A

CASL Quick Reference

This appendix provides an overview of the (concrete) syntax of each part of CASL.

Basic specifications

- declarations, definitions:
  - sorts, subsorts
  - functions: total, partial
  - constants: total, partial
  - predicates
  - datatypes
  - sort generation constraints
- variables, axioms
  - formulas
  - terms
- symbols
- comments
- annotations

Structured specifications

- specification structure
  - translation
  - hiding, revealing
  - union, extension
  - free extension, initiality
  - hiding local symbols
  - reference
  - instantiation
- named, generic specifications
  - fitting arguments
- named, generic views
  - fitting views
- symbol lists, maps

Architectural specifications

- named architectures, units
- architectural specifications
- unit specifications
- unit declarations, definitions
- unit expressions, terms

Libraries

- named libraries
- downloadings
- library names, versions

M. Bidoit and P.D. Mosses: CASL User Manual, LNCS 2900, pp. 193–201, 2004.

A.1 Basic Specifications

$\ldots ; \ldots$	list of items (';' optional)
sorts ...	sort declarations and definitions
ops ...	operation declarations and definitions
preds ...	predicate declarations and definitions
types ...	datatype declarations and definitions
generated { ... **}**	sort generation constraint
vars ...	global variable declarations
$\forall \ldots \bullet F_1 \ldots \bullet F_n$	universally-quantified list of axioms
$\bullet F_1 \ldots \bullet F_n$	unquantified list of axioms

A.1.1 Declarations and Definitions

Sort Declarations and Definitions

sort s	sort declaration
sorts $s_1, \ldots, s_n$	sorts declaration
sorts $s < s'$	subsort declaration
sorts $s_1, \ldots, s_n < s'$	subsorts and supersort declaration
sorts $s < s_1; \ldots; s < s_n$	subsort and supersorts declaration
sorts $s_1 = \ldots = s_n$	isomorphic sorts declaration
sort $s = \{v : s' \bullet F\}$	subsort definition

Function Declarations and Definitions

op $f : s_1 \times \cdots \times s_n \to s$	total function declaration
op $f : s_1 \times \cdots \times s_n \to ? s$	partial function declaration
op $f : s \times s \to s, assoc$	associative binary function
op $f : s \times s \to s', comm$	commutative binary function
op $f : s \times s \to s, idem$	idempotent binary function
op $f : s \times s \to s, unit\ T$	unit term for binary function
op $f : s \times s \to s, \ldots, \ldots$	multiple function attributes
ops $f_1, \ldots, f_n : \ldots$	functions declaration
op $f(v_1 : s_1; \ldots; v_n : s_n) : s = T$	total function definition
op $f(v_1 : s_1; \ldots; v_n : s_n) :? s = T$	partial function definition
op $f(\ldots v_{i_1}, \ldots, v_{i_m} : s_i \ldots) \ldots$	abbreviated arguments
ops ... ; ...	multiple declarations/definitions

Constant Declarations and Definitions

op $c : s$	constant declaration
op $c :? s$	partial constant declaration
ops $c_1, \ldots, c_n : s$	constants declaration
op $c : s = T$	constant definition
op $c :? s = T$	partial constant definition
ops ... ; ...	multiple declarations/definitions

Predicate Declarations and Definitions

pred $p : s_1 \times \cdots \times s_n$	predicate declaration
pred $p : ( )$	constant predicate declaration
preds $p_1, \ldots, p_n : \ldots$	predicates declaration
pred $p(v_1 : s_1; \ \ldots; \ v_n : s_n) \Leftrightarrow F$	predicate definition
pred $p \Leftrightarrow F$	constant predicate definition
pred $p(\ldots v_{i_1}, \ldots, v_{i_m} : s_i \ldots) \ldots$	abbreviated arguments
preds $\ldots; \ \ldots$	multiple declarations/definitions

Datatype Declarations

type $s ::= A$	datatype declaration with alternatives
types $s_1 ::= A_1;$	multi-sorted datatype declaration
$\ldots;$	
$s_n ::= A_n$	
generated types ...	generated datatype declaration
free types ...	free datatype declaration

Alternatives (A)

$f(s'_1; \ \ldots; \ s'_k)$	total constructor function
$f(s'_1; \ \ldots; \ s'_k)?$	partial constructor function
$f(\ldots f_i : s_i \ldots)$	total constructor and selector functions
$f(\ldots f_i :? s_i \ldots)$	total constructor, partial selector functions
$f(\ldots f_{i_1}, \ldots, f_{i_m} : s_i \ldots)$	abbreviated selectors
c	constant constructor value
sort s	subsort
sorts $s'_1, \ldots, s'_k$	subsorts
$A_1 \mid \ldots \mid A_m$	multiple alternatives

