Automatic Feature Generation for Machine Learning Based Optimizing Compilation

Hugh Leather, Edwin Bonilla,
Michael O'Boyle

Institute for Computing Systems Architecture
University of Edinburgh, UK

Overview

- **Introduction to machine learning in compilers**
- Difficulties choosing features
- A feature space for a motivating example
- Searching the feature space
- Features for GCC
- Results
- Further and on going work

Introduction to machine learning in compilers

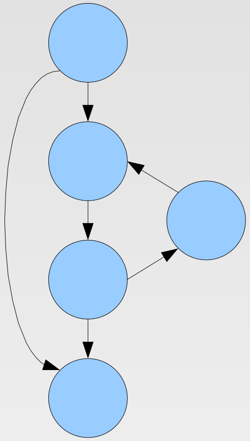
- Problem:
  - Tuning heuristics is hard
  - Architectures and compilers keep changing
- Goal:
  - Replace an heuristic with a Machine Learned one
  - ML performs very well

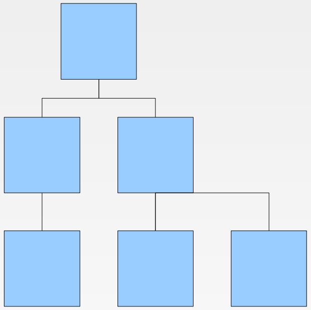
Introduction to machine learning in compilers

- How it works
  - Summarise data before heuristic (features)
  - Collect examples
  - Learn a model
  - Model predicts heuristic for new program

Introduction to machine learning in compilers

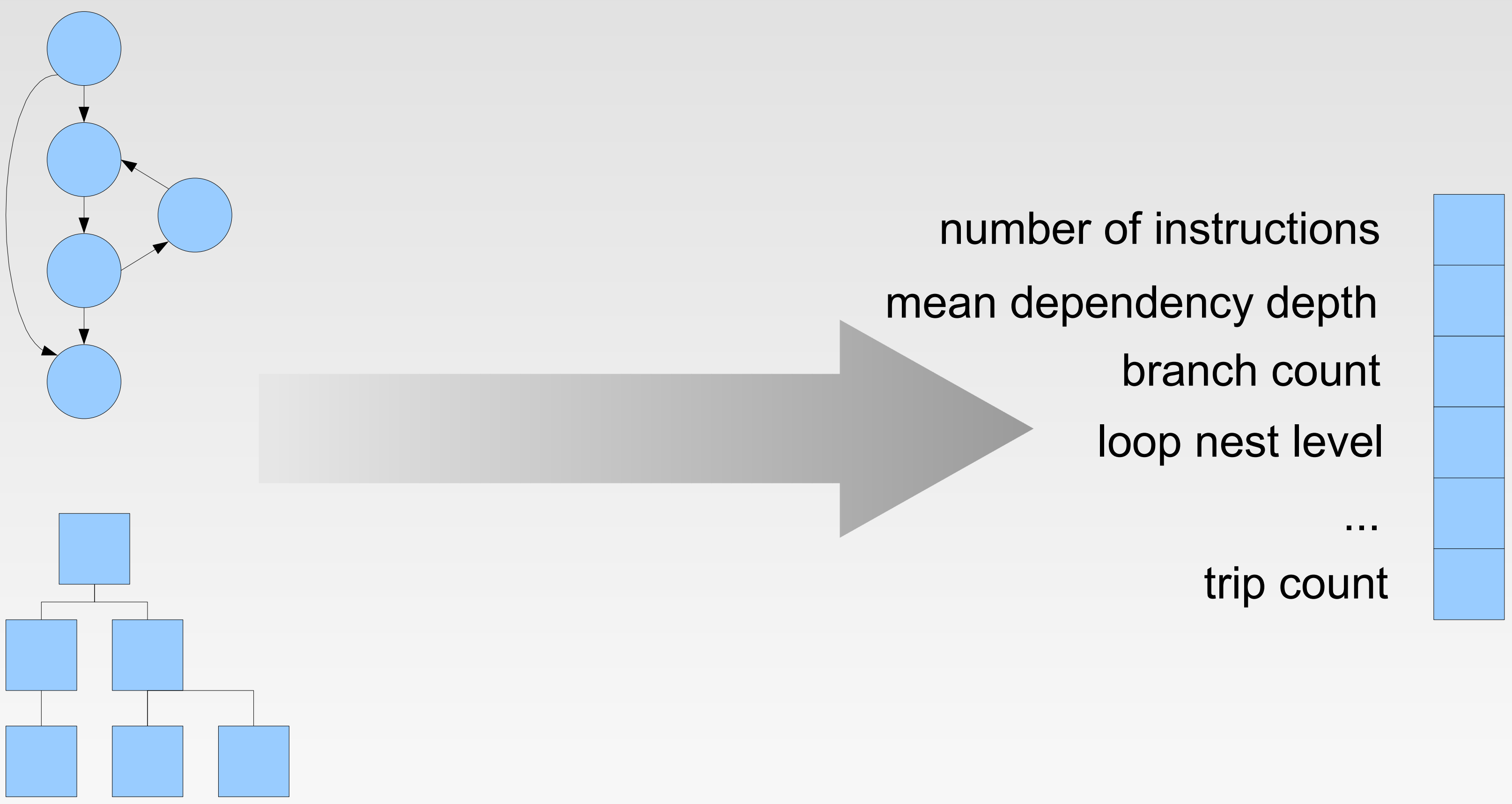
Start with compiler data structures
 AST, RTL, SSA, CFG, DDG, etc.





Introduction to machine learning in compilers

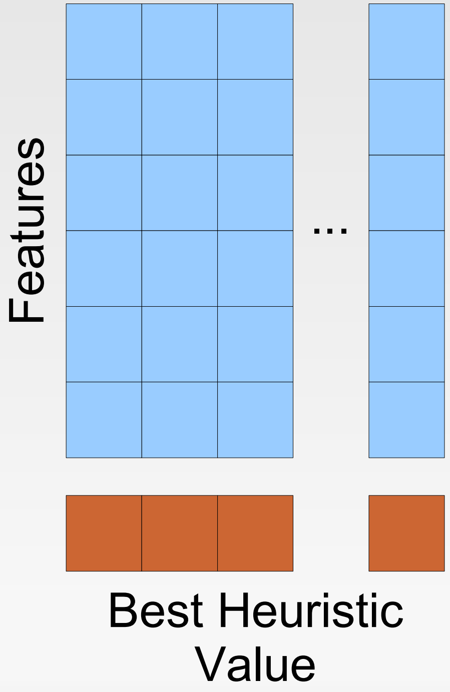
Human expert determines a mapping to a feature vector



Introduction to machine learning in compilers

Now collect many examples of programs, determining their feature values

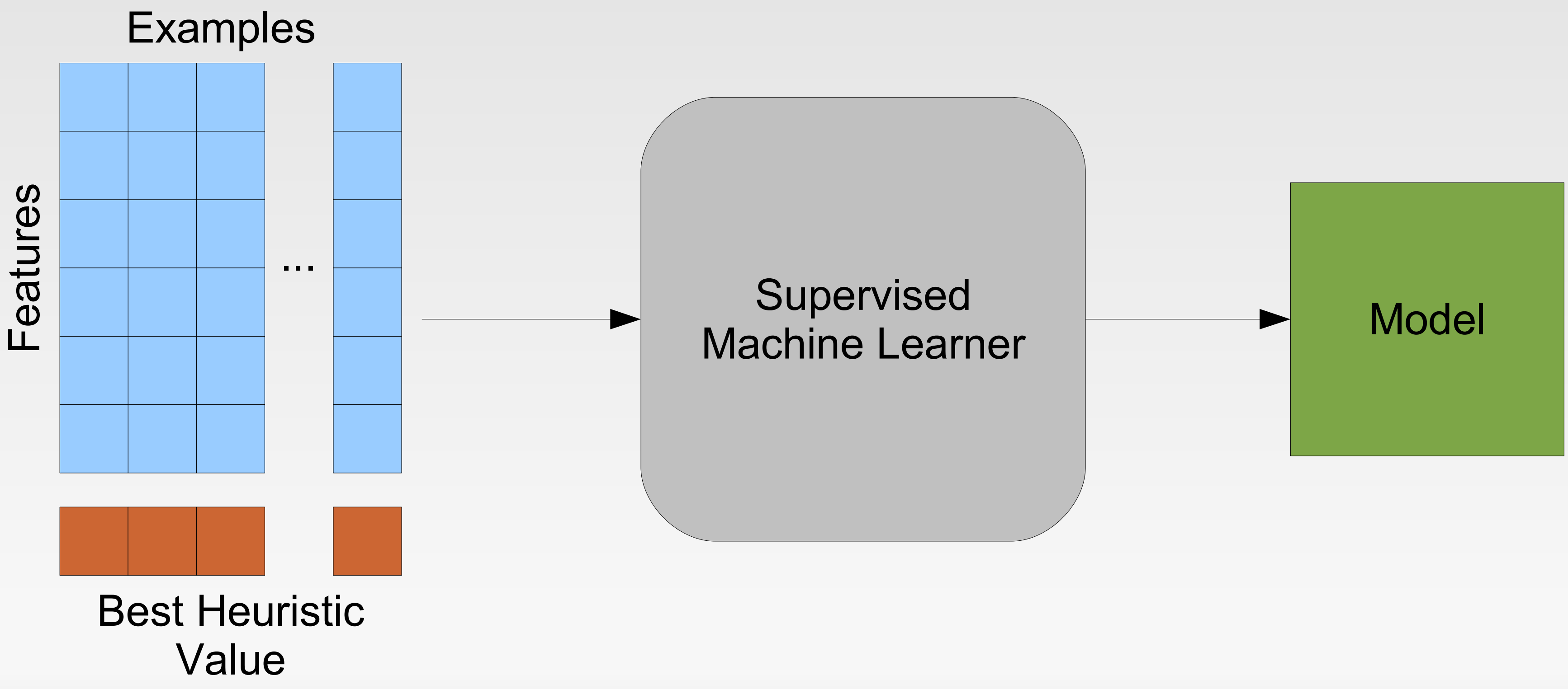
Execute the programs with different compilation strategies and find the best for each



Introduction to machine learning in compilers

Now give these examples to a machine learner

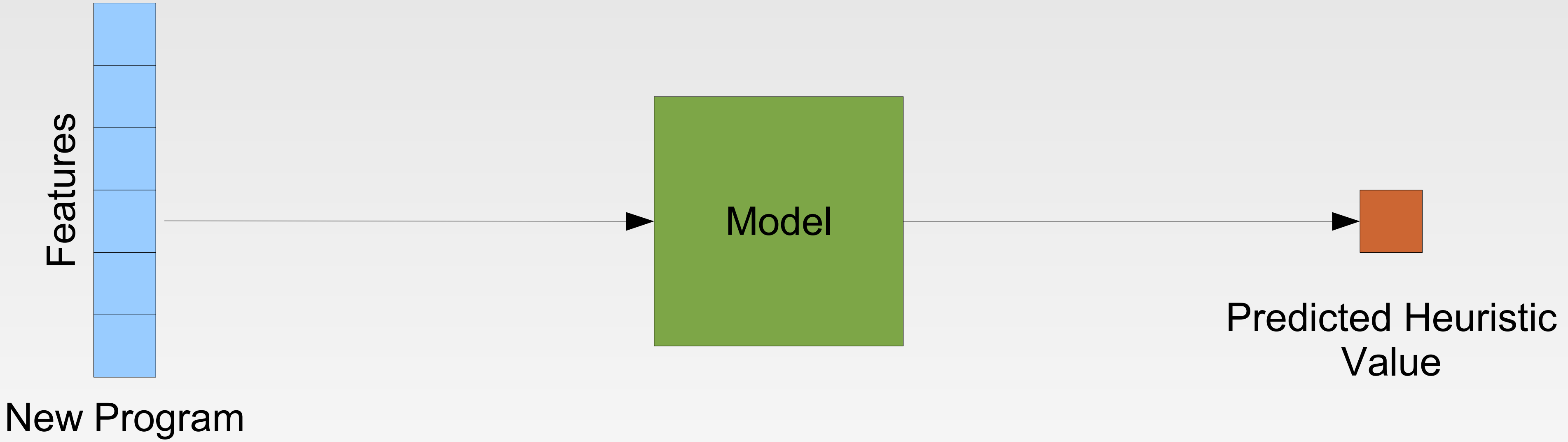
It learns a model



Introduction to machine learning in compilers

This model can then be used to predict the best compiler strategy from the features of a new program

