Hspell's Short Road, from Ideas into Perl Code

Nadav Har'El*

May 11, 2003†

Abstract

Hspell is a free Hebrew linguistic project, whose primary objective was to create a free Hebrew spell-checker. In this paper we shall give a short introduction to the problem of Hebrew spell-checking, highlight our approach, and then focus on one part of the code, the noun inflector, giving examples and explaining why Perl was chosen from day one as the language used by this project. We will show how Perl allowed us very quick prototyping, and how its built-in and powerful string-handling capabilities allowed us to easily transfer our linguistic ideas into code.

1 Introduction

The term *free software* was adopted [12] by Richard Stallman in 1984, for software whose authors decided to share it with the rest of the world, *freely*, as well as to give the users the *freedom* to see the program's source code, modify it, and share the original program or the modified versions with others.

Since then, many people wrote more and more free software. Perl, the topic of this conference is one example of free software. A major boost to free software came in 1991 when Linus Torvalds wrote the first complete free kernel, Linux [10]. Later came "Linux distributions", large collections of free software meant to be easy to install by end users, user-friendly desktop environments (like KDE and Gnome), and software that desktop users have come to expect, like office suites.

However, until recently, typical Israeli users were unable to switch over to free software because of the lack of Hebrew support throughout the system. In June of 1999, the author (Nadav Har'El) expressed the need for better Hebrew support in free software and created the *Ivrix* mailing list [11]. By January 2000, it became clear that while support for Hebrew was planned in the big free software projects (such as Web browsers, desktop environments and word processors), there were a few Hebrew-specific programs that were not likely to

*The work described here was a joint effort with Dan Kenigsberg. For more information, see the project's website, `http://www.ivrix.org.il/projects/spell-checker/`

†first appeared in proceedings of Yet Another Perl Conference, Haifa, May 2003.

miraculously appear. One of these important missing pieces was a Hebrew spell-checker, and so we (Dan Kenigsberg and Nadav Har'El) set out to write one.

Setting our sights on a modest initial goal and choosing Perl for the project's code allowed us to produce a prototype in about a week. This prototype only recognized a tiny minority of the Hebrew language, and did not include a usable front-end, but it proved that our approach was viable and had a lot of potential.

After this prototype, the project was abandoned until we returned to working on it in October of 2002. Less than two months later, the first fully functional release was made, recognizing a useful percentage of the correct Hebrew words. Perl's string-handling power and its simplicity made it easy for us to extend the original prototype, and to spend our time and efforts worrying about linguistic issues, rather than spending them on programming details. Later, when we found existing spell-checking front ends unsuitable for our needs at the time, Perl allowed us to write a different one in a mere few hours.

This article attempts to give an outline of how the Hspell spell-checker works, and a taste of some techniques and Perl code used to implement one part of the whole system, the noun inflector. Section 2 will give a short introduction to our approach for Hebrew spell-checking. Section 3 will explain in a little more detail one important part of the Hspell code, the noun-inflector, written (as was the rest of the code) in Perl.

Understanding section 3 and the examples it contain requires familiarity with Hebrew. Familiarity with Perl is also assumed for the Perl code examples.

2 Hspell's approach

2.1 Assumptions

Keeping in mind a few assumptions can help the reader better understand Hspell's approach (please read this a few times before crying out "why doesn't Hspell do ... ?"):

- Hspell is the labor-of-love of two individuals - it is not a commercial enterprise with dozens or employees or a twenty-year magnum opus. In fact, my original estimate was that Hspell would be done in 2 person-months. This estimate has proved to be somewhat optimistic, but even if the final version will have taken 6 months of work, that is nothing compared to the 100 person-years that the Rav-Millim project boasts on its web-site [7], for example.
- Hspell's *raison d'être* is being a spell-checker for niqqud-less text. While we have ideas on how to later extend this project, such as to analyze syntax, understand and add niqqud, and a full dictionary (definitions or translations), these will not be done in this phase of the project.

