The EASEA manual

This manual

This document is intended to programmer working on the EASEA platform and to everyone working with it. It contain the language and idioms description, the concept behind it and the documentation related to the compiler and its genetic algorithm library.

About EASEA

Introduction

EASEA and EASEA-CLOUD are Free Open Source Software (under GNU Affero v3 General Public License) developed by the SONIC (Stochastic Optimisation and Nature Inspired Computing) group of the BFO team at Université de Strasbourg. Through the Strasbourg Complex Systems Digital Campus, the platforms are shared with the UNESCO CS-DC UniTwin and E-laboratory on Complex Computational Ecosystems (ECCE).

EASEA (EAsy Specification of Evolutionary Algorithms) is an Artificial Evolution platform that allows scientists with only basic skills in computer science to implement evolutionary algorithms and to exploit the massive parallelism of many-core architectures in order to optimize virtually any real-world problems (continous, discrete, combinatorial, mixed and more (with Genetic Programming)), typically allowing for speedups up to x500 on a $3,000 machine, depending on the complexity of the evaluation function of the inverse problem to be solved.

Then, for very large problems, EASEA can also exploit computational ecosystems as it can parallelize (using an embedded island model) over loosely coupled heterogenous machines (Windows, Linux or Macintosh, with or without GPGPU cards, provided that they have internet access) a grid of computers or a Cloud.

Capabilities and features

The platform

Runs can be distributed over cluster of homogeneous AND heterogeneous machines. Distribution can be done locally on the same machine or over the internet (using a embedded island model). Parallelization over GPGPU cards leading to massive speedup (x100 to x1000). C++-like description language.

EASEA use CUDA to parallelize over GPGPU card. There is, as absurd as it sound, no parallelization over CPU at the time of redaction, but a working openMP prototype enters its final testing phase.

EASEA offer a high level of parametrization and the possibility to include large chunk of C/C++ code with little to no change at all.

Current genetic algorithm implementation

- Darwinian and baldwinian approach
- Genetic programming
- CMA-ES (Covariance Matrix Adaptation Evolution Strategy)
- Memetic approach

EASEA also offer out-of-the-box a wide range of selector: Max/Min deterministic, Max/Min Random, Max/Min tournament and MaxRoulette.

Three stopping criterion are possible : by generation, by time and by user defined control.
The main advantage of EASEA

Support

In order to have a working grapher, java jre 1.6 is required. Without it, an error appears at the start of easea's compiled programs but can be safely ignored.
EASEA had been compiled and tested with those following compilers:

- Gcc 4.4 to 4.8.2

- Clang 3.0 to 3.3
- Mingw-gcc 4.8.2
- CUDA SDK > 4.1

Workflow

The workflow of an easea program is the following:
easea code (.ez) easea compile C++ generated sourceC++ compileexecutable

The easea compiler generate high-performance C++ sources from the user's easea source code and predefined templates.

The language

EASEA defined sections

Genome specific fields

Genome Initialiser

Description
Uses the Genome EASEA variable to access the individual's genome.

EASEA syntax

```
\GenomeClass::initialiser :
...
\end
```

Example

```
\GenomeClass::initialiser :
for(int i=0; i(y)?(x):(y))     //Definition of the MAX global function
#define MIN(x,y) ((x)<(y)?(x):(y))      //Definition of the MIN global function
#define PI 3.141592654                 //Definition of the PI global variable
\end
```

User functions

Description

This is the section where the user can declare the different functions he will need.

EASEA syntax

```
\User declarations :
...
\end
```

Example

```
\User declarations :
float gauss()
/* Generates a normally distributed random value with variance 1 and 0 mean.
Algorithm based on “gasdev” from Numerical recipes’ pg. 203. */
{
int iset = 0;
float gset = 0.0;
float v1 = 0.0, v2 = 0.0, r = 0.0;
float factor = 0.0;
if (iset) {
iset = 0;
return gset;
}
else {
do {
v1 = (float)random(0.,1.) * 2.0 - 1.0;
v2 = (float)random(0.,1.) * 2.0 - 1.0;
r = v1 * v1 + v2 * v2;
}
while (r > 1.0);
factor = sqrt (-2.0 * log (r) / r);
gset = v1 * factor;
iset = 1;
return (v2 * factor);
}
}
\end
```

User Classes

Description

This is the section where the user will be able to declare:

1. The genome of the individuals trough the GenomeClass class
2. Different classes that will be needed.

The GenomeClass field has to be present and ideally not empty. All the variables defined in this field (the genome) will be accessible in several other fields using the 'Genome' variable. Other "hidden" variables can be accessed such as:

1. (a) The fitness (variable: fitness. Ex: Genome.fitness)

EASEA syntax

```
\User Class :
...
GenomeClass{
...
}
\end
```

Example

```
\User classes:
Element     {
int     Value;
Element *pNext; }
GenomeClass {
Element *pList;
int     Size;   }
\end
```

User Makefile

Description

This section allows the user to add some flags to the compiler typically to include some custom libraries.

