First-class Runtime Generation of High-performance Types using Exotypes

Presenter: Cunyuan

February 23, 2023

Before We Start

- This paper is based on the Terra paper.

Before We Start

- This paper is based on the Terra paper.
- You can find my previous slide here: `https://github.com/NeilKleistGao/my-presentations`

What's Exotype?

- So what is Exotype?

What's Exotype?

- So what is Exotype?
- exo- $\rightarrow$ outside.

What's Exotype?

- So what is Exotype?
- exo- $\rightarrow$ outside.
- Exotype $\rightarrow$ type from outside(e.g. a database schema).

An Example

- Considering reading data from a database.

An Example

- Considering reading data from a database.
- If we use a static language(e.g. C), we need to define the data structure beforehand.

An Example

- Considering reading data from a database.
- If we use a static language(e.g. C), we need to define the data structure beforehand.
- If we use a dynamic language, we can not control how the data would be stored.

An Example

- Considering reading data from a database.
- If we use a static language(e.g. C), we need to define the data structure beforehand.
- If we use a dynamic language, we can not control how the data would be stored.
- Data may be boxed, stored in a hash table...

Solution

- MOP + MSP.

Solution

- MOP + MSP.
- MOP: Meta-Object Protocol

Solution

- MOP + MSP.
- MOP: Meta-Object Protocol
- In Lua, we use metatables to extend the normal semantics of objects.

Metatables

```
local myobj = {}
setmetatable(myobj,
{ __index = function(self, field)
    return field end
})
print(myobj.somefield)
```

Metatables

```
local myobj = {}
setmetatable(myobj,
{ __index = function(self, field)
    return field end
})
print(myobj.somefield)
```

- If the key doesn't exist, the Lua interpreter will call the *__index* function.

Metatables

```
local myobj = {}
setmetatable(myobj,
{ __index = function(self, field)
    return field end
})
print(myobj.somefield)
```

- If the key doesn't exist, the Lua interpreter will call the *__index* function.
- In this case, we can get string "*somefield*".

How To Define An Exotype

- $(() \rightarrow \mathit{MemoryLayout}) \star (\mathit{OP}_1 \rightarrow \mathit{Quote}) \star \cdots \star (\mathit{OP}_n \rightarrow \mathit{Quote})$

How To Define An Exotype

- $(() \to \mathit{MemoryLayout}) \star (\mathit{OP}_1 \to \mathit{Quote}) \star \cdots \star (\mathit{OP}_n \to \mathit{Quote})$
- Firstly, compute the memory layout.

How To Define An Exotype

- $(() \rightarrow \mathit{MemoryLayout}) \star (\mathit{OP}_1 \rightarrow \mathit{Quote}) \star \cdots \star (\mathit{OP}_n \rightarrow \mathit{Quote})$
- Firstly, compute the memory layout.
- Given an instance of a primitive operation(OP_i), generate a concrete expression quote to implement it.

Memory Layout

```
Student = terralib.types.newstruct()
Student.metamethods.__getentries = function()
return { {field = "name", type = rawstring },
  {field = "year", type = int} }
end
```

Memory Layout

```
Student = terralib.types.newstruct()
Student.metamethods.__getentries = function()
return { {field = "name", type = rawstring },
  {field = "year", type = int} }
end
```

- We use the metatable *__getentries*.

Memory Layout

```
Student = terralib.types.newstruct()
Student.metamethods.__getentries = function()
return { {field = "name", type = rawstring },
  {field = "year", type = int} }
end
```

- We use the metatable *__getentries*.
- So we can define the data structure by using external information.

Memory Layout

```
Student = terralib.types.newstruct()
Student.metamethods.__getentries = function()
return { {field = "name", type = rawstring },
  {field = "year", type = int} }
end
```

- We use the metatable *__getentries*.
- So we can define the data structure by using external information.
- The keyword *struct* is a syntax sugar.

Memory Layout

```
Student2.metamethods.__getentries = function()
  local file = io.open("data.csv","r")
  --e.g. name,year
  local titles = split(",",file:read("*line"))
  local data = split(",",file:read("*line"))
  local entries = {}
  --loop over entries in titles
  for i,field in ipairs(titles) do
    --is the data a string or an integer?
    local type =
      tonumber(data[i]) and int or rawstring
    entries[i] = { field = field, type = type}
  end
  return entries
end
```

Reference

- Zachary DeVito, Daniel Ritchie, Matt Fisher, Alex Aiken, and Pat Hanrahan. 2014. First-class runtime generation of high-performance types using exotypes. In Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '14). Association for Computing Machinery, New York, NY, USA, 77–88. https://doi.org/10.1145/2594291.2594307
- dictionary.com. https://www.dictionary.com/browse/exo
- Zachary DeVito, James Hegarty, Alex Aiken, Pat Hanrahan, and Jan Vitek. 2013. Terra: A multi-stage language for highperformance computing. In Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'13). ACM, New York, 105–116. DOI:https://doi.org/10.1145/2491956.2462166
- Terra: A low-level counterpart to Lua. https://terralang.org/