  A spell-checker checks words one at a time, without trying to understand context. This brings about, and even more so in Hebrew than in languages like English, false negatives, i.e., misspellings which are recognized as a valid word. But this will be true of every Hebrew spell-checker that does not attempt to understand context.
- For Hspell to be *free* of other people's copyright restrictions, it is a clean-room implementation, not based on other companies' word lists, on other companies' spell checkers, or on copying of printed dictionaries.

However, we did use books like [1, 5, 6, 4] to help us design our algorithms, dictionaries like [7, 3, 2] were used to verify certain words, and various Hebrew newspapers and books, both printed and online, were used for inspiration and for finding words which we still had not recognized.

2.2 Hspell's basic structure

The most fundamental design decision behind Hspell is that it is *word-list based*. This means that the Hspell spellchecker front-end is a rather dumb spell-checker, which basically only needs to look up words in a list to see if they are valid. Very few, if any[1] , linguistic rules need to be built into the spell-checker, and instead all the "brains" go into the *word-list generators*, the programs which generate the lists of valid words. Compare this to another possible design, where a smart spell-checking algorithm looks at a given word, and checks if it can be correctly derived from a known base-word.

Section 3 will discuss in greater depth the word-list generators, written in Perl. Our current `hspell` front-end (also written in Perl) is unfortunately beyond the scope of this article.

This basic structure is one of the most controversial aspects of Hspell's design, but it was chosen for good reason. Firstly, once a word-list is available, it could be used by existing spell-checking applications like `ispell`[8] and `aspell`[9]. Secondly, we found *forward* word derivation, finding for a given word its inflections (נטיות), easier to program than *backward* derivation (given an inflection, trying to infer which rules could have been used to built it). Forward derivation is also very easy to split into cases (different sections or the code, or even separate programs, inflect different types of words), creating simpler and more readable code.

One of the downsides of a wordlist-based spell-checker is relatively-high memory use[2] for holding the huge lists. This is why in the future, Hspell will most likely use an interesting cross of the *word list* and the *backward derivation* approaches: The first step will be building a word list, but the second step will be *affix-compressing* it, automatically recognizing base words which accept common sets of prefixes and and suffixes. The nice thing is that this approach keeps all of the word-list approach's benefits. In particular, all of the linguistic information and algorithms are kept in the word-list generators. Also, existing spell checkers like `ispell`[3] can already deal with affix-compressed word-lists efficiently.

2.3 Hspell's spelling standard

As already mentioned, at this stage Hspell works only on niqqud-less texts. Hspell was designed to be *strictly* compliant with the official niqqud-less spelling rules ("הכתיב חסר הניקוד", colloquially known as "כתיב מלא"), published by the Academy of the Hebrew Language. This is both an advantage and a disadvantage, depending on the user's viewpoint. It's an advantage because it encourages a correct and consistent spelling style throughout the user's

[1] the issue of particle prefixes will be discussed later

[2] disk usage of the word list was never an issue because of a compression technique we use that is beyond the scope of this article. A 350,000 word list fits in about 90K - only about 2 bits per word!

[3] `MySpell`, the spellchecker used by OpenOffice.org and Mozilla, is based on `ispell` and thus also supports affix-compression. Affix-compression support in `aspell` does not yet exist, but is planned.

writing. It is a disadvantage, because a few of the Academia's official spelling decisions are relatively unknown to the general public. Future versions of Hspell might include an option for alternative spelling standards, and the design of the word-list generators makes this relatively easy to do.

3 The word-list generators

3.1 Introduction

A list of valid Hebrew words cannot be built solely by collecting the list of words in available Hebrew documents, because there is no way to guarantee that such a list will be correct (not contain misspellings, useless proper names, slang, and so on), complete (certain inflections might not appear in the chosen samples) or consistent in its spelling standard.

Instead our idea was to build (manually, using online texts for ideas and dictionaries for verification) a list of *base words*, which are automatically inflected by a program, a *word-list generator* which will generate all the valid inflections.