Two flags of the Makefile are accessible to the user in this section:

1. CPPFLAGS
2. LDFLAGS

EASEA syntax

```
\User Makefile options :
...
\end
```

Example

```
\User Makefile options :
CPPLAGS += -llibrary
LDFLAGS += -I/path/to/include
\end
```

Miscellaneous fields

Before everything else function

Description

This function will be called at the start of the algorithm before the parent initialization.

EASEA syntax

```
\Before everything else function :
...
\end
```

After everything else function

Description

This function will be called once the last generation has finished.

The population can be accessed.

EASEA syntax

```
\After everything else function :
...
\end
```

At the beginning of each generation function

Description

This function will be called every generation before the reproduction phase.

The population can be accessed.

EASEA syntax

```
\At the  beginning of each generation function :
...
\end
```

At the end of each generation function

Description

This function will be called at every generation after the printing of generation statistics.

The population can be accessed.

EASEA syntax

```
\At the end of each generation function :
...
\end
```

At each generation before reduce function

Description

This function will be called at every generation before performing the different population reductions.

The parent population AND the offspring population can be accessed (merged into a single population).

EASEA syntax

```
\At every generation before reduce function :
...
\end
```

Memetic specific field

Genome Optimiser

Description

The optimiser field is a Genome specific field. It is meant to contain the function defining the way an individual will be locally optimized n times. The function will hence be called sequentially as many times as the user desires for each individual.

EASEA gives the user two possibilities when designing their local optimization function :

1. The user can choose to design the function that will enhance the genome of their individuals only in which case the rest of the local optimizer (i.e. creating the local optimizing loop, checking if an individual has improved or not, storing temporary individuals, calling of the evaluation function, etc ...) will be taken care of by the EASEA memetic algorithm. The function will have to be called as many times as specified by the Number of optimisation iterations parameter.
2. The user can choose to write the complete local optimizer. This way, he will have the complete freedom to design a more complex and specific optimizer, but he will also have to deal with the creation of the local optimization loop, the management of temporary individuals, the calling of the evaluation function etc... The Number of optimisation iterations parameter will have to be set to 1 as the function desigend by the user will contain it's own optimization loop requiring it's own specific number of optimization iterations.

EASEA syntax

```
\GenomeClass::optimiser :
...
\end
```

Examples

The two following examples will expose the two different ways the optimizer field can be used. The first example will show a simple *mutation* function. The second example will show the design of a complete local optimizer. Both examples are GPU compatible.

Genome optimization only

```
\GenomeClass::optimiser :
float pas = 0.001;
Genome.x[currentIteration%SIZE]+=pas;
\end
```

This example shows a simple *mutation* function that will add a small variation to one of the genes of an individual. The call to this function will be followed by a call to the evaluation function, and a replacement process. If the modification of the genome has improved the individual, it will replace the original one. This is being taken care of by the EASEA memetic algorithm.

Complete local optimizer

```
\GenomeClass : : optimiser :   //  Optimises   the Genome
float pas=0.001;
float fitnesstmp = Genome.fitness ;
float tmp[SIZE];
int index = 0;
for(int i=0; ifitness
```

Best individual Returns a pointer to the best individual found to this point. All the genome field can be accessed as well as the fitness field. All the genome fields can be modified.

EASEA syntax

```
bBest (-> operator to access variables)
```

Genome Returns the genome of an individual (can only be used in genome specific EASEA subsections such as mutation, initialisation, evaluation and . All the fields can be modified.

EASEA syntax

```
Genome (. operator to access variables)
```

Parent 1 Returns the genome of the first selected parent. Can only be used in the crossover genome subsection.

EASEA syntax

```
parent1
```

Parent 2 Returns the genome of the second selected parent. Can only be used in the crossover genome subsection.

