Alloy: A Quick Reference

Martin Monperrus
with contributions of Marc Frappier

February 23, 2015

This document presents the syntax and intuitive meaning of the main concepts of the Alloy modeling language. It somehow focuses on the "navigation expression style " of Alloy (see Jakson's "Software Abstractions", p.34) and refers to classical object-oriented knowledge. It is inspired from Mohammad Mousavi's "Z and Alloy Notations: A Quick Reference".

Contributions are welcome as pull requests on:
https://github.com/monperrus/alloy-quick-reference

1 Basics

Notation	*Intuitive Meaning*
`sig Book { }`	Class/type definition
`sig Dictionary extends Book { ... }`	Inheritance
`sig Book { author: Author }`	Field, reference, cannot be null
`b.author`	Field access
`pred predicate_name { ... }`	Predicate (returns true or false), property definition
`pred has_author[b : Book] { }`	Parametrized property definition
`fact { ... }`	Predicate that always holds, a formula which must be satisfied by all instances of an Alloy model
`run { ... }`	Find instances
`run { ... } for 3 but 1 Author`	Find instances with constraints on of instances[1]
`pred foo[b:Book] { .... }` `run foo for 3 but 1 Author`	Find instances satisfying predicate "foo"
`assert GaGb { good_author implies good_book }` `check GaGb`	Find counter-examples violating the assertion
`check { good_author implies good_book }`	Anonymous assertions
`fun books(a: Author) : set Book ...`	Function definition (side-effect free), returns an expression of some type (here `set Book`)
`let x: E | ...`	Identifier definition

2 First Order Logic

Notation	*Intuitive Meaning*
`p and q, p && q`	Conjunction, logical and
`p or q, p || q`	Disjunction, logical or
`p implies q, p => q`	Implication
`p iff q, p <=> q`	Equivalence
`not p, !p`	Negation
`all b:Book | has_author[b]`	Universal quantification, for all

[1] except for intergers (Int) where n is the bitwidth

`some b:B | has_author[b]`	Existential quantification, at least one
`some disjoint b1, b2 : B | ...`	Existential quantification, forcing removal of case $b1 = b2$

3 Set Theory

Notation	*Intuitive Meaning*
`Book`	All instances of Book
`sig Dictionary extends Book { ... }`	Set Dictionary is a subset of set Book, all extension signatures (subclasses) are disjoint.
`univ`	All instances of all types (the universe)
`none`	Empty set {}
`no x`	Set x is empty
`#Book`	Cardinality
`a in b`	Subset or equal
`a = b`	Set equality
`some x`	Set not empty, $\|x\| \geq 1$
`one x`	$\|x\| = 1$
`lone x`	$\|x\| \leq 1$
`x & y`	Intersection
`x + y`	Union
`x - y`	Difference

4 Objects

Notation	*Intuitive Meaning*
`abstract sig Book {}`	Abstract class, cannot be instantiated
`one sig Bible extends Book {...}`	Singleton, $\|Bible\| = 1$, Bible subset of Book
`enum Vegetable {Potato, Carrot, Tomato}`	Enumeration
`sig Book { author: Author }`	Field, reference, cannot be null
`b.author`	Field access
`b.isGood`	Method call without parameters (with `pred isGood[x:Book]`)
`b.isBetterThan[b2]`	Method call with parameters (with `pred isGood[x,y: Book]`)
`sig Book { author: Author }`	Multiplicity $[1 \ldots 1]$
`sig Book { author: set Author }`	Multiplicity $[0 \ldots \star]$
`sig Book { author: lone Author }`	Multiplicity $[0 \ldots \star]$
`sig Library { books: Author -> Book }`	Dictionary, hashtable, cartesian product
`myLibrary.books[a]`	Dictionary member access
`dom[Library.books]`	Keys of the dictionary (with "open util/relation")
`ran[Library.books]`	Values of the relation (with "open util/relation")

5 Relations

Notation	*Intuitive Meaning*
`a->b`	Cartesian product $a \times b$
`a.b`	Relational product
`a<:b`	Domain restriction of relation b by set a
`a:>b`	Range restriction of relation b by set a
`joe.^friends`	Transitive closure
`~x`	Inverse
`T-> one U`	Total function from T to U

`T-> lone U`	Partial function from T to U
`T one -> one U`	Bijection from T to U

6 Integer Arithmetic

When using integer, add `open util/integer` in preamble and always specify the integer bitwidth (`run {...} for 5 Int`). There is no real or floating-point arithmetic in Alloy.

Notation	*Intuitive Meaning*
`plus[a,b]`	Addition
`minus[a,b]`	Substraction
`mul[a,b]`	Multiplication
`div[a,b]`	Division
`rem[a,b]`	Remainder of a divided by b
`sum[a]`	Returns the sum of the integers of set a
`a < b, a = b, a > b, a =< b, a >= b`	Integer comparison

7 Ordering

Notation	*Intuitive Meaning*
`open util/ordering[State] as states`	Declares a total order on State
`states/first`	First element
`states/last`	Last element
`states/next[s]`	Next element
`states/prev[s]`	Previous element
`states/nexts[s]`	All elements after s
`states/prev[s]`	All elements before s

8 Sequences / Lists

Notation	*Intuitive Meaning*
`s :  seq A`	Ordered and indexed sequence of elements
`s.first`	Head of the list
`s.rest`	Tail of the list
`s.elems`	All elements as an unordered set
`s.idxOf [x]`	Returns the first index where x appears in s, if x does not appear in s, it returns the empty set.
`s.insert[i, x]`	Returns a new list where x is inserted at index i

9 Precedence

In increasing order; operators on the same line have the same priority.

Expressions (operands are not booleans)

1. ~ , ^ , *
2. .
3. []
4. <: , :>
5. ->
6.
7. ++
8.
9. +, -
10. no , some , lone , one , set
11. ! , not
12. in , = , < , > , = , =< , =>

Logical expressions (operands are booleans)

1. ! „ not
2. && , and
3. => , implies , else
4. <=> , iff
5. || , or
6. let , no , some , lone , one