

Magnifier

Un outil d'inspection et d'aide à la compréhension des
problèmes d'arithmétique flottante.

The logo for 'numalis' is rendered in a bold, blue, sans-serif typeface. The letter 'n' is stylized with a thick stroke and a small dot above it. The 'a' is a simple, rounded shape. The 'l' is a single vertical stroke. The 'i' has a dot above it. The 's' is a simple, rounded shape. The overall design is clean and modern.

Mars 2018

Table des matières

1	Introduction	2
1.1	A propos du Magnifier	2
1.2	Un premier exemple	3
1.3	Gestion des boucles	5
1.4	Gestion des fonctions	6
2	Les Graphiques	7
2.1	Création d'un graphique	7
2.1.1	Exemple	7
2.2	Gestion des indices	8
2.2.1	Boucle simple	8
2.2.2	Boucle imbriquée	9
2.2.3	Représentation des valeurs spéciales	10
3	Perturbations	12
3.1	Exécution avec perturbations	12
3.1.1	Effectuer une perturbation avec l'éditeur HSL	12
3.1.2	Effectuer des perturbation avec le Magnifier	13
3.2	Perturbations graphics	13
4	Gradients	14
4.1	Les types de gradients	14
5	Formatage	15
6	Autorun	16
7	Syntaxe du langage HSL	17
7.1	Généralités du langage	17
7.2	Variables	17
7.3	Opérateurs	18
7.4	Structure de contrôle	19
7.4.1	Instruction If	19
7.4.2	Instruction For	19
7.4.3	Instruction While	20
7.4.4	Fonctions	21
7.4.5	Instruction With Prec	24
8	Glossaire	25

1 Introduction

1.1 A propos du Magnifier

Le Magnifier est un outil qui vise à améliorer la compréhension du comportement d'algorithmes. Le principal objectif est de surveiller l'apparition et la propagation d'erreurs sur les opérations en arithmétique flottante.

A l'heure actuelle, le Magnifier est restreint aux programmes écrits dans le langage HSL. Le Magnifier peut être utilisé comme un éditeur de code, mais il utilise un outil externe pour effectuer la compilation et l'exécution de programme.

Le rôle du Magnifier est de permettre à l'utilisateur de visualiser graphiquement les points d'intérêts d'un programme (variable, opérateur, fonction...). Pour chaque élément assimilable à une expression, l'utilisateur peut consulter toutes les valeurs et les erreurs qui lui ont été associés au cours de l'exécution du programme.

Le Magnifier propose plusieurs fonctionnalités telles que :

- L'exécution d'un programme en utilisant deux précisions : une précision de type primitive classique, utilisant soit le type float (24 bits de mantisse), soit le type double(53), et une précision arbitraire entre 5 et 65536 bits de mantisse.
- La surveillance des valeurs de tous les éléments du programme à n'importe quel moment de l'exécution. Ainsi que la comparaison avec la valeur de référence que prend cet élément dans la précision arbitraire. Il y a deux types d'erreurs qui peuvent être affichées. L'erreur brute (la différence entre la valeur de référence et la valeur constatée) ou bien l'erreur relative (pourcentage de l'erreur par rapport à la valeur de référence).
- La recherche automatique des éléments possédant la plus grande erreur.
- L'analyse de la sensibilité du programme par rapport aux perturbations de données en entrée.
- La génération de graphique représentant l'évolution des valeurs ou des erreurs d'un élément lors de l'exécution du programme.
- La simulation de l'exécution d'un programme pour avoir un aperçu du chemin d'exécution.

1.2 Un premier exemple

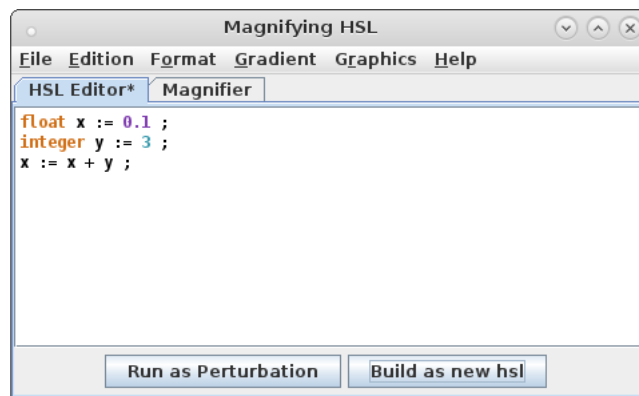
Le Magnifier peut être lancé via le raccourci ou en ligne de commande. Il accepte optionnellement en argument un fichier HSL. Une fois lancé, l'outil ouvre deux onglets.

Le premier est une zone de saisie qui fonctionne comme un éditeur classique. Le deuxième va permettre l'analyse du code saisi.

Pour pouvoir commencer à utiliser l'outil, il est nécessaire de saisir et d'exécuter un premier programme. Dans l'onglet "HSL Editor", nous pouvons saisir le programme suivant :

```
float    x := 0.1 ;
integer y := 3 ;
x := x + y ;
```

Pour compiler et exécuter le programme, il suffit de cliquer sur le bouton "Build as new Hsl" en bas à droite.



Le Magnifier va exécuter le HSL et la version enrichie par les résultats de l'exécution va être chargée dans l'onglet Magnifier. En cliquant sur cet onglet, on peut maintenant observer plusieurs informations intéressantes :

En plaçant le curseur sur chaque élément, on peut observer la valeur qu'il prend. On s'aperçoit notamment que le literal "0.1" n'a pas la même valeur selon la précision sélectionnée. En effet la valeur 0.1 n'est pas représentable de manière finie en arithmétique flottante. Même la valeur en précision 512 ne fait que l'approximer.