Our heuristic is replaced

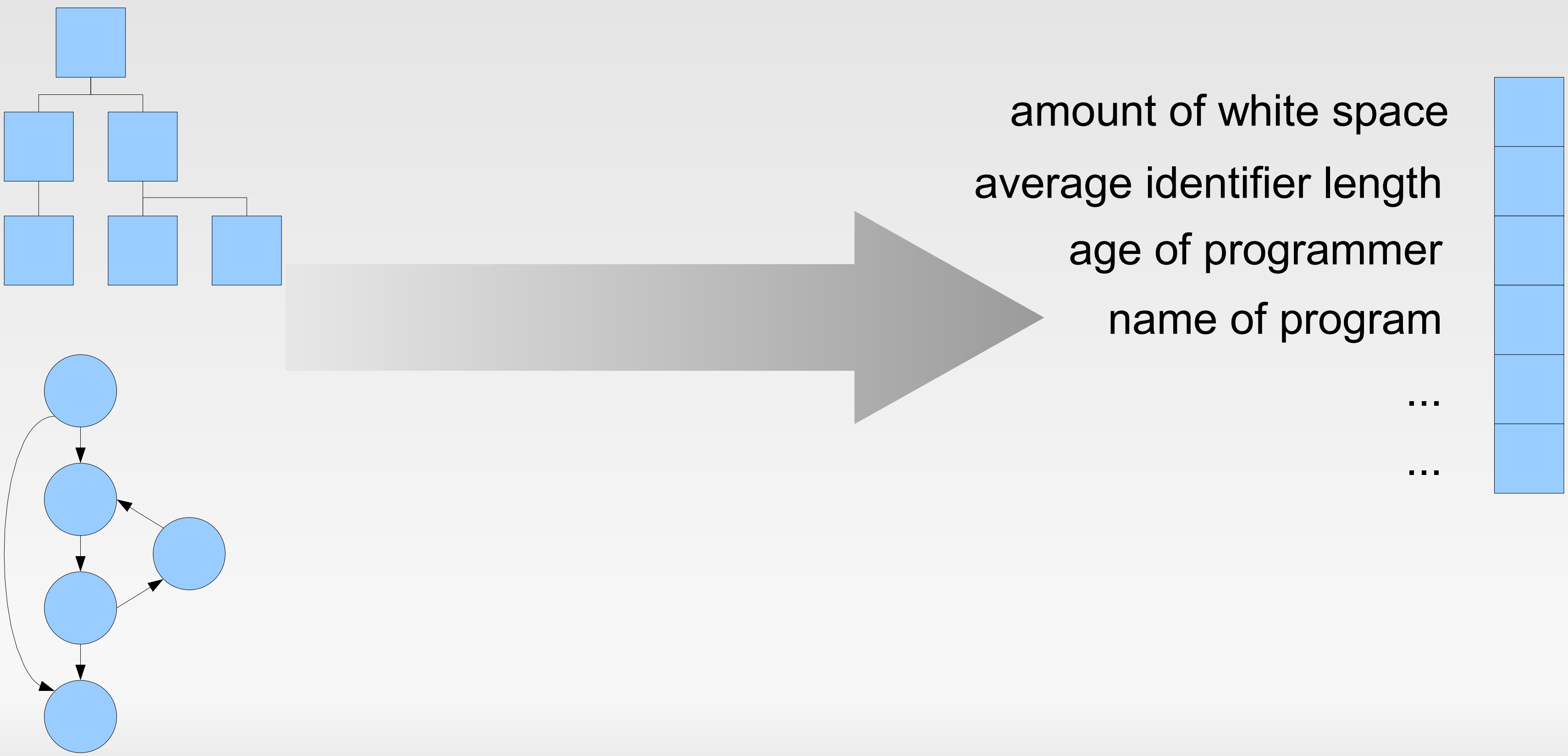


Overview

- Introduction to machine learning in compilers
- **Difficulties choosing features**
- A feature space for a motivating example
- Searching the feature space
- Features for GCC
- Results
- Further and on going work

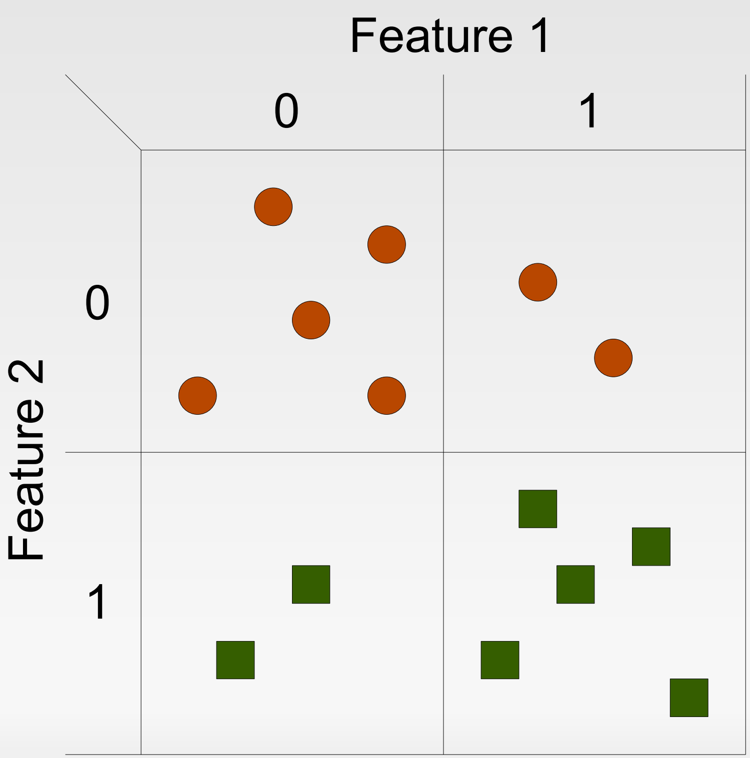
Difficulties choosing features

- The expert must do a good job of projecting down to features



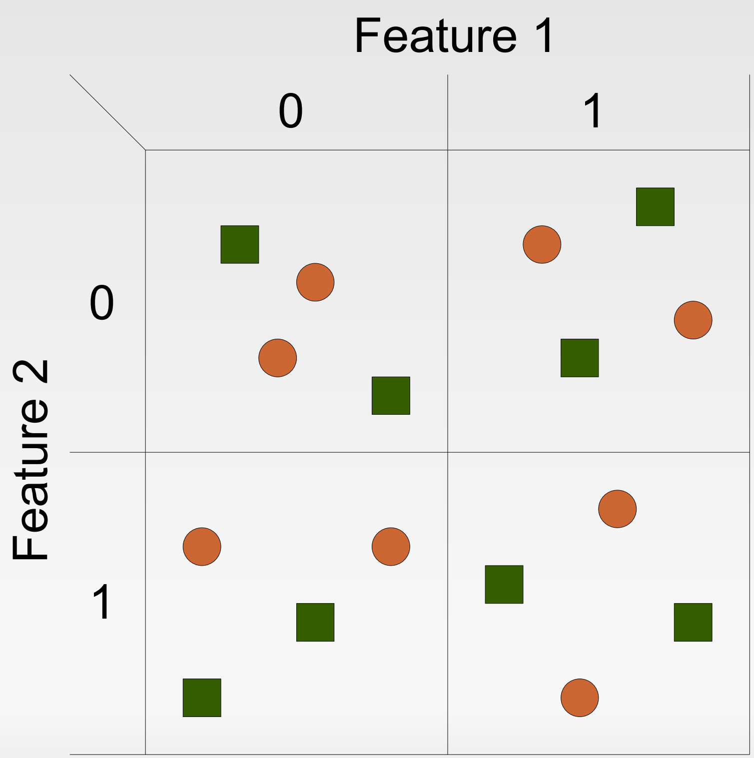
Difficulties choosing features

- Machine learning works well when all examples associated with one feature value have the same type



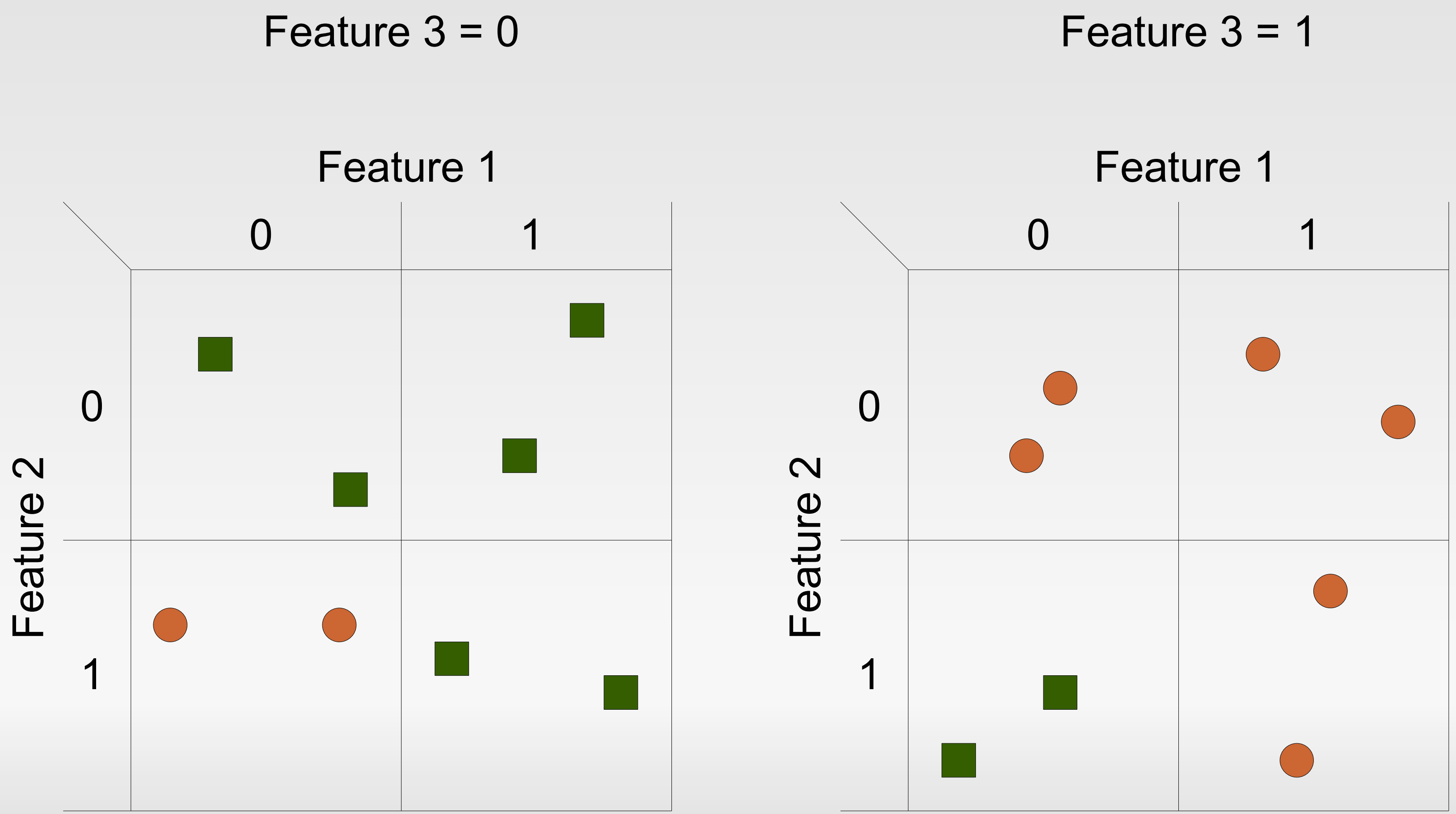
Difficulties choosing features

- Machine learning doesn't work if the features don't distinguish the examples



Difficulties choosing features

- Better features might allow classification



Difficulties choosing features

- There are much more subtle interactions between features and ML algorithm
  - Sometimes adding a feature makes things worse
  - A feature might be copies of existing features
- There is an infinite number of possible features

Overview

- Introduction to machine learning in compilers
- Difficulties choosing features
- **A feature space for a motivating example**
- Searching the feature space
- Features for GCC
- Results
- Further and on going work

