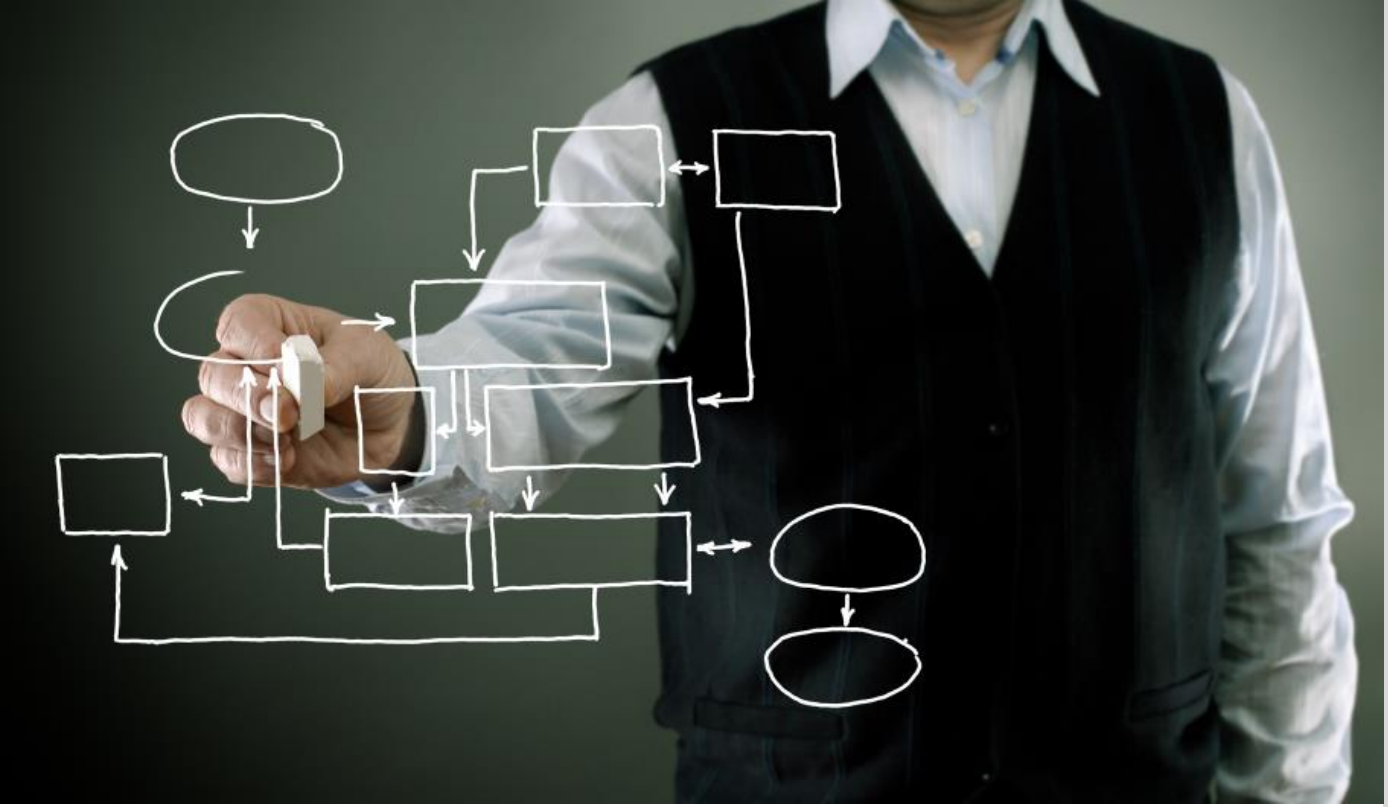


GPDL: A Framework-Independent Problem Definition Language for Grammar-Guided Genetic Programming

Gabriel Kronberger, Michael Kommenda, Stefan Wagner, Heinz Dobler

HEAL
HEURISTIC AND EVOLUTIONARY ALGORITHMS LABORATORY

HeuristicLab

Josef Ressel-Zentrum HEUREKA!

Contact:
Gabriel Kronberger
Heuristic and Evolutionary Algorithms Lab (HEAL)
Softwarepark 11
A-4232 Hagenberg

e-mail:
gabriel.kronberger@heuristiclab.com
Web:
http://heal.heuristiclab.com
http://heureka.heuristiclab.com

It is hard to use GP Frameworks

Several good GP implementations are available

Frequently implemented: symbolic regression, artificial ant, Boolean problems, …

Implementation of new problems is often cumbersome

Implementation of several classes is necessary (e.g. for HeuristicLab or ECJ)

Often a lot of boiler-plate code is necessary

Several questions on our mailing list about the implementation of custom problems

Lawn-Mower problem in HL

Implemented classes:

- Symbols
- TreeNodes
- Grammar
- Interpreter
- Evaluation operator

Effort: two days

Source Code of the Lawn-Mower-Problem for HeuristicLab

```
using HeuristicLab.Common;
using HeuristicLab.Encodings.SymbolicExpressionTreeEncoding;
using HeuristicLab.Persistence.Default.CompositeSerializers.Storable;

namespace HeuristicLab.Problems.LawnMower {
  // final implementation of the left symbol
  // with persistence and cloning patterns
  [StorableClass]
  public class Left : Symbol {
    public override int MinimumArity { get { return 0; } }
    public override int MaximumArity { get { return 0; } }

    // storable constructor for persistence
    [StorableConstructor]
    private Left(bool deserializing) : base(deserializing) { }
    // cloning constructor
    private Left(Left original, Cloner cloner)
      : base(original, cloner) {
      // nothing needs to be cloned
    }
    public Left()
      : base("Left", "Turns the lawn mower to the left.") {
    }
     // clone method override
    public override IDeepCloneable Clone(Cloner cloner) {
      return new Left(this, cloner);
    }
  }
}
```

