Lisaac

*The power of simplicity at work for you*

Benoît Sonntag – benoit.sonntag@lisaac.org

Object Prototype

http://www.lisaac.org
## Why a new language? (1/2)

- **C language**

  **Advantages**
  - Memory mapping, interrupt management, ASM glue, multiple kinds of integer, compiled, very good performance

  **Inconveniences**
  - Not high level language

- **SmartEiffel language**

  **Advantages**
  - High level language, genericity, uniformity, static type, programming by contract, compiled, good performance

  **Inconveniences**
  - Not prototype object oriented, lack of OS programming facility
Why a new language? (2/2)

- **Self language**

  **Advantages**
  Uniformity, expressivity, simplicity, prototype object oriented

  **Inconveniences**
  Not compiled, lack of protection (no type), lack of OS programming facility

- **Java language**

  **Advantages**
  C-like syntax, static type, internet facility

  **Inconveniences**
  Not prototype object oriented, lack of OS programming facility, not good performance, lack of uniformity and expressivity
History: Lisaac for IsaacOOS Language

In the past...

- **C** language
  - \(\downarrow\)
- **Unix** system

The future...

- **Lisaac**
  - *Prototype based Object Oriented Language*
  - \(\downarrow\)
- **IsaacOOS**
  - *Prototype Object Operating System*
Let them sink in a bigger box?
High-level vs Hardware
Object Oriented for Hardware

Software / High-Level / Expressivity

Hardware support

- ASM
- C
- Eiffel
- Self
- Java
- Lisaac
Class vs Prototype (1/3)

Class

1 Skeleton (=class)  →  1 Object

Prototype

1 Object prototype (=the One)  →  1 other Object
Class vs Prototype (2/3)

Class

Class A

Class B

B Instance

1 Object with A and B definition

Prototype

A object (Prototype or not)

B object (Prototype or not)

B.Clone

1 other Object
Class vs Prototype (3/3)

Dynamic inheritance

XOR
Inherit Lisaac

- **Self**: Flexibility, simplicity and prototype concept

- **Eiffel**: Static type, programming by contract

- **C**: Interrupt management, memory mapping

- **Lisaac**: Full prototype object for hardware
The grammar of Lisaac

<table>
<thead>
<tr>
<th>Language</th>
<th>Number of Grammatical Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lisaac</td>
<td>19</td>
</tr>
<tr>
<td>Self</td>
<td>25</td>
</tr>
<tr>
<td>Java</td>
<td>49</td>
</tr>
<tr>
<td>Smalltalk</td>
<td>56</td>
</tr>
<tr>
<td>C</td>
<td>63</td>
</tr>
<tr>
<td>Pascal</td>
<td>140</td>
</tr>
<tr>
<td>C++</td>
<td>156</td>
</tr>
<tr>
<td>Eiffel</td>
<td>181</td>
</tr>
</tbody>
</table>

Number of grammatical rules
## Syntax rules

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Low case &amp; mono space-name environment</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Example:</em> main, factorial</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Upper case for a first character, low case else</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Example:</em> Section, Old, Private, Header</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type/prototype</th>
<th>Upper case</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Example:</em> STRING, CHARACTER, INTEGER</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Comment</th>
<th>Like C++</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Example:</em> /* Comment multiline */ or // Comment line</td>
<td></td>
</tr>
</tbody>
</table>
### Base type (1/2)

#### INTEGER
- **Hexadecimal**: 0Bh, 0B80_0000h
- **Decimal**: 12, 12d, 100_000
- **Octal**: 14o, 777o, 7_333o
- **Binary**: 01b, 1101b, 1010_1111b

#### REAL
- **Simple**: 1.1, 0.05
- **Scientific**: 5E-2

#### CHARACTER
- **Simple**: ’a’, ’k’
- **Escape**: ’\n’, ’\t’
- **Code**: ’\10’, ’\0Ah’

#### STRING_CONSTANT
- **Simple**: “Hello world\n”
- **Multiline**: “Hello \nworld\n”

#### BLOCK
- **Encapsulate code**: { ... }
  
  See after...
Even base types are full objects!

**INTEGER**

10.factorial.print;

**REAL**

2.7E-5.print;

**CHARACTER**

'a'.to_upper.print;

**STRING_CONSTANT**

"Hello world\n".print;

**BLOCK**

{ ... }.value;
Prototype

- One prototype = one file
- The name’s prototype = the name’s file
  
  Example:
  
  The file name `string.li` contains the `STRING` prototype.

- One prototype is a set of Section:
  1. Section Header (Mandatory)
  2. $n \times$ Section Inherit or Section Insert
  3. $n \times$ Section Public or other sections...