A feature space for a motivating example

- Simple language the compiler accepts:
  - Variables, integers, '+', '*', parentheses

- Examples:
  - a = 10
  - b = 20
  - c = a * b + 12
  - d = a * ((b + c * c) * (2 + 3))

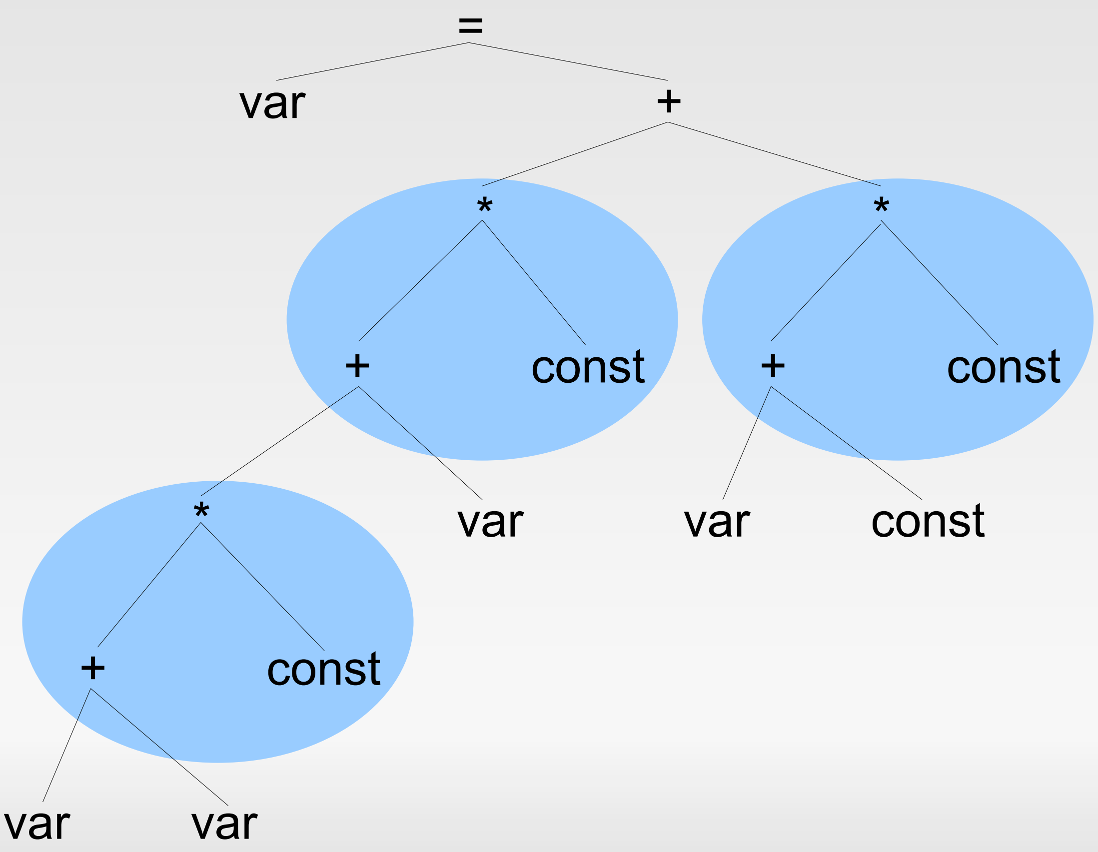
A feature space for a motivating example

- What type of features might we want?

```
count−nodes−matching(
    is−times &&
    left−child−matches(
        is−plus
    )&&
    right−child−matches(
        is−constant
    )
)
```

Value = 3

a = ((b+c)*2 + d) * 9 + (b+2)*4



A feature space for a motivating example

- Define a simple feature language:

```
<feature>    ::= ”count−nodes−matching(” <matches> ”)”
<matches>  ::= ”is−constant”
                  |   ”is−variable”
                  |   ”is−any−type”
                  |   ( ”is−plus” | ”is−times” )
                       ( ”&& left−child−matches(” <matches> ”)” ) ?
                       ( ”&& right−child−matches(” <matches> ”)” ) ?
```

A feature space for a motivating example

- Now generate sentences from the grammar to give features
- Start with the root non-terminal

Grammar
<A> ::= <A><A><A>
| "b"

Sentence
A

A feature space for a motivating example

- Now generate sentences from the grammar to give features
- Choose randomly among productions and replace

Grammar		Sentence
<A> ::=	<A><A><A>	AAA
|	"b"	

A feature space for a motivating example

- Now generate sentences from the grammar to give features
- Repeat for each non-terminal still in the sentence

Grammar		Sentence
<A> ::=	<A><A><A>	bAAAb
|	“b”	

A feature space for a motivating example

- Now generate sentences from the grammar to give features
- Continue until there are no more non-terminals

Grammar		Sentence
<A> ::=	<A><A><A>	bbbbb
|	"b"	

Overview

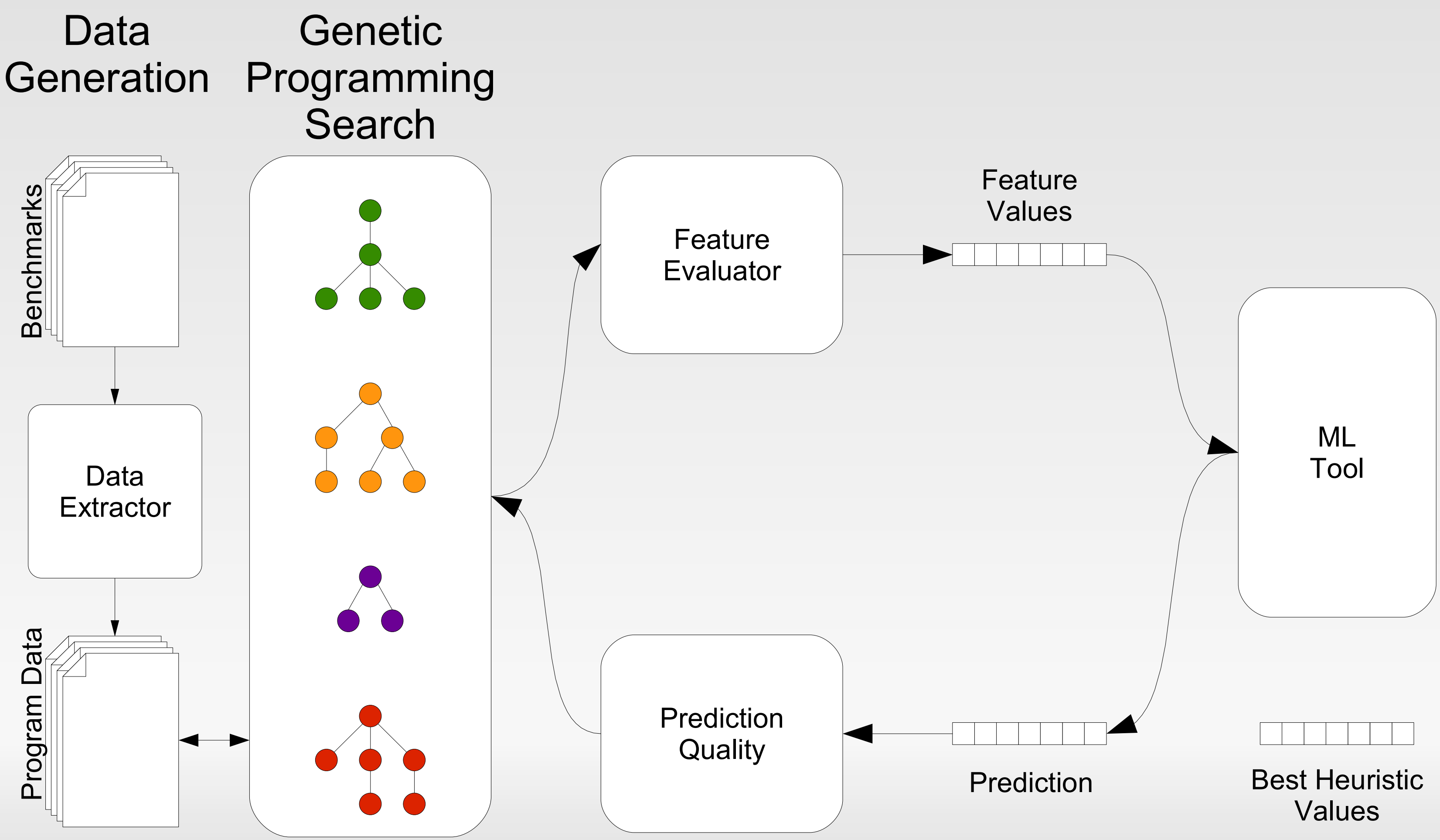
- Introduction to machine learning in compilers
- Difficulties choosing features
- A feature space for a motivating example
- **Searching the feature space**
- Features for GCC
- Results
- Further and on going work

Searching the Feature Space

- Search space is parse trees of features
- Genetic programming searches over feature parse trees
- Features which help machine learning are better

