Can Functional Programmers Make `make` Make Sense?

Norman Ramsey
Tufts University

(A Worse is Better production)

Can Functional Programmers Make `make` Make Sense?

Norman Ramsey
Tufts University

(A Worse is Better production)

$[\![$  $]\!]$ $\xrightarrow{?}$ λ

Application: class web site

Justifications:

- **Derived web pages (schedule, etc)**
- **Platform-specific binaries**
- **Minimize collisions among staff**

Surprise! 10,000–12,000 source lines

Dependencies for lecture notes

Ingredients:

- **Markdown file with embedded LaTeX**
- **Photos of blackboard**

Both computed dynamically!

Write notes using lightweight markup

```
Inductively defined data type: smallest set satisfying

$$
\mathrm{LIST}(A) = \{ \lit{'()} \} \cup \{ \lit{(cons a as)} \mid
    \lit{a} \in A, \lit{as} \in \mathrm{LIST}(A) \}
$$

**Any** list is therefore *constructed* with nil or with cons.

  - How can you tell the difference?
  - What, therefore, is the structure of a function
    that consumes a list?

Example: 'length'
```

(Turns into readable HTML)

TEX fragment turns into image

Can be shown on screen

$$\mathrm{LIST}(A) = \{\texttt{'()}\} \cup \{\texttt{(cons a as)} \mid \texttt{a} \in A, \texttt{as} \in \mathrm{LIST}(A)\}$$

(Also spliced into HTML)

Dynamic dependencies using `mk`

What images are needed?

```
<|$REQUIRE/bin/xtex -rules $WWW/notes
```

Script emits “rules” and “recipes”

- Notes depend on image (rule):

```
www/notes.html: www/xtex/slide-070a8c.png
```

- Image built from TEX file (rule + recipe):

```
www/xtex/slide-070a8c.ps: www/xtex/slide-070a8c.dvi
        dvips -E -x 1000 -o $target $prereq
```

mk is a simpler, better make

Engineered by Andrew Hume

- **Make vars are Unix env vars are shell vars**
- **“Recipes” are shell scripts**
- **No built-in rules**

Dynamic dependencies using `mk`

What images are needed?

```
<|$REQUIRE/bin/xtex -rules $WWW/notes
```

Script emits “rules” and “recipes”

- Notes depend on image (rule):

```
www/notes.html: www/xtex/slide-070a8c.png
```

- Image built from TEX file (rule + recipe):

```
www/xtex/slide-070a8c.ps: www/xtex/slide-070a8c.dvi
        dvips -E -x 1000 -o $target $prereq
```

Notes built using static rules

The build:

```
$WWW/&.mdx:D: $WWW/& $REQUIRE/bin/xtex \
              $REQUIRE/lib/lua/5.1/notes.lua
    $REQUIRE/bin/xtex -markdown $WWW/$stem > $target

$WWW/notes.html:DQ: $WWW/notes.mdx \
                    $TOP/bin/add-notes-photos \
                  $PHOTOSTOP ...
    (insert shell code from hell)
```

Dependencies for the dependencies

`xtex -markdown` produces the notes

`xtex -rules` produces the dependencies

But

- `xtex` requires platform-specific binaries
- Can't call `xtex` until binaries built
- Software is evolving; binaries change
- `./configure; make` a complete nonstarter

Hack: conditional inclusion

```
<$TOP/require/src/base.mk
<$TOP/require/base.mk
<|$TOP/require/bin/include-if-present \
        $RLIBLUA/posix.so $RLIBLUA/md5.so \
        $TOP/www/base.mk
```

Hack: conditional inclusion

```
<$TOP/require/src/base.mk
<$TOP/require/base.mk
<|$TOP/require/bin/include-if-present \
        $RLIBLUA/posix.so $RLIBLUA/md5.so \
        $TOP/www/base.mk
```

“Consistency” model:

```
    mk; mk
```

Hack: conditional inclusion

```
<$TOP/require/src/base.mk
<$TOP/require/base.mk
<|$TOP/require/bin/include-if-present \
        $RLIBLUA/posix.so $RLIBLUA/md5.so \
        $TOP/www/base.mk
```

“Consistency” model:

```
mk; mk
```

How can we do better?

