Algebraic specification and verification with CafeOBJ

Part 1 – Introduction

Norbert Preining

ESSLLI 2016 Bozen, August 2016
Intro
**Famous Bugs**

- Ariane 5 rocket, Flight 501 (overflow, 1996, $370 million!)
- Mars Climate Orbiter: (lbf vs. N, 1999)
- Intel Pentium F00F bug: planning error
- Heartbleed bug (OpenSSL, 2012-2014)
**How to deal with bugs? – Formal methods**

**post-coding**
Analysis and verification of already developed program code

**pre-coding**
Analysis and verification of domains, models, specifications, requirements, design, etc. – all before coding starts
CURRENT STATUS OF FORMAL METHODS

post-coding
model checking – big success, but limitations due to infinite to finite state transformation
CURRENT STATUS OF FORMAL METHODS

post-coding
model checking – big success, but limitations due to infinite to finite state transformation

pre-coding
interactive theorem provers – acceptance of software engineers/developers?
Our Approach

- reasonable blend of user and machine capabilities
- allow intuitive modelling while preserving a rigorous formal background
- various levels of modelling – from high-level to hard-core
- not fully automated – understanding of design and problems necessary
Our Approach

- reasonable blend of user and machine capabilities
- allow intuitive modelling while preserving a rigorous formal background
- various levels of modelling – from high-level to hard-core
- not fully automated – understanding of design and problems necessary

CafeOBJ and proof scores
Selling Points

- rigid formal background
- order-sorted equational theory
- executable semantics via rewriting
- high-level programming facilities (inheritance, templates and instantiations, …)
- freedom of “language” – syntactic elements can freely defined (postfix, infix, mixfix; overloading, …)
Logical Foundation

Order sorted algebras
partial order of sorts

Hidden algebras
co-algebraic methods, infinite objects

Rewriting logic
transitions as first class objects
Logical Foundation

Order sorted algebras
partial order of sorts

Hidden algebras
co-algebraic methods, infinite objects

Rewriting logic
transitions as first class objects

plus … executable
via rewriting engine
Proof Score Approach

- Domain/design engineers construct proof scores hand-in-hand with formal specification
- Proof scores are executable instructions
- Evaluating/computing/rewriting proof scores provides proofs of the specification and related properties
**Proof Score Approach**

- Domain/design engineers construct proof scores hand-in-hand with formal specification
- Proof scores are executable instructions
- Evaluating/computing/rewriting proof scores provides proofs of the specification and related properties

Proof by construction – proof by rewriting

<table>
<thead>
<tr>
<th>Model and describe a system in order-sorted algebraic specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Construct proof score and verify the specification by rewriting</td>
</tr>
</tbody>
</table>
Functional and Logic programming
Who has progammed here ...?
Who has programmed here ...?

C, Pascal, Python, Java, Vala, Perl, ...
Who has programmed here ...?

C, Pascal, Python, Java, Vala, Perl, ...

versus

Lisp, Scheme, Prolog, Maude, Coq, CafeOBJ, ...
**Imperative versus Functional Programming**

**Imperative programming**

- mixture of *what* is computed and *how* it is computed
  - `push a ; push b ; push c ; add ; mul`
- variables indicate data locations
- sequence of states that is changing
- result is the final state
Imperative versus Functional Programming

Imperative programming

- mixture of *what* is computed and *how* it is computed
  - `push a ; push b ; push c ; add ; mul`
- variables indicate data locations
- sequence of states that is changing
- result is the final state

Functional programming

- separate *what* from *how* (to some degree)
- variables indicate universal properties (for all $x$)
- computation is evaluation of terms
- result is the evaluation of the term
EXAMPLE: GCD

Imperative style

```c
int gcd ( int a, int b ) {
    int c;
    while ( a != 0 ) {
        c = a ; a = b%a ; b = c ;
    }
    return b;
}
```

Here we describe a procedure that computes a result.
**Example: GCD**

**Functional style**

```plaintext
open NAT .
  op gcd : Nat Nat -> Nat .
  var a : Nat . var b : NzNat .
  eq gcd(0,a) = a .
  eq gcd(b,a) = gcd(a rem b, b) .
  reduce gcd(12, 8) .
close NAT .
```

Here we describe mathematical properties of the function to be computed.
(Abstract) Data Types
Data types