Source Code of the Lawn-Mower-Problem for HeuristicLab

```
using HeuristicLab.Common;
using HeuristicLab.Encodings.SymbolicExpressionTreeEncoding;
using HeuristicLab.Persistence.Default.CompositeSerializers.Storable;

namespace HeuristicLab.Problems.LawnMower {
  // final implementation of the left symbol
  // with persistence and cloning patterns
  [StorableClass]
  public class Left : Symbol {
    public override int MinimumArity { get { return 0; } }
    public override int MaximumArity { get { return 0; } }

    // storable constructor for persistence
    [StorableConstructor]
    private Left(bool deserializing) : base(deserializing) { }
    // cloning constructor
    private Left(Left original, Cloner cloner)
      : base(original, cloner) {
      // nothing needs to be cloned
    }
    public Left()
      : base("Left", "Turns the lawn mower to the left.") {
    }
     // clone method override
    public override IDeepCloneable Clone(Cloner cloner) {
      return new Left(this, cloner);
    }
  }
}
```

Overhead of programming language

Boiler-plate code

Source Code of the Lawn-Mower-Problem for HeuristicLab

```
using HeuristicLab.Common;
using HeuristicLab.Encodings.SymbolicExpressionTreeEncoding;
using
HeuristicLab.Persistence.Default.CompositeSerializers.Storable;

namespace HeuristicLab.Problems.LawnMower {
  [StorableClass]
  public class Prog : Symbol {
    public override int MinimumArity { get { return 2; } }
    public override int MaximumArity { get { return 2; } }

    [StorableConstructor]
    private Prog(bool deserializing) : base(deserializing) { }
    private Prog(Prog original, Cloner cloner)
      : base(original, cloner) {
    }

    public Prog()
      : base("Prog",
             "Prog executes two branches sequentially and
returns the result of the last branch.") {
    }

    public override IDeepCloneable Clone(Cloner cloner) {
      return new Prog(this, cloner);
    }
  }
}
```

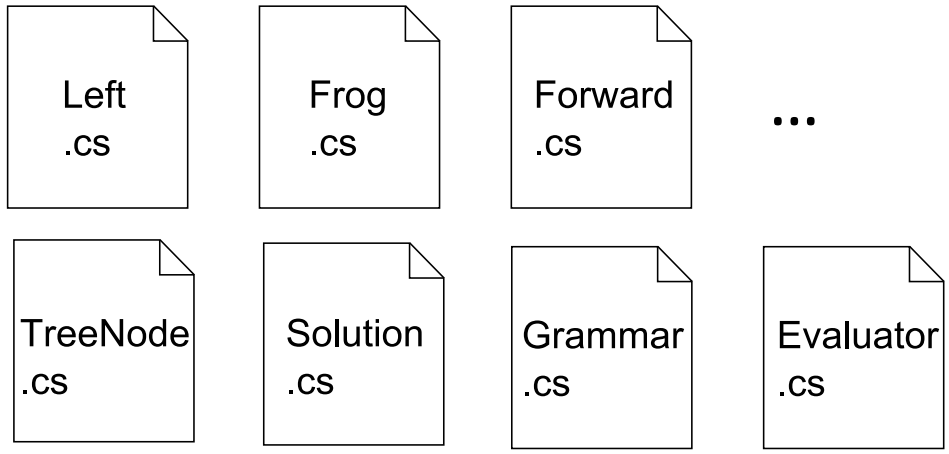
```
using HeuristicLab.Common;
using HeuristicLab.Encodings.SymbolicExpressionTreeEncoding;
using
HeuristicLab.Persistence.Default.CompositeSerializers.Storable;

namespace HeuristicLab.Problems.LawnMower {
  [StorableClass]
  public class Sum : Symbol {
    public override int MinimumArity { get { return 2; } }
    public override int MaximumArity { get { return 2; } }

    [StorableConstructor]
    private Sum(bool deserializing) : base(deserializing) { }
    private Sum(Sum original, Cloner cloner)
      : base(original, cloner) {
    }

    public Sum()
      : base("Sum", "Sum of two integer vectors.") {
    }

    public override IDeepCloneable Clone(Cloner cloner) {
      return new Sum(this, cloner);
    }
  }
}
```

Source Code of the Lawn-Mower-Problem for HeuristicLab

```
namespace HeuristicLab.Problems.LawnMower {
  [StorableClass]
  class ConstantTreeNode : SymbolicExpressionTreeTerminalNode {
    // the random integer vector is stored as a tuple of ints.
    private Tuple<int, int> value;
    [Storable]
    public Tuple<int, int> Value {
      get { return value; }
      private set { this.value = value; }
    }

    [StorableConstructor]
    private ConstantTreeNode(bool deserializing) :
                                      base(deserializing) { }
    private ConstantTreeNode(ConstantTreeNode original,
                             Cloner cloner)
      : base(original, cloner) {
      // important here we need to implement the cloning
      // of the local data (integer vector)
      this.value =
        new Tuple<int, int>(original.value.Item1,
                            original.value.Item2);
    }

    // default constructor, the symbol of a ConstantTreeNode is
    // always
    // a Constant.
    public ConstantTreeNode()
      : base(new Constant()) {
    }

    public override IDeepCloneable Clone(Cloner cloner) {
      return new ConstantTreeNode(this, cloner);
    }

    // This tree node holds data (integer vector).
    // Thus, this property must be overridden to return true
    // to indicate to the algorithm that the data in this node
    // must be reset initially.
    public override bool HasLocalParameters {
      get { return true; }
    }
    // ResetLocalParameters is called by the algorithm to reset
    // the data of the tree node when initializing the
population.
    // Here we want to create a random integer vector where
both
    // components are distributed uniformly.
    public override void ResetLocalParameters(IRandom random) {
      value = new Tuple<int, int>(random.Next(-20, 20),
                                  random.Next(-20, 20));
    }
  }
}
```

Source Code of the Lawn-Mower-Problem for HeuristicLab

… 10 more pages!



Let‘s solve the same problem using a different GP implementation

We need to implement the whole thing again!

- Different programming language
- Different API
- Different GP-system (e.g. grammatical evolution)

When implementing a new problem it‘s hard to know:

- which GP-approach works well,
- which implementation can be used, and
- if the problem can be solved by GP at all

We need to make it easier!

- Our users are no experts in GP
- Our users have no clue how to set algorithm parameters
- Our users have no clue which framework to use
- Our users don‘t want to learn the API of a GP framework
- Our users don‘t want to write boiler-plate code

- … but they know how solutions should look like

What is a GP problem?

