Efficient Heuristic Regex Matching

Gábor Kövesdán
<gabor@kovesdan.org>

20 Oct 2012

Introduction - History

- **Summer of Code 2008**: Porting BSD-licensed Text-Processing Tools from OpenBSD
- **July - Aug 2010**: BSD grep committed but has a poor performance

  Conclusion is that libc regex is not efficient enough. Furthermore, POSIX API is inefficient by design. (GNU grep)
- **Summer of Code 2011**: Replacing the old regex implementation

  Porting TRE and applying some improvements. Single pattern cases covered, multiple patterns for future development.
- **Currently**: Individual library on top of TRE. Both single and multiple pattern cases targeted and development in progress.

Trick #1: Longest literal fragment

- `regexec()` takes `const char *`, not byte-counted buffer.
- This implies reading character by character.
- Cannot use efficient algorithms: Quick Search, Boyer-Moore and such can shift more characters in each iteration but they are literal matching algorithms.
- Idea: take longest literal fragment and use that for finding potentially matching lines. Only check the corresponding line.
- Needs alternative interface, e.g. `regnexec()` that takes buffer length.

Trick #2: Prefix heuristics

- **grep** is line-oriented, that is, matches cannot overlap two lines.
- The literal fragment trick can only be applied in the line-oriented case.
- E.g. `[^f]*foo` may contain `\n` in general case and we do not even know how many characters will be caught by `*`.
- But for `foo[^f]*` we can find at least a potential start position. This is another easy case.

Trick #3: Multiple patterns

- Normal approach: reading *n* times for *n* pattern...
- Wu-Manber: Boyer-Moore variant for multiple patterns.
- Idea: extract longest literal fragment from each pattern and approximate with that for line-oriented case.
- More general case: using all literal prefixes if available
- Most general case: even if we cannot use a shortcut, at least hide the iteration logic and return the first match from a single call.