- One section is a set of slots (datas or functions).
# Sections

## Inheritance sections after Header section
- **Inherit**: Inheritance definition *(Private)*
- **Insert**: Non-conforming inheritance *(Private)*

## Simple sections
- **Public**: Services with public access
- **Private**: Services with private access
- **Directory**: Services with prototype’s directory access
- **prototype list**: Services with selective access

## Specific sections
- **Mapping**: Mapping structure object
- **Interrupt**: Hardware interruption handler
- **External**: External of Lisaac’s slot to C function
Example: Hello world!

```li
Section Header
  + name := HELLO;  // Mandatory
Section Public
  - main <-
    (       
      '"Hello world !\n".print;
    )
```

*Command line:* lisaac hello.li

*Executable result:* hello (ou hello.exe for windows)
Slot identifier

```plaintext
\- qsort tab:COLLECTION from low:INTEGER to high:INTEGER \<-
  ( + i,j:INTEGER;
    + x,y:OBJECT;
    i := low;
    j := high;
    x := tab.item ((i + j)>> 1);
  )
  { ... 
    (i <= j).if { 
      tab.swap j and i;
      ...
    }
  }.do_while {i <= j};
  (low < j).if { qsort tab from low to j; }
  (i < high).if { qsort tab from i to high; }
 );
```
Slot identifier

- `qsort tab:COLLECTION from low:INTEGER to high:INTEGER ←`

  (`+ i,j:INTEGER;
  + x,y:OBJECT;
  i := low;
  j := high;
  x := tab.item ((i + j)>> 1);
  { ...
   (i <= j).if {
     tab.swap j and i;
     ...
   }
  } do_while {i <= j};
  (low < j).if { qsort tab from low to j; }
  (i < high).if { qsort tab from i to high; }
);
Slot identifier: if

```latex
\texttt{- qsort tab:COLLECTION from low:INTEGER to high:INTEGER} \leftarrow
\begin{array}{l}
\texttt{( + i,j:INTEGER;}
\texttt{ + x,y:OBJECT;}
\texttt{ i := low;}
\texttt{ j := high;}
\texttt{ x := \texttt{tab.item ((i + j)>> 1);}
\texttt{ \{ ...}
\texttt{ (i <= j). if \{}
\texttt{ \hspace{1cm} \texttt{tab.textcolorblueswap j and i;}
\texttt{ \}}}
\texttt{ \}}
\texttt{ \}.do\_while \{i <= j\};}
\texttt{(low < j).if \{ qsort tab from low to j; \}}
\texttt{(i < high).if \{ qsort tab from i to high; \}}
\end{array}
\right)
```
Slot identifier: loop

```c
-qsort tab:COLLECTION from low:INTEGER to high:INTEGER ←
(  + i,j:INTEGER;
    + x,y:OBJECT;
    i := low;
    j := high;
    x := tab.item ((i + j)>> 1);
    {
    ...  
    (i <= j).if {  
      tab.swap j and i;
      ...  
    }
    }
    .do_while {i <= j};
(low < j).if { qsort tab from low to j; };
(i < high).if { qsort tab from i to high; };
);
```
Arguments/results definition

Argument

- Simple: `qsort tab:COLLECTION`
- Vector: `put_pixel (x,y:INTEGER)`

Result

- Simple: `is_even:BOOLEAN`
- Vector: `get_coord:(INTEGER,INTEGER)`
Operator slot: Unary (1/3)

**Prefix**

- `'-'` Self:SELF :SELF ←

  zero - Self; // Self ≡ this

*Example:* `(-3).print;`

**Postfix**

- Self:SELF '!' :SELF ←

  ( + result:INTEGER;
    (Self = 0).if { result := 1; } 
    else { result := (Self - 1) !; }
    result
  );

*Example:* `(10 !).print;`
**Operator slot: Binary (2/3)**

<table>
<thead>
<tr>
<th>Infix associativity</th>
<th>left</th>
<th>priority</th>
<th>Example</th>
</tr>
</thead>
</table>
| - Self:SELF ' +'   | Left | 80       | Self - - other;  
|                     |      |          | Example: 2 + 3 + 4 = ((2 + 3) + 4) |

<table>
<thead>
<tr>
<th>Infix associativity</th>
<th>left</th>
<th>priority</th>
<th>Example</th>
</tr>
</thead>
</table>
| - Self:SELF ' * '   | Left | 90       | ...          
|                     |      |          | Example: 2 + 3 * 4 = (2 + (3 * 4)) |

<table>
<thead>
<tr>
<th>Infix associativity</th>
<th>right</th>
<th>priority</th>
<th>Example</th>
</tr>
</thead>
</table>
| - Self:SELF ' ^ '  | right| 90       | ...          
|                     |      |          | Example: 2 ^ 3 ^ 4 = (2 ^ (3 ^ 4)) |
### Priority

