mozart

Content

- Facts about Oz
- History of Oz
- Introduction to Oz
- Oz compared to Ocaml
- Examples

Oz Facts

- Oz supports most of the major programming paradigms
  - Including: Object-oriented, functional, logic, constraint, distributed and concurrent programming
- Oz was influenced by Erlang, Lisp and Prolog
- Oz open source, there is actually a github repository
  - Actually there quite few commits in 2015 while it has been actively updated in 2014
- Nowadays the only implementation of Oz is the Mozart Programming System

History

1991. Oz was initially started by the DFKI (Deutsches Forschungszentrum für Künstliche Intelligenz) as „Project Hydra“

1992. The language Oz has been designed and implemented in a very first version

1995. The DFKI Oz System was published, the first publicly available implementation of Oz

1996. Oz is continually developed in cooperation with the Swedish Institute of Computer Science

1998. The Oz Consortium was started, consisting of DFKI, SICS and the Université catholique de Louvain

History

- In 2000 and later, oz was further developed, the latest investigations concern distributed computing with Oz
- There are 3 versions of Oz: Oz1, Oz2 and Oz3
- The major change from Oz1 to Oz2 were Changes in the threading model they used (will be explained later)
- The major additional features of Oz3 compared to Oz2 are functors and futures (will be explained later)
- The DFKI Oz System has been redeemed by the Mozart system
- The current stable Version of Mozart implements Oz3

Introduction

- Why should one use Oz?
  - „Well, one rough short answer is that, compared to other existing languages, it is magic!” [1)]
  - It provides a wide range of programming abstractions
  - It combines multiple programming paradigms into one single language
  - If you need or want to use alot programming paradigms, Oz is the right language for you
  - “More is not better (or worse) than less, just different.” [2)]

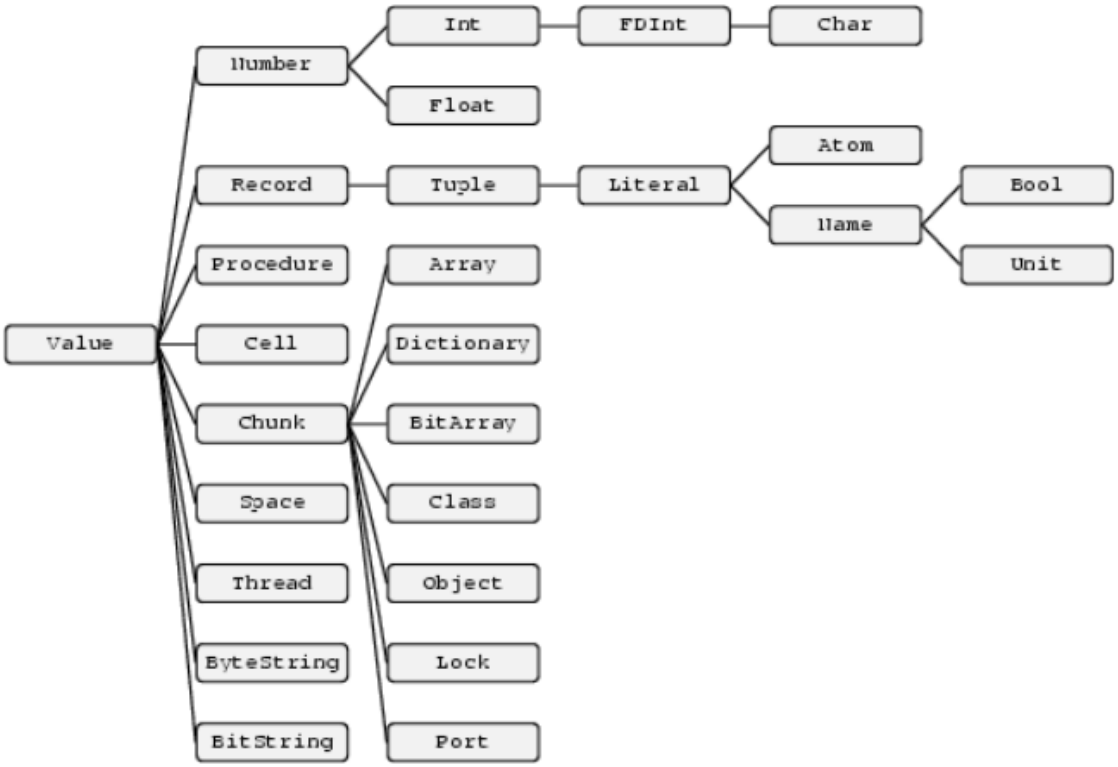
1) Source: Mozart Documentation Chapter 1.1
2) Van Roy, Peter. "Programming paradigms for dummies: What every programmer should know."

