Casanova 2.0

Doing nothing with style

Dr. Giuseppe Maggiore   Dino Dini

NHTV University of Applied Sciences
Breda, Netherlands

Agenda

1. A short history of computing
2. The non-halting problem
3. The Casanova language for game development

A short history of computing

Original goals

- Questions about automation of mathematical tasks
- Automated procedures to compute algorithm results
- Automated procedures to prove theorems

A short history of computing

First tools

- A series of theoretical (and later physical) machines
- Turing machine [10]
- λ-calculus (Alonzo Church) [1]
- μ-recursive functions [7]
- ...

Turing machine

First tools

- A tape of cells; each cell has a symbol
- A head that can read and write symbols on the tape and move the tape left and right by one
- A state register that stores the state of the Turing machine, one of finitely many
- A table of instructions that, given the state the machine is currently in and the symbol it is reading on the tape tells the machine to do the following in sequence:
  - Either erase or write a symbol
  - Move the head (`L`, `N`, `R`)
  - Change the state

λ-calculus

First tools

- Three grammatical elements:
  - `x`
  - `t s`
  - λ `x.t`

λ-calculus

First tools

- Three grammatical elements:
  - `x`
  - `t s`
  - λ `x.t`
- β reduction:
  - (λ `x.t`) `s` reduces to `t[x` $\rightarrow$ `s]`

A short history of computing

Expressive power

- First question was *how powerful are these things?*
- Second question was *is any of those more powerful than the others?*

A short history of computing

Expressive power

- First question was *how powerful are these things?*
- Second question was *is any of those more powerful than the others?*
- All computational processes (recursion, the λ-calculus, and the Turing machine) are **equivalent** [4]

A short history of computing

Expressive power and the Turing-Church thesis

- Hypothesis about *effectively calculable* functions [4]
- Functions are *effectively calculable* when computable by a Turing machine
- Computability $\equiv$ those three equivalent processes

A short history of computing

Expressive power and the Turing-Church thesis

- Hypothesis about *effectively calculable* functions [4]
- Functions are *effectively calculable* when computable by a Turing machine
- Computability $\equiv$ those three equivalent processes
- Further research: real computer $\equiv$ Turing machine
  - Any type of subroutine
  - Recursive procedures
  - Any of the known parameter-passing mechanisms
- *Disregarding IO*[a]

[a] we will get back to this

A short history of computing

Core focus

Disregarding IO
Disregarding IO
Disregarding IO
Disregarding IO
Disregarding IO
Disregarding IO
Disregarding IO
Disregarding IO
Disregarding IO
Disregarding IO
...

A short history of computing

Core focus

- Goal of programs was to:
  - Compute results
  - Go straight from *question* (input) to *answer* (output)
- This led to lots of effort towards ensuring *termination of programs*
  - A program that loops forever is clearly *stuck*
  - Unambiguously considered not useful: will never produce an answer

A short history of computing

Termination and IO

- Termination is, yet today, a fundamental property of computation
- *will my X terminate?*:
  - SQL query
  - Web-page rendering
  - Path-finding algorithm
  - Image processing
  - ...

A short history of computing

Halting problem

"Given a description of an arbitrary computer program, decide whether the program finishes running or continues to run forever". [9]

A short history of computing

Halting problem

- Turing proved in 1936 that a general algorithm to solve the halting problem for all possible program-input pairs cannot exist
- The halting problem is undecidable over Turing machines

A short history of computing

Halting problem

- Static analysis techniques to approximately determine termination
  - Type systems [2]
  - Model checking [6]
  - Abstract interpretation [3]
  - Complexity analysis (Big-O notation)
  - ...

A short history of computing

Halting problem

- There even exist languages that only produce terminating programs
  - SQL
  - Shaders
  - ...
- They are not Turing-equivalent
- If we lose the ability to not-terminate, we lose lots of expressive power

A short history of computing

Simply-typed λ-calculus

- Terms have types: $\frac{t:a \rightarrow b,\ u:a}{t\ \ u:b}$
- Every application of β-reduction strictly reduces the type complexity of the term
- *Strongly-normalizing*: well-typed terms always reduce to a value
- Impossible to define looping terms or anything that requires complex recursion/iteration
    - Parsing (even on regular expressions)
    - The Ackermann function
    - ...