1. **Classic message**
   
   **Example:** $2 + 5.\text{factorial} \iff 2 + (5.\text{factorial})$

2. **Postfix message**
   
   **Example:** $-5! \iff -(5!)$

3. **Prefix message**
   
   **Example:** $2 + -5 \iff 2 + (-5)$

4. **Infix message Depending priority**
   
   **Example:** $2 + 3 * 5 \iff 2 + (3 * 5)$

### Character list for operator *(It’s free style!)*

<table>
<thead>
<tr>
<th>Character</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>.factorial</td>
</tr>
<tr>
<td>@</td>
<td></td>
</tr>
<tr>
<td>#</td>
<td></td>
</tr>
<tr>
<td>$</td>
<td></td>
</tr>
<tr>
<td>%</td>
<td></td>
</tr>
<tr>
<td>&amp;</td>
<td></td>
</tr>
<tr>
<td>&lt;</td>
<td></td>
</tr>
<tr>
<td>&gt;</td>
<td></td>
</tr>
<tr>
<td>/</td>
<td></td>
</tr>
<tr>
<td>?</td>
<td></td>
</tr>
<tr>
<td>~</td>
<td></td>
</tr>
<tr>
<td>*</td>
<td></td>
</tr>
<tr>
<td>+</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td></td>
</tr>
<tr>
<td>=</td>
<td></td>
</tr>
</tbody>
</table>

(\(\leftarrow\) \(\Rightarrow\) != impossible)
Style slot (1/3)

+ : No shared, clonable or call sensitif
  - Distinct for classic data slot
  - Distinct for classic local slot (Local variable)

- : Shared (\texttt{= static} in java), persistant value
  - For method slot
  - For static data slot or local
No shared vs shared

<table>
<thead>
<tr>
<th>FOO</th>
<th>BAR_1</th>
<th>BAR_2</th>
<th>BAR_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ data_1</td>
<td>- data_2</td>
<td></td>
<td>+ data_1</td>
</tr>
</tbody>
</table>

- data_1
- data_2

FOO.clone
Step #0

```
+ data_1 -> NULL
- data_2 -> NULL
```

Step #3

```java
foo2.set_data_1 BAR_3;
foo2.set_data_2 BAR_4;
```

Step #1

```java
data_1 := BAR_1;
data_2 := BAR_2;
```

Step #2

```java
foo2 := FOO.clone;
```

Step #2

```java
foo2 := FOO.clone;
```

```
+ data_1 -> BAR_1
- data_2 -> BAR_2
```

```
+ data_1 -> BAR_3
- data_2 -> BAR_4
```

```
+ data_1 -> BAR_1
- data_2 -> BAR_2
```

```
+ data_1 -> BAR_1
- data_2 -> BAR_2
```

```
+ data_1 -> BAR_3
- data_2 -> BAR_4
```
Expanded attribute = Embedded object (1/2)

**Default attribute (in header declaration)**

```
Section Header
+ name := Expanded INTEGER;

Examples: All tiny objects like CHARACTER, REALs, INTEGERS
```

**Attribute type declaration**

```
+ slot:STRING_CONSTANT;
+ slot:Expanded STRING_CONSTANT;
```

Expanded slot is never NULL!
Definition

Distinct & Expanded inheritance slot

\[\iff\]

Class inheritance system (= Java like)

Section Header

+ name := DOG;

Section Inherit

+ parent_animal:Expanded ANIMAL;

Note

All other inheritance slot combinations \(\implies\) Prototype system only
Strict attribute

Strict: Static type $\rightarrow$ dynamic type

+ data:Strict FRUIT;
...

data := APPLE.clone; // IMPOSSIBLE!!!
data := FRUIT.clone; // OK!

Note

Expanded attribute $\rightarrow$ Strict attribute
SELF type

**SELF**: Dynamic type $\implies$ static type

In FRUIT:

- clone:SELF $\leftarrow$ ...