Traditionally:

Set of function symbols + set of terminal symbols + fitness function

Many modern GP systems use grammars to define valid solutions

- Grammars can be used to defined formal languages
- Some implementations already allow definition of grammars using BNF (e.g. GEVA)
- Actually all GP systems can be considered grammar-guided
  (but often the grammar is really simple)

Alternative :

Formal language + objective function

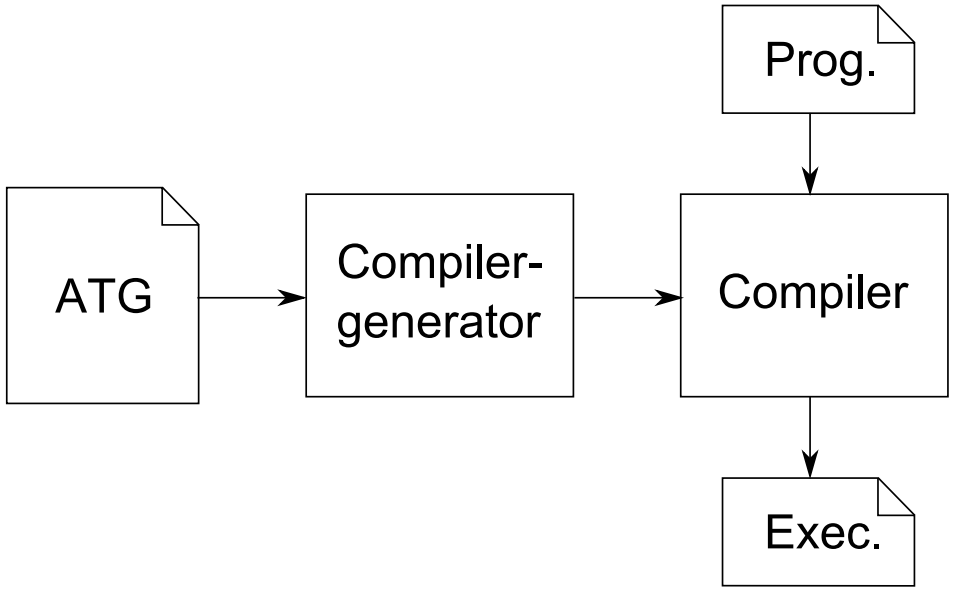
(in this context formal language = set of valid solutions)

Attributed Grammars with Semantic Actions

Commonly used in compiler-construction to define

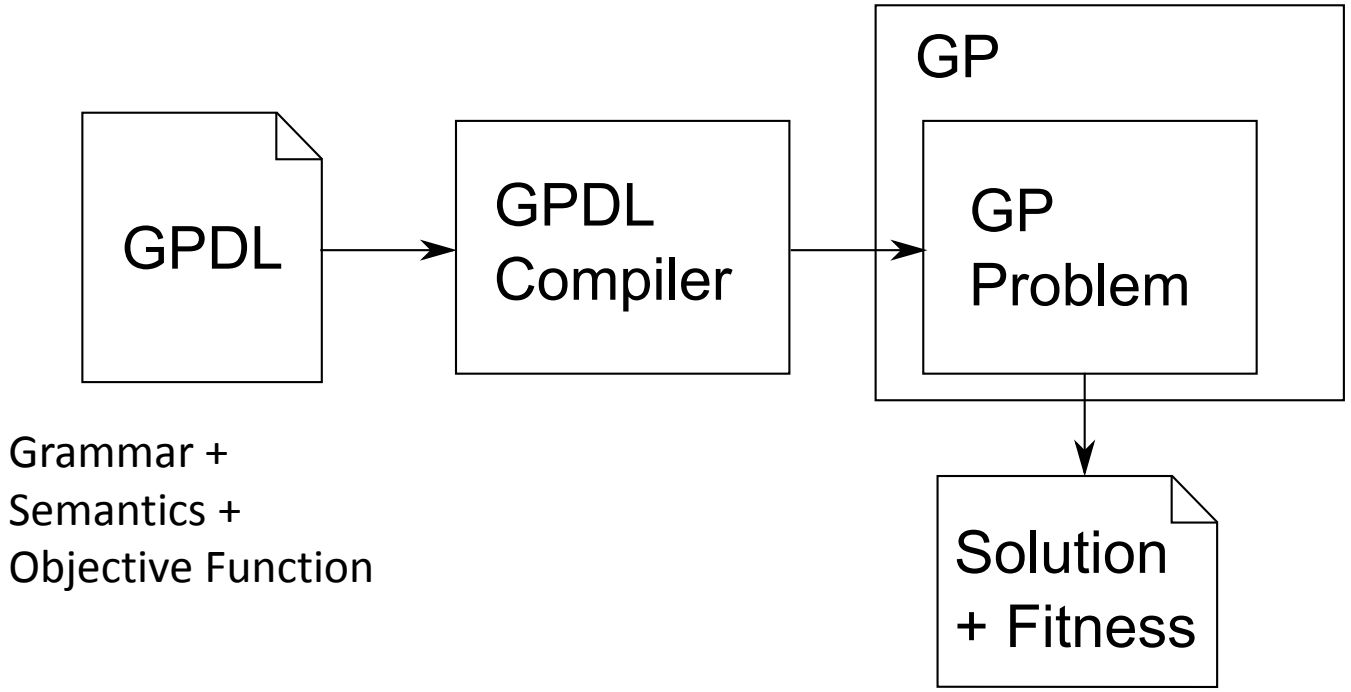
lexical structure + syntax + semantics of programming languages

Compiler-generator **translates attributed grammar into an interpreter or compiler** for the language