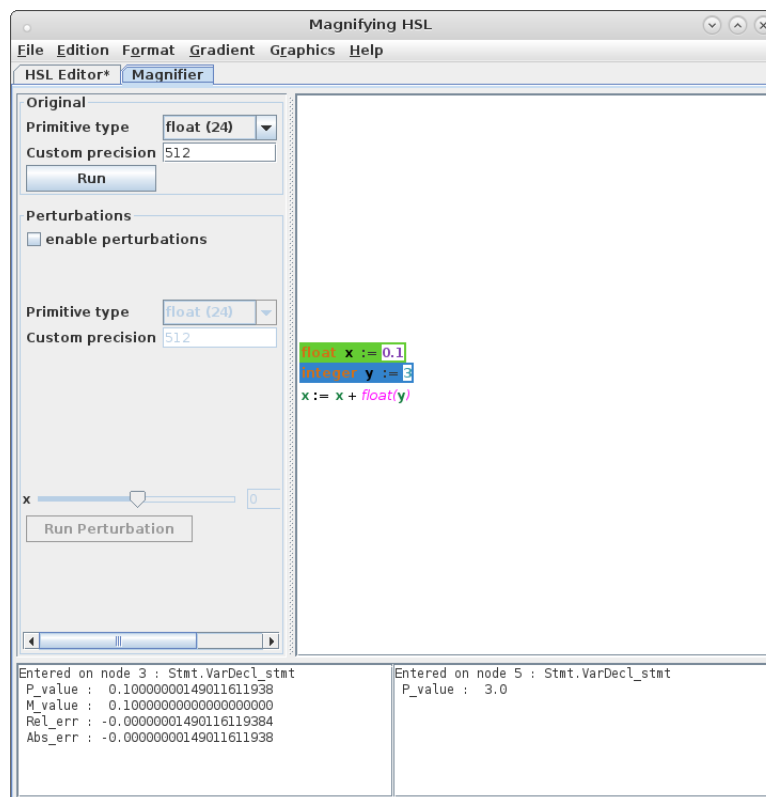
On peut modifier le nombre de décimales affichées dans le menu "Format" (voir la section consacrée).

Il est possible de verrouiller la sélection d'un élément en cliquant dessus. Le Magnifier propose deux sélections possibles, une avec le clic gauche (l'élément sera colorié en vert) et une avec le clic droit (l'élément sera colorié en bleu). Si l'utilisateur sélectionne le même élément avec les deux sélections, la deuxième sélection permet de faire une comparaison entre la valeur initiale de l'élément et la valeur que celui-ci prend dans la version perturbée (voire la section sur les perturbations).

Sur cet exemple, on peut remarquer que les entiers n'ont pas d'erreurs. En revanche les variables ou les opérateurs portent également des erreurs dès que des nombres flottants sont manipulés. Dans le cas d'une opération entre un nombre flottant et un entier, on constate que l'opérateur porte une erreur.

Dans la plupart des langages, une opérations entre deux types de nombres différents provoque une conversion (cast) de l'opérande de l'opération de précision moindre vers le type de l'autre opérande.

Ici le type entier est implicitement converti en flottant avant l'opération. Cette transformation implicite peut être mise en évidence en activant l'option "show implicit cast" dans le menu d'édition.



1.3 Gestion des boucles

Le Langage HSL propose les boucles 'For' et les boucles 'while'. Dans les deux cas, lors de l'exécution du programme, le Magnifier va récupérer les valeurs de toutes les opérations qui ont été effectuées dans le corps de la boucle, et ce pour toutes les itérations réalisées.

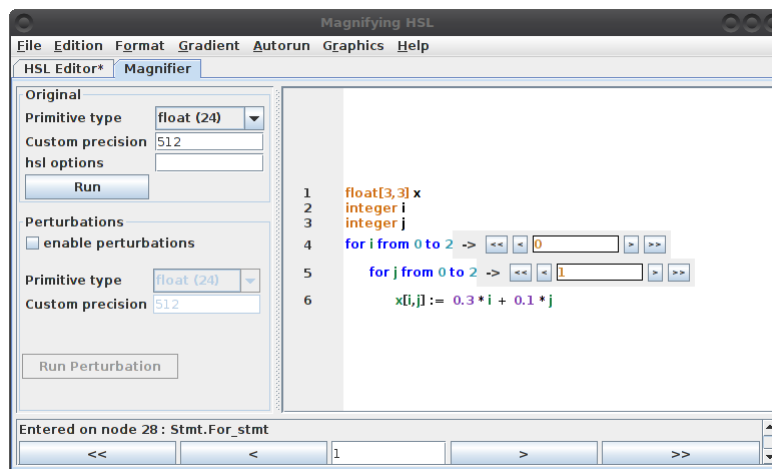
```
float[3,3] x;
integer i;
integer j;

for i from 0 to 2 :
  for j from 0 to 2 :
    x[i,j] := 0.3 * i + 0.1 * j ;
```

Afin de pouvoir sélectionner les indices de la boucle, La vue du Magnifier propose des boutons et une zone éditable à droite de la condition pour les boucles while et for. Les bornes min et max sont connues et il n'est pas possible de les dépasser. Si une valeur invalide est saisie, elle sera réinitialisée à la dernière valeur valide.

