Algebraic specification and verification with CafeOBJ

Part 1 – Introduction

Norbert Preining

ESSLLI 2016 Bozen, August 2016

Intro

Famous bugs

- Therac-25 x-ray machine: over-exposition error (5 dead, 1985–1987)
- Ariane 5 rocket, Flight 501 (overflow, 1996, $370 million!)
- Mars Climate Orbiter: (lbf vs. N, 1999)
- Intel Pentium F00F bug: planning error
- Toyota's Electronic Throttle Control System (ETCS): sudden break accidents (2009–2011)
- 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

model and describe a system in order-sorted algebraic specification

construct proof score and verify the specification by rewriting

Functional and Logic programming

Who has progammed here ...?

Who has progammed here ...?

C, Pascal, Python, Java, Vala, Perl, …

Who has progammed 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

```
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

```
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

$$S = \text{int} \times \text{char} = \{(a, b) | a \in \text{int} \wedge b \in \text{char}\}$$

- Programming: structs

```
struct S {
 int a;
 char b
}
```

DATA TYPES

Modern view

- Mathematics: algebra

$$
\begin{aligned}
T &= (S, \mathrm{fst} : S \to \mathrm{int}, \mathrm{snd} : S \to \mathrm{char}) \\
&= (\mathrm{int} \times \mathrm{char}, \lambda(a,b).a, \lambda(a,b).b)
\end{aligned}
$$