With APPLE and ORANGE inherit FRUIT:

apple := APPLE.clone; // return Strict APPLE
orange := ORANGE.clone; // return Strict ORANGE

**Note**

- Self type $\implies$ Strict attribute
- **Data slot** or **shared local** variable with SELF type is impossible!
Genericity type

Declaration in header

Section Header

+ name := ARRAY(E);

Note

E is parameter type. Syntax: [A..Z][0..9]*

Example

+ bucket:ARRAY(FRUIT);
bucket := ARRAY(FRUIT).create 2;
bucket.put ORANGE to 1;
bucket.put APPLE to 2;
Parameters’ types used in the method (without genericity)

Example

- \( \text{max a:E and b:E :E} \leftarrow \)
  
  \( (\ + \ \text{result:E}; \)
  
  \( (a > b).\text{if} \) \{ \)
  
  \( \text{result := a;} \)
  
  \} \text{else} \{ \)
  
  \( \text{result := b;} \)
  
  \}; \)

result

Note

All parameter type must be defined in arguments function.
Same prototype name

Example

DIRECTORY.FOO.message;
DIRECTORY1.DIRECTORY2.DIRECTORY3.FOO
DIRECTORY1...FOO
Assignment: data (1/3)

Example

```haskell
(f::FRUIT;
   a::APPLE;
a := APPLE;
f := a;
);
```

Note

- Assignment is statically ok, if the static type is an identical or a sub-type.
- Simple data assignment ':=' is the '==' in Java, C++, ...
- Warning with **Strict attribute** type (*see before …*)
Assignment: data, if possible (2/3)

Example

```c
(f:FRUIT;
 + a:APPLE;
(test).if {
    f := APPLE;
} else {
    f := ORANGE;
};
a ?= f; // a=f, if f is APPLE, a=NULL else
)
```

Note
- Assignment is dynamically ok, if the dynamic type is identical or sub-type.
- This mechanism replaces the cast of Java
Assignment: code (3/3)

Example

```prolog
- color (r,g,b: INTEGER) ← -
  (true_color := r<<16|g<<8|b);
...
  (color ← - (gray_color := (r+g+b)/3);
    );
  );
```
Inheritance: Class like (1/6)

+ & Expanded = Class system

Section Inherit

+ parent_animal:Expanded ANIMAL;

+ parent_animal:ANIMAL

ProTOTYPE

+ parent_animal:ANIMAL

PROTOTYPE.clone
Inheritance: Prototype “trait” (Self like) (2/6)

Full shared

Section Inherit

- parent_object : OBJECT := OBJECT;

Diagram:

```
OBJECT

- parent_object

PROTOTYPE

- parent_object

PROTOTYPE.clone
```
Inheritance: No shared & dynamic (*Lisaac inside*) (3/6)

+ = Full dynamic

Section Inherit

+ parent_object: OBJECT := OBJECT;

Section Public

... 

parent_object := FILE;

...

parent_object := DIRECTORY;

```
+ parent_object
```

PROTOTYPE

FILE

```
+ parent_object
```

PROTOTYPE.clone

DIRECTORY
Inheritance: Shared & Embedded (*Lisaac inside*) (4/6)

Section Inherit

- parent_object: Expanded OBJECT;

```
- parent_object
PROTOTYPE

- parent_object
PROTOTYPE.clone
```

& Expanded (*uniformity form*)
Inheritance: Dynamic compute parent (*Lisaac inside*)

(5/6)

For each lookup

Section Inherit

```plaintext
+ parent:OBJECT ←
( + result:OBJECT;
   ...// compute my parent
     result
   );
```

Warning

Endless Recursivity caused by the lookup algorithm.
Inheritance: Dynamic once compute parent (*Lisaac inside*)

(6/6)

Once execution dynamic parent evaluation

Section Inherit

```+ parent:OBJECT ←
( + result:OBJECT;
   ...// compute my parent
   parent := result // my parent is a data now!!
 );
```

Note

- The first lookup, the parent is dynamically defined
- The next lookup, the parent is a simple data value
Non-conforming inheritance

**Insert keyword**

```
Section Header
  + name := HUMAN;

Section Insert
  + parent_mammal:Expanded MAMMAL;
```

**Example**

```
+ a:MAMMAL;
a := HUMAN.clone; // Impossible!!!
```

**Warning**

The Expanded default object has always non-conforming inheritance
List: Set of Instructions & immediate evaluation (1/3)

**Without return value**

\[
( \langle \text{Local} \rangle; \langle \text{Expr1} \rangle; \langle \text{Expr2} \rangle; \langle \text{Expr3} \rangle; )
\]

**With one return value**

\[
( \langle \text{Local} \rangle; \langle \text{Expr1} \rangle; \langle \text{Expr2} \rangle; \langle \text{result} \rangle )
\]

**With \( n \) return value**

\[
( \langle \text{Local} \rangle; \langle \text{Expr1} \rangle; \langle \text{Expr2} \rangle; \langle \text{result1} \rangle, \langle \text{result2} \rangle )
\]
List: Examples (2/3)

For expressions

\((2 + 4) \times 7\)

For procedures

- \texttt{foo} \leftarrow \\
  ( \\
  \texttt{''Hello''.print; } \\
  ) ;