The Kernel Language

- Oz is based on the Oz Calculus, a mathematical model which is based on the π-calculus

```
<Statement> ::= <Statement1> <Statement2>
              | X = f(l1:Y1 ... ln:Yn)
              | X = <number>
              | X = <atom>
              | X = <boolean>
              | {NewName X}
              | X = Y
              | local X1 ... Xn in S1 end
              | proc {X Y1 ... Yn} S1 end
              | {X Y1 ... Yn}
              | {NewCell Y X}
              | Y=@X
              | X:=Y
              | {Exchange X Y Z}
              | if B then S1 else S2 end
              | thread S1 end
              | try S1 catch X then S2 end
              | raise X end
```

Types in Oz


Value
Number
Int
FDInt
Char
Float
Record
Tuple
Literal
Atom
Name
Bool
Unit
Procedure
Cell
Chunk
Array
Dictionary
BitArray
Class
Object
Lock
Port
Space
Thread
ByteString
BitString

Threading

- In Oz1 every statement could possibly be executed in parallel
  - {Statement1} {Statement2} {Statement3}
- The Oz1 model was hard to debug since composition does not always lead to a thread
- This has been reworked in Oz2 to ease the control of threading for programmers
  - thread Y = X*X end
- With the new threading model, each thead keyword actually spawns a thread

Threading example

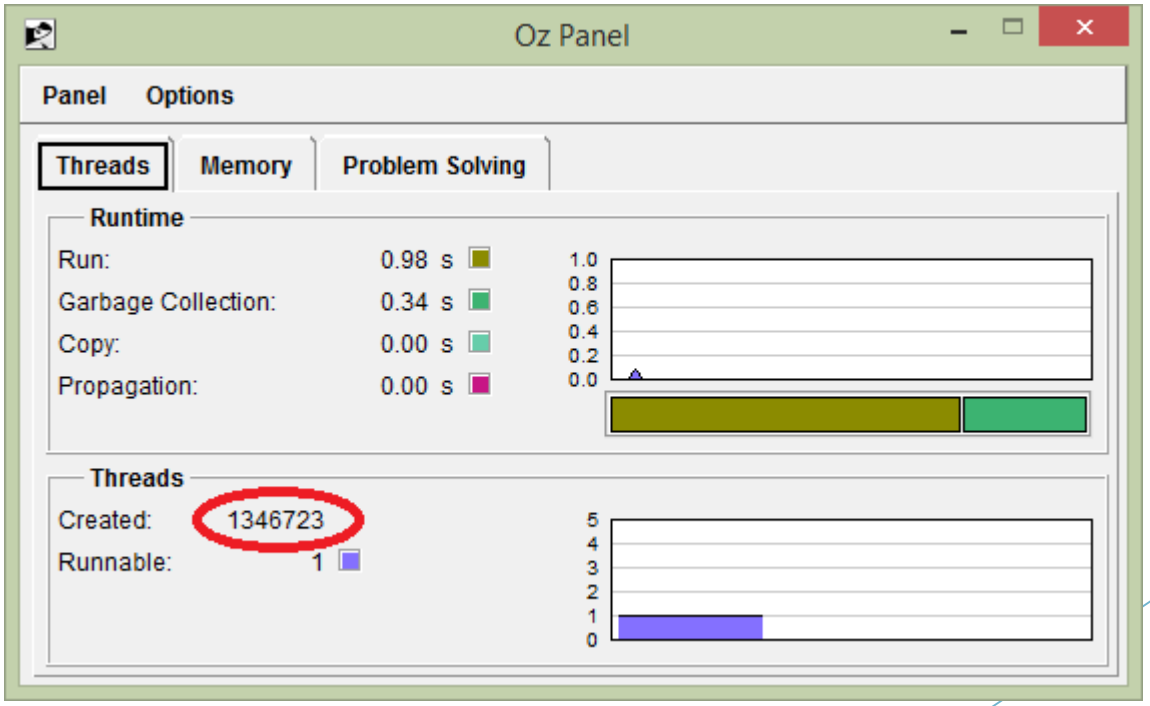
- Parallel map function

```
fun {Map Xs F}
      case Xs of nil then nil
      [] X|Xr then thread {F X} end | {Map Xr F}
      end
end
```