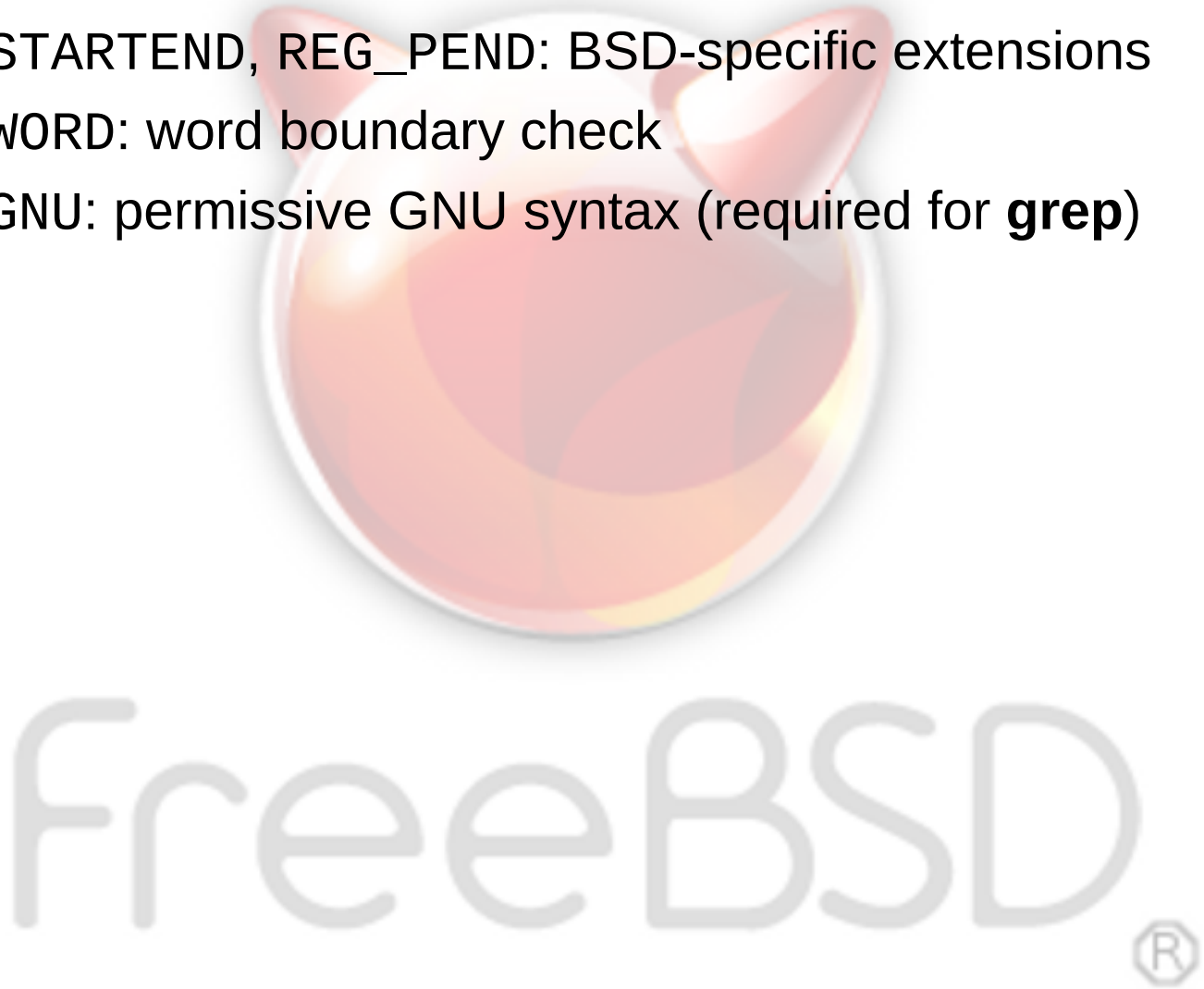
API overview

- `fastreg_regcomp()`, `fastreg_regexec()`, `fastreg_regfree()`: The usual stuff...
- `fastreg_regnexec()`: Can shift quickly, does not need to call `strlen()`
- `fastreg_mregcomp()`, `fastgreg_mregexec()`, `fastreg_mregfree()`: takes different state structure and expects an array of patterns to work with.
- `fastreg_mregnexec()`: also takes an array of pattern lengths.

Published: 20 Oct 2012

Some Extras

- This high level library is a proper place for convenience features.
- `REG_STARTEND`, `REG_PEND`: BSD-specific extensions
- `REG_WORD`: word boundary check
- `REG_GNU`: permissive GNU syntax (required for **grep**)

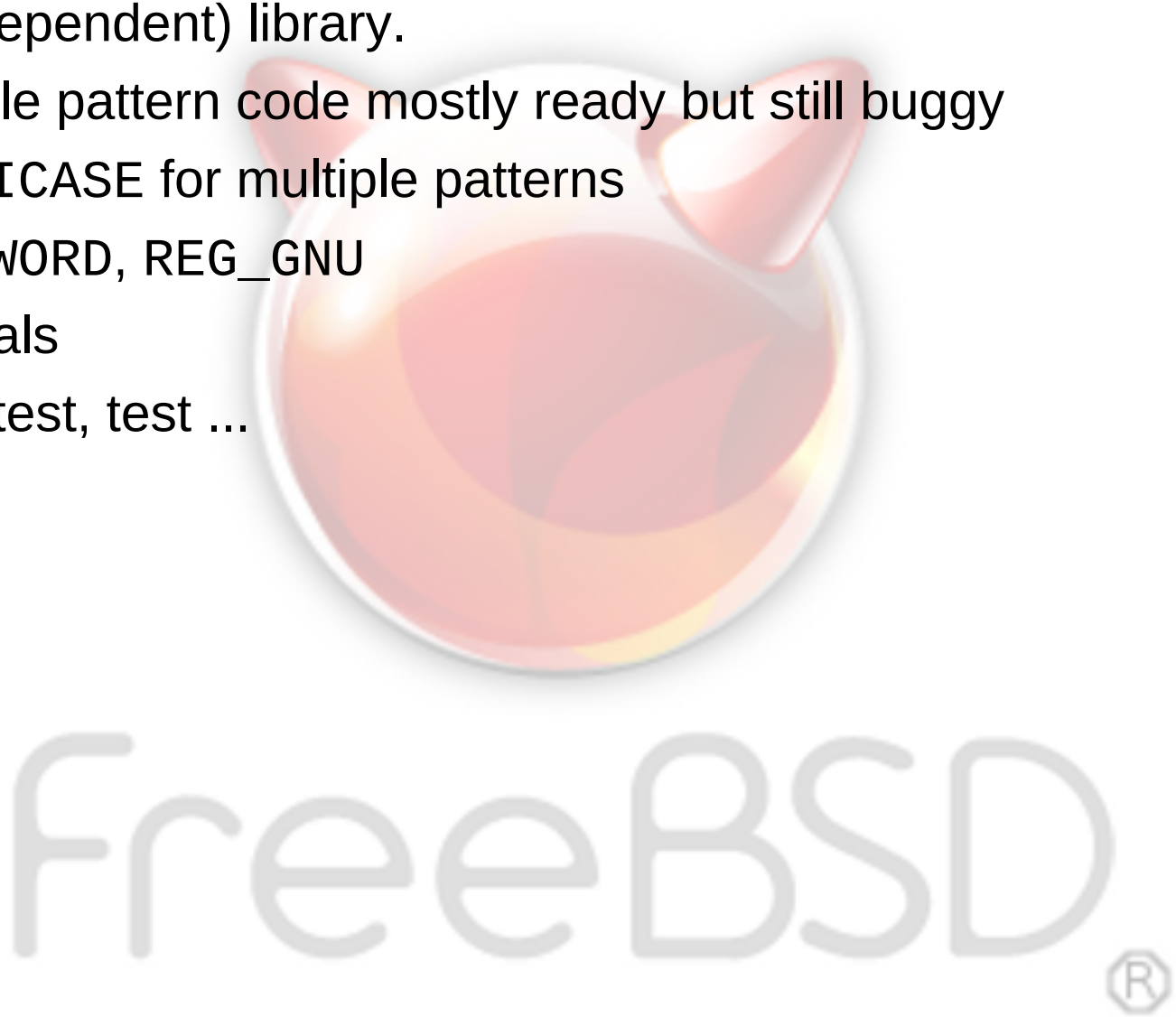


Summary

- Hides complexity.
- Underlying POSIX-compliant implementation can be really simple...
- ... and is easy to replace.
- Further improvements can be added here; let us keep the main implementation simple.