GP Problem =
ATG + Semantic Actions + Objective Function

We can extend the concept for the definition of GP problems



Example: Symbolic Regression Problem

```
PROBLEM SymbRegKoza
CODE <<
  double[,] inputValues;
  double[] targetValues;
  string[] variableNames;

  double GetValue(double[,] data, string varName, int row)
    { /* */  }
  double RSquared(double[] xs, double[] ys)
    { /* */  }
  void LoadData(string fileName, out double[,] x,
                out string[] varName, out double[] target)
    { /* */  }
>>

INIT <<
  LoadData("filename.csv",
         out inputValues, out variableNames, out targetValues);
>>

/************************************************/
NONTERMINALS
  Model<<int row, out double val>>.
  RPB<<int row, out double val>>.
  Add<<int row, out double val>>.
  Sub<<int row, out double val>>.
  Mul<<int row, out double val>>.
  Div<<int row, out double val>>.


/************************************************/
TERMINALS
  ERC<<out double val>>
    CONSTRAINTS
      val IN RANGE <<-100>> .. <<100>>
  .

  Var<<out string varName>>
    CONSTRAINTS
      varName IN SET <<variableNames>>
  .
```

```
/************************************************/
RULES
  Model<<int row, out double val>> =
    RPB<<row, out val>> .

  RPB<<int row, out double val>> =  LOCAL << string varName; >>
    Add<<row, out val>>
    | Sub<<row, out val>>
    | Div<<row, out val>>
    | Mul<<row, out val>>
    | Var<<out varName>>        SEM << val =
                                  GetValue(inputValues, varName, row);
                                >>
    | ERC<<out val>>
  .

  Add<<int row, out double val>> =           LOCAL << double x1, x2; >>
      RPB<<row, out x1>> RPB<<row, out x2>>  SEM<< val = x1 + x2; >>
  .
  Sub<<int row, out double val>> =           LOCAL << double x1, x2; >>
      RPB<<row, out x1>> RPB<<row, out x2>>  SEM<< val = x1 - x2; >>
  .
  Div<<int row, out double val>> =           LOCAL << double x1, x2; >>
      RPB<<row, out x1>> RPB<<row, out x2>>  SEM<< val = x1 / x2; >>
  .
  Mul<<int row, out double val>> =           LOCAL << double x1, x2; >>
      RPB<<row, out x1>> RPB<<row, out x2>>  SEM<< val = x1 * x2; >>
  .

/************************************************/

MAXIMIZE                /* could also use the keyword MINIMIZE here */
  <<
    var rows = Enumerable.Range(0, inputValues.GetLength(0));
    var predicted = rows.Select(r => {
      double result;
      Model(r, out result);  /* we can call the root symbol directly */
      return result;
    });
    return RSquared(predicted, targetValues);
  >>
END SymbRegKoza.
```

Our contribution

Definition of a language for the definition of GP problems

- Based on the idea of ATGs with semantic actions
- Extended by a fitness function

Reference implementation using HeuristicLab as the backend

- Compiler-compiler: Coco-2 (available in many languages)

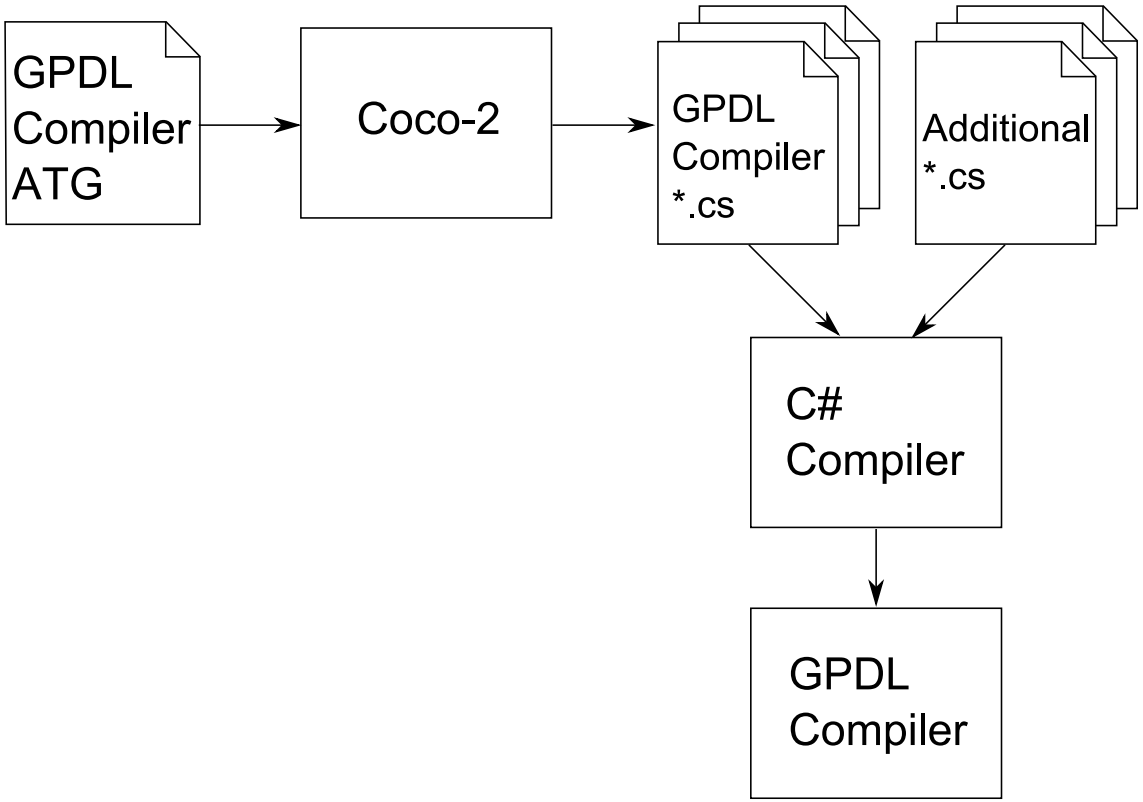
First step to facilitate discussion in the community

- Contributions are welcome!
- Open development of language specification and reference implementation
- Send us comments and feedback on syntax and semantics