Il est important de noter que les nombres sélectionnables dans les indices des boucles font références aux indices d'itérations et non pas aux variables utilisées dans la condition de la boucle. Ainsi une boucle " from 10 to 0 step -1 " contiendra 11 itérations, mais celles ci seront accessibles avec la zone de saisie dans l'ordre de 0 à 10.

Les zones de saisies pour le contrôle des boucles peuvent êtres masquées avec l'option "show loop index" dans le menu "Edition". Les Indices peuvent toujours êtres changés en cliquant sur un des mots clé de la boucle et en modifiant l'indice dans la zone en bas de l'écran.




1.4 Gestion des fonctions

Avec l'introduction des fonctions, la consultation des valeurs doit maintenant prendre en compte l'état courant de la pile d'appel.

Dans l'exemple ci dessous, la fonction "bar" n'est appelée qu'à un seul endroit, la consultation de ces paramètres et des variables contenues sont donc directement reliés à un état hérité de son appellant. En revanche, la fonction "foo" est appelée à trois endroits distincts.

Afin de confirmer le contexte d'appel, on peut sélectionner l'identifiant de l'expression appelante via une liste déroulante. Cette liste déroulante apparaît automatiquement dès lors qu'il y a plus d'une expression appelante possible. Les labels sélectionnables dans la liste déroulante contiennent le nom de la fonction, la ligne où l'appel à lieu, et si nécessaire, l'ordre d'apparition sur la ligne.

```
float a := 5.0
float b := 30.0
float c := 0.0
function float foo (float x, float y)
{
  float c := x + y
  return c
}
function float bar ( )
{
  float a := 5.0
  float b := 30.0
  float x := foo(0.1,0.3)
  float y := foo(a,b)
  return x + y
}
integer i
for i from 0 to 100
{
  c := foo(a,b)
  float d := bar() + 2
}
```



2 Les Graphiques

2.1 Création d'un graphique

Le Magnifier permet de créer des graphiques en points ou en lignes représentant des données issues de l'exécution d'un programme.

Pour générer un graphique après une exécution, il suffit de sélectionner un token en cliquant dessus dans l'onglet "Magnifier", puis de cliquer dans le menu Graphic -> "new from left/right selection". Un nouvel onglet est ainsi généré.

Contrairement aux onglets HSL Editor et Magnifier, les onglets de graphique peuvent être fermés à l'aide d'une croix rouge présente à droite du nom de l'onglet.

2.1.1 Exemple

Les graphiques ne sont utilisables que sur des tokens qui peuvent être associées à plusieurs valeurs au cours de l'exécution du programme.

Typiquement, une variable ou un opérateur à l'intérieur d'une boucle. Dans l'exemple ci-dessous, on s'intéresse à l'évolution de la variable x en fonction de la variable i

```
integer i ;  
float x := 0;  
for i from 0 to 100 :  
    x := i + (i % 10) ;
```