For functions

- \texttt{zero:INTEGER} \leftarrow \\
  ( \\
  \texttt{''Call zero''.print; } \\
  0 \\
  ) ;
List: Examples (3/3)

For vector assignment

\[(a,b) := (3,7);\]

For functions with results

\[- \text{coord} : (\text{INTEGER,INTEGER}) \leftarrow (\ x_{\text{current}},y_{\text{current}} \ );\]

For vector argument

\[\text{put\_pixel} (x,y) \text{ color } 0;\]

Plugin of vectors

\[(x,y) := \text{get\_coord};\]
\[\text{put\_pixel} (x,y) \text{ color } 0;\]
\[\equiv\]
\[\text{put\_pixel \ get\_coord \ color } 0;\]
BLOCK: Set of instructions & late evaluation (1/4)

Without return value

\[
\{ 
\langle \text{Args} \rangle; \\
\langle \text{Local} \rangle; \\
\langle \text{Expr1} \rangle; \\
\langle \text{Expr2} \rangle; \\
\}
\]

With one return value

\[
\{ 
\langle \text{Args} \rangle; \\
\langle \text{Local} \rangle; \\
\langle \text{Expr1} \rangle; \\
\langle \text{Expr2} \rangle; \\
\langle \text{result} \rangle \\
\}
\]

With \( n \) return value

\[
\{ 
\langle \text{Args} \rangle; \\
\langle \text{Local} \rangle; \\
\langle \text{Expr1} \rangle; \\
\langle \text{Expr2} \rangle; \\
\langle \text{result}_1 \rangle, \\
\langle \text{result}_2 \rangle \\
\}
\]
### BLOCK vs List (2/4)

<table>
<thead>
<tr>
<th>List</th>
</tr>
</thead>
<tbody>
<tr>
<td>( &lt; Local &gt;; &lt; Expr1 &gt;; &lt; Expr2 &gt;; &lt; Expr3 &gt;; )</td>
</tr>
</tbody>
</table>

\[
\text{BLOCK}.\text{value} \equiv \{ < Local >; < Expr1 >; < Expr2 >; < Expr3 >; \}.\text{value}
\]
Embedded code in object

```plaintext
+ display:{(INTEGER,INTEGER); INTEGER};
display := {
  (x,y:INTEGER); // Vector parameter
  + sum:INTEGER; // One local variable
  x.print;
  ',',print;
  y.print;
  sum := x + y;
  sum // The result block
};
...
display.value (3,4).print;
```
For expressions

(a != NULL) && {a.value = 3}

For conditionals

(a > b).if {
    'y'.print;
} else {
    'n'.print;
}

For loops

{j := j + 1; j.print;}.do_while {j < 10};

For iterations

1.to 10 do { j:INTEGER;
    j.print;
};
C like Switch statement (1/3)

For vector assignment

```c
foo.switch
    .case 1 do {
        ''Case 1'' . print;
    }.break
    .case 2 do {
        ''Case 2'' . print;
    }
    .case 3 do {
        ''Case 3'' . print;
    }
    .default {
        ''Default case'' . print;
    };
```
C like Switch statement (2/3)

- Self:SELF.switch:(SELF,INTEGER_8) <- (Self, 0);

- (Self:SELF, stat:INTEGER_8).case
  value:SELF do body:{} :(SELF,INTEGER_8) <-
  ( + new_stat:INTEGER_8;
  Self,
  (((stat = 0) && {value = Self}) || {stat = 1}).if {
    new_stat := 1;
    body.value;
  };
  new_stat
);
C like Switch statement (3/3)

- (Self:SELF, stat:INTEGER_8).break:(SELF,INTEGER_8) <-
  (+ new_stat:INTEGER_8;
   Self,
   (stat = 1).if {
     new_stat := 2;
   }
   new_stat
  );

- (Self:SELF, stat:INTEGER_8).default body:{ } <-
  (
   (stat = 0).if body;
  );
Auto-conversion: export (1/3)

Example

Section Header

+ name := Expanded CHARACTER;
- export := INTEGER_8;

Section Public

- to_integer_8:INTEGER_8 ← ...

...

( + a:CHARACTER;
+ b:INTEGER_8;
...

b := a; // ⇔ b := a.to_integer_8;

Note

- export primitive is not transivity
- ARRAY(INTEGER) type ➔ to_array_of_integer slot
Example

Section Header
  + name := Expanded CHARACTER;
  - import := INTEGER_8;

Section Public
  - from_integer_8 a:INTEGER_8 :SELF ← ...

...