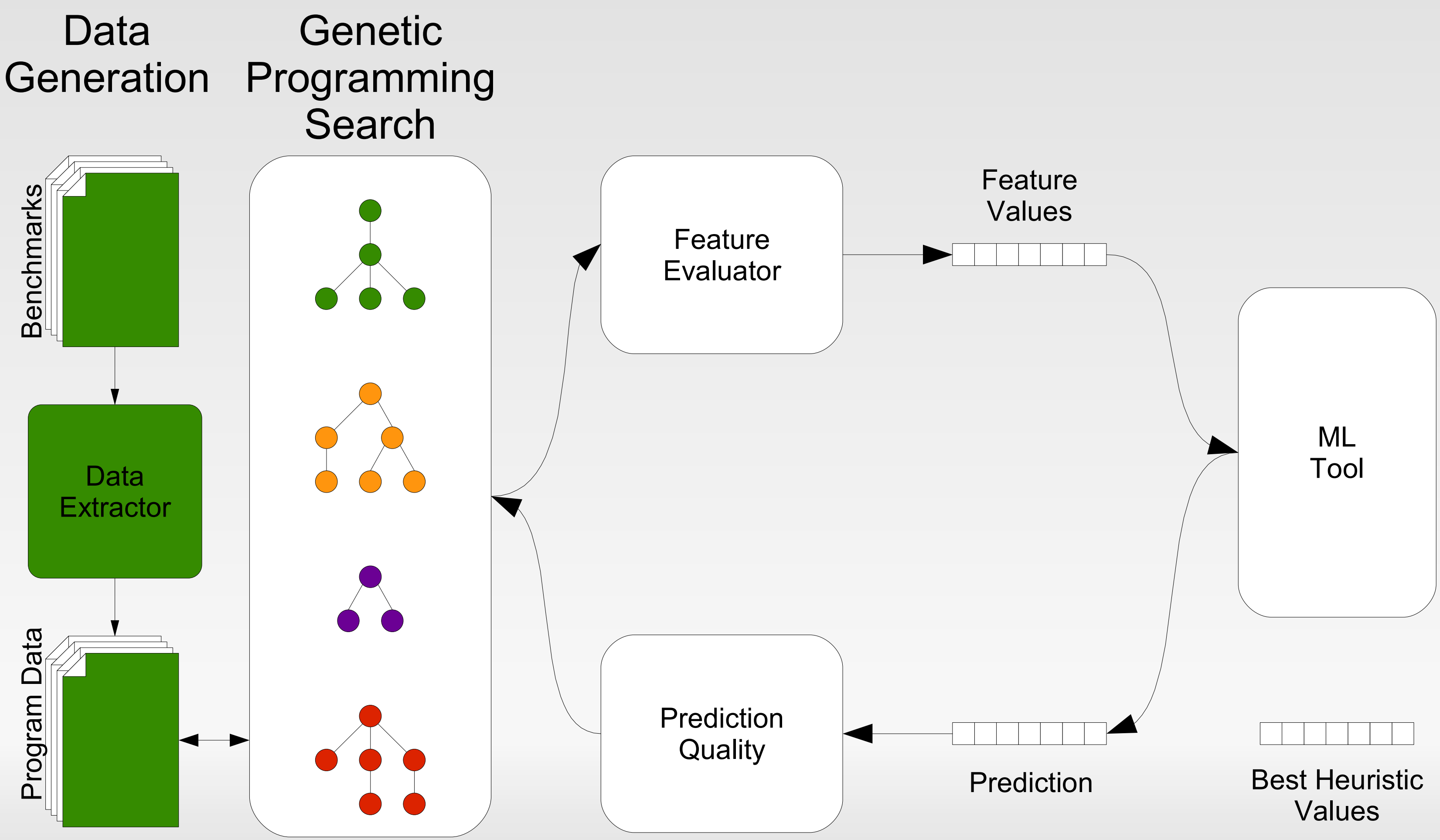
Searching the Feature Space

- Overview of searching



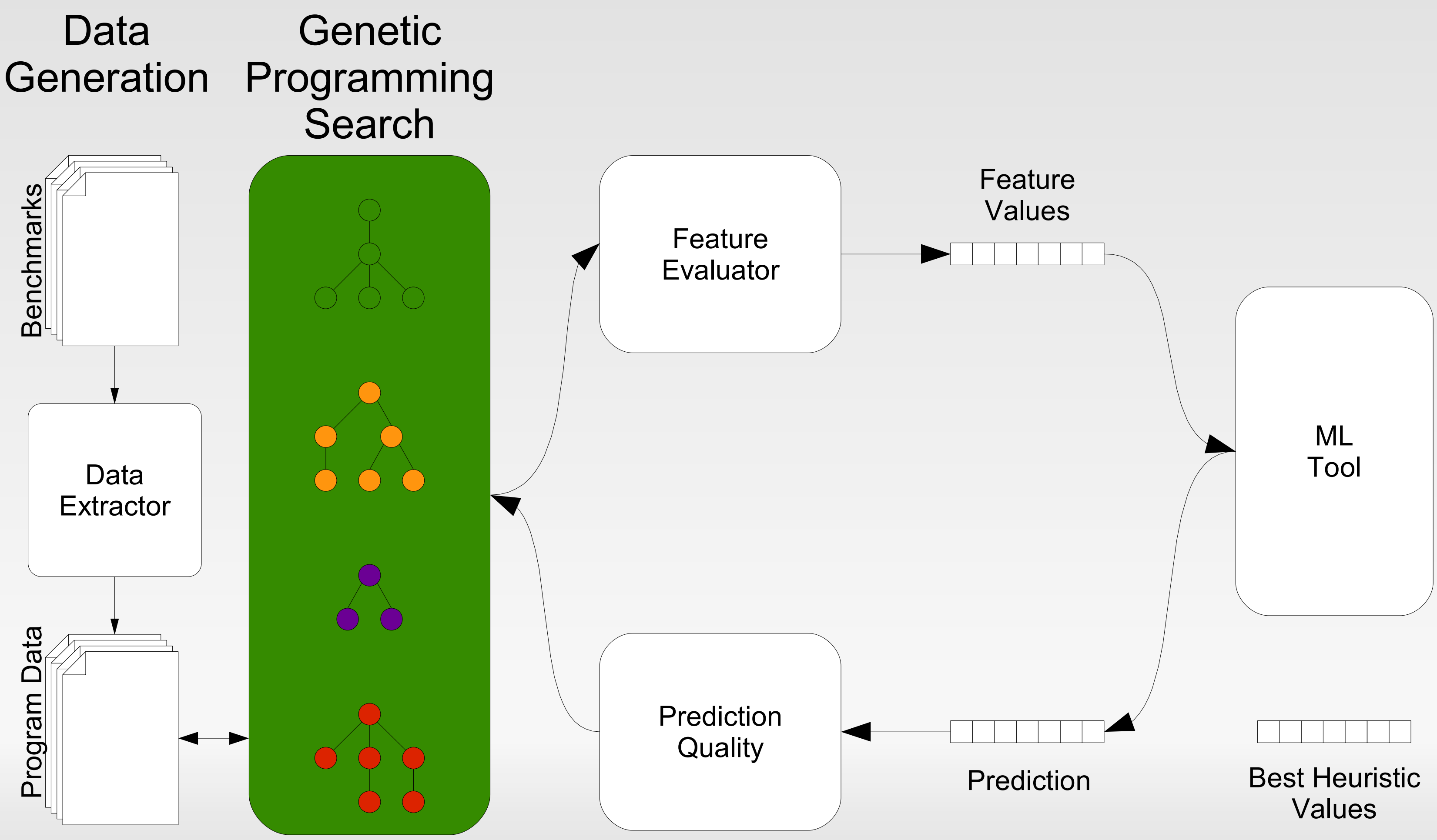
Searching the Feature Space

- Data structures extracted from benchmarks



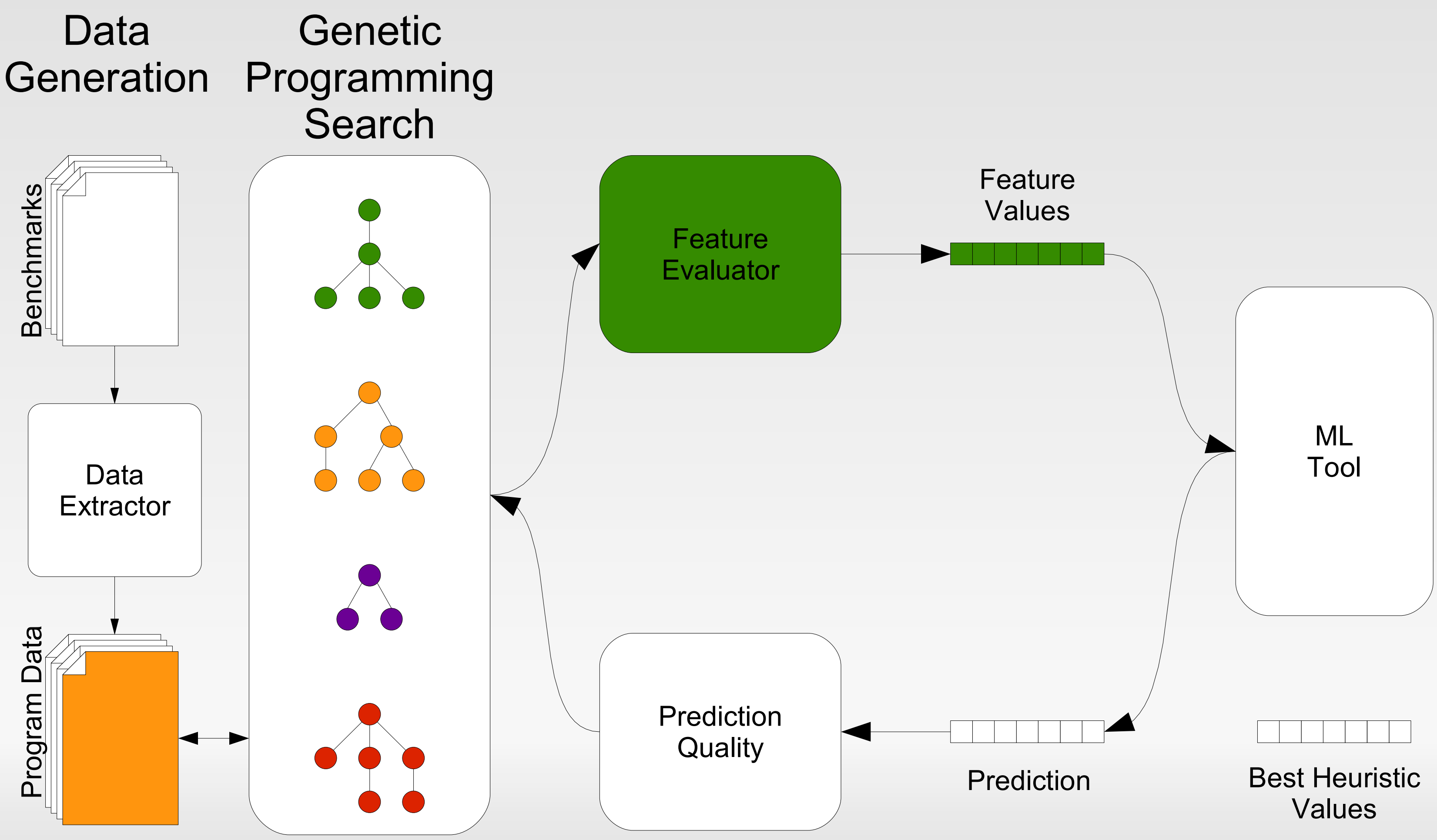
Searching the Feature Space

- Population of parse trees evolved with genetic programming



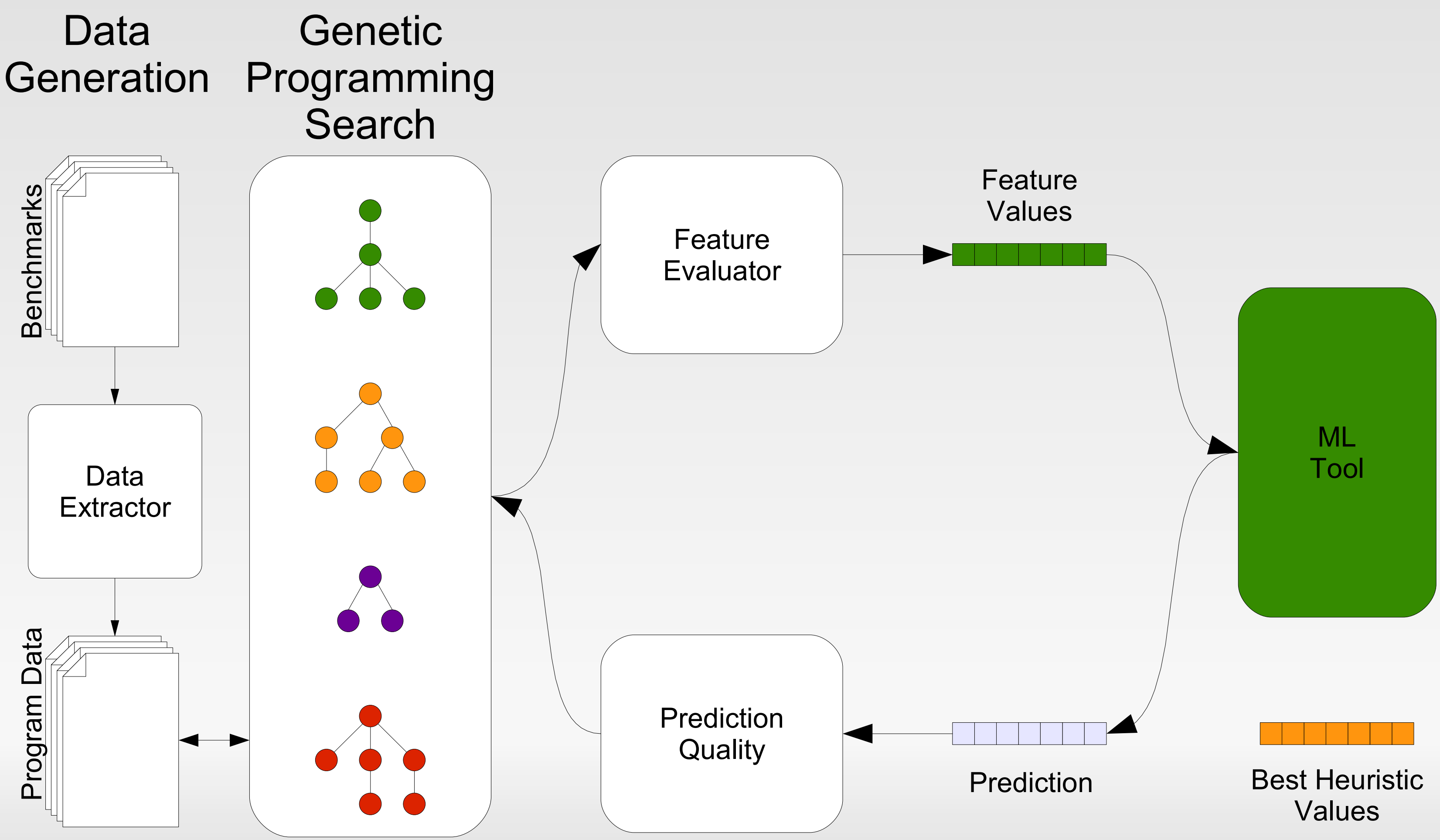
Searching the Feature Space

- Parse trees are interpreted over program data giving feature values



