CFML Cheat List

Types

```
b   bool          boolean value
n   int           idealized integer
l   loc           memory location
x   var           variable
v   val           closed value
t   trm           term
s   state         state                 := fmap loc val
h   heap          piece of state        := state
H   hprop         heap predicate        := heap->Prop
Q                 postcondition         := val->hprop  or  := A->hprop
E   ctx           substitution context  := list (var*val)
```

Entailment

```
H1 ==> H2         := forall h, H1 h -> H2 h
Q1 ===> Q2        := forall x, Q1 x ==> Q2 x
```

Core heap predicates

```
\[]             hempty                 empty state predicate     := fun h => h = fmap_empty /\ P
\[P]            hpure P                pure heap predicate       := fun h => h = fmap_empty /\ P
\Top            htop                   any heap predicate        := fun h => True
H1 \* H2        hstar H1 H2            separating conjunction    := fun h => exists h1 h2, h = h1 \u h2
                                                                              /\ fmap_disjoint h1 h2
                                                                              /\ H1 h2 /\ H2 h2
\exists x, H    hexists (fun x => H)   existential on hprop      := fun h => exists x, H
Q \*+ H         (fun x => Q x \* H)    star with a postcondition
p ~~~> v        hsingle p v            singleton heap            := fun h => h = fmap_single p v
p ~~> V         Hsingle p V            lifted singleton heap     := hsingle p (enc V)
p ~> MList L    MList L p              mutable list predicate
p ~> Record`{ f1 := V1; f2 := V2 }
```

Advanced heap predicates

```
\GC             hgc                    any heap predicate             := exists H, H \* \[haffine H]
\forall x, H    hforall (fun x => H)   universal on hprop             := fun h => forall x, H
H1 \-* H2       hwand H1 h2            magic wand                     := exists H, H \* [H \* H1 ==> H2]
Q1 \--* Q2      qwand Q1 Q2            magic wand on postconditions   := \forall x, Q1 x \-* Q2 x
                hand H1 H2             non-separating conjunction     := fun h => H1 h /\ H2 h
```

Judgments

```
red s t s' v     evaluation judgment
hoare t H Q      total correctness Hoare triple, on the whole state
triple t H Q     total correctness SL triple, on a piece of state
Triple t H Q     lifted SL triple
H ==> wp t Q     weakest-precondition style SL triple
H ==> Wp t Q     lifted wp-style SL triple
TRIPLE t PRE H POST Q       := Triple t H Q
```

Wp operators

```
formula        result of wp                := (val->hprop)->hprop
wpgen E t      formula generator
subst x v t    substitution
isubst E t     iterated substitution
structural F   structural formula
mkstruct F     structural wrapper
```

Lifted wp operators

```
Formula        result of Wp                := forall A, Enc A -> (A->hprop)->hprop
`F             structural wrapper          := MkStruct F
^F Q           applied formula             := F _ _ Q
Structural F
```

Syntax for values and terms

```
VFun 'x := t1
VFix 'f 'x := t1
'x
Cstr C x1 xn
If_ t0 Then t1 Else t2
If_ t0 Then t1 End
t1 '; t2
Let 'x := t1 in t2
Let Rec 'f 'x := t1 in t2
Match t With p1 '=> t1 | pn '=> tn End
Fail
New`{ f1 := t1; f2 := t2 }
t1 '. f
Set t1 '. f := t2
'ref t
'! t
t1 ':= t2
'not t
t1 '+ t1
t1 '= t2
t1 '< t2
..
```

Tactics for entailments

```
xpull
   applies e.g; to: \exists x, \[x = 3] \* H1 ==> H2
   produces:        forall x, x = 3 -> (H1 ==> H2)

xsimpl
   applies to:     H1 ==> H2
   invokes xpull, then cancel out items on both sides

xsimpl X1 XN
   applies to:     H1 ==> \exists x1 xn, H2
   set x1 := X1 and x2 := X2, then call xsimpl

xchange M
   where M:      H1 ==> H2         or     H1 = H2
   applies to:   H1 \* H3 ==> H4
   produces:     H2 \* H3 ==> H4

xchange <- M
   where M:      H1 = H2
   applies to:   H2 \* H3 ==> H4
   produces:     H1 \* H3 ==> H4

xchanges E
   invokes "xchange E" then "xsimpl"

xunfold R
   applies to:   p ~> R X
   changes to:   R X p
```

Tactics for structural rules

```
xwp         TRIPLE (f v) PRE H POST Q
   turns the goal into H ==> wpgen (f v) Q
   useful to establish a specification

xtriple     TRIPLE (f v) PRE H POST Q
   turns the goal into H ==> `App f v Q
   useful to prove a derived specification

xgc             H ==> ^F Q
   turns into:  H ==> ^F Q \* \GC

xcast           H ==> ^(Cast V) Q
   turns into:  H ==> Q V

xpost Q'        H ==> ^F Q
   turns into:  H ==> ^F Q'   and   Q' ===> Q
```