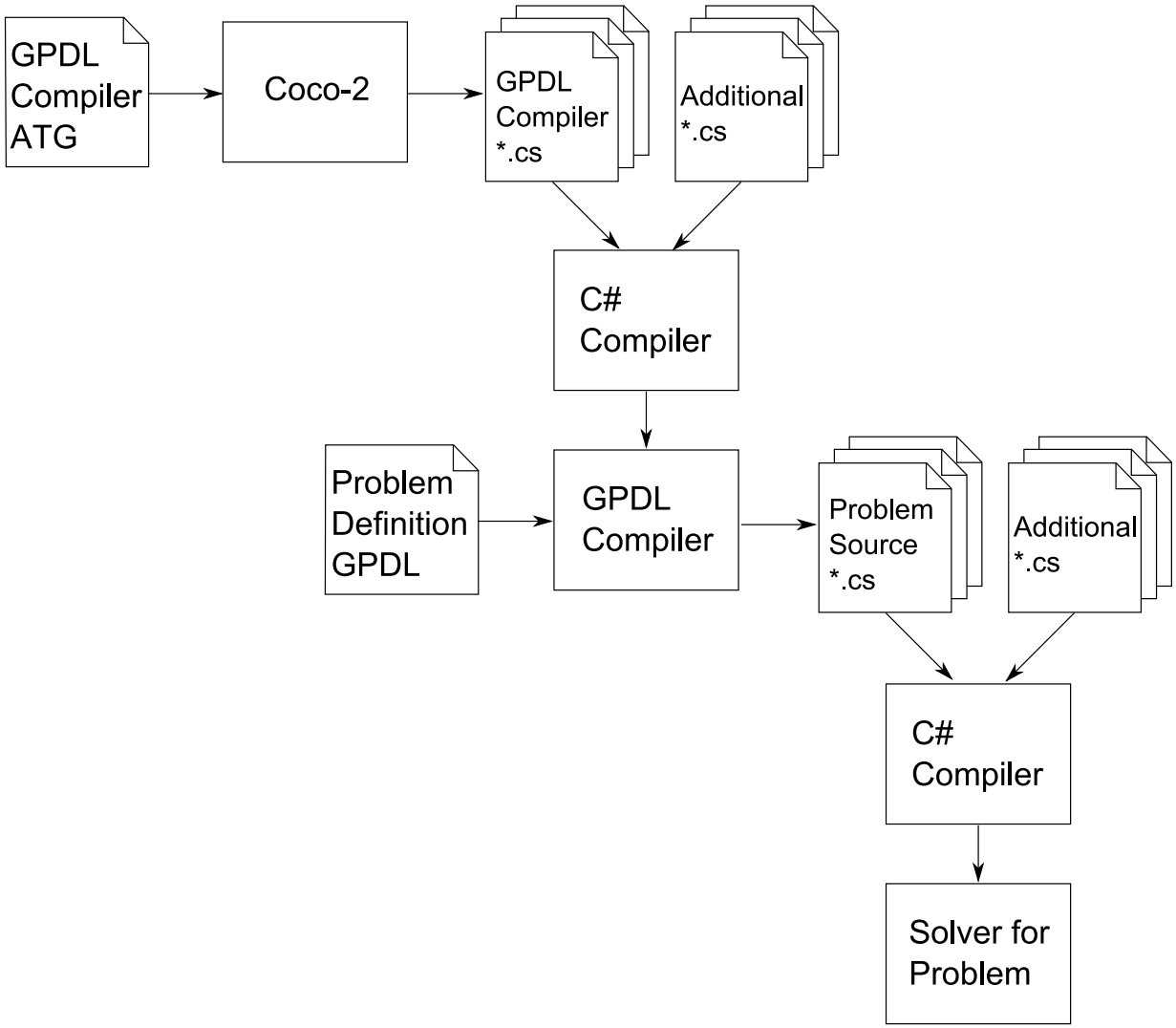
Syntax of GPDL (excerpt)

```
COMPILER GPDefSyntaxAnalyzer

KEYWORDS
  'PROBLEM'. 'END'. 'EPS'. 'LOCAL'. 'NONTERMINALS'. 'RULES'.
  'SEM'. 'MAXIMIZE'. 'MINIMIZE'. 'TERMINALS'. 'CONSTRAINTS'. 'INIT'. 'CODE'.
  'IN'. 'SET'. 'RANGE'.

TOKENS
  '='. '|'. '.'. '('. ')'. '['. ']'. '{'. '}'. '>>'. '..'.

NONTERMINALS
  GPDefSyntaxAnalyzer.
  SemDecl. SemAction. NonterminalDecl.
  RuleDef. SynExpr. SynTerm. SynFact. TerminalDecl.
  ConstraintDef. ConstraintRule. SetDefinition.

RULES
  GPDefSyntaxAnalyzer =
    'PROBLEM' ident
            ['CODE' /* SourceText */]
            ['INIT' /* SourceText */]
    'NONTERMINALS' { NonterminalDecl }
    'TERMINALS' { TerminalDecl }
    'RULES' { RuleDef }
    ('MAXIMIZE' | 'MINIMIZE') /* SourceText */
    'END' ident '.'.

END GPDefSyntaxAnalyzer.
```

