

Implementing Constraint Imperative Languages with Higher-order Functions

Martin Grabmüller

`magr@cs.tu-berlin.de`

Technische Universität Berlin

Fakultät IV – Elektrotechnik und Informatik

Franklinstr. 28/29, 10587 Berlin, Germany

Outline

- Constraint imperative programming
- The constraint imperative language Turtle
- Implementation
- Summary

Constraint Imperative Programming

- Imperative programming
 - Statement oriented
 - State-changing program execution
 - Dependent on time
- Constraint programming
 - Expression oriented
 - Stateless execution
 - Time-independent
- Constraint imperative programming aims at combining both

Introduction to Turtle

- Imperative programming
 - Assignment, loops, procedures
- Functional programming
 - Higher-order functions
 - Algebraic data types
- Constraint programming
 - Constraining variables
 - Constraint statements
 - User-defined constraints
 - Constraint solvers

Turtle – An Example

```
1  constraint all_different (l: list of !int)
2    while (tl l <> null) do
3      var ll: list of !int := tl l;
4      while (ll <> null) do
5        require hd l <> hd ll;
6        ll := tl ll;
7      end;
8      l := tl l;
9    end;
10 end;
    ...
11 var a: !int := var 0, b: !int := var 1, c: !int := var 2;
12 require all_different ([a, b, c]) in ... end;
```

Implementation

- Compiler
- Run-time system
- Library Modules

Compiler

- Compiles to ANSI C
- Traditional imperative compiler
- Static type checking with overload resolution
- Parametrized data types and modules
- Algebraic data types with constructor/access function generation
- Constraint compilation
 - Constraint analysis
 - Code generation

Constraint Analysis

```
var y: int ← 4;  
var x: ! int ← var 0;  
require 10 * x + 10 > 3 * y - 1;
```

Constant expression:

$$-10 + 3*y - 1$$

Constrainable variable with coefficient:

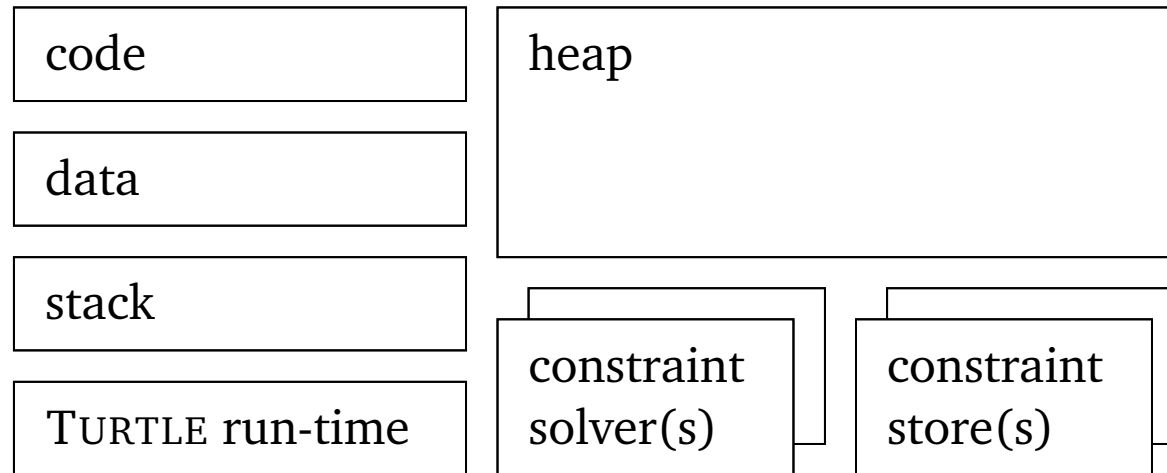
$$10*x$$

Code Generation

$$10*x > -10+3*y - 1$$

```
1  push-constant  0  // constraint strength '0'
2  push-constant  3  // constraint kind '>'
3  push-constant  1  // number of constrainable variables
4  push-variable  y  // calculate the constant term...
5  load-constant  3
6  mul
7  push
8  load-constant -11
9  add
10 push
11 push-variable  x  // load the constrainable variable object
12 push-constant  10
13 add-constraint // add the constraint to the store
```

Run-time system



- Constraint solvers
- Garbage collection
- Restricted foreign function interface

Library Modules

Collection of abstract data types, utility functions and interface modules to the operating system.

- Modules for built-in types for array, list sorting, string and number manipulation,
- Data types, like binary trees, hash tables, ...
- Imperative I/O.
- Low level library modules including interfacing with the OS, such as process management and network programming.

Summary

- Turtle is a multiparadigm language integrating constraint, functional and imperative programming.
- Techniques for implementing each of these paradigms can be combined.
- Problems/Sources for future considerations:
 - Slow/simple proof-of-concept implementation of the constraint solvers.
 - Limited use of constraints for controlling program execution.

Turtle: Layout Example Program

```
module layout;
import io;
fun main (args: list of string): int
  var lm, rm, gap, pw, col: !real := var 0.0;
  require lm = 2.0 and rm = 2.0 and pw = 21.0 and
    gap >= 0.5 and gap <= 2.0 and
    gap = 0.5 : medium and col <= 7.0 : strong and
    gap + lm + 2.0 * col + rm = pw in
    io.put ("lm="); io.put (!lm); io.nl ();
    ...
    io.put ("col="); io.put (!col); io.nl ();
  end;
  return 0;
end;
```