EASEA syntax

```
parent2
```

Child Returns the genome of the newly created individual. Can only be used in the crossover genome subsection.

EASEA syntax

```
child
```

Here is a quick presentation of the various parameters that can be found and used in a .ez file

EASEA defined functions

Random Number Generators

TossCoin

Description Simulates the toss of a coin. There are two different definitions of the tossCoin function:

1. A simple toss of a coin.
2. A biased toss of a coin.

EASEA syntax `bool tossCoin()  // SImple tosscoin`

`bool tossCoin(float bias)  // Biased tossCoin`

Example
```
if( tossCoin(0.1)){
...
}
```

Random

Description Generates a random number. There are several definitions to the random function:

1. Random function with Min Max Boundary
2. Random function with Max Boundary only

In the case of a Max boundary only, the Min boundary will be 0.

EASEA syntax `int random( int Min, int Max)`

`int random( int Max)`

`float random( float Min, float Max)`

`double random( double Min, double Max)`

EASEA defined variables

Current Generation

Return the current generation number. This variable can be modified.

EASEA syntax `currentGeneration`

Number of Generation

Returns the generation limit. This variable can be modified.

EASEA syntax `NB_GEN`

Population Size

Returns the size of the population. This variable cannot be modified.

EASEA syntax `POP_SIZE`

Mutation Probability

Returns the mutation probability. This variable can be modified.

EASEA syntax `MUT_PROB`

Crossover Probability

Returns the crossover probability. This variable can be modified.

EASEA syntax `XOVER_PROB`

Minimise

Returns a boolean indicating of the algorithm performs a minimization of not. Returns true if minimizing.

EASEA syntax `MINIMISE`

Population

Returns a pointer to the main population. This variable cannot be used everywhere. If misused, it can provoque unexpected behaviours or compile errors.

EASEA syntax `pPopulation ([i] to access individuals)`

fitness

`pPopulation[i]->fitness`

Best individual

Returns a pointer to the best individual found to this point. All the genome field can be accessed as well as the fitness field. All the genome fields can be modified.

EASEA syntax `bBest (-> operator to access variables)`

Genome

Returns the genome of an individual (can only be used in genome specific EASEA sections such as mutation, initialisation, [[EASEA defined sections#Genome Evaluationevaluation and display). All the fields can be modified.

EASEA syntax `Genome (. operator to access variables)`

Parent 1

Returns the genome of the first selected parent. Can only be used in the crossover genome section.

EASEA syntax `parent1`

Parent 2

Returns the genome of the second selected parent. Can only be used in the crossover genome section.

EASEA syntax `parent2`

Child

Returns the genome of the newly created individual. Can only be used in the crossover genome section.

EASEA syntax `child`

EASEA defined parameters

Basic Parameters

Number of generations

Description :
Gives the maximum number of generations during wich the evolutionary algorithm will run.

EASEA syntax :

`Number of generations:`

Values : Integer strictly over 0.

Command line syntax :

`--nbGen=`

Values : Integer strictly over 0.

Time limit

Description :
Sets the maximum amount of time in seconds during wich the evolutionary algorithm will be allowed to run. Setting this parameter to 0 deactivates the time limit.

EASEA syntax :

`Time limit :`

Values : Positive Integer.

Command line syntax :

`--timeLimit=`

Values : Positive Integer.

Population size

Description :
Sets the size of the population that will be evolved.

EASEA syntax :

`Population size :`

Values : Integer strictly over 0.
Command line syntax :

`--popSize=`

Values : Integer strictly over 0.

Offspring size

Description :
Sets the number of individual that will be produced trough evolutionary crossover and/or mutation.

EASEA syntax :

`Offspring size :`

Values : Integer strictly over 0.
Command line syntax :

`--nbOffspring=`

Values : Integer strictly over 0.

Mutation probability

Desciption :
This parameter determines the probability that an individual will be mutated during it's creation process.

EASEA syntax :

`Mutation probability :`

Values : Real number between 0.0 and 1.0 .

Command line syntax :
This parameter is not avaible in command line yet.

Crossover probability

Description :
This parameter determines the probability that an individual will be the result of a crossover during it's creation process.

EASEA syntax :

`Crossover probability :`

Values : Real number between 0.0 and 1.0 .

Command line syntax :
This parameter is not avaible in command line yet.

Evaluator goal

Description :
This parameter will set the goal of the evolutionary algorithm.