The non-halting problem

Always doing something?

- Most programming languages only deal with termination
- `Do-something` operations
- Always take a step towards the final goal
- Abuse the language to do nothing
  - `while C do x += 0;`

The non-halting problem

The non-halting problem

- Termination sometimes makes no sense
  - OS
  - Games
  - ...
- Anything interactive where *IO determines when we are done*

The non-halting problem

Interaction and non-termination

- Show information at a human pace (much slower than computer ability to generate it)
- Wait for human to be ready or to express desire to interact
- **Example**

The non-halting problem

Doing nothing with style :)

- Games and other non-terminating applications need the ability to **selectively do nothing**
- Some entities/parts of the world/sub-systems/sub-algorithms are just **waiting**
  - For time to pass
  - For input
  - For specific conditions in the game world
  - For *complex combinations* of the above

Casanova

Our goal

Design a programming language[a] where *time is a first-class primitive*. [8]

[a] centred around computer games

Casanova

Time, data, and computations

- Move from handling *data and computations* to handling *data, computations, and the flow of time*
- Implicit concept of the *uninterruptible flow of time* (the main loop)
- Automated updates of entities through first class transformation operations (`rule`)
- `wait`/`when` as a first-class concept

Simple ball dynamics

```
world Ball = {
  S : Sprite
  P : Vector2<m>
  V : Vector2<m/s>
  G : Vector2<m/s^2>
  rule P = yield P + dt * V
  rule V = yield V + dt * G
}
```

Syntax

```
<Program> ::= <worldDecl> {<entityDecl>}

<worldDecl>  ::= world id = <entityBody>
<entityDecl> ::= entity id = <entityBody>
<entityBody> ::= "{" {<fieldDecl>} {<ruleDecl>} "}"

<fieldDecl>  ::= id ":" <type>
<ruleDecl>   ::= rule id {"," id} "=" <expr>
<type>       ::= int | float | Vector2 | ...

<expr > ::= ...(* typical expressions : let , if ,
                  for , while , etc. *)
           | <queryExpr > | "wait" <expr> | "yield" <
              expr>
           | <arithExpr > | <boolExpr > | <literal>
```

Semantics

```
E = { Field_1 = f_1; ...; Field_n = f_n
      Rule_1 = r_1; ...; Rule_m = r_m }

tick(e:E, dt) =
 { E with Field_1=tick(f_1^m, dt); ...; Field_n=tick(f_n^m, dt)
      ;
          Rule_1=r'_1; ...; Rule_m=r'_m }

where
   f_1^m, ..., f_n^m, r'_m = step(f_1^(m-1), ..., f_n^(m-1), r_m)
   ⋮
   f_1^2, ..., f_n^2, r'_2 = step(f_1^1, ..., f_n^1, r_2)
   f_1^1, ..., f_n^1, r'_1 = step(f_1, ..., f_n, r_1)
```

Casanova

A "small" challenge

A light-switch.[a]

[a] Serious kudos to Dino Dini for the idea.

An empty slide for you guys to think :)

Simple light-switch

```
world LightSwitch = {
  S : Sprite
  P : bool
  rule P =
    when(IsKeyDown(Keys.Space))
    yield true
    yield false
    when(IsKeyUp(Keys.Space))
  rule S.Color =
    yield Color.White
    when P
    yield Color.Black
    when P
}
```

Three state light-switch

```
world LightSwitch = {
  S : Sprite
  P : bool
  rule P = ...
  rule S.Color =
    yield Color.Green
    when P
    yield Color.Orange
    when P
    yield Color.Red
    when P
}
```