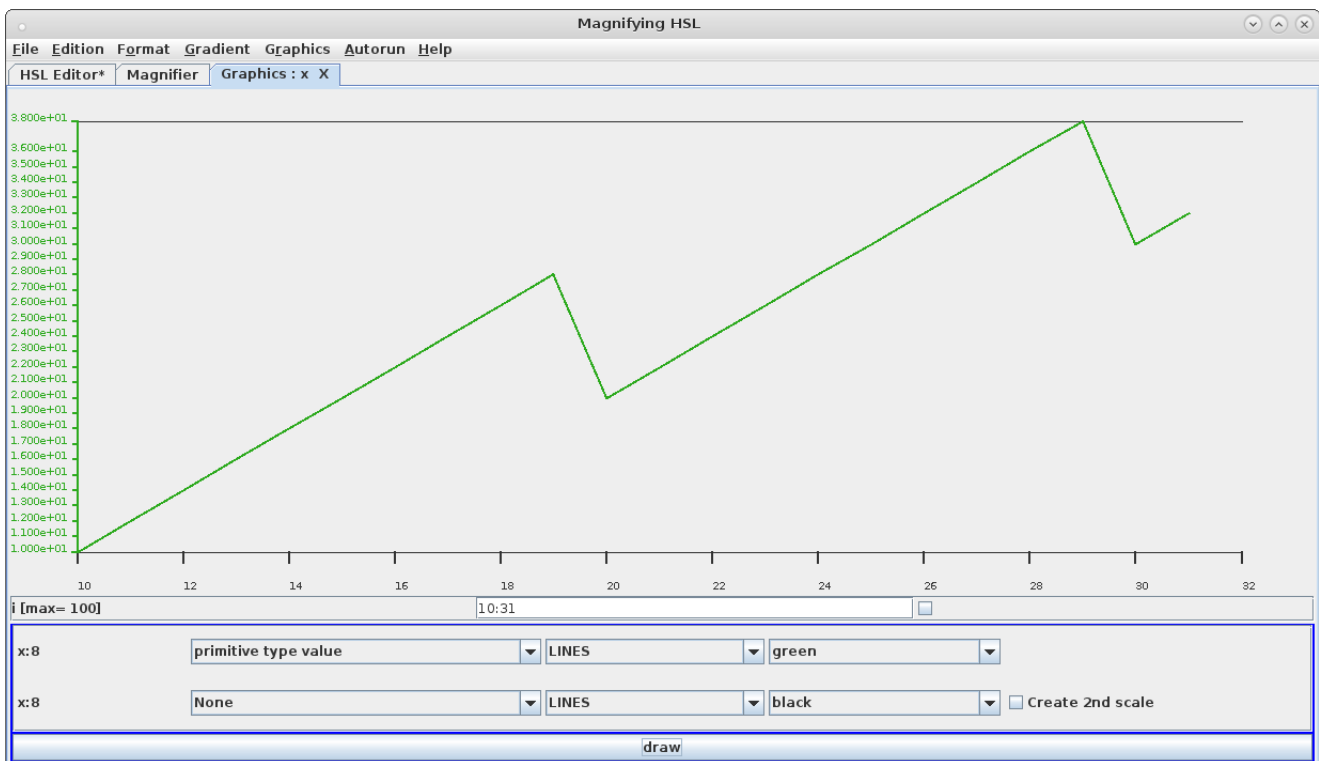

2.2 Gestion des indices

2.2.1 Boucle simple

Dans le cas précédent on observe un motif récurrent. Pour isoler ce motif, on s'intéresse plus particulièrement au comportement de x entre les itérations 10 et 31. On peut restreindre les valeurs affichées en agissant sur les indices considérés.

Ces indices sont visibles entre le graphique et la zone de saisie des séries. Dans ce cas particulier, Il n'y a qu'un seul indice : 'i' qui a une valeur maximum de 100. Par défaut, la plage de sélection considère tous les indices pour cette dimension qui est "libre".

Pour effectuer une restriction sur les indices, il suffit de décocher la case associée à cette dimension et de rentrer au choix une valeur d'indice, ou un intervalle sous la forme "borne_min :borne_max". Pour cet exemple, on va donc modifier les bornes et rafraîchir le graphique en cliquant sur "Draw"



2.2.2 Boucle imbriquée

Si l'on s'intéresse à un programme contenant des boucles imbriquées, on peut rencontrer des cas où les boucles contenues n'ont pas toujours le même nombre d'itération en fonction de l'itération englobante. Par exemple dans le cas ci dessous :

```
integer i ;
integer j ;
float x := 0;
for i from 0 to 100 :
{
    j := i;
    while (j > 0 ):
    {
        j := j -1;
        x := i + (i % 10 ) + j;
    }
}
```

On remarque de manière évidente que la boucle while n'aura pas le même nombre d'itérations en fonction de l'indice i de la boucle englobante. Lorsque l'on lance le Magnifier, les indices valides de la boucle while correspondent aux indices minimum et maximum rencontrés lors de l'exécution. Ainsi, pour i = 100, on a une boucle while qui effectuera 99 itérations.

Les indices valides pour la boucle for sont donc de 0 à 100 et de 0 à 99 pour la boucle while. Cependant, toutes les combinaisons d'indices ne sont pas valides. Par exemple, on ne peut pas avoir de valeur ayant un indice $j > i$ ou $i = j = 0$.

Si l'on sélectionne le nœud x dans le Magnifier en configurant les boucles pour avoir x[0][1], on constate effectivement qu'il n'y a pas de valeur pour ces coordonnées. Au niveau de l'affichage des valeurs, le Magnifier nous affiche un message informatif "no value found for coordinates 0;0"

Lors de la génération d'un graphique, les points à ces coordonnées ne sont pas affichés.

Dans cette representation, on peut également spécifier les indices pour isoler une partie du graphique.

2.2.3 Représentation des valeurs spéciales

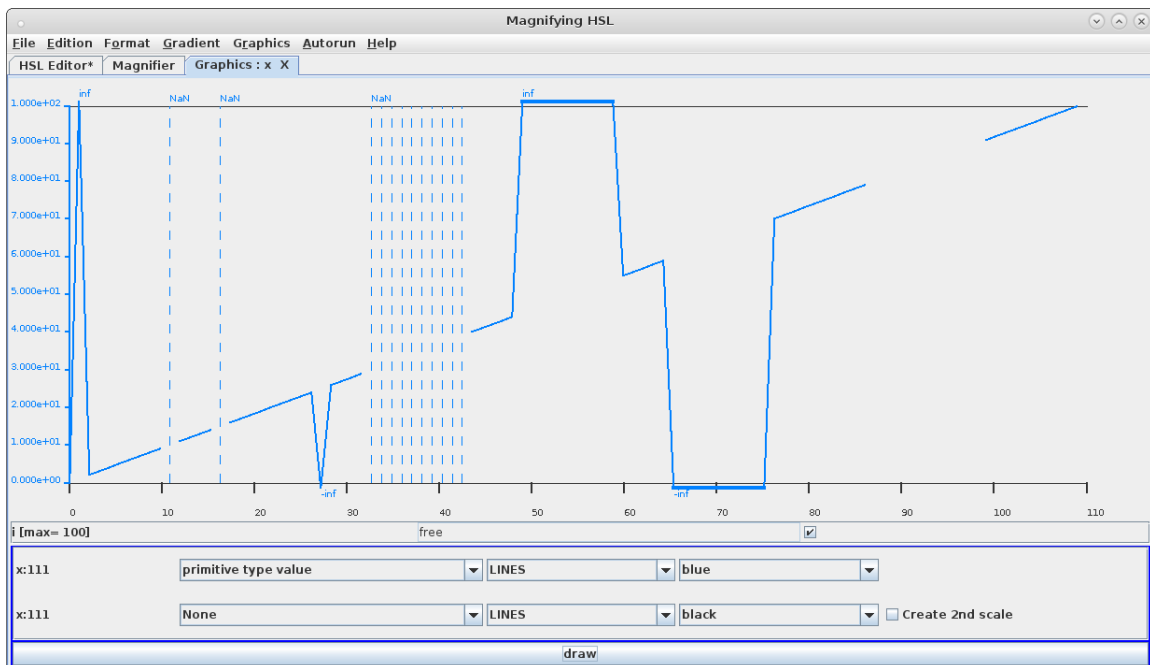
Comme nous l'avons vu précédemment, lorsque les coordonnées de boucle d'un point ne correspondent pas à une combinaison valide à l'exécution, il n'y a pas de données à afficher pour ce point. D'un point de vue graphique, le point a une valeur nulle et n'est pas représenté. Un point peut également prendre d'autre valeur qui nécessite de le représenter de façon spéciale :