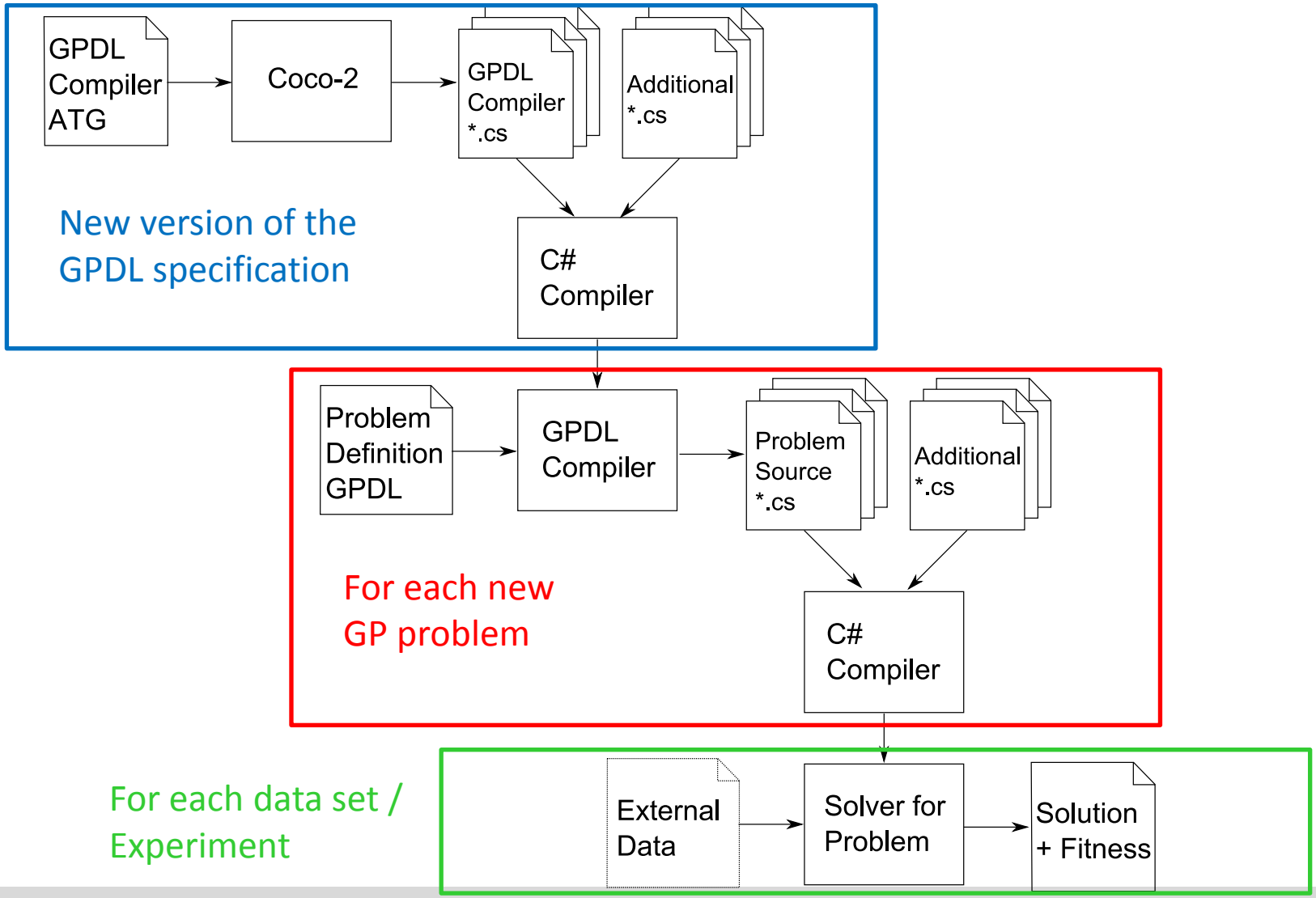
Reference Implementation for HeuristicLab: Building the Compiler



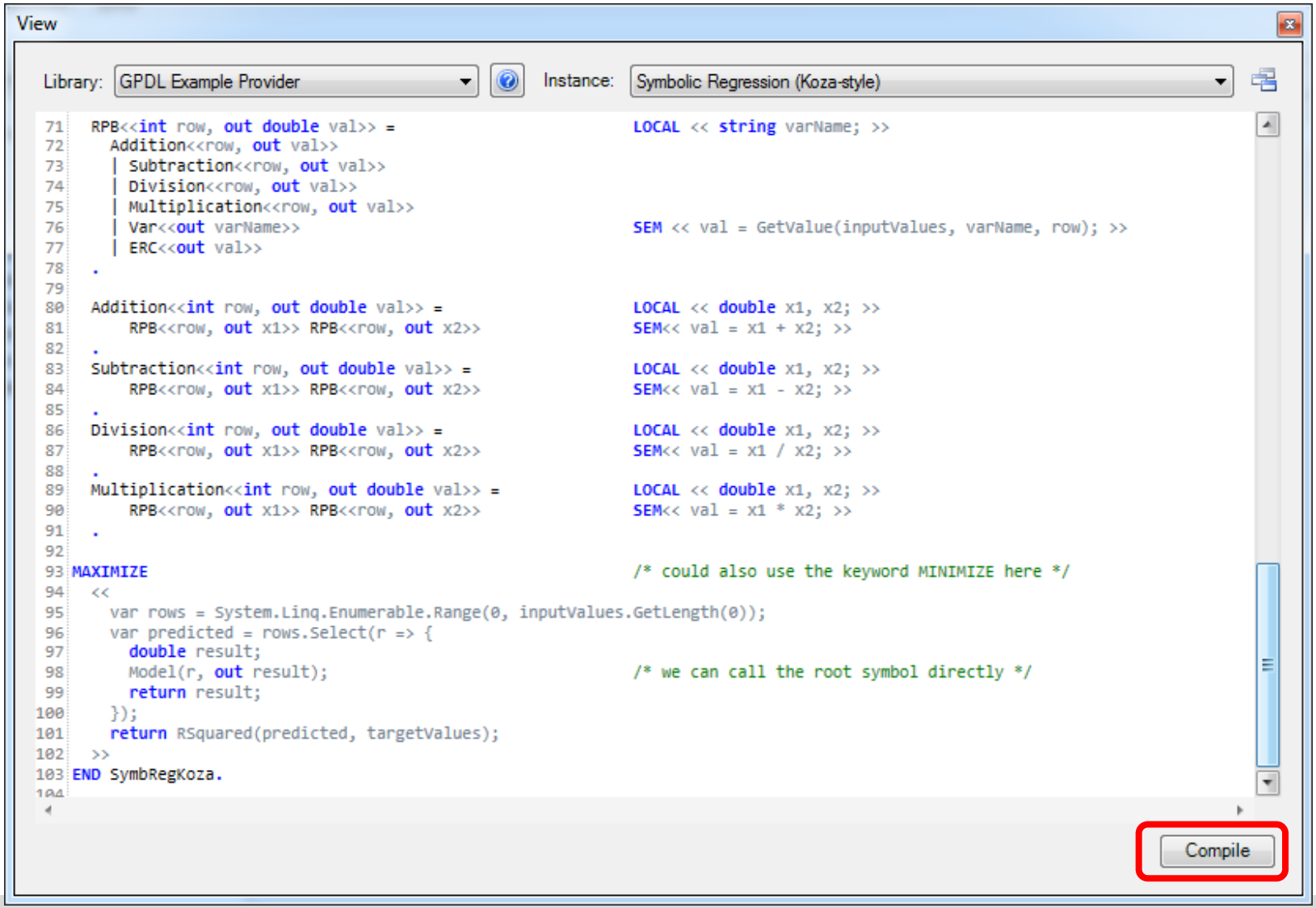
Reference Implementation for HeuristicLab: Building a Solver for a Problem