EASEA syntax :

`Evaluator goal :`

Values : *minimis/ze* or *maximis/ze*

Command line syntax :
This parameter is too fundamental to the behaviour of the algorithm to be changed in command line.

Selection operator

Description :
This parameter decides of the way individuals of the parent population will be selected to create the new individuals. Several operator are avaiable in EASEA:

Tournament (need a selection pressure)

Deterministic

Roulette (only when Evaluator goal = maximise)

Random

When the selection pressure is an integer over 0, the best individual from the n will be selected.
When the selection pressure is a real number between 0.0 and 1.0, the best individual of 2 will be selection with the probability p given by the selection pressure.

EASEA syntax :

`Selection operator: OPERATOR SELECTION_PRESSURE`

Values for operators : Tournament Deterministic Roulette Random
Values for selection pressure : Integer strictly over 0 **or** Real number between 0.0 and 1.0 .

Compile line syntax :

```
--selectionOperator=
--selectionPressure=
```

Values for operators : Tournament Deterministic Roulette Random
Values for selection pressure : Integer strictly over 0 **or** Real number between 0.0 and 1.0 .

Surviving parents

Description :
This parameter will determine the number of children that will be participating in the run for the next generation (IL VA FALLOIR MODIFIER CETTE DESCRITPION)

EASEA syntax :

```
Surviving offspring:
```

Values : Integer strictly positive **or** Percentage (example: 100%)

Command line syntax :

```
--survivingOffspring=
```

Values : Integer strictly positive **or** Real number value between 0.0 and 1.0

Reduce parents operator

Description :
This parameter decides of the way individuals of the parent population will be selected to participate in the next selection process. Several operator are avaiable in EASEA:

Tournament (need a selection pressure)

Deterministic

Roulette (only when Evaluator goal = maximise)

Random

When the selection pressure is an integer over 0, the best individual from the n will be selected.
When the selection pressure is a real number between 0.0 and 1.0, the best individual of 2 will be selection with the probability p given by the selection pressure.

EASEA syntax :

```
Reduce parents operator: OPERATOR SELECTION_PRESSURE
```

Values for operators : Tournament Deterministic Roulette Random
Values for selection pressure : Integer strictly over 0 **or** Real number between 0.0 and 1.0 .

Compile line syntax :

```
--reduceParentsOperator=
--reduceParentsPressure=
```

Values for operators : Tournament Deterministic Roulette Random
Values for selection pressure : Integer strictly over 0 **or** Real number between 0.0 and 1.0 .

Reduce offspring operator

Description :
This parameter decides of the way individuals of the offspring population will be selected to participate in the next selection process. Several operator are avaiable in EASEA:

Tournament (need a selection pressure)

Deterministic

Roulette (only when Evaluator goal = maximise)

Random

When the selection pressure is an integer over 0, the best individual from the n will be selected.
When the selection pressure is a real number between 0.0 and 1.0, the best individual of 2 will be selection with the probability p given by the selection pressure.

EASEA syntax :

```
Reduce offspring operator: OPERATOR SELECTION_PRESSURE
```

Values for operators : Tournament Deterministic Roulette Random
Values for selection pressure : Integer strictly over 0 **or** Real number between 0.0 and 1.0 .

Compile line syntax :

```
--reduceOffspringOperator=
--reduceOffspringPressure=
```

Values for operators : Tournament Deterministic Roulette Random
Values for selection pressure : Integer strictly over 0 **or** Real number between 0.0 and 1.0 .

Final reduce operator

Description :
This parameter decides of the way individuals will be selected to participate in the next generation. Several operator are avaiable in EASEA:

Tournament (need a selection pressure)

Deterministic

Roulette (only when Evaluator goal = maximise)

Random

When the selection pressure is an integer over 0, the best individual from the n will be selected.
When the selection pressure is a real number between 0.0 and 1.0, the best individual of 2 will be selection with the probability p given by the selection pressure.

EASEA syntax :

```
Final reduce operator: OPERATOR SELECTION_PRESSURE
```

Values for operators : Tournament Deterministic Roulette Random
Values for selection pressure : Integer strictly over 0 **or** Real number between 0.0 and 1.0 .

Compile line syntax :

```
--reduceFinalOperator=
--reduceFinalPressure=
```

Values for operators : Tournament Deterministic Roulette Random
Values for selection pressure : Integer strictly over 0 **or** Real number between 0.0 and 1.0 .

