14. Conses

14.1 Cons Concepts

A *cons* is a compound data *object* having two components called the *car* and the *cdr*.

```
car  cons    rplacd
cdr  rplaca
```

Figure 14-1. Some defined names relating to conses.

Depending on context, a group of connected *conses* can be viewed in a variety of different ways. A variety of operations is provided to support each of these various views.

14.1.1 Conses as Trees

A *tree* is a binary recursive data structure made up of *conses* and *atoms*: the *conses* are themselves also *trees* (sometimes called "subtrees" or "branches"), and the *atoms* are terminal nodes (sometimes called *leaves*). Typically, the *leaves* represent data while the branches establish some relationship among that data.

```
caaaar  caddar  cdar       nsubst
caaadr  cadddr  cddaar     nsubst-if
caaar   caddr   cddadr     nsubst-if-not
caadar  cadr    cddar      nthcdr
caaddr  cdaaar  cdddar     sublis
caadr   cdaadr  cddddr     subst
caar    cdaar   cdddr      subst-if
cadaar  cdadar  cddr       subst-if-not
cadadr  cdaddr  copy-tree  tree-equal
cadar   cdadr   nsublis
```

Figure 14-2. Some defined names relating to trees.

14.1.1.1 General Restrictions on Parameters that must be Trees

Except as explicitly stated otherwise, for any *standardized function* that takes a *parameter* that is required to be a *tree*, the consequences are undefined if that *tree* is circular.

14.1.2 Conses as Lists

A *list* is a chain of *conses* in which the *car* of each *cons* is an *element* of the *list*, and the *cdr* of each *cons* is either the next link in the chain or a terminating *atom*.

A *proper list* is a *list* terminated by the *empty list*. The *empty list* is a *proper list*, but is not a *cons*.

An *improper list* is a *list* that is not a *proper list*; that is, it is a *circular list* or a *dotted list*.

A *dotted list* is a *list* that has a terminating *atom* that is not the *empty list*. A *non-nil atom* by itself is not considered to be a *list* of any kind---not even a *dotted list*.

A *circular list* is a chain of *conses* that has no termination because some *cons* in the chain is the *cdr* of a later *cons*.

```
append      last           nbutlast  rest
butlast     ldiff          nconc     revappend
copy-alist  list           ninth     second
copy-list   list*          nreconc   seventh
eighth      list-length    nth       sixth
endp        make-list      nthcdr    tailp
fifth       member         pop       tenth
first       member-if      push      third
fourth      member-if-not  pushnew
```

Figure 14-3. Some defined names relating to lists.

14.1.2.1 Lists as Association Lists

An *association list* is a *list* of *conses* representing an association of *keys* with *values*, where the *car* of each *cons* is the *key* and the *cdr* is the *value* associated with that *key*.

```
acons  assoc-if      pairlis  rassoc-if
assoc  assoc-if-not  rassoc   rassoc-if-not
```

Figure 14-4. Some defined names related to assocation lists.

14.1.2.2 Lists as Sets

Lists are sometimes viewed as sets by considering their elements unordered and by assuming there is no duplication of elements.

```
adjoin         nset-difference    set-difference    union
intersection   nset-exclusive-or  set-exclusive-or
nintersection  nunion             subsetp
```

Figure 14-5. Some defined names related to sets.

14.1.2.3 General Restrictions on Parameters that must be Lists

Except as explicitly specified otherwise, any *standardized function* that takes a *parameter* that is required to be a *list* should be prepared to signal an error of *type* **type-error** if the *value* received is a *dotted list*.

Except as explicitly specified otherwise, for any *standardized function* that takes a *parameter* that is required to be a *list*, the consequences are undefined if that *list* is *circular*.