`mk` is attractive at small/medium scale

Attractive features

- **Easy to integrate panoply of Unix tools**
- **Easy to integrate custom tools**
- **Appears declarative**
- **Caching (incremental rebuild) sometimes works**

Deficiencies emerge gradually

Examples:

- **Maintenance of dependencies**
- **Portability may involve devil's bargain**
- **Caching doesn't always work**

The language picture is bleak

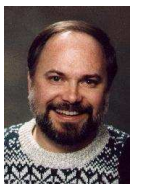
Low-level, algorithmic semantics

Makefiles don’t compose

- ***Recursive Make Considered Harmful*** **(Miller 1998)**
- **Textual substitution too low-level**

Existing solutions not for medium scale

Prior art:

- Nmake/iffe (1985+)  / 
- Odin (1990) 
- Vesta (2000) 
- OMake (2006) 
- Shake (2010) 

Room left in the design space

	language	add tool?	“run”	note
Nmake	rules++++	easy	std shell	scales, portable
Odin	C?	no	—	caches
Vesta	functional	hard	“bridge”	scales, proven
OMake	rules+fp	easy	custom	small delta
Shake	Haskell	?	Haskell	consistent
wanted	rules(+)	easy	std shell	composable

Part II: Ideas, Questions

Low-hanging fruit

Interpretation of relative pathnames

- **Today: relative to where `mk` is invoked**
- **Correct: Relative to location of mention**

Assignment to variables

- **No mutation**
- **Limited in scope (OMake)?**
- **Union semantics for sets of names?**

Controlled destruction of derived files

- **Automate `mk clean`**

Medium challenge

Dependencies computed by tools:

- **Easy integration**
- **Caching**

Replace textual inclusion

Harder problems I: Dependencies

Can we get this right?

- **Dependencies for dependency generator (like `xtex`)**
- **Eliminate phase distinction?**
- **Cache results properly?**

Harder problems II: Metarules

Like type classes, but based on names

- **Ways to disambiguate?**
  (e.g., choose metarule by name?)

Really hard problem: Fixed points

Example: LaTeX

Harder example: textbook with index, cross-reference, and mini-index

3.9.2 A polymorphic, higher-order sort

Sorting is another example of a polymorphic computation. Industrial-strength sorts are highly tuned for performance, and it's desirable to create a single sort that can sort any kind of object. In sorting, each kind of object needs it own ordering function, just as in the set example, each kind of object needs its own equality function. To sort a list of numbers, the primitive function `<` probably suffices, but to sort a list of lists, this function no longer makes sense. We can define a polymorphic, higher-order function `mk-insertion-sort`, which when passed an argument `lt`, returns a function to sort a list of elements into nondecreasing order according to function `lt`. The sort itself is based on the algorithm in chunk 71a.

93a ⟨*transcript* 66⟩+≡ ◁92d 93b▷

```
-> (define mk-insertion-sort (lt)
     (letrec (
       (insert (lambda (x l)
                 (if (null? l) (list1 x)
                     (if (lt x (car l))
                         (cons x l)
                         (cons (car l) (insert x (cdr l)))))))
       (sort   (lambda (l)
                 (if (null? l)
                     '()
                     (insert (car l) (sort (cdr l)))))))
       sort))
```

The great thing about `mk-insertion-sort` is that it is easy to reuse. For example, we can easily sort numbers in increasing or decreasing order.

93b ⟨*transcript* 66⟩+≡ ◁93a 93c▷

```
-> (val sort< (mk-insertion-sort <))
-> (val sort> (mk-insertion-sort >))
-> (sort< '(6 9 1 7 4 3 8 5 2 10))
(1 2 3 4 5 6 7 8 9 10)
-> (sort> '(6 9 1 7 4 3 8 5 2 10))
(10 9 8 7 6 5 4 3 2 1)
```