Timed-on/off lightswitch

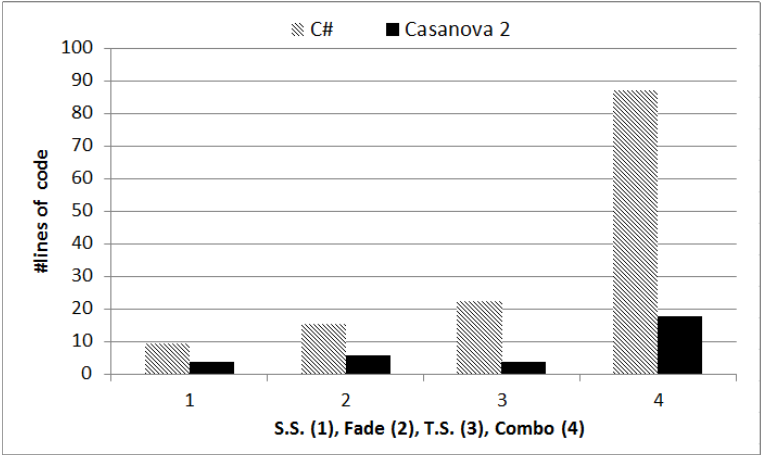
```
world LightSwitch = {
  S : Sprite
  rule S.Color =
    yield Color.Green
    wait 2.0f<s>
    yield Color.Orange
    wait 1.0f<s>
    yield Color.Red
    wait 3.0f<s>
}
```

Lightswitch in C#

```
public class FlatStateMachine : MonoBehaviour {
  bool state;
  bool previousInput;

  void Update () {
    bool currentInput = Input.GetMouseButton(0);
    if (!previousInput && currentInput) {
      state = !state;
      renderer.material.color = state ? new Color(0,
          0, 0) : new Color(1, 1, 1);
    }
    previousInput = currentInput;
  }
}
```

Comparison with C#



Casanova

Readability

- Programming languages are description tools
- Their main goal is to *tell another human what the machine should do* [5]
- Assessing the *readability of a language* is crucial

Casanova

Readability

- Test: students[a] read Casanova sources

Table: Feedback from students

Syntax is unfamiliar at first	3
Syntax is clear	8
Indentation instead of parentheses is a downside	2
List processing with queries is very effective	1
Rules are a good abstraction for games	2

[a] Games students, programming and design

Casanova

Compiler structure

- Parse
- Analyse and optimize queries
- Analyse and optimize `waits` with state machines
    - Event-system
    - Large `switch`-statement for each state machine
- Generate main loop
- Generate code
    - Using the C# syntax tree and Roslyn
    - Can also output C# instead of binaries

Casanova

Usage in practice

- All nice and all, but does it work in reality?

Casanova

Usage in practice

- All nice and all, but does it work in reality?
- Yup
- DEMO (RTS game)

Casanova

Future endeavours

Networking in games

Networking

<pre>local{
  rule X =
    send(b)
    if b then
      yield receive()
    else
      wait 1.0f<s>
      send(c)
}</pre>
	<pre>remote{
	  rule X =
	    let b = receive()
	    if b then
	      wait 1.0f<s>
	      send(c)
	    else
	      yield receive()
	}</pre>

Casanova

Conclusions

- Interaction negates termination, at least globally
- Modern programming languages suffer from this
- Need for better languages that handle time
- Concept of "do nothing" at the language level
  - High expressive power
  - Clean code
  - We argue results in a more pleasant "game development experience"

That's it

Thank you!

References I

Alonzo Church.
A Set of Postulates for the Foundation of Logic.
Annals of Mathematics, 2(33):346–366, 1932.

Alonzo Church.
A formulation of the simple theory of types.
Jurnal of Symbolic Logic, 5(2):56–68, June 1940.

Patrick Cousot and Radhia Cousot.
Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints.
In *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '77, pages 238–252, New York, NY, USA, 1977. ACM.

References II

M. Davis.
The Undecidable: Basic Papers on Undecidable Propositions, Unsolvable Problems and Computable Functions.
Dover Books on Mathematics Series. Dover Publications, 2004.

Edsger Dijkstra.
Programming considered as a human activity.
In *Classics in software engineering*, pages 1–9. Yourdon Press, 1979.

E. Allen Emerson and Edmund M. Clarke.
Characterizing correctness properties of parallel programs using fixpoints.
volume 85 of *Lecture Notes in Computer Science*, pages 169–181. Springer, 1980.

References III

S.C. Kleene.
Introduction to Metamathematics...
Bibliotheca mathematica. North-Holland Publishing Company.

Giuseppe Maggiore.
Casanova: a language for game development.
PhD thesis, Università Ca' Foscari di Venezia, Italy, April 2013.

Alan M. Turing.
On computable numbers, with an application to the entscheidungsproblem.
Proceedings of the London Mathematical Society, 42:230–265, 1936.

References IV

Alan M. Turing.
Computability and lambda-definability.
Journal of Symbolic Logic, 2(4):153–163, 1937.