```
cbf:              datablock            { cbf_failnez (cbf_find_parent (&($$), $1, CBF_ROOT)) }
                ;
cbfstart:                              { $$ = ((void **) context) [1]; }
                ;
datablockstart:   cbfstart             { cbf_failnez (cbf_make_child (&($$), $1, CBF_DATABLOCK, NULL)) }
                | cbf datablockname    { cbf_failnez (cbf_make_child (&($$), $1, CBF_DATABLOCK, $2)) }
                ;
datablock:        datablockstart       { $$ = $1; }
                | assignment           { cbf_failnez (cbf_find_parent (&($$), $1, CBF_DATABLOCK)) }
                | loopassignment       { cbf_failnez (cbf_find_parent (&($$), $1, CBF_DATABLOCK)) }
                ;
category:         datablock categoryname  { cbf_failnez (cbf_make_child (&($$), $1, CBF_CATEGORY, $2)) }
                ;
column:           category columnname  { cbf_failnez (cbf_make_child (&($$), $1, CBF_COLUMN, $2))  }
                | datablock itemname   { cbf_failnez (cbf_make_new_child (&($$), $1, CBF_CATEGORY, NULL))
                                        cbf_failnez (cbf_make_child (&($$), $$, CBF_COLUMN, $2)) }
                ;
assignment:       column value         { $$ = $1;
                                        cbf_failnez (cbf_set_columnrow ($$, 0, $2, 1)) }
                ;
loopstart:        datablock loop       { cbf_failnez (cbf_make_node (&($$), CBF_LINK, NULL, NULL))
                                         cbf_failnez (cbf_set_link ($$, $1)) }
                ;
loopcategory:     loopstart categoryname  { cbf_failnez (cbf_make_child (&($$), $1, CBF_CATEGORY, $2))
                                          cbf_failnez (cbf_set_link ($1, $$))
                                          $$ = $1; }
                | loopcolumn categoryname { cbf_failnez (cbf_find_parent (&($$), $1, CBF_DATABLOCK))
                                          cbf_failnez (cbf_make_child (&($$), $$, CBF_CATEGORY, $2))
                                          cbf_failnez (cbf_set_link ($1, $$))
                                          $$ = $1; }
                ;

loopcolumn:       loopstart itemname    { cbf_failnez (cbf_make_new_child (&($$), $1, CBF_CATEGORY, NULL))
                                          cbf_failnez (cbf_make_child (&($$), $$, CBF_COLUMN, $2))
                                          cbf_failnez (cbf_set_link ($1, $$))
                                          cbf_failnez (cbf_add_link ($1, $$))
                                          $$ = $1; }
                | loopcolumn itemname   { cbf_failnez (cbf_find_parent (&($$), $1, CBF_DATABLOCK))
                                          cbf_failnez (cbf_make_child (&($$), $$, CBF_CATEGORY, NULL))
                                          cbf_failnez (cbf_make_child (&($$), $$, CBF_COLUMN, $2))
                                          cbf_failnez (cbf_set_link ($1, $$))
                                          cbf_failnez (cbf_add_link ($1, $$))
                                          $$ = $1; }
                | loopcategory columnname  { cbf_failnez (cbf_make_child (&($$), $1, CBF_COLUMN, $2))
                                          cbf_failnez (cbf_set_link ($1, $$))
                                          cbf_failnez (cbf_add_link ($1, $$))
                                          $$ = $1; }
                ;
loopassignment:   loopcolumn value      { $$ = $1;
                                          cbf_failnez (cbf_shift_link ($$))
                                          cbf_failnez (cbf_add_columnrow ($$, $2))  }
                | loopassignment value  { $$ = $1;
                                          cbf_failnez (cbf_shift_link ($$))
                                          cbf_failnez (cbf_add_columnrow ($$, $2))  }
                ;
loop:             LOOP
                ;
datablockname:    DATA                  { $$ = $1; }
                ;
categoryname:     CATEGORY              { $$ = $1; }
                ;
columnname:       COLUMN                { $$ = $1; }
                ;
itemname:         ITEM                  { $$ = $1; }
                ;
value:            STRING                { $$ = $1; }
                | WORD                  { $$ = $1; }
                | BINARY                { $$ = $1; }
                ;
```

Fig. 5.1.3.5. Example of *bison* data defining a CIF parser (taken from *CBFlib*).

preloading an internal data structure that holds the entire CIF may not be the optimal choice for a given application. When reading a CIF it is difficult to avoid the need for extra data structures to resolve the issue of CIF order independence. However, when writing data to a CIF, it may be sufficient simply to write the necessary tags and values from the internal data structures of an application, rather than buffering them through a special CIF data structure.

It is tempting to apply the same reasoning to the reading of CIF and create a fixed ordering in which data are to be processed, so that no intermediate data structure will be needed to buffer a CIF.