- Programming:

```
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

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

Q 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, …

Q How do we make sure that the NSA does not interfere?

Data types and implementations

More formally

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

abstract data type class of Σ-algebras closed under isomorphims (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 Σ-algebras
(e.g., class of all Boolean algebras isomorphic to $\{\top, \bot\}$)

abstract data type class of Σ-algebras closed under isomorphims
(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

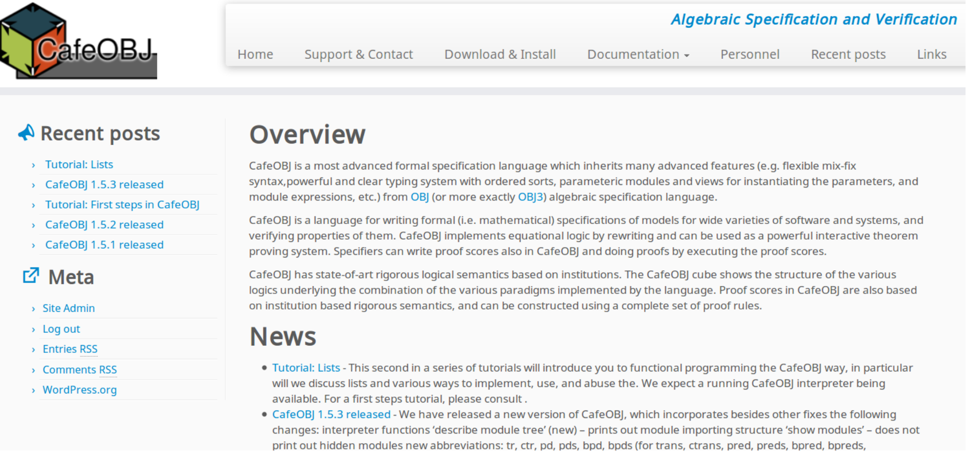
- algebraic specification language based on equational theory
- origin in CLEAR (Burstall and Goguen, early 70s) and OBJ language (Goguen et at., 70-80s SRI and UCI San Diego)
- OBJ2 (Futatsugi, Goguen, Jouannaud, Meseguer at UCI San Diego, 1984) – based on Horn logic, sub-sorts, parametrized modules, …
- 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



CafeOBJ

Algebraic Specification and Verification

Home Support & Contact Download & Install Documentation Personnel Recent posts Links

Recent posts

- Tutorial: Lists
- CafeOBJ 1.5.3 released
- Tutorial: First steps in CafeOBJ
- CafeOBJ 1.5.2 released
- CafeOBJ 1.5.1 released

Meta

- Site Admin
- Log out
- Entries RSS
- Comments RSS
- WordPress.org

Overview

CafeOBJ is a most advanced formal specification language which inherits many advanced features (e.g. flexible mix-fix syntax,powerful and clear typing system with ordered sorts, parameteric modules and views for instantiating the parameters, and module expressions, etc.) from OBJ (or more exactly OBJ3) algebraic specification language.

CafeOBJ is a language for writing formal (i.e. mathematical) specifications of models for wide varieties of software and systems, and verifying properties of them. CafeOBJ implements equational logic by rewriting and can be used as a powerful interactive theorem proving system. Specifiers can write proof scores also in CafeOBJ and doing proofs by executing the proof scores.

CafeOBJ has state-of-art rigorous logical semantics based on institutions. The CafeOBJ cube shows the structure of the various logics underlying the combination of the various paradigms implemented by the language. Proof scores in CafeOBJ are also based on institution based rigorous semantics, and can be constructed using a complete set of proof rules.

News

- Tutorial: Lists - This second in a series of tutorials will introduce you to functional programming the CafeOBJ way, in particular will we discuss lists and various ways to implement, use, and abuse the. We expect a running CafeOBJ interpreter being available. For a first steps tutorial, please consult .
- CafeOBJ 1.5.3 released - We have released a new version of CafeOBJ, which incorporates besides other fixes the following changes: interpreter functions 'describe module tree' (new) – prints out module importing structure 'show modules' – does not print out hidden modules new abbreviations: tr, ctr, pd, pds, bpd, bpds (for trans, ctrans, pred, preds, bpred, bpreds,

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

$$
\begin{tikzcd}
 & \mathrm{HOSA} \arrow[dl] \arrow[dd] & & \mathrm{HOSRWL} \arrow[ll] \arrow[dl] \arrow[dd] \\
\mathrm{HA} \arrow[dd] & & \mathrm{HRWL} \arrow[ll] \arrow[dd] & \\
 & \mathrm{OSA} \arrow[dl] & & \mathrm{OSRWL} \arrow[ll] \arrow[dl] \\
\mathrm{MSA} & & \mathrm{RWL} \arrow[ll] &
\end{tikzcd}
$$

M = many
S = sorted
O = order
H = hidden
A = algebra
RWL = rewriting logic

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

$$
\begin{aligned}
\mathsf{append}(\mathsf{nil}, ys) &\to ys \\
\mathsf{append}(x : xs, ys) &\to x : \mathsf{append}(xs, ys)
\end{aligned}
$$

CafeOBJ basics

Term rewriting

$$\begin{aligned} \mathsf{append}(\mathsf{nil}, ys) &\to ys \\ \mathsf{append}(x : xs, ys) &\to x : \mathsf{append}(xs, ys) \end{aligned}$$

e.g.

$$\begin{aligned} \mathsf{append}(1 : 2 : 3 : \mathsf{nil}, 4 : 5 : \mathsf{nil}) &\to 1 : \mathsf{append}(2 : 3 : \mathsf{nil}, 4 : 5 : \mathsf{nil}) \\ &\to 1 : 2 : \mathsf{append}(3 : \mathsf{nil}, 4 : 5 : \mathsf{nil}) \\ &\to 1 : 2 : 3 : \mathsf{append}(\mathsf{nil}, 4 : 5 : \mathsf{nil}) \\ &\to 1 : 2 : 3 : 4 : 5 : \mathsf{nil} \end{aligned}$$

CafeOBJ BASICS

Sorts and order

- arbitrary sorts (distinct universes)
  `[ sort1 sort2 sort3 ...]`
- may be partially ordered, order is set inclusion
  `[ Nat < Int < Rat, Int < Float ]`
- operator overloading depending of type (domain/codomain)
- inheritance
  `op f : Int -> Int`
  works also for `Nat`.
- strictly typed language
  `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 $f : s_1 \times \ldots \times s_k \to s$
  - $s_1, \ldots, s_k, s$ 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

CafeOBJ BASICS – OSA

order signature (S, F) such that

- S is a set of sorts (sort names)
- F set of operations of the form $f : s_1 \times \ldots \times s_k \to s$
  - $s_1, \ldots, s_k, s$ 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.

CafeOBJ basics – OSA

order signature (S, F) such that

- S is a set of sorts (sort names)
- F set of operations of the form $f : s_1 \times \ldots \times s_k \to s$
  - $s_1, \ldots, s_k, s$ 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)

```
class      description
-----------------------------------------------------------
decl       CafeOBJ top-level declarations, such as 'modul
element    Declarations of module constructs, such as 'op
parse      Commands parsing a term in the specified conte
rewrite    Invokes term rewriting engine in various manne
inspect    Inspecting everthing you want.
```

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 t
'<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'

*
```

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`
 . qualified sort/operator/parameter
 . `parameterized module`
 . `[sys:]module[!|*] <modname> [ ( <params> ) ] [ <principal_sort_s
_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
```

Simple computations