Sort Generation Constraints

generated { **sorts** ...	generated sorts
ops ...	generating operations
preds ...	
types ...	generated sorts
}	and generating constructors

A.1.2 Variables and Axioms

var $v : s$	global variable declaration
vars $v_1 : s_1; \ldots; v_n : s_n$	global variables declaration
vars $\ldots v_1, \ldots, v_n : s_n \ldots$	abbreviated variables declaration
vars $\ldots; \ldots$	multiple global variable declarations
$\forall v : s \bullet F_1 \ldots \bullet F_n$	universally-quantified list of axioms
$\forall v_1, \ldots, v_n : s \bullet \ldots$	abbreviated quantifications
$\forall \ldots; \ldots \bullet \ldots$	multiple quantifications
$\bullet F_1 \ldots \bullet F_n$	unquantified list of axioms

Formulas (F)

$\forall \ldots \bullet F$	universal quantification on formula
$\exists \ldots \bullet F$	existential quantification
$\exists! \ldots \bullet F$	unique-existential quantification
$F_1 \wedge \ldots \wedge F_n$	conjunction
$F_1 \vee \ldots \vee F_n$	disjunction
$F \Rightarrow F'$	implication
F' *if* F	reverse implication
$F \Leftrightarrow F'$	equivalence
$\neg F$	negation
true	truth
false	falsity
$p(T_1, \ldots, T_n)$	predicate application
$t_0\ T_1\ t_1 \ldots T_n\ t_n$	mixfix predicate application
q	constant predicate
$T = T'$	ordinary (strong) equality
$T \stackrel{e}{=} T'$	existential equality
def T	definedness
$T \in s$	subsort membership

Terms (T)

$f(T_1, \ldots, T_n)$	application
$t_0\ T_1\ t_1 \ldots T_n\ t_n$	mixfix application
$t_0\ T_1, \ldots, T_n\ t_1$	literal syntax
c	constant
v	variable
$T : s$	sorted term
T *as* s	projection to subsort
T *when* F *else* T'	conditional choice

A.1.3 Symbols

Character set: ASCII (with optional use of ISO Latin-1).

Key Words and Signs

Reserved key words (always lowercase):

and arch as axiom axioms closed def else end exists false fit
forall free from generated get given hide if in lambda library local
not op ops pred preds result reveal sort sorts spec then to true
type types unit units var vars version view when with within

Reserved key signs:

: :? ::= = => <=> ¬ . · | |−> \/ /\

Unreserved key signs:

< ∗ × −> ? ! [ ] { }

Key words and signs representing mathematical symbols:

forall exists exists! *not in lambda* $=e=$ −> => <=> . · |−> /\ \/
∀ ∃ ∃! ¬ ∈ λ $\stackrel{e}{=}$ → ⇒ ⇔ • • ↦ ∧ ∨

Identifiers

Identifiers for sorts and variables are simple words (other than reserved words) possibly containing digits, primes, and *single* underscores:

Elem *Y_1* *Z2′* *A_Rather_Long_Identifier*

Sort identifiers can also be compound:

List[*Int*] *Map*[*Index*, *Elem*]

Identifier for operations and predicates can moreover be sequences of (unreserved) signs, with any brackets [ ] { } balanced:

$+ - * / \backslash \& = < > [\,] \{\,\} ! ? : . \$ @ \# \hat{} \tilde{} ¡ ¿ \times \div £ © \pm ¶ § {}^{1}\,{}^{2}\,{}^{3} \cdot \not< {}^{\circ} \neg \mu |$

or single decimal digits *1 2 3 4 5 6 7 8 9 0*, or single quoted characters $'c'$.

The signs ( ) ; , ‘ ” % are *not* allowed in identifiers, nor are the ISO Latin-1 signs for general currency, yen, broken vertical bar, registered trade mark, masculine and feminine ordinals, left and right angle quotes, fractions, soft hyphen, acute accent, cedilla, macron, and umlaut.

Operation and predicate identifiers can also be compound:

order[__ < __]

Function and predicate identifiers can also be infixes, prefixes, postfixes, and general mixfixes, formed from words and/or sequences of signs separated by *double* underscores (indicating the positions of the arguments), with any brackets [ ] { } balanced:

__++__ ||__|| {[__]} push_onto__ __.select1

Invisible mixfix identifiers (such as ____) with two or more arguments are allowed. (Subsort embeddings give the effect of invisible unary functions.)

An operation, or predicate identifier can be compound, with a list of identifiers appended to its final token.

Literal Strings and Numbers

"this is a string" 42 3.14159 1E−9 27.3e6

Library Identifiers

Names of libraries are either paths, e.g.:

BASIC/NUMBERS BASIC/ALGEBRA_II

or URLs formed from A...ZA...z0...9$-_@.&+!*"'(),~ and hexadecimal codes %XX, and prefixed by HTTP://, FTP://, or FILE:///.