Traditional view

- Mathematics: set

\[ S = \text{int} \times \text{char} = \{(a, b) | a \in \text{int} \land b \in \text{char}\} \]

- Programming: structs

```c
struct S {
    int a;
    char b
}
```
DATA TYPES

Modern view

- **Mathematics: algebra**

\[
T = (S, \text{fst} : S \to \text{int}, \text{snd} : S \to \text{char}) \\
= (\text{int} \times \text{char}, \lambda (a, b).a, \lambda (a, b).b)
\]

- **Programming:**

```plaintext
mod S {
    protect(INT + 2TUPLES)
    ...
    ops fst snd : 2Tuple -> Int .
    eq fst( << a ; b >> ) = a .
    eq snd( << a ; b >> ) = b .
}
```
**Data types (cont)**

**Specification** Company X needs a program that does the following things:
- if the customer requests *foo*, assign him a slot
- if all slots are consumed, put the customer into a waiting loop and give him a slot as soon as one becomes free
- if the customer is finished, free the slot
Data types (cont)

specification  Company X needs a program that does the following things:

- if the customer requests $foo$, assign him a slot
- if all slots are consumed, put the customer into a waiting loop and give him a slot as soon as one becomes free
- if the customer is finished, free the slot

implementation  first-in-first-out queues
Company X needs a program that does the following things:

- if the customer requests `foo`, assign him a slot
- if all slots are consumed, put the customer into a waiting loop and give him a slot as soon as one becomes free
- if the customer is finished, free the slot

**implementation** first-in-first-out queues

How do we verify that the program matches the specification?
Data types (cont)

specification a program that implements asymmetric encryption
implementation various key formats, hash algorithms, ...
Data types (cont)

specification a program that implements asymmetric encryption
implementation various key formats, hash algorithms, ...

How do we make sure that the NSA does not interfere?
DATA TYPES AND IMPLEMENTATIONS

More formally

**datatype** equivalence class of isomorphic $\Sigma$-algebras
(e.g., class of all Boolean algebras isomorphic to $\{\top, \bot\}$)

**abstract data type** class of $\Sigma$-algebras closed under isomorphisms
(e.g., class of Boolean algebras that are either 2-valued or 4-valued)
DATA TYPES AND IMPLEMENTATIONS

More formally

**datatype** equivalence class of isomorphic $\Sigma$-algebras (e.g., class of all Boolean algebras isomorphic to $\{\top, \bot\}$)

**abstract data type** class of $\Sigma$-algebras closed under isomorphisms (e.g., class of Boolean algebras that are either 2-valued or 4-valued)

**implementation** is a (concrete) data type
- concrete data representation
- provide realizations of the methods/operations

**specification** is an abstract data type
- allows for different implementations
- no particular data representation
(Algebraic) Specification Languages

Programming languages describe the actual implementation. Verification needs an additional description language for the specification.

Specification languages describe both the specification as well as the implementation. Verification can be carried out within the system.
CafeOBJ
CafeOBJ HISTORY, BACKGROUND, AND RELATIVES

- algebraic specification language based on equational theory
- origin in CLEAR (Burstall and Goguen, early 70s) and OBJ language (Goguen et al., 70-80s SRI and UCI San Diego)
- OBJ2 (Futatsugi, Goguen, Jouannaud, Meseguer at UCI San Diego, 1984) – based on Horn logic, sub sorts, parametrized modules, ...
- OBJ3 (Claude-Kirchner), Maude (Meseguer, full Horn logic)
- formal background based on rewrite logic, initial semantics, and institutions
- similar but unrelated languages: Coq (Jouannaud, based on Marin-Löf’s type theory)
EXAMPLES SPECIFICATIONS

- classical mutual exclusion protocols (QLock)
- simplified cloud protocol
- real time algorithms (Fischer’s mutual exclusion protocol)
- railway signaling systems
- authentication protocols (NSLPK, STS, Otway-Rees)
- key secrecy PACE protocol (German passport)
- e-commerce protocols (SET – practical sized 60000loc)
- UML semantics
- formal fault tree analysis
- secure workflow models
Availability

http://cafeobj.org
Available pages

- Home: Welcome and latest news
- Support & Contact: mailing list, bug tracker, email
- Download & Install: source, binary, installation instructions
- Documentation: reference manual, user manual, some other documents
  Sub-pages: Tutorials, examples, reference manual wiki
- Personnel: list of people
- Recent posts: list of all recent posts
- Links: some links
GETTING INFORMATION

