Cheat Sheet

www.rascal-mpl.org

Module

```
module Example
import N;
data Day = mon() | tue();
public int double(int n) {
   return 2 * n;
}
private str S ="rascal";
alias Money = int;
```

Declarations

import *N*	Import module *N*
data *N* = *P* | ...	Data type *N*
T N(*T V*, ...) *S* *T N*(*T V*, ...) *S* throws *N*, *N*	Function *N*
T V = *E*	Variable *V*
alias *N* = *T*	Alias *N*
anno *T T* @ *N*	Annotation *N*
rule *N P* => *P* rule *N P* : *S*	Rewrite rule *N*

Control Statements

if(*E*) *S*
if(*E*) *S* else *S*
while(*E*) do *S*
do *S* while(*E*)
for(*E*,*E*,...) *S*

switch(*E*) {
case *P* => *P*
case *P* : *S*
...
default: *S*
}

try *S* catch *P* => *P* catch *P* : *S* finally: *S*
throw *E*
fail
return, return *E*
solve(*V*, *V*, ...) *S*

Other statements

{ *S*; *S*; ... }
assert *E* assert *E* : *E*
test *E* test *E* : *E*
append *E*
insert *E*

Types and Values

bool	true, false
int	1, 0, -1, 123456789
real	2.30E-14
str	"rascal"
loc	|file:///etc/passwd|
tuple[*T*, ...] tuple[*T V*, ...]	<"monday", 1>
list[*T*]	[1,2,3,2,1]
set[*T*]	{1,2,3}
map[*T*,*T*]	("mo":1, "tue": 2)
rel[*T*, ...] rel[*T V*, ...]	{<2001,5>, <2002, 6>}
node	f(), g("abc",[2,3,4])
void, value	

Assignment

V = *E*	Variable
A[*E*] = *E*	Subscript
A.*N* = *E*	Field
<*A*,*A*,...> = *E*	Tuple
A ? *E* = *E*	Isdefined
A @ *N* = *E*	Annotation
N(*A*,*A*,...) = *E*	Constructor
+=,-=,*=, /= &=, ?=	*A* = *A op E*

Expressions

E . *N*	Field selection: x.a
E [*N* = *E*]	Field assignment: x[a=3]
E < *N*, ... >	Field projection x<a>
E[*E*, ...]	Subscription: L["a"]
E @ *N*	Annotation
E[@*N* = *E*]	Annotation replacement
N(*E*,...)	Function call: f(3, "a")
E ? *E* : *E*	Conditional expression: (x >3) ? 30 : 40
[*E* .. *E*] [*E*, *E*, .. *E*]	Range: [1 .. 10] [1, 2 .. 10]
P <- *E*	Enumerator: n <- [1..10]
[*E*, ... | *E*, ...]	List comprehension: [n * n | int n <- [1 .. 10]]
{*E*, ... | *E*, ...}	Set comprehension { n | int n <- [1 .. 10], n%3 == 0}
(*E* : *E* | *E*, ...)	Map comprehension (k : size(k) | k <- ["a", "bcd"])

visit(*E*) {
case *P* => *P*
case *P* : *S*
...
}
	Visit (traversal:
	visit(T){
	case int n => n+1;
	}

one(*E*,...)	One *E* is true
all(*E*,...)	All *E*s are true
if, while, do, for	

Operators

E + *E*	Addition, union, concatenation
E - *E*	Subtraction, difference
E * *E*	Multiplication, product
E / *E*	Division
E % *E*	Modulo
E & *E*	Intersection
E join *E*	Join
E o *E*	Compose
E && *E*	And
E || *E*	Or
E == *E*, *E* != *E*	Equal, Not equal
E < *E*, *E* <= *E*, *E* > *E*, *E* >= *E*	Comparison
E ==> *E* *E* <==> *E*	Implies Equivalence
E in *E* *E* notin *E*	Element of Not element of
P := *E* *P* !:= *E*	Match No match
- *E* ! *E*	Arith. Negation, Logical Not
E +, *E* *	(Reflexive) transitive closure
E ?	Is defined

Abstract Patterns	
T V	Variable declaration
*V**	Multi-variable (list or set) declaration
V	Variable use
[*P*, *P*, ...]	List
{*P*, *P*, ...}	Set
<*P*, *P*, ...>	Tuple
N(*P*, *P*, ...)	Node
/ *P*	Descendant
V : *P*	Labeled
T V : *P*	Typed, Labelled
[*T*] *P*	Type constraint

Concrete Patterns	
` *L L* ...`	Quoted
T ` *L L* ...`	Typed & Quoted
L L ...	Unquoted
<*T V*>	Typed variable

Regular Expression Patterns	
/ ... /	Regular Expression
<*V* : *R*>	Named RegExp
L L ...	Unquoted
<*T V*>	Typed variable

Legend	
E	Expression
S	Statement
V	Variable
T	Type
A	Assignable
P	Pattern
N	Name
R	Regular Expression
L	Lexical token

Standard Library	
ATermIO	Read/write values as ATerms
Benchmark	Benchmarking tools
Boolean	Boolean functions
Exception	Exceptions a program can catch
Graph	Graph manipulation
Integer	Integer functions
IO	Input/output
JDT	Java fact extraction functions
LabeledGraph	Labeled graph manipulation
List	List functions
Location	Location functions
Map	Map functions
Node	Node functions
PriorityQueue	Functions on prio queues

Standard Library	
Real	Real functions
Relation	Functions on relations
Resource	Retrieve Eclipse resources
RSF	Read RSF files
Set	Functions on sets
String	Functions on strings
Tree	Functions on parse trees
Tuples	Functions on tuples
ValueIO	Read/write values as text/binary
viz::Basic	Display basic datatypes
viz::Figure::Chart	Chart drawing
viz::Figure::Core	Core figure functions
viz::Figure::Render	Rendering of figure
viz::View	Graph/tree/text display of values