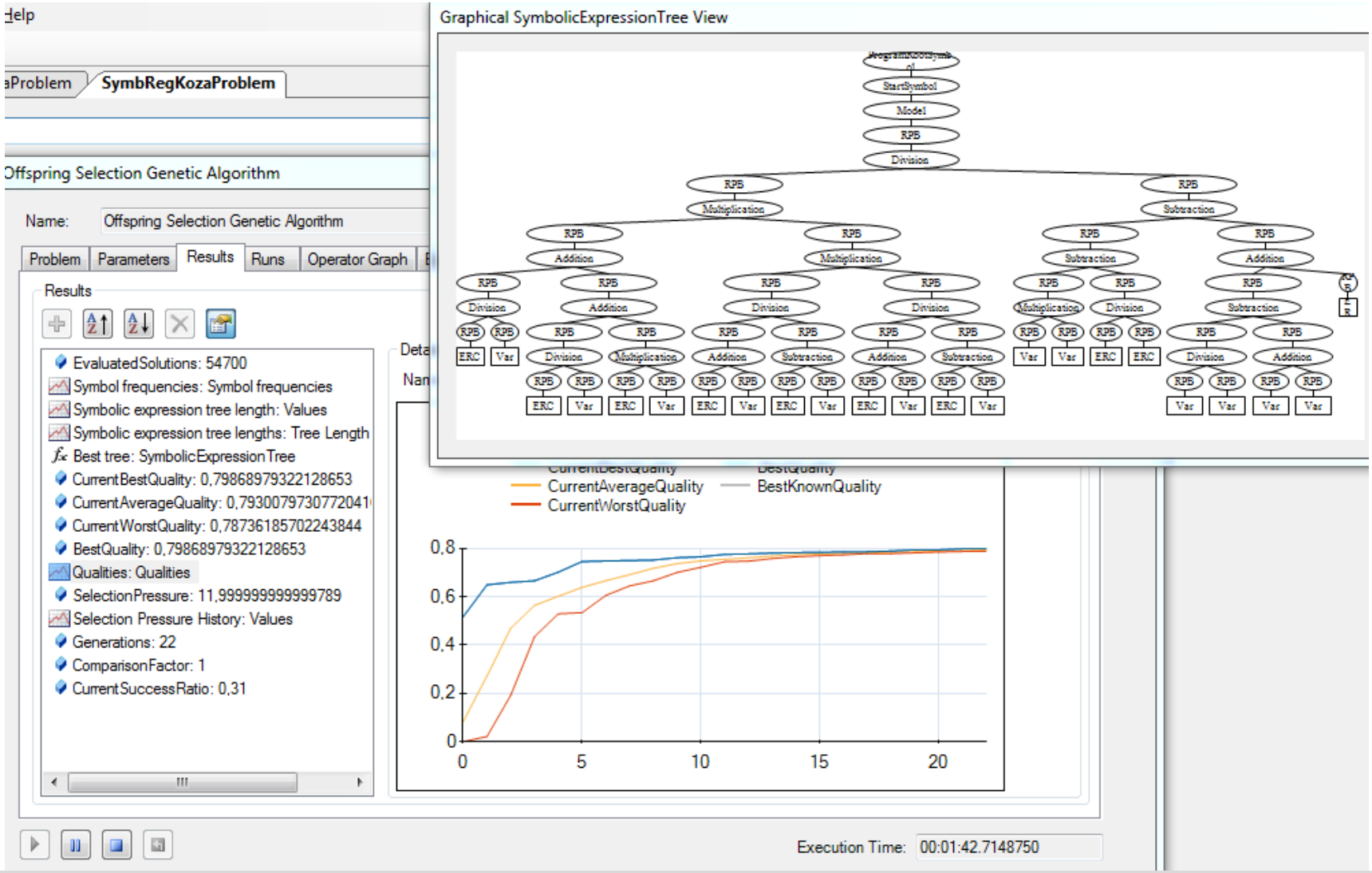
Reference Implementation for HeuristicLab: Using the Solver



Integration into HeuristicLab



Integration into HeuristicLab

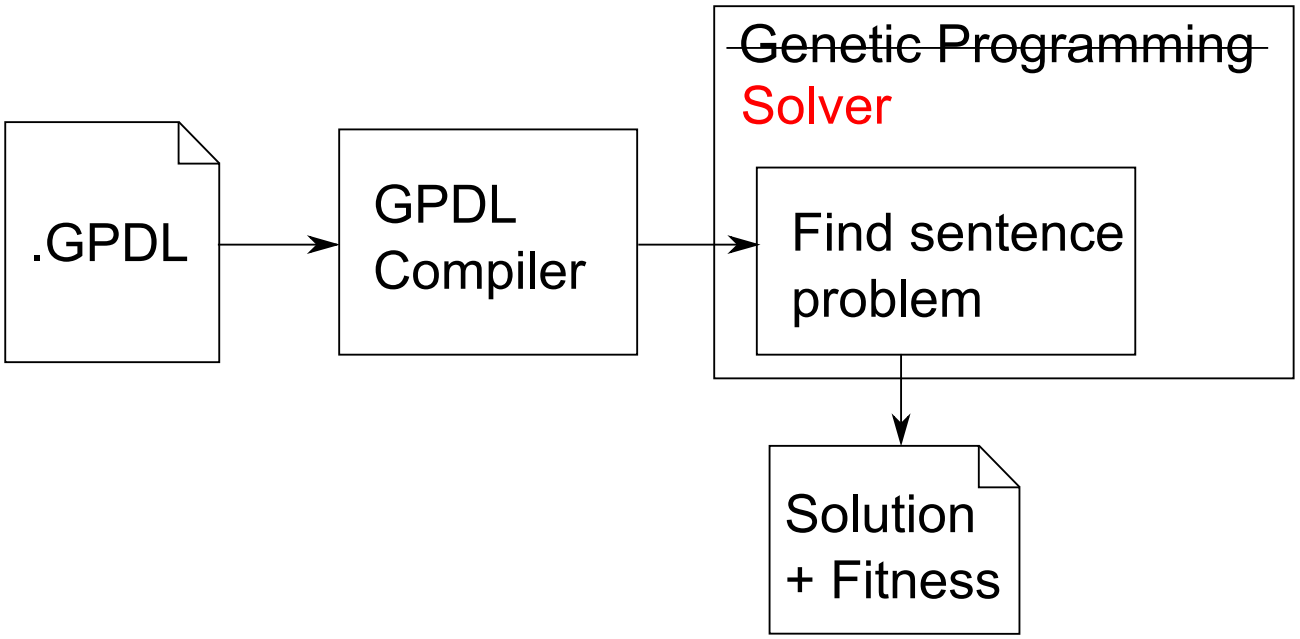


GPDL: ~~Genetic~~ General Programming Problem Definition Language?