Term tactics for term rules

```
xfail       H ==> ^(`Fail) Q
   turns the goal to [False]

xval        H ==> ^(`Val v) Q
   turns the goal to H ==> Q v

xval V      H ==> ^(`Val v) Q
   specifies the value of which v is the encoding

xfun        H ==> ^(`Val (val_fun x t)) Q
   instantiates Q as the specification for the function

xapp        H ==> ^(`App f v) Q
   exploits the registered specification Triple for f

xapp E
   enables to specify the specification triples

xapp_nosubst
   xapp with the substitution that may occur for the
reult

xappn
   repeat xapp

xseq        H ==> ^(`Seq F1 F2) Q
   remark: xapp usually applies directly

xlet        H ==> ^(`Let 'x := F1 F2) Q
   remark: xapp usually applies directly

xif         H ==> ^(`If b Then F1 Else F2) Q
   performs the case analysis

xcase       H ==>^ (`Case ..) Q
   performs the case analysis
```

```
===============
TLC Cheat List
===============

E: stands for an expression
H: stands for an existing hypothesis
X: stands for a fresh identifier
I: stands for an introduction pattern

A tactic followed with the symbol "~" triggers call to [auto] on all subgoals
A tactic followed with the symbol "*" triggers call to [jauto], a variant of [induction eauto]
A tactic arguments may have a subterm "rm X" to trigger a call to "clear X" after the tactic

Most useful tactics
-------------------

foo ;=> I1 .. IN     (shorthand for "foo; intros I1 .. IN")
introv I1 I2 .. IN   (introduction tactic that inputs only the name of hypotheses, not of variables)

inverts H             (inversion followed with substitution of all equalities produced)
inverts H as I1..IN   (similar to "inverts H", but produced hypotheses are named explictly)
invert H              (similar to "inverts H", but no substitution is performed, everything is left in the
goal)

lets I: E0 E1 ... EN        (instantiates a lemma E0 on arguments Ei and names the result)
applys E0 E1 ... EN         (instantiates a lemma E0 on arguments Ei and apply the result to the goal)
specializes H E1 ... EN     (instantiates an hypothesis H in-place on the arguments Ei)
forwards I: E0 E1 .. EN     (instantiates a lemma on all its arguments, "lets I: E0 E1 .. EN __ __ .. __")
rewrites (>> E0 E1 ... EN)  (instantiates a lemma E0 arguments Ei, then perform a "rewrite" with the result)
applys_eq E0 i1 .. iN       (applys a lemma E0 up to equality on arguments at specified indices i1 .. iN)

In all tactics above, "E0 E1 .. EN" may be written "(>> E0 E1 ... EN)", as shown for the tactic "rewrites".
Any of the arguments E1 .. EN may be a wildcard, written "__".

Very useful tactics
-------------------

false             (replaces the goal by "False", and kills it if obvious contradictions are found)
false E           (a shorthand for [false; applys E])
tryfalse          (solves the goal if [false] kills it, else does nothing)

math              (proves math related goal, variant of "omega")
fequals           (improved implementation of "f_equal", leveraging the "congruence" tactice)
simpls            (shorthand for "simpl in *")
unfolds R         (shorthand for "unfold R in *")

case_if           (performs a case analysis on the first "if" statement in the goal)
cases E as I      (performs a case analysis on E, remembering the equality as I)

asserts I: E      (asserts statement E as first subgoal, destruct E as I in the second goal)
cuts I: E         (asserts statement E as second subgoal, destruct E as I in the first goal)
sets X: E         (defines X as a local definition for E, and replaces occurences of E with X)
sets_eq X: E      (introduces a name X and an equality X = E, and replaces occurences of E with X)
clears H1 ... HN  (clears hypotheses Hi and their dependencies)

iff               (tactic to split an equivalence "P <-> Q")
splits            (splits an N-ary conjunction into N goals)
branch N          (selects the N-th branch of a disjunction with several branches)
exists E1 .. EN   (provides witnesses to an N-ary existential goal, wildcards "__" are supported)

inductions_wf I: E X  (well-founded induction, E is a measure or relation, X the argument, I the hypothesis)
gen_eq X: E           (generalize X as E and add "X = E" as hypothesis in the goal, useful for inductions)
gen H1 H2 .. HN       (generalizes and clears hypotheses Hi and their dependencies, = "dependent generalize")

Normalization tactics
---------------------

rew_list          (normalizes basic list functions, e.g. "++" and "length")
rew_listx         (normalizes advanced list functions, e.g. "map")
rew_heap          (normalizes Separation Logic expressions)
rew_bool_eq       (normalizes expressions involving "isTrue", in particular)
rew_fmap          (normalizes finite map expressions)

TLC notations
=============

If P then X else Y
\{}   \{x}   (E1 \u E2)  (E1 \n E2)   (E2 \- E2)   (E1 \-- x)   (x \in E)   (x \notin E)   (E1 \c E2)
M[x]    M[x := v]
forall_ x \in E, P
x <= y       (overloaded on different types using typeclasses)
```