- NaN : Dans le cas où la valeur de la variable est le résultat d'un calcul mais n'est pas défini mathématiquement (0.0/0.0, Inf-Inf, ...), le nombre flottant prend la valeur NaN (Not a number). Un nombre NaN est graphiquement représenté par une barre hachée verticale avec la mention NaN. L'apparition d'un NaN, comme avec une valeur nulle entraine une discontinuité en mode de représentation en ligne.
- Inf ou -Inf : Dans le cas de la représentation d'une valeur infinie, le point représentant la valeur est placée juste en dehors des limites du graphe (haute ou basse en fonction de son signe). La mention Inf est également écrites.

Note : Dans le cas de successions de valeurs spéciales (Inf ou Nan), seule la première valeur est accompagnée d'un libellé Nan ou Inf, les suivants sont implicites pour ne pas surcharger le graphique.

Le programme suivant permet de visualiser les différents cas de valeurs spéciales :

```
integer i; integer j;
float x := 0.1; float[2,3] y; float[11] z;
for i from 0 to 100:
{
  x := i;
  if i == 10 or i == 10:
    x := 0.0 / 0.0;
  if i == 15:
    x := -0.0 / 0.0;
  if i == 1:
    x := 1.0 / 0.0;
  if i == 25:
    x := -1.0 / 0.0;
  if i >= 30 and i < 40:
    x := 0.0 / 0.0;
  if i >= 45 and i < 55:
    x := 1.0 / 0.0;
  if i >= 60 and i < 70:
    x := -(1.0 / 0.0);
  if(i < 80 or i > 90):
    x := x;
}
```



3 Perturbations

Le principe des perturbation est d'exécuter un programme une deuxième fois en changeant certains paramètres afin d'en étudier les variations numériques. Cette pratique permet de conduire différentes études, notamment l'analyse de la sensibilité d'un programme à ses entrées.

3.1 Exécution avec perturbations

Il est possibles d'effectuer des perturbations dans le programmes de deux manières différentes. Soit directement au niveau du HSL avec l'éditeur, soit avec les commandes de perturbations disponibles au niveau de l'onglet Magnifier. Ces deux méthodes sont décrites plus amplement dans les sections ci-dessous.

Une fois qu'une perturbation a été exécutée, il est nécessaire de sélectionner un même nœud avec les deux modes de sélections (clic gauche et clic droit) pour visualiser les différences.

Attention, une version perturbée n'existe que par rapport à une exécution qui a été effectuée précédemment en tant que version originale pour la comparaison. Si l'on relance une exécution sur le code de manière classique, les données de la version perturbées sont perdues.

3.1.1 Effectuer une perturbation avec l'éditeur HSL

Dans l'onglet HSL Editor, il est possible d'effectuer des modifications directement dans le code source et de lancer une exécution en mode perturbation avec le bouton "Run as Perturbation". Cette fonctionnalité possède tout de même une sérieuse limite, car il ne faut pas changer la structure du programme.

Les seules modifications légales (le Magnifier n'effectue aucune vérification) doivent uniquement porter sur des littéraux au milieu d'expression. Une option sûre est de déclarer les littéraux que l'on souhaite modifier en tant que variables. Les motifs de déclaration de variables avec un littéral sont reconnus par le Magnifier qui propose des perturbations par défauts.

3.1.2 Effectuer des perturbation avec le Magnifier

Après l'exécution d'un programme, le Magnifier permet d'effectuer des modifications sur les variables qui sont déclarées et immédiatement initialisées avec un nombre flottant (Les entiers ou les initialisations avec des entiers ne sont pas considérés). Avec le programme minimum suivant ;

```
float x := 0.1;
```

On retrouve dans l'onglet Magnifier un panneau dédié aux perturbations sur la gauche. Par défaut ce panneau est grisé pour réduire la confusion. Pour l'activer, il suffit de cocher la case "enable perturbation" située sur ce même panneau. Ce panneau propose de nouveau de redéfinir la précision de base et la précision arbitraire qui seront utilisées lors de l'exécution de la version perturbée. Il est également possible d'effectuer des perturbations sur la variables x.

Chaque variable perturbable peut être modifiée à l'aide d'un slider qui permet une variation de -100 à 100. Un menu déroulant à droite de chaque slider permet de sélectionner la nature de la perturbation : en pourcentage de la valeur ou en ULP(unit of last precision*)

Une fois la configuration de la version perturbée définie, il suffit de cliquer sur le bouton "Run perturbation" pour lancer l'exécution

3.2 Perturbations graphics

Une fois qu'une version perturbée a été exécutée, il est possible d'afficher un graphique des données produites. Pour cela, il suffit de procéder comme pour la version non perturbée en sélectionnant le nœud d'intérêt dans le Magnifier, puis de passer par le menu "Graphic -> new from [left/right] selection".