Contact
info@cafeobj.org

Mailing list
users@cafeobj.org
Registration at: https://cafeobj.org/mailman/listinfo/users
Getting information

Contact
info@cafeobj.org

Mailing list
users@cafeobj.org
Registration at: https://cafeobj.org/mailman/listinfo/users

Reference wiki
Reference manual split into Wiki pages

After logging in (various options): ability to change documentation, add entries, clarify, add examples
GETTING INFORMATION

Contact
info@cafeobj.org

Mailing list
users@cafeobj.org
Registration at: https://cafeobj.org/mailman/listinfo/users

Reference wiki
Reference manual split into Wiki pages

After logging in (various options): ability to change documentation, add entries, clarify, add examples

Bug tracker
If you find a bug, or have a feature request:
http://tracker.cafeobj.org/
**ALGEBRAIC SEMANTICS – THE CafeOBJ CUBE**

Logical systems and morphisms are formalised as institutions and institution morphism
**WHAT IS CafeOBJ**

**Algebraic specification language**

Logic foundation:
- order sorted algebra
- co-algebra, hidden algebra
- rewriting logic
What is CafeOBJ

Algebraic specification language
Logic foundation:
- order sorted algebra
- co-algebra, hidden algebra
- rewriting logic

Verification/Programming language
Executable semantics
- equational theory
- rewriting engine (conditional, order-sorted, AC)
- module system
- parametrized modules
- inheritance (module reuse)
- completely free syntax (prefix, postfix, infix, mixfix)
Computational semantics

- equational theory
- axioms are directed
- order-sorted rewriting
Computational semantics

- equational theory
- axioms are directed
- order-sorted rewriting

CafeOBJ as programming language

- module system
- parametrized modules
- inheritance
Computational semantics

- equational theory
- axioms are directed
- order-sorted rewriting

CafeOBJ as programming language

- module system
- parametrized modules
- inheritance
- completely free syntax
CafeOBJ BASICS

Term rewriting

\[
\text{append}(\text{nil}, y) \rightarrow y \\
\text{append}(x : xs, y) \rightarrow x : \text{append}(xs, y)
\]
CafeOBJ BASICS

Term rewriting

append(nil, ys) → ys
append(x : xs, ys) → x : append(xs, ys)

e.g.

append(1 : 2 : 3 : nil, 4 : 5 : nil) → 1 : append(2 : 3 : nil, 4 : 5 : nil)
→ 1 : 2 : append(3 : nil, 4 : 5 : nil)
→ 1 : 2 : 3 : append(nil, 4 : 5 : nil)
→ 1 : 2 : 3 : 4 : 5 : nil
CafeOBJ BASICS

Sorts and order

- arbitrary sorts (distinct universes)
  \[ \text{sort1 sort2 sort3 ...} \]
- may be partially ordered, order is set inclusion
  \[ \text{Nat < Int < Rat, Int < Float} \]
- operator overloading depending of type (domain/codomain)
- inheritance
  \[ \text{op f : Int -> Int} \]
  works also for Nat.
- strictly typed language
  \[ \text{f( X:Float )} \]
  does not work out
CafeOBJ BASICS – OSA

order signature \((S, F)\) such that

- \(S\) is a set of sorts (sort names)
- \(F\) set of operations of the form \(f : s_1 \times \ldots \times s_k \rightarrow s\)
  - \(s_1, \ldots, s_k, s\) are sorts
  - operation name: \(f\)
  - argument sorts: \(s_1 \times \ldots \times s_k\)
  - target sort: \(s\)
  - arity: \(s_1 \times \ldots \times s_k \rightarrow s\)
  - If \(k = 0\) then it is a constant: \(c : \rightarrow s\) of sort \(s\)
CafeOBJ BASICS – OSA

order signature \((S, F)\) such that

- \(S\) is a set of sorts (sort names)
- \(F\) set of operations of the form \(f : s_1 \times \ldots \times s_k \rightarrow s\)
  - \(s_1, \ldots, s_k, s\) are sorts
  - operation name: \(f\)
  - argument sorts: \(s_1 \times \ldots \times s_k\)
  - target sort: \(s\)
  - arity: \(s_1 \times \ldots \times s_k \rightarrow s\)
  - If \(k = 0\) then it is a constant: \(c : \rightarrow s\) of sort \(s\)

