DRAFT DO NOT COPY OR CITE

Introduction

This document collects together a number of potentially useful, general-purpose CLIF axioms which can be used to do 'structural' abbreviations, re-arrangements and so on. They are therefore not special to any particular topic, and can be used (or adapted to use) in a wide range of ontologies. They also may serve as a useful source of examples of the flexibility and power of CLIF syntax.

Syntax reminders

In CLIF, any name can be used for any syntactic role: it may denote a relation or a funciton or an individual. If this seems counter-intuitive, it is harmless, though often redundant, to state the various cases separately. CLIF does not distinguish syntactically between names and variables; a 'variable' is merely a bound name. Any name may be bound by a quantifier. All quantifiers must be explicit.

Names are case-sensitive, can use any Unicode characters, and must be enclosed by double quotes if they contain spaces or parentheses. Single quotes (apostrophes) indicate character strings. Numerals are decimal. (Some axioms below use *plus* or *succ* without defining them; any reasonably powerful axiomatic arithmetic may be assumed.) Any name beginning with the three-character sequence '...' is a CL sequence marker, analogous to a sequence variable in KIF (written there with the prefix '@'). If only one sequence marker is used in an axiom, it is usually simply written as '...'. Quantification over sequences alwways includes the empty sequence, often giving a case that requires its own additional axiom.

Sentences containing sequence markers should be used only as axiom schemas, never set as goals to be proved, in order to stay within first-order logic.

AXIOMS

(cl-text

(cl-comment 'n-ary equality and inequality, the latter corresponding to the OWL *allDifferent*'

(forall (x)(same x))
(forall (x y ...)(iff (same x y ...)(and (= x y)(same x ...))))

(forall (x) (different x))
(forall (x y ...)(iff (different x y ...)(and (not (= x y))(different x ...)(different y ...))))

)

(cl-comment 'two useful ways of extending unary and binary relations to sequences'

(forall (f)(iff (distributive f)(and (f)(forall (x ...)(iff (f x ...)(and (f x)(f ...)))))))

(forall (f)(iff (chained f)(forall (x)(and (f x)(forall (y ...)(iff (f x y ...)(and (f x y)(f y ...)))))))

)
(cl-comment '
examples:
(distributive Human) (Human Bill Jill Jean Harry Osama)
(distributive distributive) (forall (x)(if (ClassifierPredicate x)(distributive x)))
(chained lessThan) (lessThan 45 x 78 y 122)
')

(cl-comment 'Some useful functionals on predicates. These all correspond to OWL primitives, for example *And* is owl:intersectionOf'

(forall (x)((And) x))
(forall (x y ...)(iff ((And x ...) y)(and (x y)((And ...) y))))

(forall (x)(not ((Or) x)))
(forall (x y ...)(iff ((Or x ...) y)(or (x y)((Or ...) y))))

(forall (x)(not ((OneOf) x)))
(forall (x y ...)(iff ((OneOf x ...) y)(or (= x y)((OneOf ...) y)))

(forall (x y)(iff ((Not x) y)(not (x y))))

)
(cl-comment '
examples
(= Man (And Human Male)) (Man Joe)
(= PersonSeen030908 (OneOf Joe Bill Osama Harry)) (PersonSeen030908 x)
(forall ((x (Not Human)))(dumb x))
')

(cl-comment 'Names may be called with empty argument sequences by enclosing them in parentheses. No special meaning is given to this by the logic itself, but conventions can be imposed by suitable axioms, such as requiring a no-argument function to yield a unary relation which is true just of the thing itself. Note that conventions stated like this are universal, so must be used with care, as they can easily become inconsistent with other conventions and usages.'

(forall (x y)(iff ((x) y)(= x y)))

)
(cl-comment '
alternative axiom

(forall (x)(= (x)(OneOf x)))
')

(cl-comment 'Similarity/identity between relations. Note, this is a very strong statement but weaker than a direct equation between two relation names.'

(forall (f g)(iff (Equiv f g)(forall (...)(iff (f ...)(g ...)))))

)

(cl-comment 'Explicit lists.
CL argument sequences are very like 'flat' LISP lists We can make this explicit by using a variadic list function to denote this list, thereby placing it into the universe of discourse. Then it is routine to define construction and selection functions on these lists in several commonly used ways. Here we give various vocabularies.'

(forall (x)(iff (List x)(exists (...)(= x (list ...)))))

(= nil (list))

(forall (x ...)(and (= x (first (list x ...))) (= (list ...)(rest (list x ...)))))

(same first car hd head)

(same rest cdr tl tail)

(forall (f)(iff (Listable f)(forall (...)(and (= (f ...)(f (list ...)))(iff (f ...)(f (list ...)))))))

)

(cl-comment 'The last axiom means that if F is Listable, then (F x), (F (list x)), (F (list (list x))), and so on are *all* equivalent, as are (F x y z)(F (list x y z)), (F (list (list x y z))), etc.. Treating function and relation argument sequences as lists in this way, therefore, effectively makes it impractical to use such relations and functions *on lists themselves*. This means for example that it would be a serious **mistake** to assert (Listable rest). In general, one should decide to use *either* argument lists *or* argument sequences, and then stick to that decision throughout an ontology.

To make this construction apply only to 'flat' lists of arguments, replace the last sentence above by:

(distributive (Not List))
(forall (f)(iff (Listable f)(forall (...)(if ((Not List) ...)(and (= (f ...)(f (list ...)))(iff (f ...)(f (list ...))))))))

To prevent the just-mentioned idempotence behavior on singleton argument lists, it is necessary to treat the zero, one and two-or-more cases separately:

```
(forall (f)(iff (Listable f)(and
                        (= (f) (g))
                        (iff (f) (g))
                        (forall ((x (Not List)))(and (= (f x)(f (list x)))(iff (f x)(f (list x))) ))
                        (forall (... x y)(and (= (f x y ...)(f (list x y ...)))(iff (f x y ...)(f (list x y ...))) ))
)))
```

With this axiom, (F x) and (F (list x)) are equivalent when x is not a list, but not (F x) and (F (list (list x))),

It should be clear how to adapt such axioms to apply only to relations or only to functions, if required.
End of comment. ')

```
(cl-comment  'Applying relations to many arguments in sequence.'

(forall (r ...1 x ...2)(iff ((AppAll r ...1) x ...2)(and (r ...1 x)((AppAll r ...1) ...2)) ))
(forall (...)((AppAll ...)) )

)
(cl-comment 'Use is best explained by an example:

((AppAll LessThan 12) a b c d)
```

says that all of a, b, c and d are greater than 12:

```
(and (LessThan 12 a)(LessThan 12 b)(LessThan 12 c)(LessThan 12 d))

(forall (...)(Listable (ApAll ...))
((AppAll Duty Harry Afghanistan) (DeployList Harry))
(= (DeployList Harry)(list (period 010104 053105)(period 011306 060106)(period 010107 053107)))
```

together imply

```
(and
(Duty Harry Afghanistan (period 010104 053105))
(Duty Harry Afghanistan (period 011306 060106))
(Duty Harry Afghanistan (period 010107 053107))
)
```

End of comment. ')