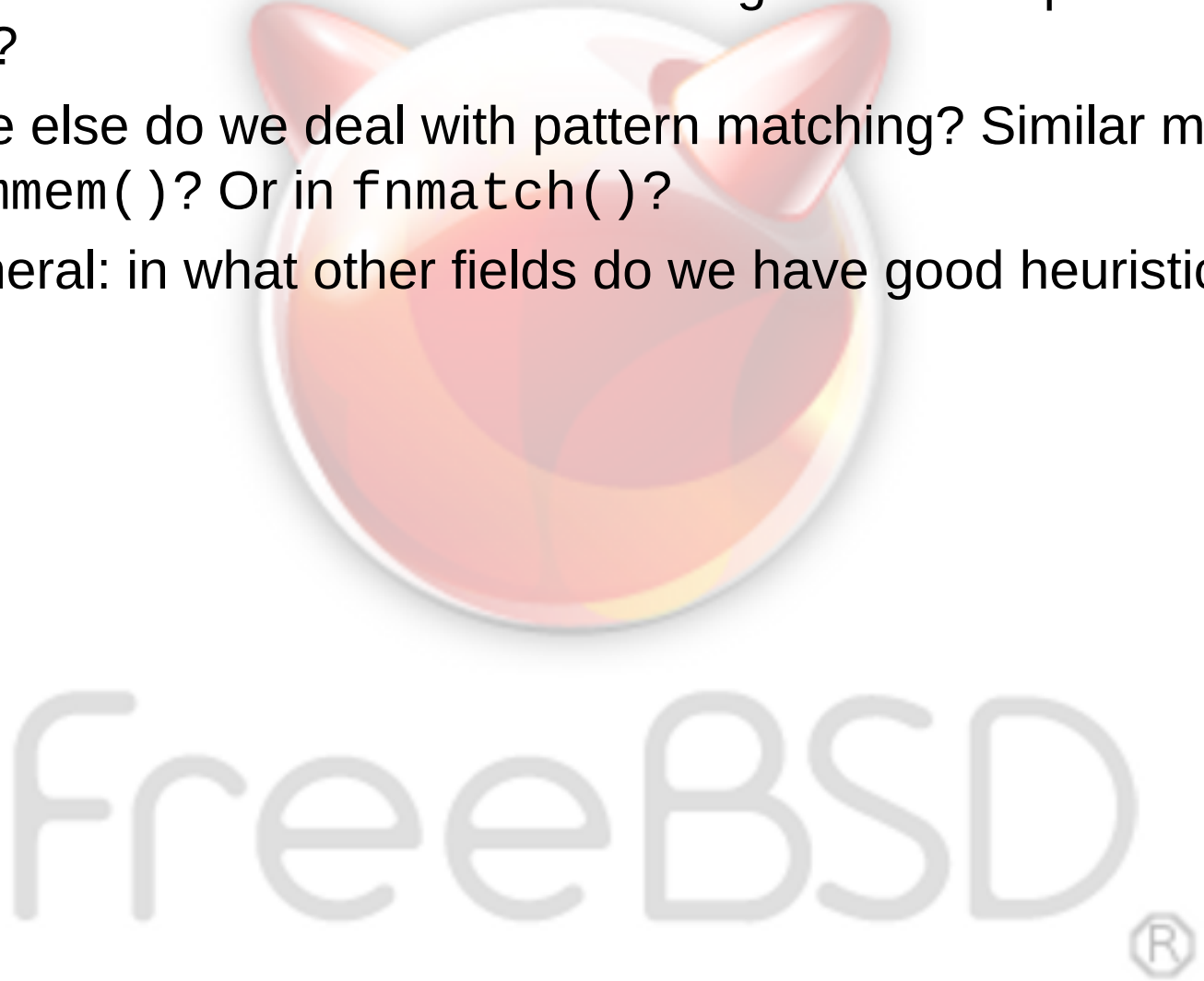
Status

- Started as a TRE extension, being refactored to be a separate (but dependent) library.
- Multiple pattern code mostly ready but still buggy
- `REG_ICASE` for multiple patterns
- `REG_WORD`, `REG_GNU`
- Manuals
- Test, test, test ...



Future Plans

- BSD **grep** will surely use it...
- What about BSD **sed**? Would it be significant the performance boost?
- Where else do we deal with pattern matching? Similar method in `memmem()`? Or in `fnmatch()`?
- In general: in what other fields do we have good heuristics?



Published: 20 Oct 2012

Questions?

Published: 20 Oct 2012

Thanks for listening...

Gábor Kövesdán

gabor@FreeBSD.org

Published: 20 Oct 2012