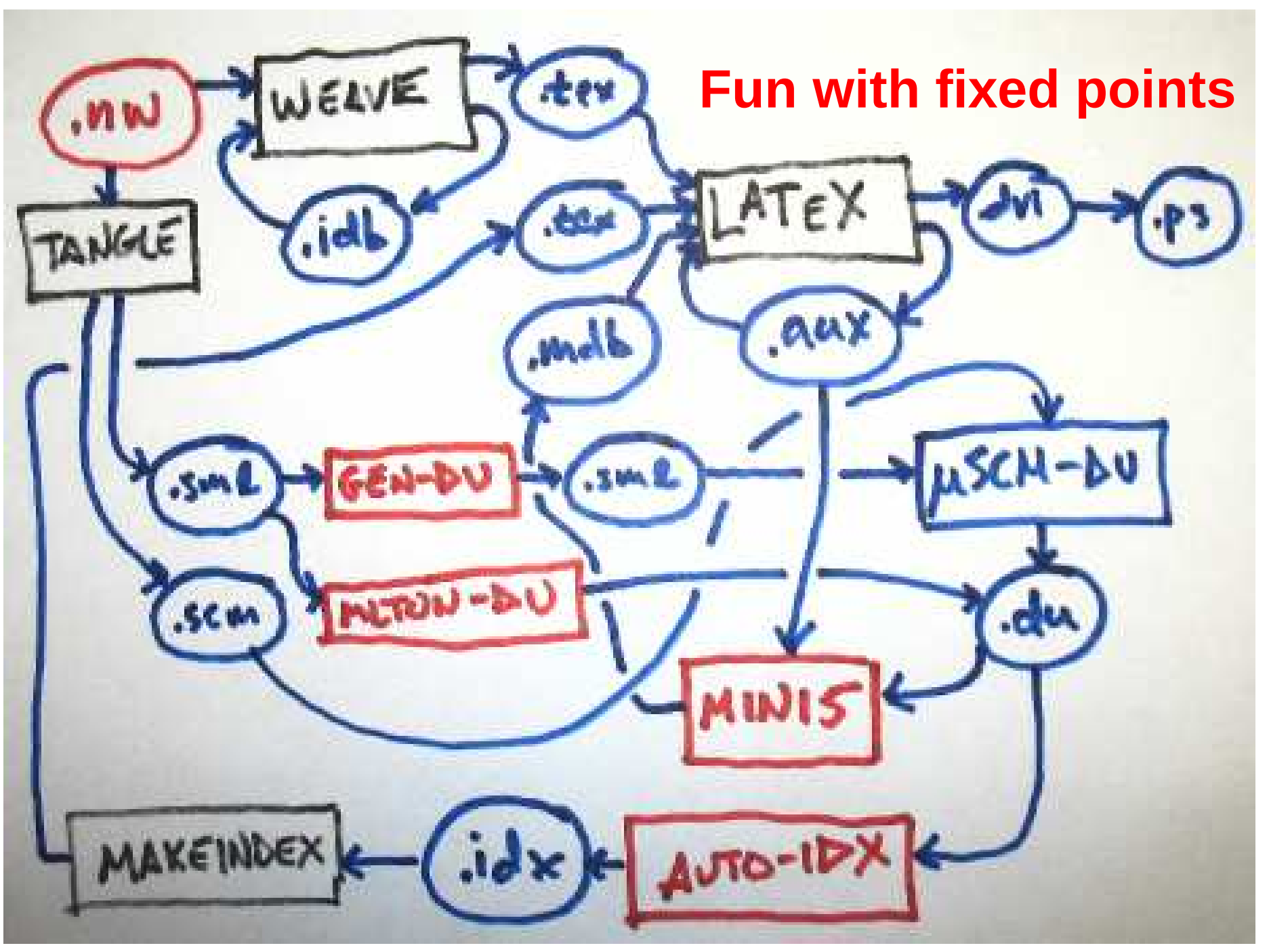
We can also use `mk-insertion-sort` to sort pairs of integers lexicographically.

93c ⟨*transcript* 66⟩+≡ ◁93b 94b▷

```
-> (define pair< (p1 p2)
     (or (< (car p1) (car p2))
         (and (= (car p1) (car p2))
              (< (cadr p1) (cadr p2)))))
-> ((mk-insertion-sort pair<) '((4 5) (2 9) (3 3) (8 1) (2 7)))
((2 7) (2 9) (3 3) (4 5) (8 1))
```


3.10 Practice V: Continuation-passing style

Our implementation of association lists suffers from a problem with `find`: we cannot distinguish between a key that is bound to *nil* and a key that is not bound at all. We could solve this problem by returning a pair: one element telling whether found and another telling what the answer is. But this interface would be awkward, and using it would force us to test results both inside `find` and in clients of `find`.


Fun with fixed points
.nw
WEAVE
.tex
.idb
TANGLE
.scn
LATEX
dvi
.ps
.mdb
.aux
.sml
GEN-DU
.sml
µSCM-DU
.scm
MLTOU-DU
.du
MINIS
MAKEINDEX
.idx
AUTO-IDX

Just plain fun

Real semantics:

- **Makefile as value? What does it denote?**
- **Makefile target as function?**
- **Composition?**
- **Fine-grained memoization?**

I want your advice

What's interesting?

Is there science?

How to get research credit?