```
$  cafeobj
-- loading standard prelude
...
```

Simple computations

```
$  cafeobj
-- loading standard prelude
...
CafeOBJ>
```

Simple computations

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

Simple computations

```
$  cafeobj
-- loading standard prelude
...
CafeOBJ> open NAT .
..
%NAT> red 10 * 20 + 30 .
```

Simple computations

```
$  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

$$\mathsf{square} : \mathbb{N} \to \mathbb{N} \qquad \mathsf{f} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$$

$$\mathsf{square}(a) = a * a \qquad \mathsf{f}(a, b) = a * a + b * b$$

Function Declaration

- mathematical definition

$$\mathsf{square} : \mathbb{N} \to \mathbb{N} \qquad \mathsf{f} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$$

$$\mathsf{square}(a) = a * a \qquad \mathsf{f}(a, b) = a * a + b * b$$

- CafeOBJ:

```
open NAT .
 vars A B : Nat
 op square : Nat -> Nat .
 eq square(A) = A * A .
 op f : Nat Nat -> Nat .
 eq f(A, B) = A * A + B * B .

 red square(10) .
 red f(10,20) .
close
```

RECURSION

- mathematical definition

$$\mathsf{sum} : \mathbb{N} \to \mathbb{N}$$

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

Recursion

- mathematical definition

$$\mathsf{sum} : \mathbb{N} \to \mathbb{N}$$

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

- CafeOBJ:

```
open NAT .
 var A : Nat .
 op sum : Nat -> Nat .
 eq sum(A) = if A == 0 then 0 else A + sum(p A) fi

 red sum(10) .
close
```

Recursion

- mathematical definition

$$\mathsf{sum} : \mathbb{N} \to \mathbb{N}$$

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

- CafeOBJ:

```
open NAT .
 var A : Nat .
 op sum : Nat -> Nat .
 eq sum(A) = if A == 0 then 0 else A + sum(p A) fi

 red sum(10) .
close
```

NOTE

p A is $A - 1$, called predecessor function

Conditional Equalities

- mathematical definition

$$\mathsf{sum} : \mathbb{N} \to \mathbb{N}$$

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

Conditional Equalities

- mathematical definition

$$\mathsf{sum} : \mathbb{N} \to \mathbb{N}$$

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

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

```
open NAT .
 op sum : Nat -> Nat .
 eq sum(0) = 0 .
 ceq sum(A:Nat) = A + sum(p A) if A > 0 .
 red sum(10) .
close
```

Conditional Equalities

- mathematical definition

$$\mathsf{sum} : \mathbb{N} \to \mathbb{N}$$

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

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

```
open NAT .
 op sum : Nat -> Nat .
 eq sum(0) = 0 .
 ceq sum(A:Nat) = A + sum(p A) if A > 0 .
 red sum(10) .
close
```

CafeOBJ warns sort mismatch for p A:

```
[Warning]: axiom : ... contains error operators..* done.
```

Case Distinctions by Sorts

- mathematical definition

NB. $\mathbb{N} = \{0\} \cup \{1, 2, 3, \ldots\}$

$$\mathsf{sum} : \mathbb{N} \to \mathbb{N}$$

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

Case Distinctions by Sorts

- mathematical definition

NB. $\mathbb{N} = \{0\} \cup \{1, 2, 3, \ldots\}$

$$\mathsf{sum} : \mathbb{N} \to \mathbb{N}$$

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

- code simplified by subsort `NzNat` of `Nat`

```
open NAT .
 op sum : Nat -> Nat .
 eq sum(0) = 0 .
 eq sum(A:NzNat) = A + sum(p A) .
 red sum(10) .
close
```

Computational Model

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

Computational Model

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

```
open NAT .
 op sum : Nat -> Nat .
 eq sum(0) = 0 .
 eq sum(N:NzNat) = N + sum(p N) .

 red sum(3) .
close
```

Computational Model

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

```
open NAT .
  op sum : Nat -> Nat .
  eq sum(0) = 0 .
  eq sum(N:NzNat) = N + sum(p N) .

  red sum(3) .
close
```

$$
\begin{aligned}
\mathsf{sum}(3) &\rightarrow 3 + \mathsf{sum}(2) \\
&\rightarrow 3 + (2 + \mathsf{sum}(1)) \\
&\rightarrow 3 + (2 + (1 + \mathsf{sum}(0))) \\
&\rightarrow 3 + (2 + (1 + 0)) \\
&\rightarrow \cdots \\
&\rightarrow 6
\end{aligned}
$$

Challenge

Implement the following functions in CafeOBJ

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