Si le Magnifier détecte la présence d'un jeu de données d'une version perturbée, de nouvelles options apparaissent dans le menu déroulant des valeurs à afficher. Il est possible d'afficher côte à côte des valeurs de la version originale et de la version perturbée pour visualiser les différences.

4 Gradients

Les gradients sont des outils d'aide pour la recherches des lignes générant les plus grandes erreurs. Le principe est d'effectuer une coloration en fonction des données associées à chaque token. Il est possible d'activer un gradient dans le menu "Gradient".

Par défaut, les bornes minimum et maximum du gradient sont détectées automatiquement pour les indices sélectionnés. Cela permet de visualiser rapidement les valeurs limites. Il est cependant possible de décocher la case auto présente dans le menu et de sélectionner ses propres bornes. Cela permet notamment de visualiser les différents degrés d'erreur des expressions qui pourraient être invisibles en mode automatique si la borne maximum détectée est beaucoup plus grande que toutes les autres et provoque un effet d'écrasement.

Une autre valeur peut être manuellement entrée : le seuil (threshold) à partir duquel on considère qu'une valeur est correcte. Une valeur dont la valeur absolue est inférieure à ce seuil apparaîtra en vert.

4.1 Les types de gradients

- Primitive value : Il s'agit de la valeur dans le type primitif (float ou double de la norme IEEE754)
- Custom prec value : Il s'agit de la valeur dans le type de la précision arbitraire
- Relative error : Il s'agit de l'erreur brute exprimée en pourcentage de la valeur de référence (valeur de référence - valeur mesurée) / valeur de référence. Cette mesure est généralement la plus intéressante pour trouver les sources d'erreurs car elle n'est pas biaisée par l'amplitude de la valeur (une erreur relative de 0.1 sur une valeur de 1 est beaucoup plus importante qu'une erreur de 1 sur 100).
- Raw error : Il s'agit de l'erreur brute mesurée entre la valeur de référence et la valeur mesurée (valeur de référence - valeur mesurée)
- Variation p : C'est l'écart constaté sur les valeurs des types primitifs entre la version originale et la version perturbée
- Variation m : C'est l'écart constaté sur les valeurs des types en précision arbitraires entre la version originale et la version perturbée.
- Variation rel err : C'est l'écart entre l'erreur relative observée sur la version originale et celle observée sur la version perturbée
- Variation raw err : C'est l'écart entre l'erreur brute observée sur la version originale et celle observée sur la version perturbée

5 Formatage

Dans la barre de menu, le menu format permet de sélectionner le nombre de décimales à afficher lors de la consultation des nœuds dans le Magnifier. Il est également possible de choisir parmi trois types de formatages :

- Formatage normal : Dans ce mode de formatage, les valeurs sont affichées en utilisant soit le mode scientifique soit le mode précision fixe. Le mode sélectionné est celui dont l'application donne un résultat occupant un nombre minimum de caractères. C'est un mode généralement plus intuitif et compact, mais qui n'est pas uniforme d'une valeur à l'autre.
- Formatage Scientifique : En representation scientifique, tous les nombres sont représentés avec une partie entière non nulle de un caractère, et une partie décimale suivie d'un exposant en puissance de 10.
- Formatage en précision fixe : En précision fixe, aucun exposant n'est appliqué, c'est le mode par défaut.

6 Autorun

La fonction Autorun disponible dans le menu Autorun->start permet de lancer une simulation de l'exécution du programme.

Ce mode est utile pour comprendre le chemin d'exécution. Chaque passage sur une instruction est rendue visible par la création d'une bordure, la couleur de fond de chaque élément parcouru est également changée en fonction de la configuration du gradient.

Contrairement à l'activation manuelle du gradient, qui affiche les valeurs en fonctions des coordonnées de boucles sélectionnées, l'autorun va faire évoluer les indices de boucles. Il est donc recommandé de sélectionner manuellement les bornes minimum et maximum dans le menu gradient avant de lancer l'autorun.

Lors du parcours, si la valeur d'un élément est inférieure à la valeur seuil sélectionnée dans le menu Gradient >Threshold, la couleur d'arrière plan de l'élément sera changée en vert.

Attention, étant donné que l'autorun utilise l'erreur relative, si la valeur de référence vaut 0, on peut facilement avoir des valeurs d'erreur relative valant NaN ou l'infinie.

Les instructions sont alors colorisées d'une façon spéciale :

- Rouge pour +Inf
- bleu pour -Inf
- violet pour un NaN

Si l'élément n'a pas de valeur associée, l'arrière plan de l'élément courant ne sera pas colorisé.

L'exécution de l'autorun peut nécessiter un temps de chargement avant de se lancer. Une fois lancé, l'autorun peut être interrompu avec la commande Autorun->stop. Avant de lancer une exécution de l'autorun, il est possible de choisir le temps de pause entre l'inspection de deux éléments. Par défaut, le temps d'attente est de 100 millisecondes.

L'autorun accepte des vitesses d'exécution entre 10ms et 10 000ms. Il n'est cependant pas possible de changer la vitesse une fois l'autorun lancé.

7 Syntaxe du langage HSL

7.1 Généralités du langage

Le langage HSL est un langage fortement typé qui propose deux types :

- integer : Le type integer est un entier signé encodé sur 64 bits
- float : Le type float est défini à l'exécution. Il est remplacé par un type primitif (float ou double de la norme IEEE 745) et par un type en précision étendue spécifiée par l'utilisateur. La valeur par défaut est 512 bits (contre 24 pour le type float et 53 pour le type double)

Le langage HSL propose également le support des tableaux multidimensionnels.