Elitism

Description :
This parameter determines the type of elitism of the evolutionary algorithm.

EASEA syntax :

```
Elitism :
```

Values : Strong **or** Weak

Compile line syntax :

```
---eliteType=
```

Values : 1 (for Strong) **or** 0 (for Weak)

Elite

Description :
This parameter sets the number of individuals that are going to be "elites". Set this parameter to 0 to deactivate the elitism

EASEA syntax :

```
Elite :
```

Values : Positive Integer. 0 to deactive elitism.

Command line syntax :

```
--elite=
```

Values : Positive Integer. 0 to deactive elitism.

Remote Island Model Parameters

Remote island model

Description :
Boolean that will activate the migration of individuals towards other EASEA instances of the same problem that are located on remote machines. The address of the machines has to be specified in the *IP file* parameter.

EASEA syntax :

```
Remote island model :
```

Values : true **or** false.

Command line syntax :

```
---remoteIslandModel=
```

Values : 1 (for true) **or** 0 (for false).

IP file

Description :
This parameter contains the path to the File containing the IP addresses and port of all the remote machines that can receive individuals. This file needs to be structure as such:
a1.b1.c1.d1:p1
a2.b2.c2.d2:p2
...
an.bn.cn.dn:pn

No blank line is allowed at the end of the file.
An individual can be send to a machine that isn't active.
The IP address and port of the local machine can be present in the file. EASEA will recognize and ignore it.
If there are no IP addresses in the file, or if only the local machine's address is present in that file, the *Remote island model* Parameter will be set to *false*.

EASEA syntax :

`IP file :`

Values : The path to the file containing the IP addresses (example: ./ip.txt)

Command line syntax :

`--ipFile=`

Values : The path to the file containing the IP addresses (example: ./ip.txt)

Server Port

'Description :
This parameter sets the port that will be used by the island server to listen for the arrival of newcomers.

EASEA syntax :

`Server port :`

Values : A valid port number.

Command line syntax :

`---serverPort=`

Values : A valid port number.

Migration probability

Description :
This parameter sets the probability for and individual to migrate to a remote machine every generation.

EASEA syntax :

`Migration probability :`

Values : A real number between 0.0 and 1.0 .

Command line syntax :

`--migrationProbability=`

Values : A real number between 0.0 and 1.0 .

Memetic Parameters

Number of optimisation iterations

Description :
This parameters gives the number of iterations the local search algorithm will perform.

EASEA syntax :

```
Number of optimisation iterations :
```

Values : A positive integer.

Command line syntax :

```
--optimiseIterations=
```

Values : A positive integer.

Baldwinism

Description :
This parameter will set the behaviour of the memetic algorithm. Setting it to *true* will turn the algorithm into a **Baldwinian** memetic algorithm. Setting it to *false* will turn the algorithm into a **Lamarckian** memetic algorithm. This parameter's default value is *true*.

EASEA syntax :

```
Baldwinism :
```

Values : true **or** false.

Command line syntax :

```
--baldwinism=
```

Values : 1 (for true) **or** 0 (for false).

Miscalleous Parameters

Print stats

Description :
This parameter set to *true* will show different stats every generations:

Generation number

Time passed since the start of the algorithm

Number of evaluations completed

Fitness of the best individual found

Avegrage fitness

Standard deviation of the fitness

This parameter's default value is *true*.

EASEA syntax :

`Print stats :`

Value : true **or** false.

Command line syntax :

`printStats=`

Values : 1 (for true) **or** 0 (for false).

Plot stats

Description
This parameter set to *true* will open a **gnuplot** process, that will plot the curves representing the evolution of:

The best fitness

The avegage fitness

The standard deviation of the fitness

To be able to use this parameter, gnuplot has to be installed on the machine. This parameter is only avaiable for Linux users.
The ploting of the curves will happen simultaniously to the evolution. At the end of the evolution, the resulting curves will be saved in a **.png** file. It's name will be the same as the project name. This parameter's default value is *false*.

EASEA syntax :

`Plot stats :`

Value : true **or** false.

Command line syntax :

`plotStats=`

Values : 1 (for true) **or** 0 (for false).

Generate csv stats file

Description :
This parameter set to *true* will save the different stats shown with the Print stats parameter to a **.csv** file at the end of the evolution. The file name will be the same as the project name. This parameter's default value is *false*.

EASEA syntax :

```
Generate csv stats file :
```

Values : true **or** false.