order sorted signature \((S, \leq, F)\) such that

- \((S, F)\) is a order signature
- a partial ordering \(\leq\) on \(S\) such that monotonicity condition holds: order in the argument sorts implies order in the target sort.
order signature \((S, F)\) such that
- \(S\) is a set of sorts (sort names)
- \(F\) set of operations of the form \(f : s_1 \times \ldots \times s_k \to s\)
  - \(s_1, \ldots, s_k, s\) are sorts
  - operation name: \(f\)
  - argument sorts: \(s_1 \times \ldots \times s_k\)
  - target sort: \(s\)
  - arity: \(s_1 \times \ldots \times s_k \to s\)
  - If \(k = 0\) then it is a constant: \(c : \to s\) of sort \(s\)

order sorted signature \((S, \leq, F)\) such that
- \((S, F)\) is a order signature
- a partial ordering \(\leq\) on \(S\) such that monotonicity condition holds: order in the argument sorts implies order in the target sort.

order sorted algebra sets for sorts with proper order, operations follow the sorts, monotonicity condition
ADVANTAGES OF OSA

- polymorphism (parametric, subsort) and overloading
- error definition and handling via subsorts
- multiple inheritance
- operational semantics that executes equations as rewrite rules (executable specifications)
- rigorous model-theoretic semantics based on institutions
First steps in CafeOBJ
STARTING CafeOBJ

$ cafeobj
-- loading standard prelude

-- CafeOBJ system Version 1.5.6(PigNose0.99,b3) --
built: 2016 Jan 20 Wed 14:12:49 GMT
prelude file: std.bin
***
2016 Jan 23 Sat 11:09:58 GMT
Type ? for help
***
-- Containing PigNose Extensions --
---
built on SBCL
1.3.1.debian

CafeOBJ>
GETTING HELP

Documents
Reference manual, user manual, some specific manuals (CITP, PigNose, mostly in Japanese)
GETTING HELP

Documents
Reference manual, user manual, some specific manuals (CITP, PigNose, mostly in Japanese)

Help system
CafeOBJ has a built-in documentation and help system, available commands:

- ? – general help
Getting help

Documents
Reference manual, user manual, some specific manuals (CITP, PigNose, mostly in Japanese)

Help system
CafeOBJ has a built-in documentation and help system, available commands:

- `?` – general help
- `?com <class>` – shows available commands classified by `<class>`
  (list of classes when no class is passed in)

<table>
<thead>
<tr>
<th>class</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>decl</td>
<td>CafeOBJ top-level declarations, such as 'module'</td>
</tr>
<tr>
<td>element</td>
<td>Declarations of module constructs, such as 'op'</td>
</tr>
<tr>
<td>parse</td>
<td>Commands parsing a term in the specified context</td>
</tr>
<tr>
<td>rewrite</td>
<td>Invokes term rewriting engine in various manners</td>
</tr>
<tr>
<td>inspect</td>
<td>Inspecting everthing you want.</td>
</tr>
<tr>
<td>switch</td>
<td>Commands controlling system's behavior</td>
</tr>
</tbody>
</table>
GETTING HELP

Documents
Reference manual, user manual, some specific manuals (CITP, PigNose, mostly in Japanese)

Help system
CafeOBJ has a built-in documentation and help system, available commands:

- ? – general help
- ?com <class> – shows available commands classified by <class>
  (list of classes when no class is passed in)
- ? <name> gives the reference manual description of <name>

CafeOBJ> ? op
'op <op-spec> : <sorts> -> <sort> { <attribute-list>
Defines an operator by its domain, co-domain, and type.
'(<sorts>)' is a space separated list of sort names, single sort name.
GETTING HELP

Documents
Reference manual, user manual, some specific manuals (CITP, PigNose, mostly in Japanese)

Help system
CafeOBJ has a built-in documentation and help system, available commands:

- `?` - general help
- `?com <class>` - shows available commands classified by `<class>`
  (list of classes when no class is passed in)
- `? <name>` gives the reference manual description of `<name>`
- `?ex <name>` gives examples for `<name>`, if available

```
CafeOBJ> ?ex parameteri
Example(s) for 'parameterized module'
```

Algebraic specification and verification with CafeOBJ [5pt]Part 1 - Introduction
Getting help

Documents
Reference manual, user manual, some specific manuals (CITP, PigNose, mostly in Japanese)

Help system
CafeOBJ has a built-in documentation and help system, available.