Les commentaires dans le langage doivent être précédés par le symbole '#' et se terminent en fin de ligne.

7.2 Variables

Un nom de variable ne peut contenir que des caractères alphanumérique ou le caractère underscore '_'. Une Variable ne peut pas commencer par un caractère numérique. Les noms sont sensibles à la casse et doivent être définis de manière unique dans le programme. Toutes les variables sont par défaut initialisées à 0. Les variables déclarées sont supprimées à la fin de leur bloc de déclaration.

Déclaration d'une variable :

Les déclaration de variable fonctionnent de la même manière qu'en C. Le type précède le nom de la variable.

Exemples :

```
float a;  
integer b;
```

Par défaut, les variables sont initialisées à 0. L'utilisateur peut préciser leur valeur initiale lors de leur déclaration :

```
float a := 3.5;  
integer b := 8;
```

Pour déclarer un tableau, il suffit d'indiquer les dimensions à la suite du nom type : Exemples :

```
float[5] array;  
float[10, 5] matrix;  
integer[3, 3, 3] three_dimensional_array;
```

Il n'est pas possible d'initialiser un tableau lors de sa déclaration. Les éléments d'un tableau sont indexés à partir de 0. Pour accéder aux éléments d'un tableau, l'opérateur [] peut être utilisé de la façon suivante :

```

array[0] := 3.6;
array[4] := 10.1;
array[5] := 0; # ==> error: index out of bounds
matrix[0, 4] := 33.5;
matrix[b, 0] := 66.1;
three_dimensional_array[0, 0, 1] := 5;

```

7.3 Opérateurs

Voici la liste des opérateurs, trié par ordre inverse de priorité

Operator Name	Operator	Notes
Affectation operator	:=	
Logical OR	or	Unlike C/C++, operator does not exist
Logical AND	and	Unlike C/C++, operator && does not exist
Equal and Not Equal	==, !=	
Greater/Lesser, Greater or equal, Lesser or equal	<, >, <=, >=	
Operator addition and subtraction	+, -	
Operator multiply, divide and modulo	*, /, %, mod	Modulo can be written just '%' or in letters : 'mod'
Unary operator + and -	+, -	

7.4 Structure de contrôle

Le type booléen n'existe pas en HSL, les tests sont effectués par les valeurs numériques. La valeur 0 est évaluée comme étant fausse. Toute autre valeur est vraie. Cette règle concerne aussi bien les entiers que les flottants. Ainsi la valeur 0.5 est considérée comme vraie. En cas d'assignation d'une opération booléenne la variable cible est affectée à 0 si la condition est fausse, à 1 sinon.

7.4.1 Instruction If

La syntaxe de l'instruction if est :

if cond_expr : instruction_si_vraie else : instruction_si_fausse

L'instruction else est optionnelle, plusieurs instructions peuvent être regroupées dans un bloc délimité par { }. Une instruction else se réfère toujours à la dernière instruction if rencontrée au même niveau d'imbrication.

Exemples :

```
if a == 0:
    b := c;
else:
    b := c / a;
if a + 3 < 4: {
    b[a] := 5;
}
```

7.4.2 Instruction For

La syntaxe de l'instruction for est :

for variable from first to last step s : statement

l'instruction step est optionnelle, elle vaut 1 par défaut.

Exemples :

```
for b from 0 to 4 step 1:
    array[b] := a + b;
# Ce qui est équivalent en C a :
# for (b=0; b<=4; b = b + 1)
#   array[b] = a + b;

for b from 0 to 4:
    array[b] := a + b;
# Meme exemple mais avec un pas implicite de 1

for b from 100 to 0:
{
    array[b] := a + b;
}
```

```
# Il y a un pas implicite de 1. Dans ce cas,  
# aucune iteration ne sera realisee.  
# la variable b est toujours initialisee a 100  
# Puisque le pas par default est positif,  
# Le test initial de la condition renvoi faux.  
#  
#  
for b from 100 to 0 step -1: array[b] := a + b;  
# Ce qui est equivalent en C a  
# for (b = 100; b >= 0; b = b - 1)  
#   array[b] = a + b;
```

Il n'est pas possible de choisir un pas de 0, et on ne peut pas non plus modifier la variable servant à itérer dans le corps de la boucle

7.4.3 Instruction While

La syntaxe de l'instruction while est : while cond_expr : statement

Exemples :

```
while a < 3:  
    a := a + 0.1;  
while a != 3:  
{  
    a := a + 1;  
}  
while a * b + 15 <= 30 * a + 52 * b:  
{  
    a := a + 1;  
    b := b + 1;  
}
```

7.4.4 Fonctions

Il est désormais possible de définir des fonctions utilisateurs. Le hsl impose cependant quelque limitations :

- Une fonction doit être entièrement définie avant de pouvoir l'appeler (il n'est actuellement pas possible d'utiliser des déclarations seules). En conséquences, il n'est pour l'instant pas possible de définir des fonctions récursives. Les fonctions peuvent être définies à l'intérieur d'une autre fonction. Une fois définie, elle peut être appelée dans la suite du programme, y compris à l'extérieur de la fonction englobante.
- Il n'est pas possible de faire de la surcharge.
- Les types tableaux ne peuvent pas être passés en arguments.
- Le type void n'est pas supporté, une fonction doit toujours retourner une valeur.
- Les fonctions ne supportent pas de mécanisme de paramètre par défaut.
- Il n'est pas possible d'utiliser de variable globales (définies en dehors de la définition de la fonction).