( + a:CHARACTER;
  + b:INTEGER_8;

  ...

  a := b; // ⇔ a := CHARACTER.from_integer_8 b;
Auto-conversion: export/import (3/3)

Priority for resolved conflicting type

1. If source is a subtype of destination then OK, else
2. search an export in source static type to destination, else
3. search an import in destination static type for source, else
4. Error type mismatch!
Default value of prototype

Example

Section Header
- name := Expanded CHARACTER;
- default := '\0';

Section Header
+ name := STRING;
- default := STRING.clone;

Note
- By default, NULL is the default value for not Expanded prototype
- For Expanded prototype, the prototype is the default value
Definition Pre-pattern

The pattern code common at a set of the slot definition. This pattern code must be at the beginning of the code slot.

Example in the parent

```
- my_slot ←
[ // my pre-pattern
 'Call my_slot!' .println;
]
( // my body
  deferred; // abstract slot
);
```
In two children redefinition

- `my_slot ← ( 'First!' .print; );`
- `my_slot ← ( 'Second!' .print; );`

Result runtime

Call `my_slot`! First!
Call `my_slot`! Second!
Pattern code: pre-pattern (3/6)

In two children redefinition

- my_slot ←
  [ // redefine pattern
    ’’It’s me!’’.println;
  ]
( ’’First!’’.print; );

- my_slot ←
  [ // recompose pattern
    ’’Old :’’.println;
    ...
    ’’End!’’.println;
  ]
( ’’Second!’’.print; );

Result runtime

It’s me!
First!

Old:
Call my_slot!
End!
Second!
Definition Post-pattern

The pattern code common at a set of the slot definition. This pattern code must be at the end of the code slot.

Example

- my_slot ←
  ( // my body
    deferred; // abstract slot
  )

[ // my post-pattern
  ’’End of call my_slot!’’.println;
];
Definition Out-pattern

The pattern code common at a set of all output slot definition. This pattern is common for all extern call slot prototype. 

*Welcome in the Matrix!*

Definition & note

- The out-pattern is define at the end of prototype/file
- The out-pattern is executing after the execution extern call.
- call of type `my_slot`: not execute out-pattern (not extern call)
- call of type `my_object.my_slot`: execute out-pattern
- call of type `Self.my_slot`: execute out-pattern
Progress...

Why not? In the future...
The set of contract is tested during runtime.

The violation of contract implies the crash of execution and to print of the stack runtime.

The contract can be inhibited by the compiler option.

```
( // Source code . . .
  ? {j > 0}; // my assertion
  // Source code . . .
)
```
Programming by contract: Prototype level (2/5)

Note

The invariant primitive uses the "out-pattern"

Invariant to end of prototype file

Section Header
+ name := COLLECTION(E);

Section Public

... 

// The end of file:
[
  ? {lower <= upper + 1};
];
Programming by contract: slot level (3/5)

**Note**
- The **require** primitive use the “**pre-pattern**”
- The **ensure** primitive use the “**post-pattern**”

**Primitive additive for ensure**
- **Old**: compute the expression value before the call slot. This primitive can be used in the body slot too.
- **Result** or **Result_{< n }>**: send the result value of slot

**Example**:

```plaintext
? {Result = item upper};
? {count = Old count};
```
Programming by contract: Require/Ensure (4/5)

- swap idx1:INTEGER with idx2:INTEGER ←
  // Swap item at index ‘idx1’ with item at index ‘idx2’
  [  // Require
    ? {valid_index idx1};
    ? {valid_index idx2};
  ]
  ( + tmp:E; // Body slot
    tmp := item idx1;
    put (item idx2) to idx1;   put tmp to idx2;
  )
  [  // Ensure
    ? {item idx1 = Old item idx2};
    ? {item idx2 = Old item idx1};
  ];
Inheritance of contract

- By default, a prototype inherit all the contract of parent:
  - 1. Require on the slot
  - 2. Ensure on the slot
  - 3. Invariant on the prototype
- The redefine contract delete the old contract of parent
- In the redefine, you can paste the old contract with ‘…’ primitive

Note & resume...

- Require: test on arguments validity
- Ensure: test on results validity
- Invariant: test of the cohere on data set object
- Assertion: test a stat in the code (No inheritance primitive)
Memory Mapping: hardware structure (1/3)

Example for Global Descriptor Table on Intel x86

Section Header
+ name := SEGMENT_DESCRIPTOR;

Section Mapping
+ limit:UINT32;
+ address:UINT32;
+ type:UINT32;
+ level:UINT32;

... 
- gdt: NATIVE_ARRAY (Expanded SEGMENT_DESCRIPTOR);
Memory Mapping: binary file structure (2/3)

Section Header
+ name := MY_STRUCT;

Section Mapping
+ coord_x: UINTEGER_32;
+ coord_y: UINTEGER_32;
+ flags: UINTEGER_16;
+ color: UINTEGER_16;

Section Public
- move ← ...
- set_color ← ...

```
1 0 1 0 1 1 0 0 1 1 0 1 1 1 0 0
```

Diagram of the raw file structure of MY_STRUCT.
Section Header
+ name := STRUCT_1;

Section Mapping
+ code: UINTEGER_32;
+ stat: Expanded STRUCT_2;
+ type: UINTEGER_16;
...