Version numbers of libraries are hierarchical: 0, 0.999, 1, 1.0, 1.0.2.

A.1.4 Comments

%% This is a comment at the end of a line...
...%{ This is an in-line comment }% ...
...%{ This a comment that might take
several lines }%
%[This is for commenting-out text
%{ including other kinds of comment }%]%

A.1.5 Annotations

A label is of the form %(text)%.

An end-of-line annotation is of the general form **%word** ...
with a space following the word.

A possibly multi-line annotation is of the general form **%word(** ... **)%**
with no space preceding the '('.

A.2 Structured Specifications

A.2.1 Specifications (*SP*)

SP **with** *SM*	symbol translation
SP **hide** *SL*	hiding listed symbols
SP **reveal** *SM*	revealing/translating listed symbols
SP_1 **and** ... **and** SP_n	union
SP_1 **then** ... **then** SP_n	extension
free *SP*	free or initial
local *SP* **within** SP'	hiding of local symbols
closed *SP*	self-contained
SN	reference to named specification
$SN[FA_1]...[FA_n]$	instantiation of generic specification

A.2.2 Named and Generic Specifications

spec $SN = SP$ **end**	named specification (**end** optional)
spec $SN[SP_1]...[SP_n] = SP$ **end**	generic specification (**end** optional)
spec $SN[SP_1]...[SP_n]$	generic specification
given $SP''_1,...,SP''_m = SP$ **end**	with imports (**end** optional)

Fitting Arguments (*FA*)

SP **fit** *SM*	fitting by symbol map
SP	implicit fitting
FV	fitting view

A.2.3 Named and Generic Views

view $VN : SP$ **to** $SP' = SM$ **end**	named view (**end** optional)
view $VN[SP_1]...[SP_n]$	generic view
$: SP$ **to** $SP' = SM$ **end**	(**end** optional)
view $VN[SP_1]...[SP_n]$	generic view
given $SP''_1,...,SP''_m$	with imports
$: SP$ **to** $SP' = SM$ **end**	(**end** optional)

Fitting Views (*FV*)

view *VN*	reference to named view
view $VN[FA_1]...[FA_n]$	instantiation of generic view

A.2.4 Symbol Lists (*SL*) and Maps (*SM*)

$SY_1, ..., SY_n$	lists (maybe with **sorts**, **ops**, **preds**)
$SY_1 \mapsto SY'_1, ..., SY_n \mapsto SY'_n$	maps (maybe with **sorts**, **ops**, **preds**)
$..., SY_i, ...$	in a map, abbreviates $..., SY_i \mapsto SY_i, ...$

A.3 Architectural Specifications

A.3.1 Named Architectures and Units

arch spec $ASN = ASP$ **end**	named arch. spec. (**end** optional)
unit spec $SN = USP$ **end**	named unit spec. (**end** optional)

A.3.2 Architectural Specifications (*ASP*)

ASN	arch. spec. name
units $UD_1; \ldots; UD_n$ **result** UE	basic arch. spec.

A.3.3 Unit Specifications (*USP*)

SP	unit specification
$SP_1 \times \ldots \times SP_n \to SP$	generic-unit specification
closed USP	self-contained
arch spec ASP	models of arch. spec.

A.3.4 Unit Declarations and Definitions (*UD*)

$UN : USP$	unit declaration
$UN : USP$ **given** $UT_1, \ldots, UT_n$	importing units
$UN = UE$	unit definition

A.3.5 Unit Expressions (*UE*)

UT	unit term
$\lambda UN_1 : SP_1; \ldots; UN_n : SP_n \bullet UT$	unit composition

A.3.6 Unit Terms (*UT*)

UT **with** SM	symbol translation
UT **hide** SL	hiding listed symbols
UT **reveal** SM	revealing/translating listed symbols
UT_1 **and** ... **and** UT_n	amalgamation
local $UD_1; \ldots; UD_n$ **within** UT	local units
UN	unit name
$UN[UT_1]\ldots[UT_n]$	generic-unit application
$UN[UT_1$ **fit** $SM_1]\ldots[UT_n$ **fit** $SM_n]$	with fitting by symbol maps

A.4 Libraries

library *LN* ...	named library of downloadings, specifications, views

A.4.1 Downloadings

from *LN* **get** $IN_1, \ldots, IN_n$ **end**	downloads listed items
from *LN* **get** ... $IN \mapsto IN'$ ... **end**	renames downloaded items

A.4.2 Library Names (*LN*)

BASIC/NUMBERS	greatest version registered
BASIC/ALGEBRA_II **version** *0.999*	specified version registered
HTTP://...	greatest version unregistered
HTTP://... **version** *1.0.2*	specified version unregistered