The Grand Recognizer Unification

Bernd Paysan, net2o
M. Anton Ertl, TU Wien

The Benefits of Uniformity

- Unix: Everything is a file
  Files are sequences of bytes

- Object-oriented programming: `draw` graphical object

- Forth: Everything is a word
  Addresses, integers, xts are cells

Recognize string in text interpreter

	wordlist	recognizer
one	wordlist `find-name-in`	recognizer `execute`
many	search order `find-name`	recognizer sequence `recognize`

Wordlist as recognizer

- *wid* is implemented as xt of recognizer
  `( c-addr u - nt rectype-nt | rectype-null )`

- Use as wordlist:

```
: find-name-in ( c-addr u wid -- nt | 0 )
    execute rectype-nt <> if 0 then ;
```

- Lower-level implementation later

Recognizer sequences as recognizers

```
: rec-sequence ( xt1 .. xtn n "name" -- )
  create dup , dup , 0 ?do , loop
does> ( c-addr u -- ... rectype )
  {: c-addr u addr :}
  addr cell+ @ cells addr 2 cells +
  dup >r + r> ?do
    c-addr u i @ execute
    dup rectype-null <> if
      unloop exit then
    drop
  1 cells +loop
  rectype-null ;
```

Recognizer sequences as recognizers

```
: follow-defers ( xt1 -- xt2 )
  begin
    dup is-defer? while
      defer@
  repeat  ;

: get-rec-sequence ( xt -- xt1 .. xtn n )
  follow-defers dup is-rec-sequence? 0= if
    drop 0 exit then
  >body cell+ dup cell+ over @ dup 0= if
    nip nip exit then
  dup >r cells rot + do
    i @ -1 cells +loop
  r> ;
```

Recognizer sequences as recognizers

```
: set-rec-sequence ( xt1 .. xtu u xt -- )
  follow-defers
  dup is-rec-sequence? 0= -12 and throw
  >body {: u addr :}
  u addr @ > -49 and throw
  u addr cell+ !
  addr 2 cells + dup u cells + swap ?do
    i !
  1 cells +loop ;
```

Search order as recognizer

```
: rec-nothing ( c-addr u -- rectype-null )
  2drop rectype-null ;

' rec-nothing dup 2dup 2dup 2dup 2dup 2dup 2dup 2dup 16 rec-sequence search-order

: get-order ( -- wid1 ... widu u )
  ['] search-order get-rec-sequence ;

wordlist constant root-wordlist

: only ( -- )
  root-wordlist 1 search-order set-rec-sequence ;

: set-order ( wid1 ... widn n -- )
  dup -1 = if drop only exit then
  ['] search-order set-rec-sequence ;
```

Search order as recognizer

- What can be put in the search order?
  `find-name-in` must work: `( c-addr u - nt rectype-nt | rectype-null )`
  `traverse-wordlist` must work

Search order as recognizer

- `find`, `find-name` first searches locals, then search order
- locals: `rec-loc ( c-addr u - nt rectype-nt | rectype-null )`

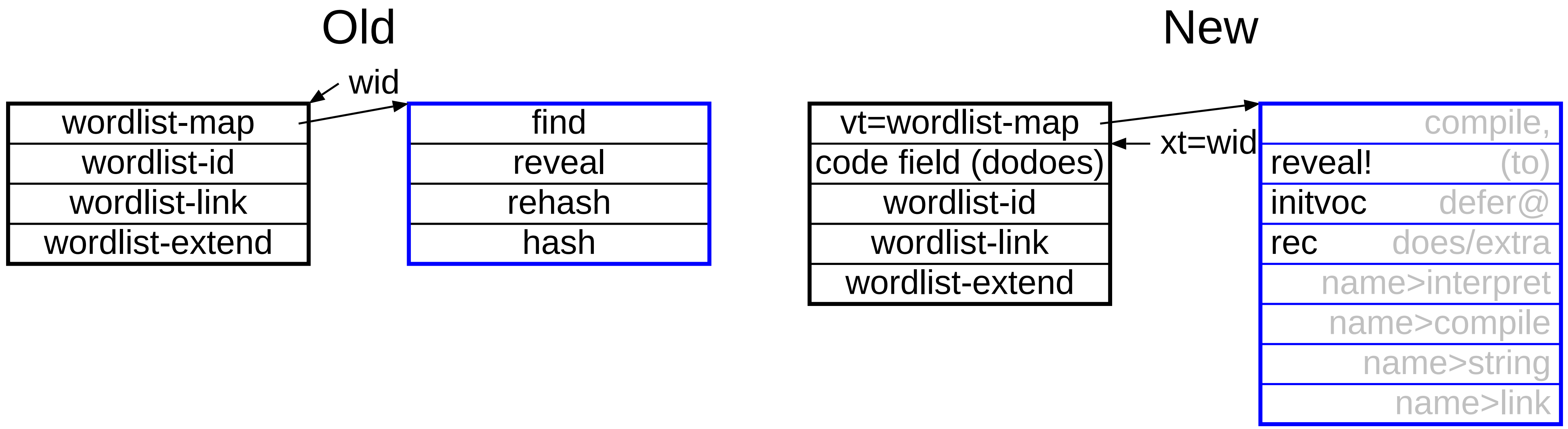
```
defer rec-nt

' search-order ' rec-loc 2 rec-sequence rec-locals

: activate-locals   ( -- ) ['] rec-locals   is rec-nt ;
: deactivate-locals ( -- ) ['] search-order is rec-nt ;
deactivate-locals

: find-name ( c-addr u -- nt|0 )
  ['] rec-nt find-name-in ;
```

Wordlist as recognizer: Implementation



Conclusion

- Uniformity allows better factoring
- Unify wordlist, search order, recognizer, recognizer sequence
  Implement them as recognizers
- search order becomes a recognizer sequence
- wordlist becomes a (recognizer) word

Alternative: Recognizer Sequence as binary operator

- `two-recs ( xt1 xt2 "name" - )`
- simpler to implement
- but does not match `get-order set-order`