CafeOBJ> ?ap parame
Found the following matches:
  . `view <name> from <modname> to <modname> { <viewelems> }`
  . `search predicates`\parameterized module.
  . qualified sort/operator/parameter
  . `\parameterized module`
  . `[sys:]module[!|]* \ <modname> [ ( <params> ) ] [ <principal_sort_sort
_elements ... }`
  . instantiation of parameterized modules

● ?ap <term> searches all available documentation strings for the terms


**GETTING HELP**

**Documents**
Reference manual, user manual, some specific manuals (CITP, PigNose, mostly in Japanese)

**Help system**
CafeOBJ has a built-in documentation and help system, available commands:

- `?` - general help
- `?com <class>` - shows available commands classified by `<class>`
  (list of classes when no class is passed in)
- `? <name>` gives the reference manual description of `<name>`
- `?ex <name>` gives examples for `<name>`, if available
- `?ap <term>` searches all available documentation strings for the terms
SIMPLE COMPUTATIONS

$  cafeobj
$\textbf{SIMPLE COMPUTATIONS}$

$\texttt{\$ cafeobj}$

\texttt{-- loading standard prelude}

\texttt{...}$
**Simple Computations**

```
$ cafeobj
-- loading standard prelude
...
CafeOBJ>
```
$ cafeobj
-- loading standard prelude
...
CafeOBJ> open NAT .
..%NAT>
Simple Computations

$ cafeobj
-- loading standard prelude
...
CafeOBJ> open NAT .
..
%NAT> red 10 * 20 + 30 .
$  cafeobj
-- loading standard prelude
...
CafeOBJ> open NAT .
..
%NAT> red 10 * 20 + 30 .
-- reduce in %NAT : ((10 * 20) + 30):NzNat
(230):NzNat
(0.0000 sec for parse, 0.0000 sec for 2 rewrites + 2 matches)
%NAT> close
CafeOBJ> quit
Function Declaration

- mathematical definition

\[
\text{square} : \mathbb{N} \rightarrow \mathbb{N} \\
\text{square}(a) = a \times a \\
\text{f} : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N} \\
f(a, b) = a \times a + b \times b
\]
FUNCTION DECLARATION

- mathematical definition

  \[ \text{square} : \mathbb{N} \rightarrow \mathbb{N} \quad \text{f} : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N} \]
  \[ \text{square}(a) = a \times a \quad \text{f}(a, b) = a \times a + b \times b \]

- CafeOBJ:

  1. open NAT .
  2. vars A B : Nat
  3. op square : Nat -> Nat .
  4. eq square(A) = A * A .
  5. op f : Nat Nat -> Nat .
  7. red square(10) .
  8. red f(10, 20) .
  9. close
Recursion

- mathematical definition

\[ \text{sum} : \mathbb{N} \rightarrow \mathbb{N} \]

\[
\text{sum}(a) = \begin{cases} 
0 & \text{if } a = 0 \\
a + \text{sum}(a - 1) & \text{otherwise}
\end{cases}
\]
Recursion

- mathematical definition

\[
\text{sum} : \mathbb{N} \rightarrow \mathbb{N}
\]

\[
\text{sum}(a) = \begin{cases} 
0 & \text{if } a = 0 \\
 a + \text{sum}(a - 1) & \text{otherwise}
\end{cases}
\]

- CafeOBJ:

\begin{verbatim}
1 open NAT .
2 var A : Nat .
3 op sum : Nat -> Nat .
4 eq sum(A) = if A == 0 then 0 else A + sum(p A) fi
5
6 red sum(10) .
7 close
\end{verbatim}
Recursion

- mathematical definition

\[
\text{sum} : \mathbb{N} \rightarrow \mathbb{N} \\
\text{sum}(a) = \begin{cases} 
0 & \text{if } a = 0 \\
 a + \text{sum}(a - 1) & \text{otherwise}
\end{cases}
\]

- CafeOBJ:

1. open NAT .
2. var A : Nat .
3. op sum : Nat -> Nat .
4. eq sum(A) = if A == 0 then 0 else A + sum(p A) fi
5.
6. red sum(10) .
7. close