Section Header
+ name := STRUCT_2;

Section Mapping
+ data_1: UINTEGER_16;
+ data_2: UINTEGER_16;
Interrupt hardware manager

Example

Section Interrupt
- my_interrupt ←
  ( // Code Lisaac ...
 );

Note

- Can’t a call direct my_interrupt slot
- my_interrupt call send a POINTER address function. It’s necessary for to put this address in Interrupt Descriptor Table.

Restriction

- Parameter or result is prohibited
- The function should not be Self dependent
Example without result

- `die_with_code code:INTEGER ← \texttt{\textasciicircum exit(@code)};`

Note

- `<identifier>` for access to local variable only (or argument)
- This access is always read only.
External C to Lisaac: with result (2/4)

Example

- **Persistant external**: 
  - `basic_getc ← 'getchar()':(CHARACTER);`
- **Non persistant external**: 
  - `Self:SELF '>>' other:SELF :SELF ← '@Self>>@other':SELF;`

Note: Warning

- **Persistant**: The persistant external means that the code will remain present even if the return value is not used. Parentheses in the type of return shows that the return value is not important, is the execution of this external is important.
- **Non persistant**: If the result external is not used, then the external is deleted by the compiler.
Example

\[- \text{Self:SELF '>}' other:SELF : BOOLEAN \leftarrow
\]\n\['@\text{Self}@\text{other}' : BOOLEAN\{\text{TRUE, FALSE}\};\]

Note

- This **static type** result is **BOOLEAN**
- The **dynamic type set** for this result is **TRUE** or **FALSE**
- Each dynamic type must be a sub type of static type
Example

Section Header
+ name := Expanded CHARACTER;
− type := ‘signed char’;

Note
The compiler translate the CHARACTER with C type signed char

Warning
With Expanded or not and the C type:
• Expanded type \(\Rightarrow\) No pointer C type
• No Expanded type \(\Rightarrow\) Pointer C type
External Lisaac to C

**Examples**

Section External

- `function_for_c(a, b: INTEGER) : INTEGER ←
  ( // Code Lisaac ...
  );`

**Note**

Here, we have a function `int function_for_c(int a, int b)` in C code product.

**Restriction**

- Several keywords for the name function is prohibited
- The function should not be Self dependent
- The vector result is prohibited
External intern of Lisaac

**Definition**
This is a fundamental external known and used by the compiler.
Syntax: `<number>` with \( number \in [0..31] \)

**Examples**
- Self:SELF '‐' Left 80 other:SELF :SELF ↵ '1';
- Self:SELF '*' Left 100 other:SELF :SELF ↵ '2';
- Self:SELF '/' Left 100 other:SELF :SELF ↵ '3';
- Self:SELF '&' Left 100 other:SELF :SELF ↵ '4';
- Self:SELF '>' Left 100 other:SELF :BOOLEAN ↵ '5';
COP: Concurrent Object Prototypes (1/4)

Possible call

Pointer on C impossible (Not possible call)
COP: Concurrent Object Prototypes (2/4)

The language

The physical screen

First Window on screen

Second Window on screen
COP: Concurrent Object Prototypes (3/4)
One file = one project

By default: lisaac/make.lip

- Communication between Compiler and Lip file: 
  *Via Intern variables*
- Full configuration of compiler options
- Subset Lisaac language Interpreter
- Dynamic description of paths directories
- Set of instructions before compilation pass (Front-end)
- Set of instructions after compilation pass (Back-end)
- Dynamic execution during compilation in live prototype context
LIP: Lip file location (2/11)

Explicite path for a Lip file

```
sonntag@isaac:~/slides/lisaac$ lisaac ../project/make.lip
```

Implicite research

1. Search lip file in current directory.
2. if failed, search in parent of directory.
3. go to (2) until the root directory
4. Else, search lip file by default (lisaac/make.lip)
Lip: Intern variables (3/11)

Compiler $\implies$ Lip (immediately)

+ \texttt{lisaac:STRING};

Example: /home/sonntag/lisaac/

\begin{itemize}
  \item Read LISAAC\_DIRECTORY environnement variable
  \item if (1) failed, search \#define LISAAC\_DIRECTORY in path.h
\end{itemize}

Compiler $\implies$ Lip (immediately)

+ \texttt{input\_file:STRING};