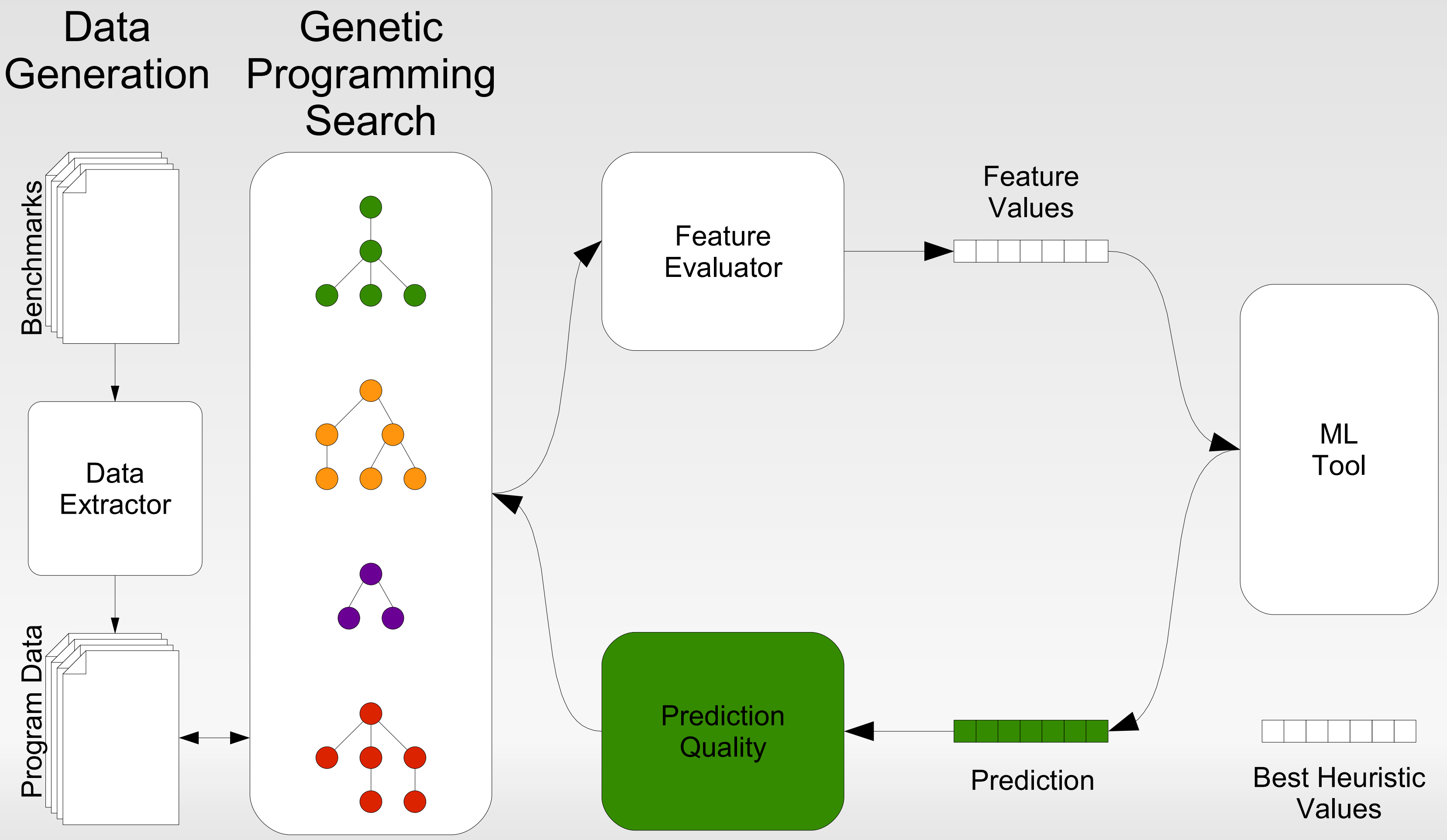
Searching the Feature Space

- ML tool learns model using feature values and target heuristic values



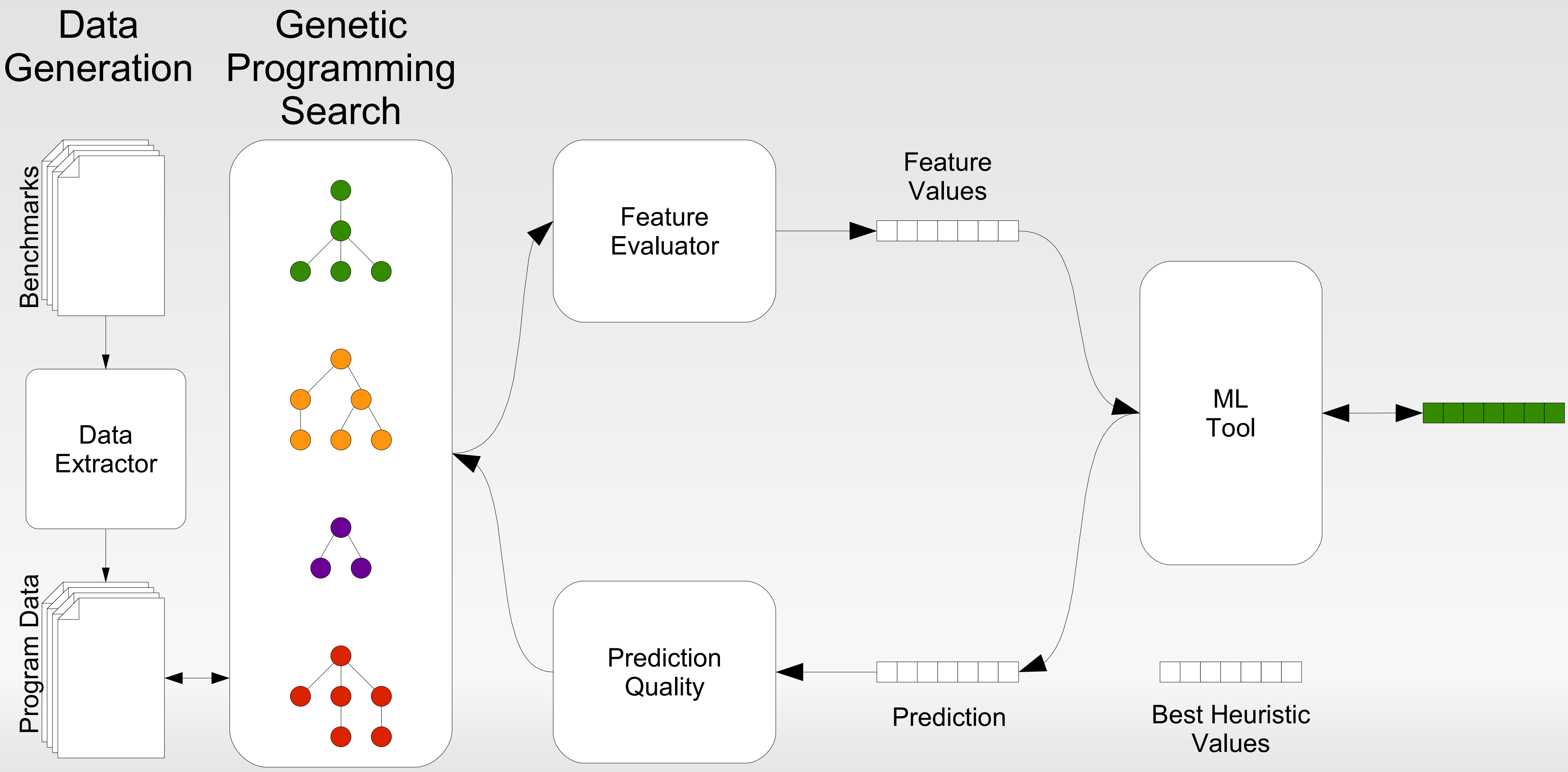
Searching the Feature Space

- Model predicts heuristic with cross validation
- Quality found (accuracy or speedup)



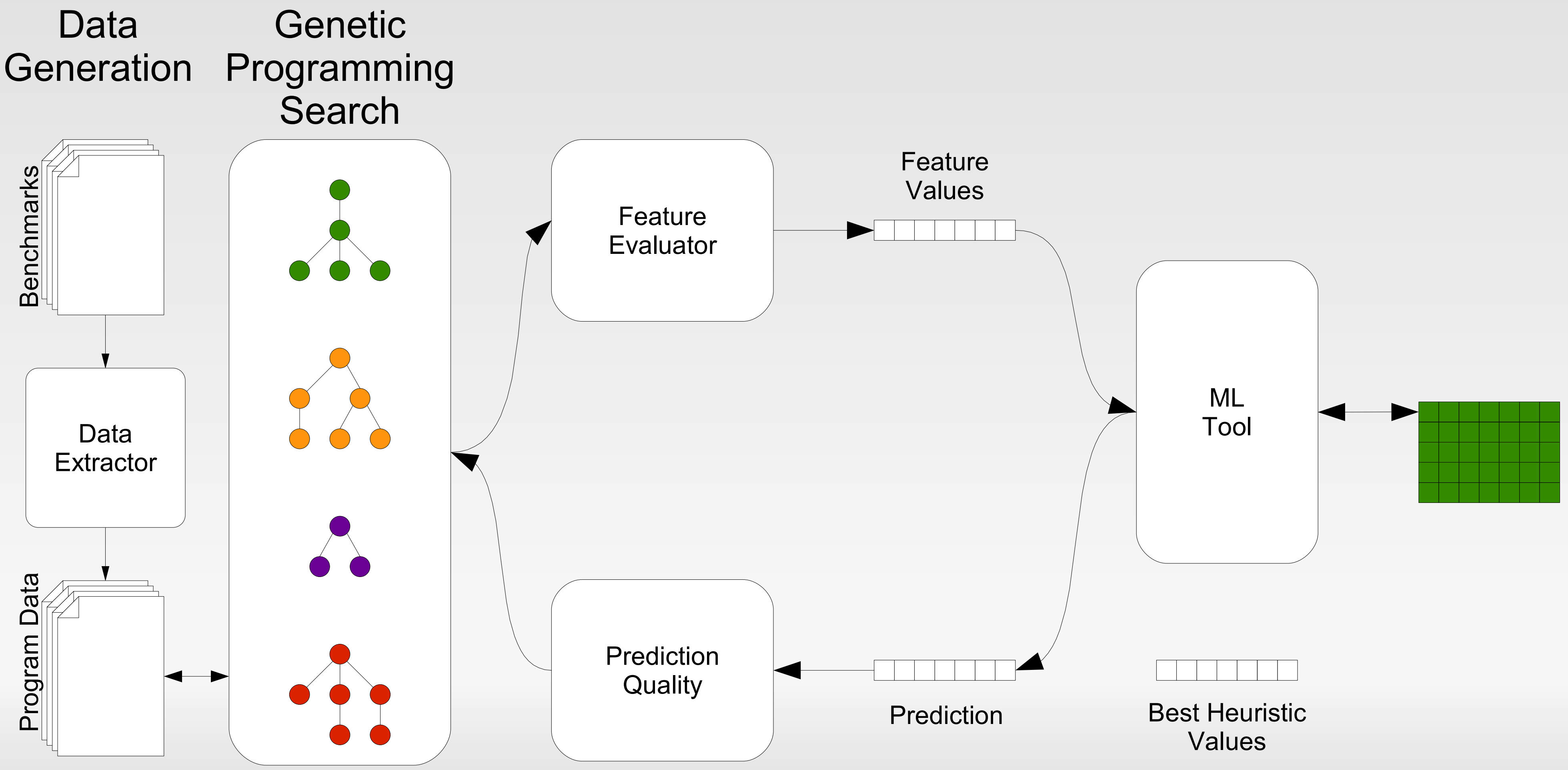
Searching the Feature Space

- After some generations first feature fixed
- Used when learning model for next feature