Command line syntax :

```
--generateCSV=
```

Values : 1 (for true) **or** 0 (for false).

Generate gnuplot script

Description :
This parameter set to *true* will create a gnuplot script that will allow to plot the evolution in a **.png** file which, name will be the same as the project name. To plot the evolution, the script will save the stats in a **.dat** file. This parameter's default value is *false*.

EASEA syntax :

```
Generate gnuplot script :
```

Values : true **or** false.

Command line syntax :

```
--generateGnuplotScript=
```

Values : 1 (for true) **or** 0 (for false).

Generate R script Description :
This parameter set to *true* will create a R script that will allow to plot the evolution in a **.png** file, which name will be the same as the project name. To plot the evolution, the script will save the stats in a **.csv** file. This parameter's default value is *false*.

EASEA syntax :

```
Generate R script :
```

Values : true **or** false.

Command line syntax :

`--generateRScript=`

Values : 1 (for true) **or** 0 (for false).

Save population Description :
This parameter set to *true* will save the final population in a **.pop** file, which name will be the same as the project name. The population will be in a EASEA serialized form. This parameter's default value is *false*.

EASEA syntax :

`Save population :`

Values : true **or** false.

Command line syntax :

`--savePopulation=`

Values : 1 (for true) **or** 0 (for false).

Start from file Description :
This parameter set to *true* will initialize the population using the population saved the **.pop** file that is in the current folder. The file has to have the save name as the project. For this initialization to work, the individuals contained in the file has to have the exact same Genome, and the number of individuals has to be equal or greater to the population size.
The individuals have to be in the EASEA serialization form.
This parameter's default value is *false*.

EASEA syntax :

`Start from file :`

Values : true **or** false.

Command line syntax :

`--saveFromFile=`

Values : 1 (for true) **or** 0 (for false).

Print initial population Description :
This parameter allow to print the initial population. This parameter's default value is 0.

EASEA syntax :
This is a command line exclusive parameter.

Command line syntax :

`--printInitialPopulation=`

Values : 1 (for true) **or** 0 (for false).

Print final population Description :
This parameter allow to print the final population. This parameter's default value is 0.

EASEA syntax :
This is a command line exclusive parameter.

Command line syntax :

```
--printFinalPopulation=
```

Values : 1 (for true) **or** 0 (for false).

User defined parameters

EASEA defines 5 parameters that are left free to be used by the ".ez" programmer.

User parameters :

```
 --u1 arg                      User defined parameter 1
--u2 arg                      User defined parameter 2
--u3 arg                      User defined parameter 3
--u4 arg                      User defined parameter 4
--u5 arg                      User defined parameter 5
```

Types : u1 and u2 are string variables, u3 to u5 are integers.

"'Usage Exemple : "' In the .ez source code

```
int a = setVariable("u3",b);
```

'where :
"a" is the variable set to the argument or default value.
"u3" is the argument name.
"b" is the default integer value, if the u3 argument has not been set.

Developer documentation

Compiler

Parsing the language

The compile does its parsing using lex and YACC, and more specifically the alex and AYACC implementation from bumble-bee software.
A good enhancement is to move from alex and AYACC to flex and Bison. The main reason is that the ayacc/alex compilers only work on windows and there is some licenses fuzziness.

The AYACC “compiler” generated the “EaseaParse.cpp” file from “EaseaParse.y” . The alex “compiler” generated the “EaseaLex.cpp” file from “Easealex.l” .

Those file don’t need to be re-generated at each compilation, only when changed. The correct command through wine are:

wine /.WINE/drive_c/Program
Files/Parser
Generator/BIN/ayacc.exe $< -Tcpp -d wine /.wine/drive_c/Program
Files/Parser
Generator/BIN/ALex.exe $< -Tcpp -i

Note When using the urrent version of ayacc or alex some problems arise. The generated code classes names changed leading to multiple error such as no “class YYPARSERNAME found” because the rest of the easea compiler defined this class name as YYPARSENAME.
The only workaround it to vimdiff the files and relie on the compiler error to manually fix the definitions problems.

EASEA library

The idea behind easea is the following: the EASEA compiler wrap predefined sections in class and function, and by polyphormism and pointer manipulation, are exploitable by evolutionary engine.

The evolutionary engine is implemented as the evolutianary library, the libeasea.

[sub:EASEA library]

Main evolutionary loop

Example

Additional ressources

Additional ressource can be found on the project wiki : and repository