Thread spawning example

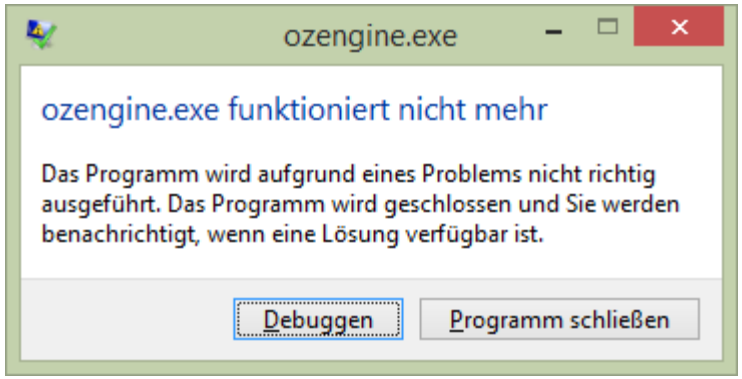
```
declare
fun {Fib X}
   case X
   of 0 then 1
   [] 1 then 1
   else thread {Fib X-1} end  + {Fib X-2} end
end

{Browse {Fib 30}}
```



Thread spawning continued

- The developers of Oz say that Theads are cheap and you can use alot of them (i.e. 100000)
- Thread spawning is about 60 times faster than in java JDK1.2
- Too many threads may actually kill the Oz environment
- {Browse {Fib 100}} leads to an abnormal exit of Oz using Windows 8.1



Futures

- Oz provides a lazy evaluation feature called Futures

```
Y = !!X
```

- A Future is evaluated when it gets accessed
- A lazy producer consumer example with futures:

```
local
   proc {Producer Xs}
      Xr in
      Xs = product|{ByNeed {Producer Xr} $}
   end
   proc {Consumer N Xs}
      if N>0 then
         case Xs of X|Xr then
            if N mod 1000 == 0 then
               {Browse 'consumed 1000 products'}
            end
            {Consumer N-1 Xr}
         end
      end
   end
in
   {Consumer 10000 thread {Producer $} end}
end
```

Oz vs Ocaml

- Oz is dynamically typed, ocaml has type inference
- Oz has logic variables (not mutable), ocaml has custom types and mutable variables
- Functional programming in Oz is syntactically different but semantically similar

```
declare fun {Average A B}
   {Float.'/' (A + B) 2.0}
end
```

```
let Average A B =
    (A+.B) /. 2.0;;
```

- Pattern matching is quite similar

```
declare fun {Fac X}
   case X of 0 then 1
   [] X then X * {Fac X-1}
   end
end
```

```
let rec Fac = function
    | 0 -> 1
    | n -> n * Fac (n-1);;
```

Examples

Books related to Oz

- Multiparadigm Programming in Mozart/Oz, Second International Conference, MOZ 2004 (2005) Peter Van Roy (Ed.)
- Concepts, Techniques, and Models of Computer Programming (2004) Peter Van Roy and Seif Haridi
- Programming Constraint Services (2002) Christian Schulte
- Concurrent Constraint Programming in Oz for Natural Language Processing (1998) Denys Duchier and Claire Gardent and Joachim Niehren
- Objects for Concurrent Constraint Programming (1997) Martin Henz

Sources

- The Documentation (.chm) delivered with Mozart/Oz Version 1.4.0
- Henz Martin, Gert Smolka, and Jörg Würtz. *„Oz - a programming language for multi-agent systems.“ IJCAI*, 1993.
- Gert Smolka. *„A calculus for higher-order concurrent constraint programming with deep guards.“* Deutsches Forschungszentrum für Künstliche Intelligenz, 1994.
- Gert Smolka. *„An Oz primer.“* in *DFKI Oz Documentation*. 1995.
- Van Roy, Peter. "Programming paradigms for dummies: What every programmer should know." *New computational paradigms for computer music*104 (2009).