Searching the Feature Space

- Build up features, one at a time
- Stop when no improvement

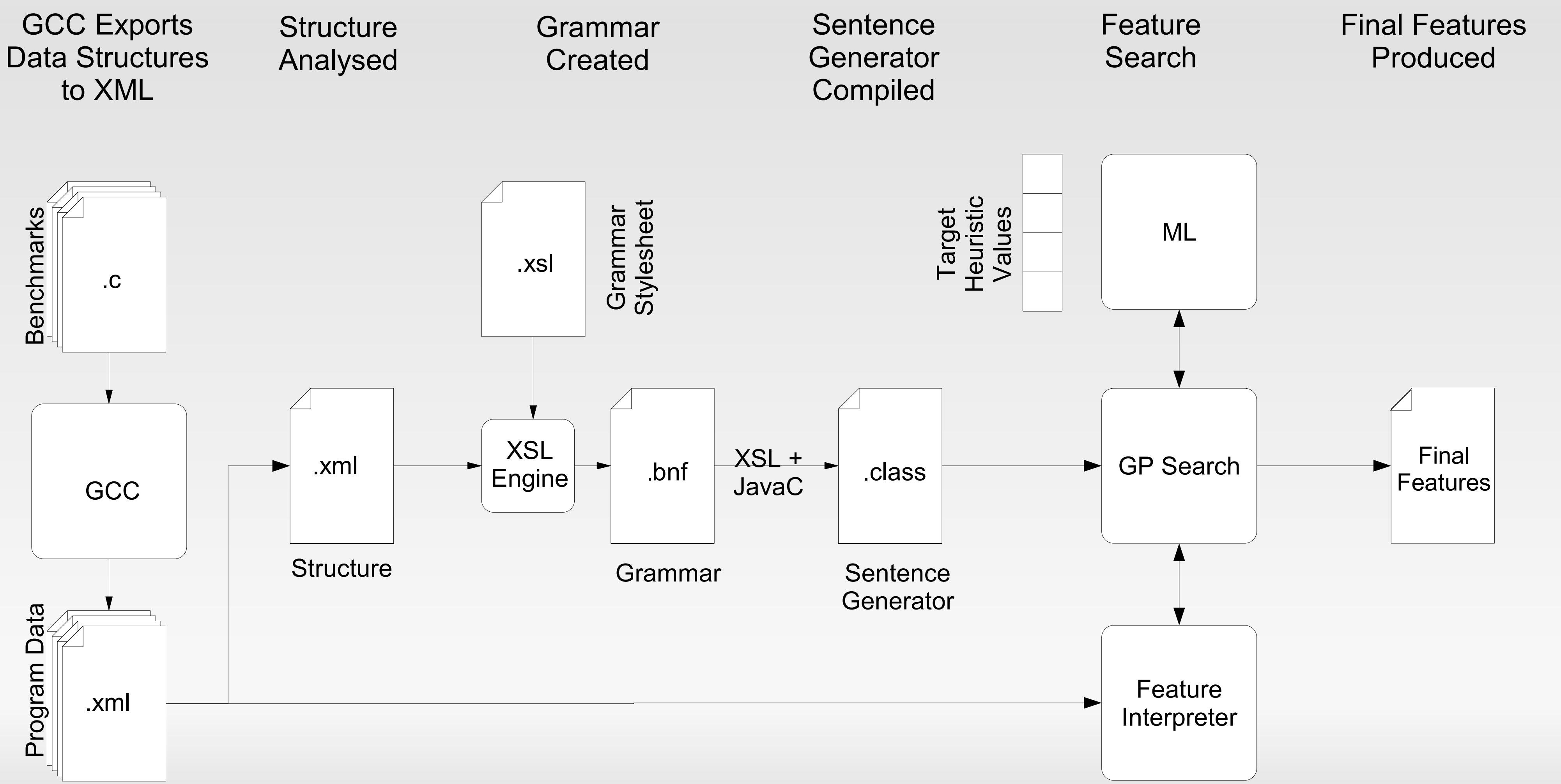


Overview

- Introduction to machine learning in compilers
- Difficulties choosing features
- A feature space for a motivating example
- Searching the feature space
- **Features for GCC**
- Results
- Further and on going work