NOTE

\(p\ A\) is \(A - 1\), called predecessor function
**Conditional Equalities**

- **Mathematical Definition**

  \[
  \text{sum} : \mathbb{N} \rightarrow \mathbb{N} \\
  \text{sum}(a) = \begin{cases} 
  0 & \text{if } a = 0 \\
  a + \text{sum}(a - 1) & \text{if } a > 0 
  \end{cases}
  \]
**Conditional Equalities**

- **mathematical definition**

  \[
  \text{sum} : \mathbb{N} \rightarrow \mathbb{N}
  \]

  \[
  \text{sum}(a) = \begin{cases} 
  0 & \text{if } a = 0 \\
  a + \text{sum}(a - 1) & \text{if } a > 0
  \end{cases}
  \]

- **code simplified by ceq and inline (on-the-fly) variable declaration :Nat**

```plaintext
1  open NAT .
2  op sum : Nat -> Nat .
3  eq sum(0) = 0 .
4  ceq sum(A:Nat) = A + sum(p A) if A > 0 .
5  red sum(10) .
6  close
```

Algebraic specification and verification with CafeOBJ [5pt]Part 1 - Introduction
**Conditional Equalities**

- **mathematical definition**

  \[
  \text{sum} : \mathbb{N} \rightarrow \mathbb{N} \\
  \text{sum}(a) = \begin{cases} 
  0 & \text{if } a = 0 \\
  a + \text{sum}(a - 1) & \text{if } a > 0 
  \end{cases}
  \]

- **code simplified by ceq and inline (on-the-fly) variable declaration** :Nat

  ```
  1  open NAT .
  2  op sum : Nat -> Nat .
  3  eq sum(0) = 0 .
  4  ceq sum(A:Nat) = A + sum(p A) if A > 0 .
  5  red sum(10) .
  6  close
  ```

  CafeOBJ warns sort mismatch for \( p A \):

  `[Warning]: axiom : ... contains error operators..* done.`
Case Distinctions by Sorts

- mathematical definition

\[ \text{sum} : \mathbb{N} \to \mathbb{N} \]

\[ \text{sum}(a) = \begin{cases} 
0 & \text{if } a = 0 \\
 a + \text{sum}(a - 1) & \text{if } a \in \{1, 2, 3, \ldots\}
\end{cases} \]

NB. \( \mathbb{N} = \{0\} \cup \{1, 2, 3, \ldots\} \)
Case Distinctions by Sorts

- mathematical definition

\[
\text{sum} : \mathbb{N} \rightarrow \mathbb{N}
\]

\[
\text{sum}(a) = \begin{cases} 
0 & \text{if } a = 0 \\
a + \text{sum}(a - 1) & \text{if } a \in \{1, 2, 3, \ldots\}
\end{cases}
\]

- code simplified by subsort \textit{NzNat} of \textit{Nat}

```
1 open NAT .
2 op sum : Nat -> Nat .
3 eq sum(0) = 0 .
4 eq sum(A:NzNat) = A + sum(p A) .
5 red sum(10) .
6 close
```
Computational Model

- program is set of (directed) equalities
- execution is rewriting
**Computational Model**

- program is set of (directed) **equalities**
- execution is **rewriting**

```plaintext
1  open NAT .
2  op sum : Nat -> Nat .
3  eq sum(0) = 0 .
4  eq sum(N:NzNat) = N + sum(p N) .
5
6  red sum(3) .
7  close
```
COMPUTATIONAL MODEL

- program is set of (directed) equalities
- execution is rewriting

```
1 open NAT.
2  op sum : Nat -> Nat .
3  eq sum(0) = 0 .
4  eq sum(N:NzNat) = N + sum(p N) .
5
6 red sum(3) .
7 close
```

\[ \text{sum}(3) \rightarrow 3 + \text{sum}(2) \]
\[ \rightarrow 3 + (2 + \text{sum}(1)) \]
\[ \rightarrow 3 + (2 + (1 + \text{sum}(0))) \]
\[ \rightarrow 3 + (2 + (1 + 0)) \]
\[ \rightarrow \ldots \]
\[ \rightarrow 6 \]
CHALLENGE

Implement the following functions in CafeOBJ

- Implement \(\text{factorial}(n) = n!\)
- Implement \(\text{fib}(n)\), \(n\)-th Fibonacci number, where \(\text{fib}(0) = 0\), \(\text{fib}(1) = 1\), and \(\text{fib}(n) = \text{fib}(n - 2) + \text{fib}(n - 1)\) otherwise