1

Data Types

Strings

Strings are sequences of characters enclosed by double quotes. For example,

```
"abc"
"84 . a%3"
"string with spaces"
```

Strings are self-evaluating, ie they evaluate to themselves. The names of atoms are stored as strings.

Strings cannot have a value, property list or function definition associated with them.

Numbers

Integer	eg.	`43 -99`
Real	eg.	`22.3 -9.9`
Bignums	eg	`1234567898`

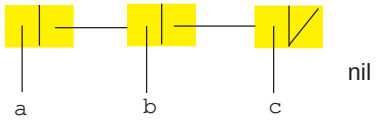
Also: **array** - blocks of contiguous storage
vector - arbitrary size, kind of property list
hunk - vector of from 1-128 lisp values
binary - machine coded functions
- (compiled , internal, external)
port - used to do I/O

2

Lists

All lists with the exception of () or nil consist of data elements of type list (also known as cons cells). Each element has two fields (named respectively `car`, `cdr`)

Lists are represented in Lisp using binary trees. For example, the list `(a b c)` is represented as:

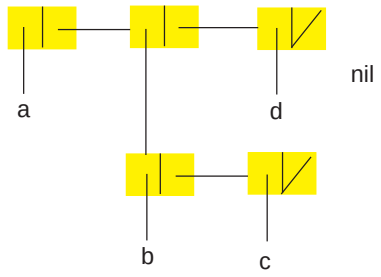


Note that the `cdr` field of the last node contains an empty pointer, representing a pointer to the atom `nil`. This convention indicates that there are no more elements in the list.

3

Lists (continued)

A list may have another list as one of its elements. For example, the list `(a (b c) d)` is represented as:



What is the representation of `(a ((()) b) c)` ???

©1993 John Mylopoulos

4

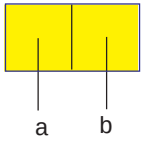
The Dotted Pair Notation for Lists

Dotted Pair Notation constructs lists by using a binary operator (.) which represents a cons cell and its two operants represent the values of the two fields of the corresponding cell..

For example:

```
'(a)     -->  '( a . nil)
'(a b c) --> '(a . (b .  (c . nil)))
'(a (b c) d) -->
            '(a.((b.(c.nil)).(d.nil))
```

In fact, this notation lets us construct binary trees that do not have a `nil` at the end of the list:



This value could be constructed using:

```
-->   '(a.b)
```

(or, for some Lisp systems

```
-->   (cons 'a 'b) )
```


©1993 John Mylopoulos

5

Symbols

Symbols are non-constant atoms that appear in the symbol table of Lisp.

Symbols have associated with them:

- **a unique "location"** - used by `eq`
- **a string name** - used to print and uniquely identify the symbol
- **a value** - used for evaluation
- **a function** - used in function evaluation
- **a property list -** used to associate attribute-value pairs to a symbol

The `read` function recognizes and parses the string input of s-expressions, and converts this into the internal representation used by Lisp.

`read` also creates entries for new symbols in the symbol table, and ensures that all atoms with the same name refer to the same symbol.

©1993 John Mylopoulos

6

Property Lists

Property lists are used to associate information to an atom. For example, consider the atom `alex` (intended to represent a particular person). To `alex` we may want to attach information such as

attribute name	attribute values
name	(Alex Borgida)
hair	black
children	((Michelle) (Anna Maria))
interests	(AI Databases (Programming Languages))
habits	nil

©1993 John Mylopoulos

7

Property List Operations

```
(putprop  symbol attr-value attr-name)
```

`putprop` replaces (assigns) to the property list of `symbol` whose attribute name is `attr-name` the value of `attr-value`. `putprop` is a function of three arguments that evaluates all of its arguments and returns the property value.

For example, after

```
(putprop 'alex 'coffee 'habits)
```

the value of `alex`'s `habits` is `coffee`

```
(get  symbol  attr-name)
```

`get` looks up the property list of `symbol` and returns the value of attribute `attr-name`. `get` is a function of two arguments that evaluates all of its arguments and returns the property value.

For example, after

```
(putprop 'alex 'coffee 'habits)

--> (get 'alex 'habits)
coffee
```


©1993 John Mylopoulos

8

Property List Operations

```
(remprop  symbol attr-name)
```

`remprop` removes from the property list of `symbol` the attribute with name is `attr-name`, along with its value. `remprop` is a function of two arguments that evaluates all of its arguments and returns `nil` if the attribute is not found.

For example after `(remprop 'alex 'habits)`:

```
-->(get 'alex 'habits)
   nil
```

Use of Property Lists:

```
--> (putprop 'alex '(Alex Borgida) 'name)
   (Alex Borgida)
--> (putprop 'alex 'black 'hair)
   black
--> (get 'alex 'name)
   (Alex Borgida)
--> (get 'alex 'mother)
   nil
--> (remprop 'alex 'name)
   ((Alex Borgida))
--> (get 'alex 'name)
   nil
```


©1993 John Mylopoulos