La définition d'une fonction se fait de la manière suivante :

```
float a := 5.0;
float b := 30.0;
float c := 0.0;

#une fonction doit commencer par le mot clé "function",
#puis le type de retour, le nom de la fonction
#et une liste de paramètres
function float foo(float x, float y )
{
    float c := x + y;
    return c ;
}

#une fonction peut ne prendre aucun argument
#et avoir plusieurs retours
function integer bar ()
{
    integer x := 1;
    if x:
        return 1 ;
    else:
        return 3 ;
}
```

```
#l'appel de fonction est assimilable à une valeur,  
#elle peut servir dans une initialisation, une affectation  
# ou une expression  
c := foo(a, b);  
integer d;  
d := bar() + 2;
```

Le Hsl propose une liste de fonction prédéfinies qui sont à l'image des fonctions mathématiques classiques du C.

```
float random()  
#Retourne un nombre flottant entre 0 et 1.  
#Le résultat est calculé en divisant un nombre aléatoire  
#entier par la plus grande valeur entière possible (231 -1)  
  
void random_seed_set(integer s)  
#Initialise le générateur de nombre aléatoire avec une graine s  
  
integer isInf(float x)  
#Retourne 1 si x représente l'infini (indépendamment du signe), 0 sinon  
  
float getInf()  
#Retourne un flottant représentant l'infini.  
#Il est possible d'obtenir un infini négatif  
#en utilisant l'opérateur unaire -  
  
integer isNaN(float x)  
#Retourne 1 si x ne représente pas un nombre (exemple, 0/0  
or inf / inf), 0 sinon  
  
float getNaN()  
#Retourne un flottant codant NaN (not a number)  
  
float getConstPI()  
#Retourne la valeur de Pi  
  
float getConstE()  
#Retourne la valeur de E
```

Hsl propose également les fonctions mathématiques suivantes :

```
float acos (float x)
float asin (float x)
float atan (float x)
float atan2(float y, float x)

float sin(float x)
float cos(float x)
float tan(float x)

float sinh(float x)
float cosh(float x)
float tanh(float x)

float asinh(float x)
float acosh(float x)
float atanh(float x)

float exp (float x)
float exp2 (float x)
float exp10(float x)
float log (float x)
float log2 (float x)
float log10(float x)

float pow (float x, float y)
float sqrt(float x)

float abs (float x)
float ceil (float x)
float floor(float x)
```


7.4.5 Instruction With Prec

Par défaut le langage HSL ne propose qu'un type flottant (float), dont la précision peut être modifiée par l'utilisateur. Il n'est pas possible d'effectuer des opérations entre différents types de flottants.

En revanche, il est possible de changer la précision du type float en cours d'exécution à l'aide de l'instruction "with prec" qui s'applique sur un bloc d'instructions.

Durant l'exécution du block statement suivant cette instruction, les variables utilisées sont castées vers la précision donnée, les calculs sont également effectués dans la nouvelle précision.

Il est important de noter que toutes les instructions exécutées dans le corps de l'instruction sont impactées par le changement de précision, cela inclut les appels de fonctions. La précision utilisée lors de l'exécution d'une fonction est donc dépendante du contexte dans lequel se trouve l'appel.

```
float a := 0.1;
with prec := my_prec :
{
    float b := 0.1;
}
```

8 Glossaire

Format flottant : Les nombres flottants sont codés sur un certain nombre de bits et se décomposent en trois parties (voir norme IEEE 754).

- Le bit de signe qui permet d'indiquer si le nombre courant est positif ou négatif.
- L'exposant : qui est une chaîne binaire représentant un exposant d'une puissance de 2 sur lequel on applique un biais
- La mantisse : qui commence habituellement par un bit implicite à 1 et qui représente les décimales du nombre, en puissance de 2.

ULP : Unit of least Precision. Pour un nombre flottant donnée, l'ULP est la valeur représentée par le dernier bit de la mantisse. Cette valeur dépend de l'exposant

HSL : Le HSL est un langage de script minimaliste spécialisée pour l'analyse de propagation d'erreur.

NaN : Not A Number : Il s'agit d'une valeur spéciale que peut prendre un nombre flottant lorsque celui-ci est le résultat d'une opération mathématiquement invalide (inf / inf , $\text{inf} - \text{inf}$, $0/0\dots$).

Erreur brute : L'erreur brute est définie comme l'écart de la valeur observée par rapport à la valeur de référence. Dans le cadre du magnifier. L'erreur brute est calculée en faisant la différence entre la valeur obtenue en utilisant la précision arbitraire et la valeur utilisant le type primitif. En utilisant une précision égale au type primitif, l'erreur sera souvent à 0. Des exceptions peuvent survenir où à précision égale, le type en précision arbitraire n'aura pas exactement la même valeur. Cela est généralement dû à l'utilisation d'une fonction mathématique qui n'utilise pas un arrondi correct. Dans ce cas, la valeur en précision arbitraire sera plus précise.

Erreur relative : L'erreur relative correspond à l'erreur brute divisée par la valeur de référence (en précision arbitraire). Cette valeur permet de comparer différentes erreurs en prenant en compte l'impact de l'erreur par rapport à l'amplitude de la valeur (en pourcentage). Cette métrique est donc généralement plus intéressante pour découvrir les expressions entraînant des dérives numériques, mais elle peut rapidement donner des résultats extrêmes lorsque la valeur de référence est proche ou égale à 0.