparaît un nouveau comportement possible pour une instruction : la *non-terminaison*. En effet, si l'expression booléenne `b` vaut éternellement `true`, alors l'instruction `p` s'exécute à l'infini et l'exécution de l'instruction `while (b) p` ne termine pas. C'est par exemple le cas de l'instruction

```
int x = 1;
while (x >= 0) {x = 3;}
```

En introduisant une instruction fictive `skip ;` dont l'exécution ne fait rigoureusement rien, on peut définir l'instruction `while (b) p` comme une abréviation pour l'instruction infinie

```
if (b) {p if (b) {p if (b) {p if (b) ...
                                      else skip;}
                            else skip;}
                  else skip;}
       else skip;
```