Example: hello\_world \textit{(Read command line argument)}

Compiler $\implies$ Lip (after compilation)

+ \texttt{is\_cop:BOOLEAN};
Lip: Intern variables (4/11)

<table>
<thead>
<tr>
<th>Compiler ↔ Lip (Debug information)</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ debug_level: INTEGER;</td>
</tr>
<tr>
<td>+ debug_with_code: BOOLEAN;</td>
</tr>
<tr>
<td>+ is_all_warning: BOOLEAN;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Compiler ↔ Lip (Optimization)</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ is_optimization: BOOLEAN;</td>
</tr>
<tr>
<td>+ inline_level: INTEGER;</td>
</tr>
</tbody>
</table>
### Lip: Intern variables (5/11)

<table>
<thead>
<tr>
<th>Compiler $\leftrightarrow$ Lip (Generate code)</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ is_java: BOOLEAN;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Compiler $\leftrightarrow$ Lip (Other)</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ is_statistic: BOOLEAN;</td>
</tr>
<tr>
<td>+ is_quiet: BOOLEAN;</td>
</tr>
</tbody>
</table>
Lip: Subset Lisaac language (6/11)

Syntax

- **Types**: BOOLEAN, STRING, INTEGER
- **Binary Operators**: | & + − < > ≤ ≥ = !=
- **Unary Operators**: - !
- **Assignment**: :=
- **Style slot**: + data slot
  - method slot
  
  *(with 0 or 1 parameter and without return value)*
**Lip: Subset Lisaac language (7/11)**

<table>
<thead>
<tr>
<th>Slot built-in</th>
</tr>
</thead>
<tbody>
<tr>
<td>• <strong>BOOLEAN.if</strong> { … }</td>
</tr>
<tr>
<td>• <strong>BOOLEAN.if</strong> { … } else { … }</td>
</tr>
<tr>
<td>• **BOOLEAN</td>
</tr>
<tr>
<td>• <strong>path</strong> text:<em>STRING</em></td>
</tr>
<tr>
<td>• <strong>run</strong> cmd:<em>STRING :INTEGER</em></td>
</tr>
<tr>
<td>• <strong>get_integer</strong>:<em>INTEGER</em></td>
</tr>
<tr>
<td>• <strong>get_string</strong>:<em>STRING</em></td>
</tr>
<tr>
<td>• <strong>exit</strong></td>
</tr>
</tbody>
</table>
In `Section Public`

- `debug level:INTEGER < -`

  // Fix debug level (default: 15)

  (          
     ((level < 1) | (level > 20)).if {
         "Incorrect debug level.".print;
         exit;
     };
     debug_level := level;
  );

**Compiler Lisaac option**

Options:

- `-debug <level:INTEGER>`:
  Fix debug level (default: 15)
Lip: Other Section (9/11)

In Section Private
- Others code slots.
- Data slot intern and others data slots.

In Section Inherit (*Multi-inheritance*)
- With lip path:
  + `parent:STRING := '../my_project/linux/';`
- Without path: Inheritance Lip file by default.
  + `parent:STRING;`

Inheritance
- Redefinition slot is authorized.
- Lookup algorithm is active.
### Lip: Particular method slot (10/11)

<table>
<thead>
<tr>
<th>front_end</th>
</tr>
</thead>
<tbody>
<tr>
<td>Executed by compiler, before compilation step.</td>
</tr>
<tr>
<td>- Detect operating system,</td>
</tr>
<tr>
<td>- Loading path set for a project,</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>back_end</th>
</tr>
</thead>
<tbody>
<tr>
<td>Executed by compiler, after compilation step.</td>
</tr>
<tr>
<td>- Added gcc options, lib, ...</td>
</tr>
<tr>
<td>- Finalize the compilation with gcc or others</td>
</tr>
</tbody>
</table>

**Warning**

back_end & front_end is mandatory in Section Private
Lip: Dynamic execution during compilation (11/11)

In the **Section Header**

```plaintext
+ name := VIDEO;
- lip <- ( add_lib "-lX11"; );
```

In **make.lip**

```plaintext
- add_lib lib:STRING <-
  ( run "echo \"int main\{ return(1); \}\" > _t.c";
    (run("gcc _t.c"+lib+" 2>/dev/null")=0).if {
      lib_gcc := lib_gcc + " " + lib;
    } else {
      ("ERROR: " + lib + "' lib not found.").print;
      run "rm _t.c"; exit;
    };
  );
```

Project manager

Conclusion
Question?

IRC

- Server: irc.oftc.net
- Channel: #isaac

Information & contacts

- Wiki: http://wiki.lisaac.org
- Mailing list:
  http://www.lisaac.org/community/contact

Good luck!