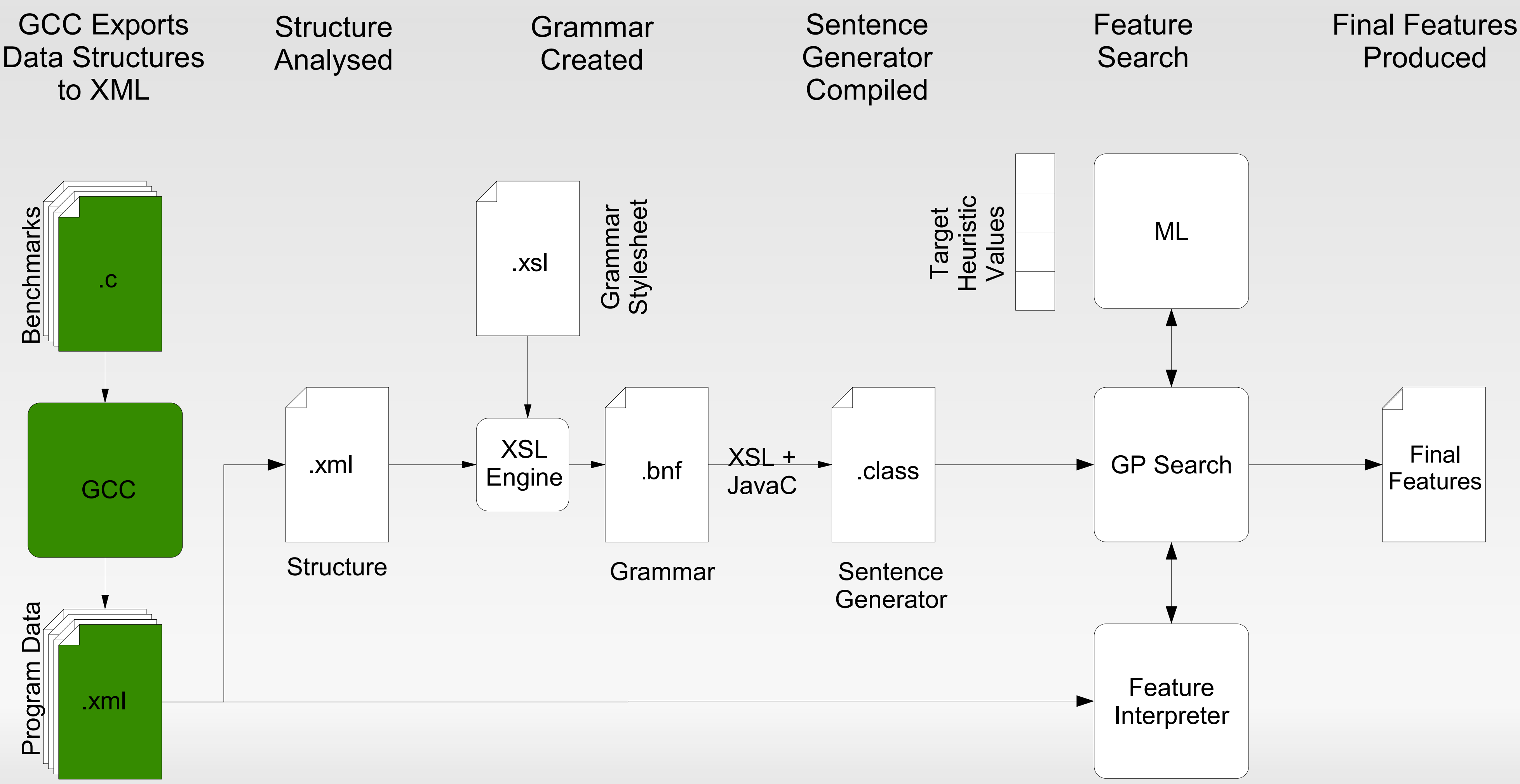
Features for GCC

- Start by dumping data structures to XML



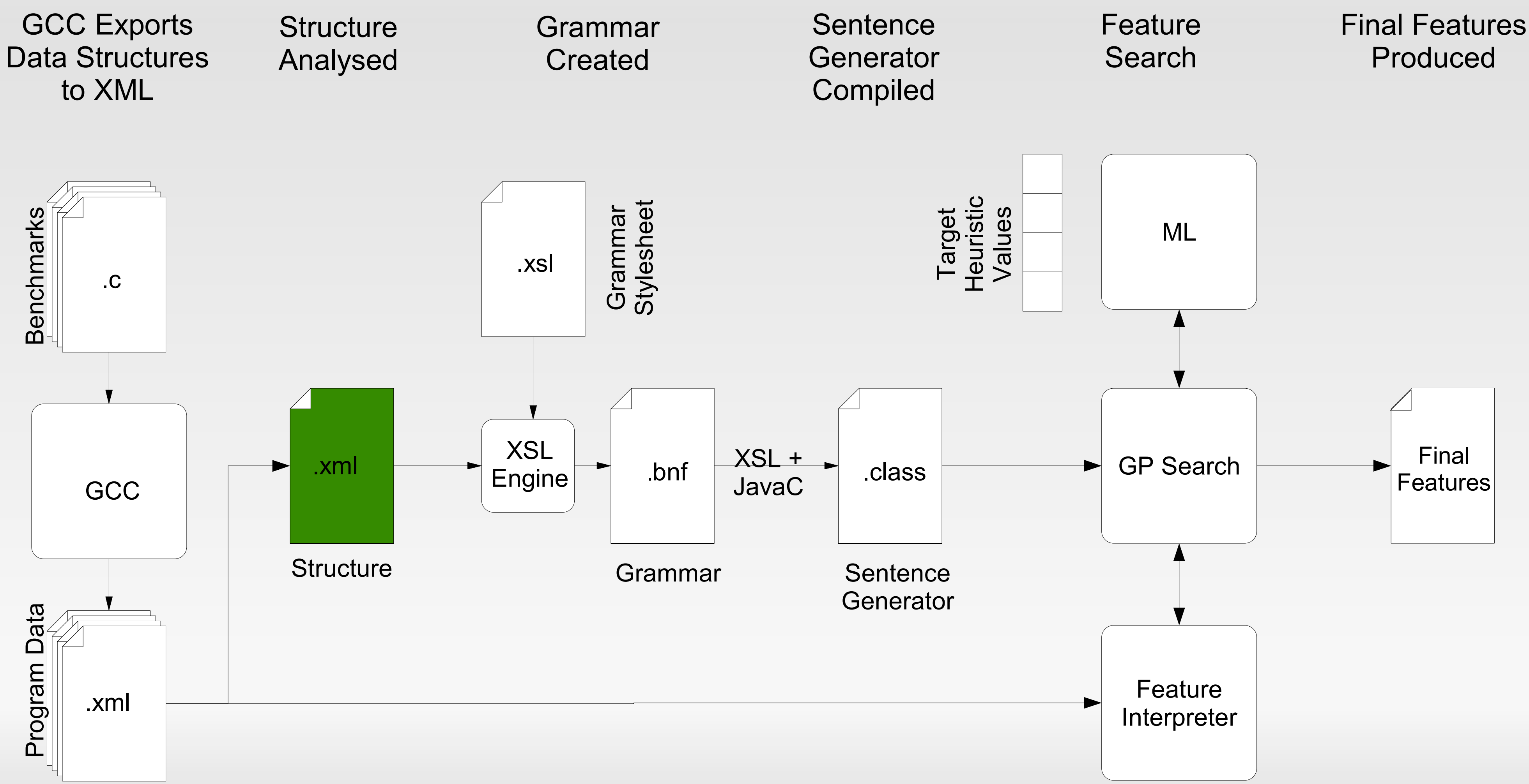
Features for GCC

- Start by dumping data structures to XML



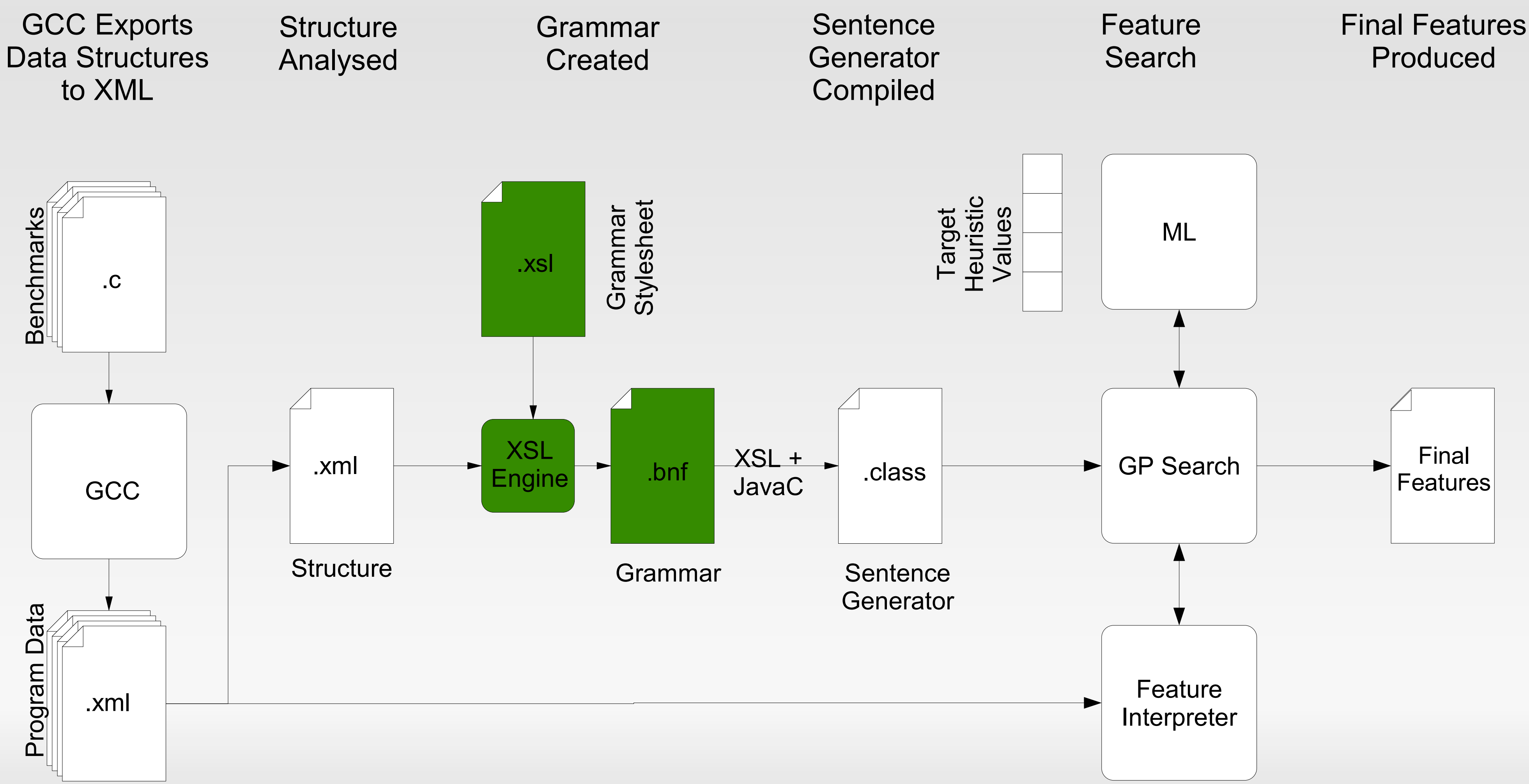
Features for GCC

- Find out the structure found in the benchmarks
- Allows system to know data format without hard coding



Features for GCC

- Grammar is constructed from structure

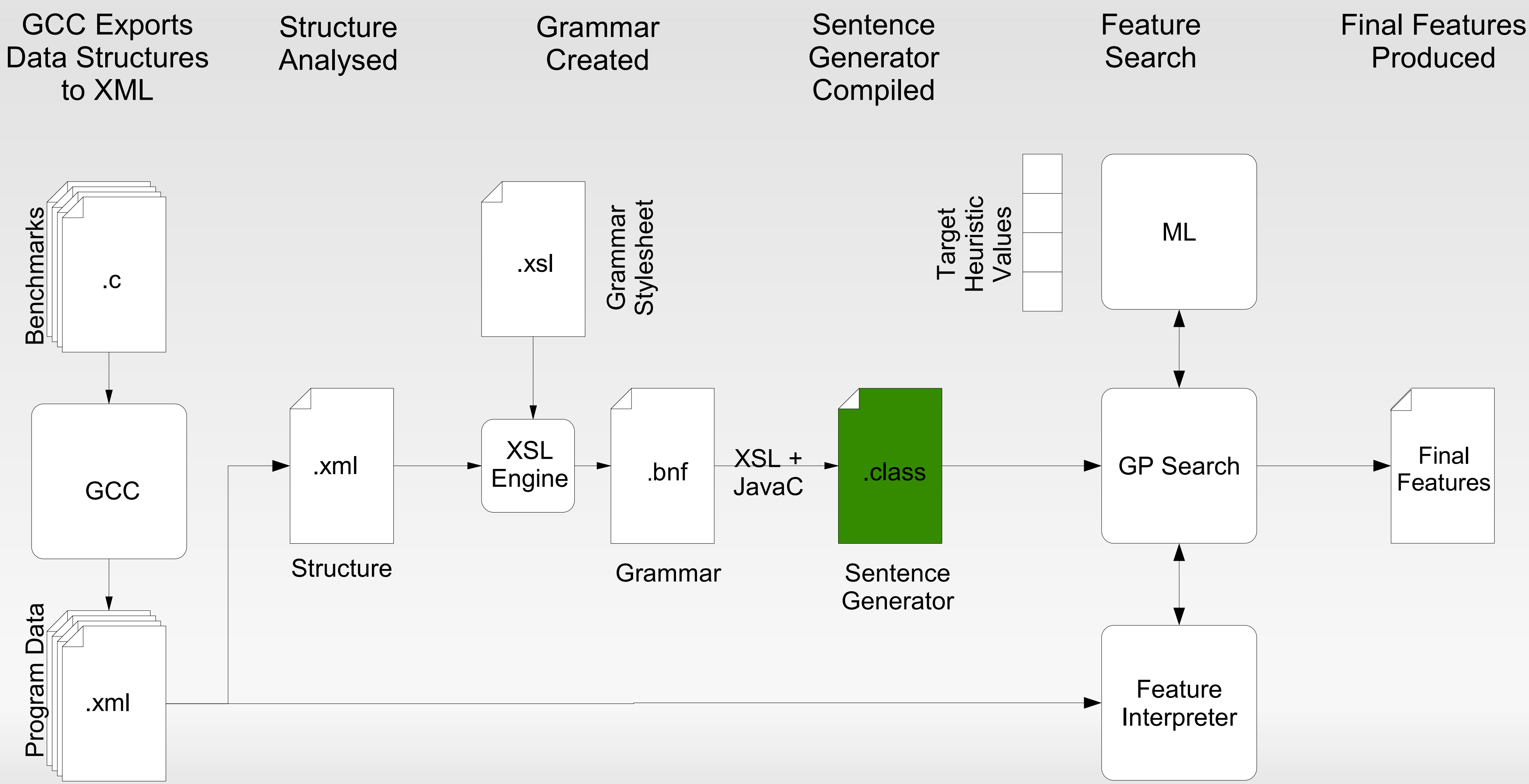


Features for GCC

- Grammar is constructed from structure
  - *Huge* grammar > 160kb
  - Ensures minimal useless features
  - Update easy if GCC changes
- Features are in interpreted language