GPDL is framework-independent

- The problem definition is fully de-coupled from the GP implementation
- Programming language of semantic actions is not fixed
  - Any language supported by Coco can be used

GPDL is also solver-independent!



GPDL: ~~Genetic~~ General Programming Problem Definition Language?

GP is only one way to find an optimal sentence

Alternatives:

- Enumeration
- Tree-Search
- Estimation of distribution algorithms

Different types of solvers can be tested easily

Compare modeling languages for MIP-formulations (e.g. GAMS)

Solvers that adapt to the problem instance?

- Explicit definition of syntactical structure and semantics of the problem language
- Information could be used by the solver

Next Steps

GPDL compiler is included in the next release of HeuristicLab

Not yet because we want to improve syntax/semantic error reporting

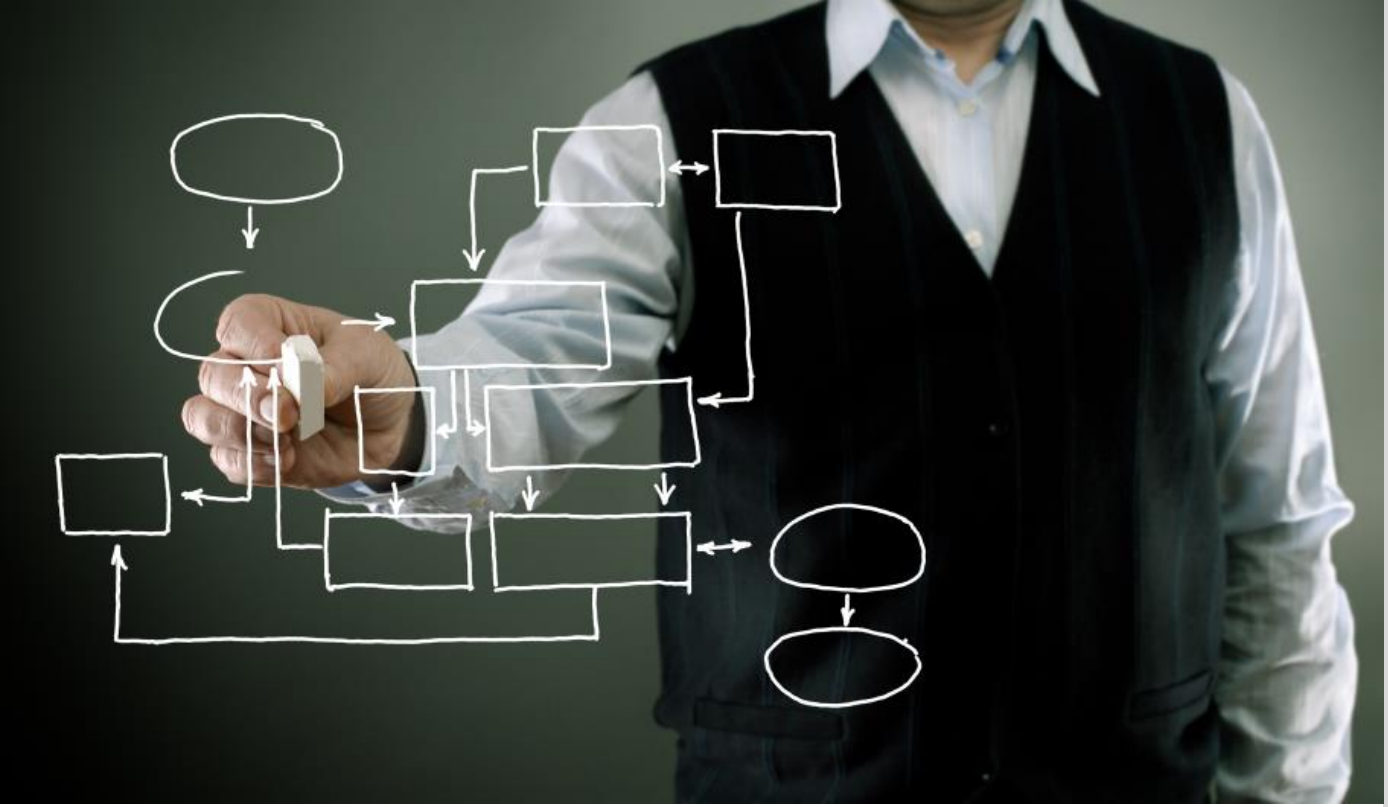
Updates of language specification and reference implementation

On our website.

Work on alternative solvers

We need help!

- **Implement additional problems**
- **Comments on the GPDL syntax / semantics**
- **Implement compilers for other backends**
  - ECJ,
  - GEVA
  - EpochX
  - DEAP
  - JGAP
  - ...



GPDL: A Framework-Independent Problem Definition Language for Grammar-Guided Genetic Programming

Gabriel Kronberger, Michael Kommenda, Stefan Wagner, Heinz Dobler

HEAL
HEURISTIC AND EVOLUTIONARY ALGORITHMS LABORATORY

HeuristicLab

Josef Ressel-Zentrum HEUREKA!

Contact:
Gabriel Kronberger
Heuristic and Evolutionary Algorithms Lab (HEAL)
Softwarepark 11
A-4232 Hagenberg

e-mail:
gabriel.kronberger@heuristiclab.com
Web:
http://heal.heuristiclab.com
http://heureka.heuristiclab.com