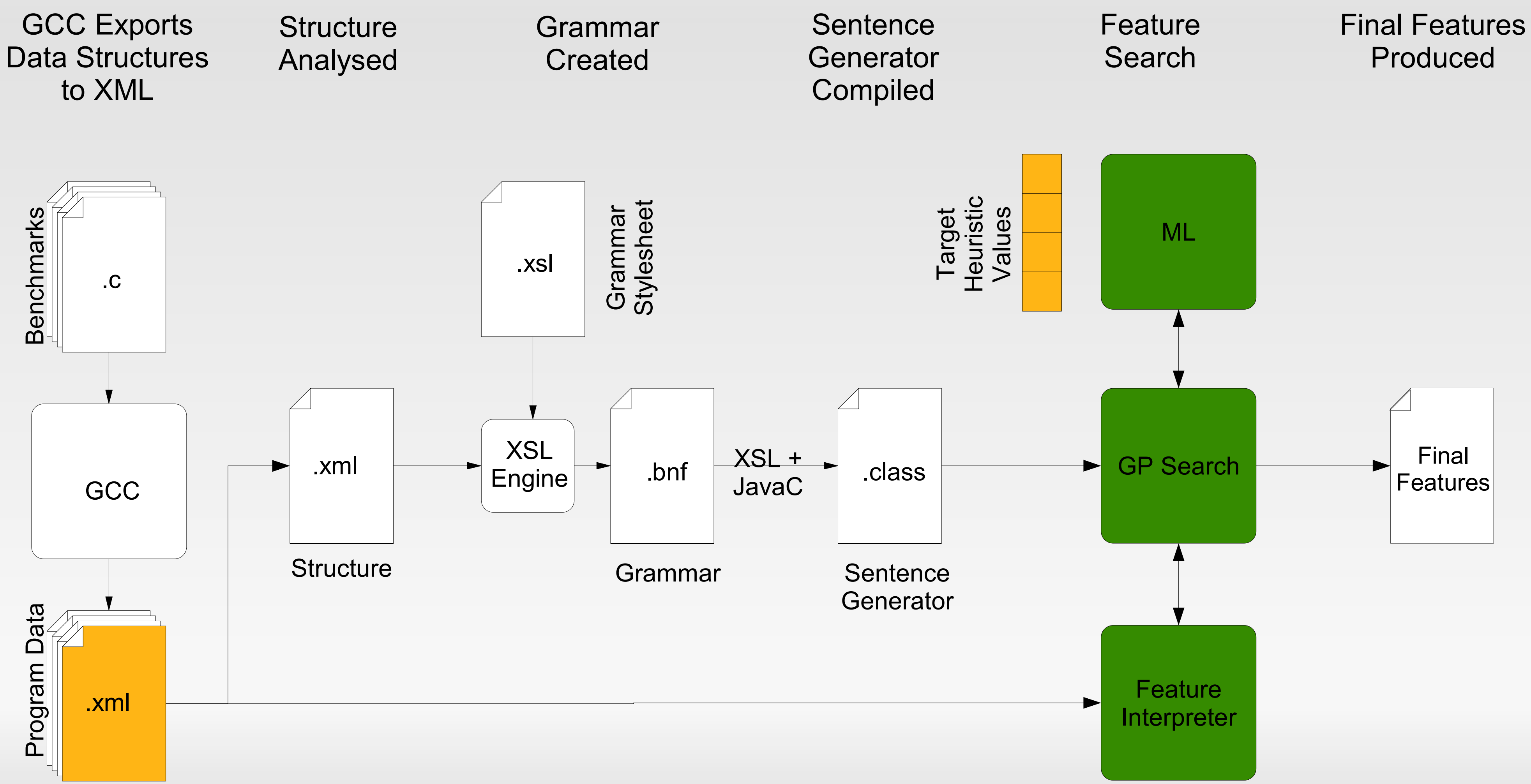
Features for GCC

- Grammar compiled down to Java



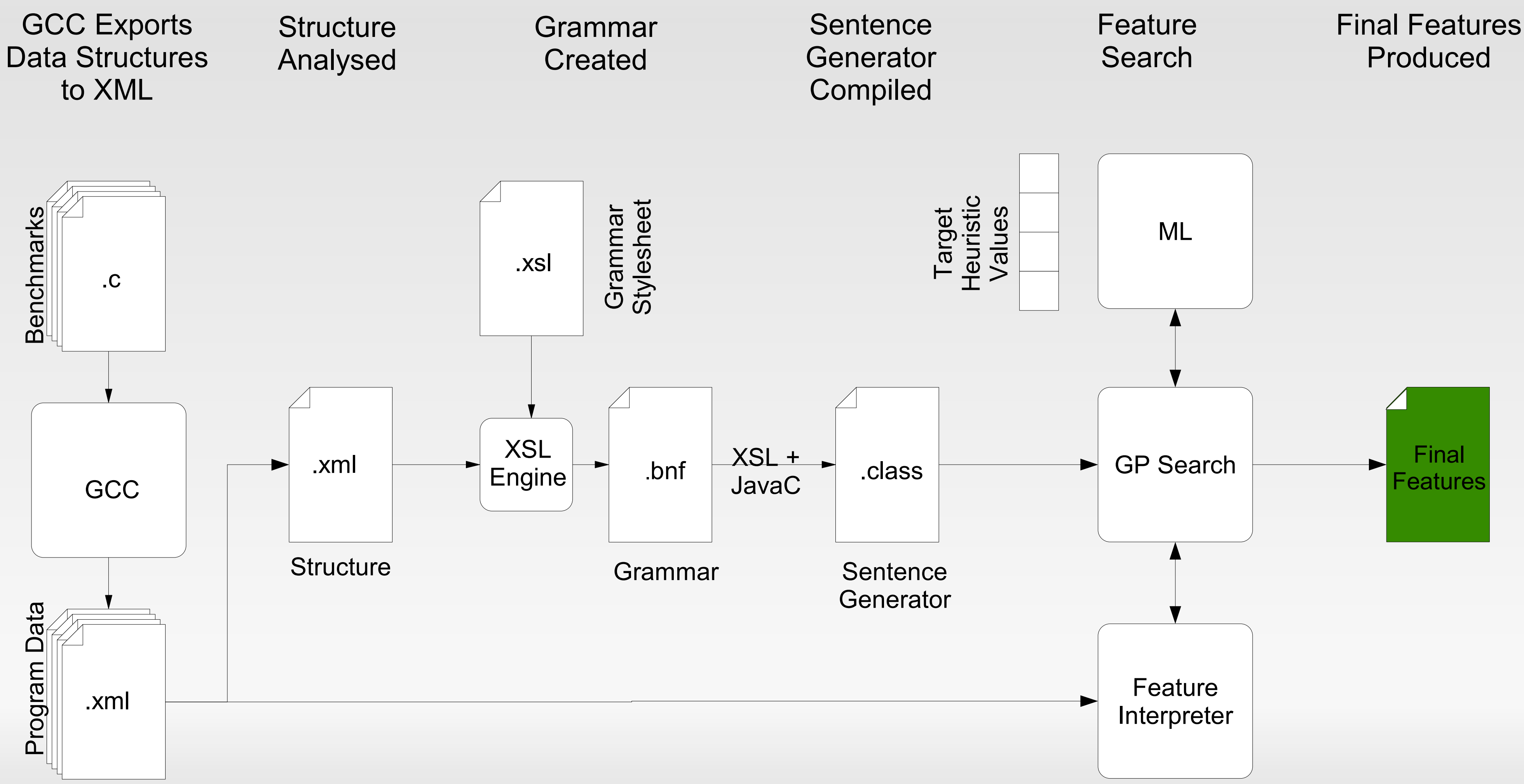
Features for GCC

- Feature search
- Features are evaluated over benchmark data



Features for GCC

- Final features outputted



Overview

- Introduction to machine learning in compilers
- Difficulties choosing features
- A feature space for a motivating example
- Searching the feature space
- Features for GCC
- **Results**
- Further and on going work

Results

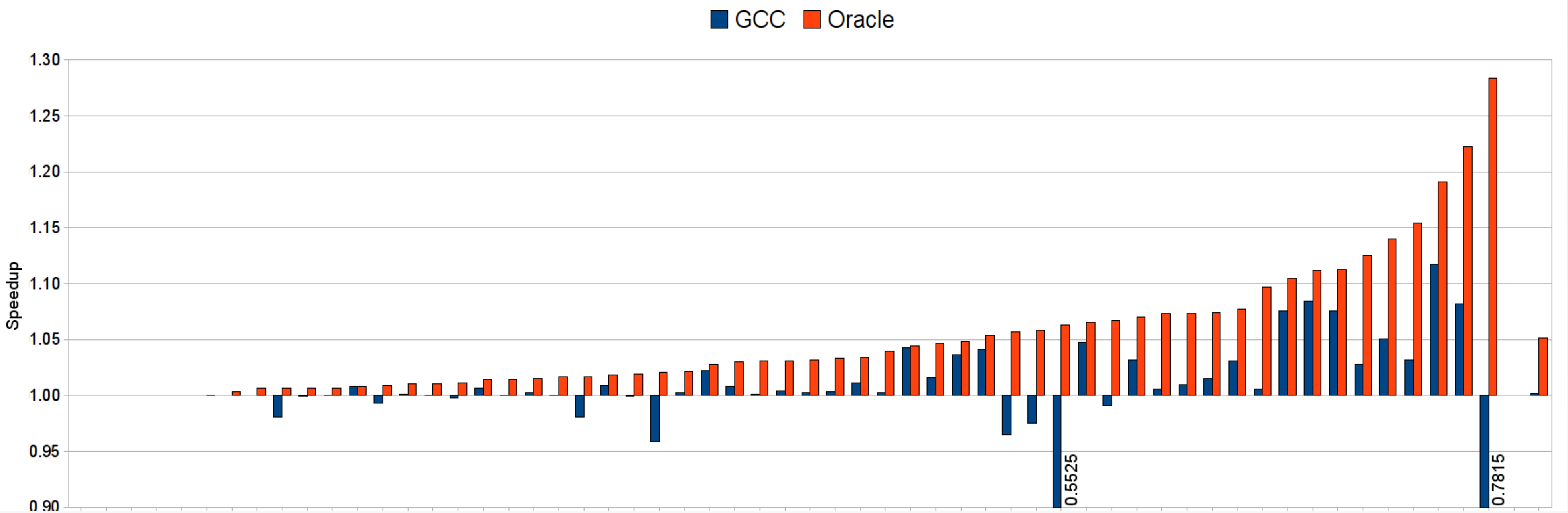
- Set up
  - Modified GCC 4.3.1
  - 57 benchmarks from MiBench, MediaBench and UTDSP
  - Pentium 6; 2.8GHz; 512Mb RAM
  - Benchmarks run in RamDisk to reduce IO variability
  - Found best unroll factor for each loop in [0-16]

Results

- Search set up
  - 100 parse trees per generation
  - Stop at 200 generations or 15 without change
  - Double Cross Validation
- Machine learning
  - Decision Trees (C4.5)

Results

- GCC default heuristic vs. oracle
- GCC gets 3% of maximum (1.05 speedup)
- On average mostly not worth unrolling

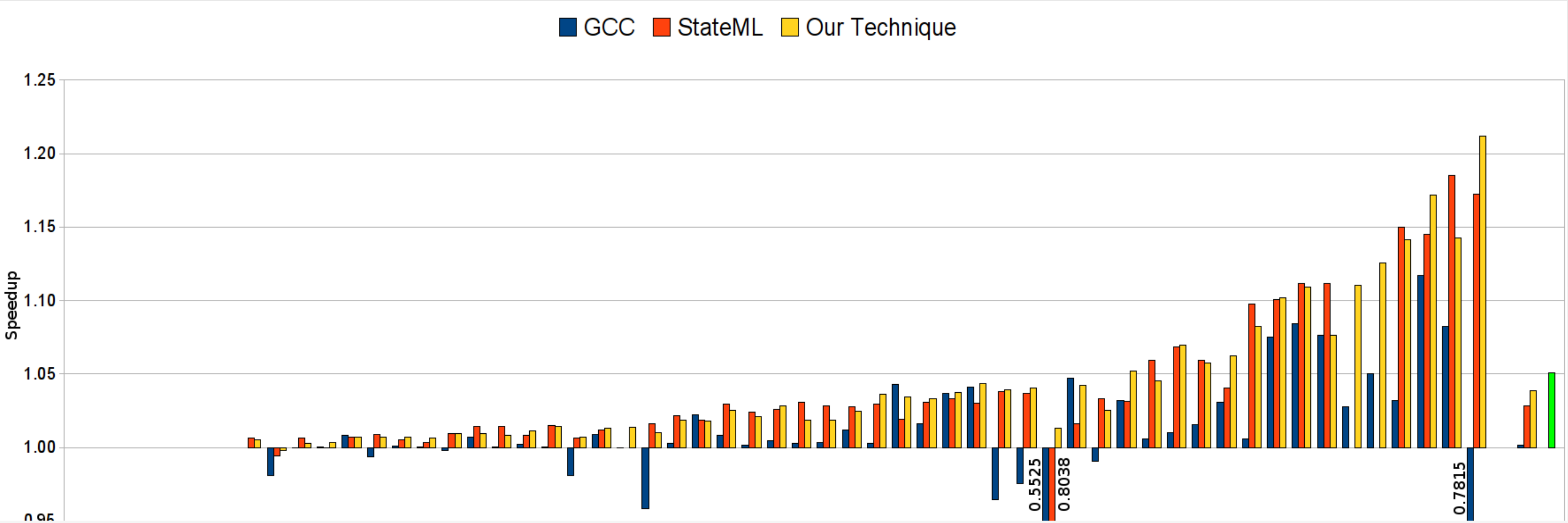


Results

- State of the art technique – Stephenson
- Hand designed features
- Uses support vector machine

Results

- GCC vs. state of the art vs. ours
- GCC 3%   Stephenson 59%   Ours 75%
- Automated features outperform human ones

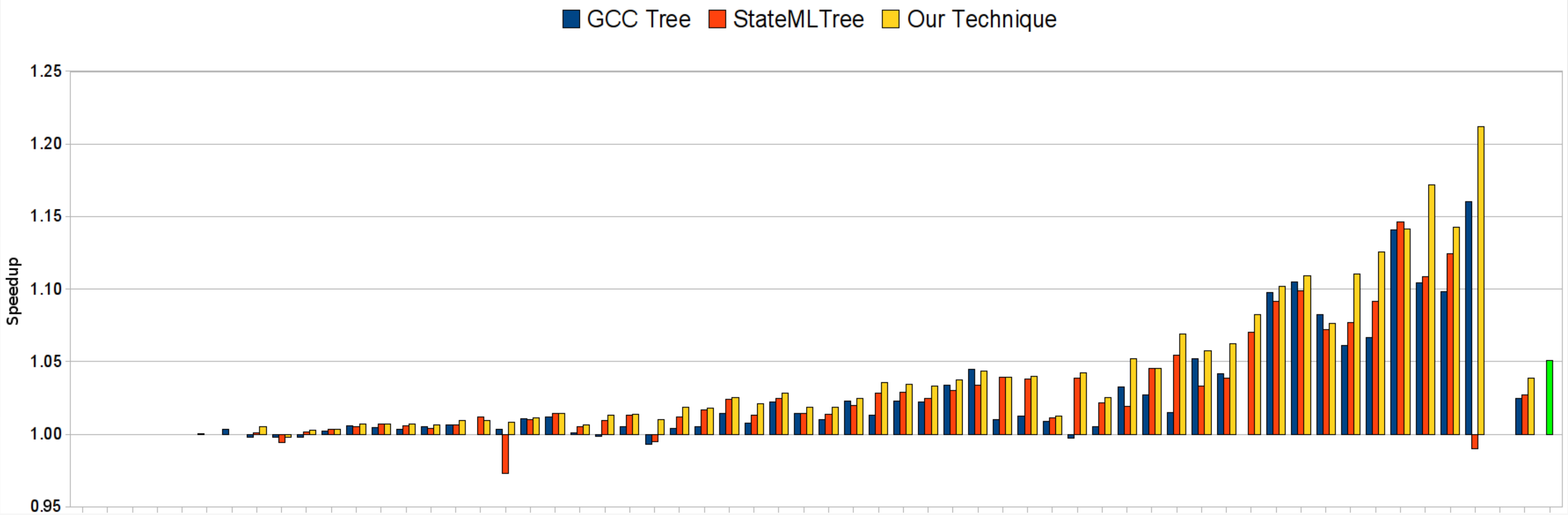


Results

- Compare all with same machine learning
  - All with C4.5 decision trees
  - Level playing field
  - GCC 'features' are data used to compute heuristic

Results

- Decision Trees
- GCC 48%   Stephenson 53%   Ours 75%
- Automated features outperform human ones



Results

- Top Features Found
  - get-attr(@num-iter)
  - count(filter(//*, !(is-type(wide-int) || (is-type(float extend) &&[(is-type(reg)]/ count(filter(//*,is-type(int))))) || is-type(union type))))
  - count(filter(/*, (is-type(basic-block) && (

```
  !@loop-depth==2 ||
  (0.0 > (
      (count(filter(//*, is-type(var decl))) -
      (count(filter(//*, (is-type(xor) && @mode==HI))) +
      sum(
          filter(/*, (is-type(call insn) && has-attr(@unchanging))),
          count(filter(//*, is-type(real type)))))) /
          count(filter(/*, is-type(code label)))))))))
```

Overview

- Introduction to machine learning in compilers
- Difficulties choosing features
- A feature space for a motivating example
- Searching the feature space
- Features for GCC
- Results
- **Further and on going work**

Further and on going work

- Make all GCC's internals available
- Integrate to Milepost GCC plug-in system and open source it
- Features for multi-core optimisation

MILEPOST

http://www.milepost.eu

Conclusion

- Shown a system which automatically finds good features
  - Searches a huge set of features
  - Allows greater experimentation in 'feature ideas'
- More flexible
  - A few feature grammars should service many heuristics
  - Retune features as well as heuristic
- Out performs expert features