For example, the following input line says that כלב (dog) is a noun:

כלב ע

In this specific case, no further hints are needed for correct inflection (this is discussed below), and the word-list generator will output all the valid inflections:

כלב כלב- כלבי כלבנו כלבך כלבך כלבכם כלבכן כלבו כלבה כלבן כלבם
כלבים כלבי- כלביי כלבינו כלביך כלבייך כלביכם כלביכן כלביו כלביה כלביהן כלביהם

It should be noted that the generated word list does not include the various forms of the word with a *particle*. Particles are prefixes formed from the single Hebrew letters משה וכל״ב that function as prepositions, conjunctions or articles. For example, כלבינו (our dogs) will be found in the list, but וכשכלבינו (and when our dogs) will not.

This decision was made for a practical reason, as it allows the generated word list to be an order of magnitude smaller than the list with all possible combinations of particles. Instead of the the word list containing all the possible words with all the possible particles, the `hspell` front end tries to remove possible particle prefixes from the words it checks. In the first release, `hspell` always allowed every prefix for every word, but this is gradually being improved; For example, while the definite article ה makes sense on a noun, it does not on the imperative form of a verb. Letting the front-end spell checker know which word accepts which set of prefixes will in fact be the first stage towards *affix compression* mentioned earlier.

Hspell now uses two separate word-list generators, both written in Perl: one that inflects nouns and adjectives, and one that inflects verbs, and we will go into a little detail on the first below. These inflectors can also feed on one another (e.g., the verb inflector generates gerunds, which are nouns and can be further inflected as such), and to the generated word list we later also add a list of *extra words*, like various prepositions, proper names, numbers, acronyms, and other miscellaneous words that cannot be inflected.

Despite their name, the word-list generators are capable of much more than just creating a list of valid Hebrew words, and are useful not only for spell-checking. They can write detailed output explaining exactly how each inflection was created, and this output could be used for everything from finding the reasonable set of particle prefixes that a word accepts, to a full morphological analysis of given words.

In the rest of this section we will explain, in a little more details, how noun inflection works. Unfortunately, it is beyond the scope of this article to cover the complete details, and we will also not be able to cover adjective or verb inflection at all.

3.2 Noun inflection

The noun and adjective inflector is a Perl program called `wolig.pl`[4] that works on an input file listing base words (nouns and adjectives), and possible hints on how to inflect them.

It is obvious that `wolig` does need hints for correct inflections. The easiest hints to understand the need for are pluralization hints: how could `wolig` possibly know that while the plural of קוף is קופים, the plural of עוף is עופות? Or that the pair שירותים,שירות is different from חירויות,חירות? It can't. To get these words correctly inflected the input file should contain the lines:

```
עוף ע,ות
שירות ע,ים
```

The hint (ע) tells `wolig` that this a noun (not an adjective), and the ות or ים tells it which plural form is appropriate for that word. Other supported pluralization types:

```
משנה ע,יות
קצבה ע,אות
גרב ע,יים
עשן ע,יחיד
בת ע,רבים=בנות
```

Where יים refers to the pair-plural (which in niqqud-less spelling will indeed look like יים), and יחיד says there is no plural form for this word. In the last example, the word has a completely irregular plural, so it is assigned explicitly, and the rest of the plural inflections will be automatically derived from it. Some words have more than one valid pluralization:

```
חודש ע,ים,יים
שעה ע,ות,יים
קבר ע,ות,ים
איש ע,ים,רבים=אנשים
שפה ע,ות,יים,רבים=שפתות
```

This hint format, a flexible and human-readable comma-separated list of flags and assignments, made it easy for us to gradually add more flags and to more easily write (and later read) our input file. In Perl, understanding this sort of hint list is easy, with code like this splitting a single line into the base word, `$word` and an associative array `%opt`:

[4] originally meaning "WOrdLIst Generator"

```
($word,$optstring)=split;
undef %opts;
foreach $opt (split /,/o, $optstring){
  ($opt, $val) = (split /=/o, $opt);
  $val = 1 unless defined $val;
  $opts{$opt}=$val;
}
```

And later in the code the hints about the current word are queried with statements like `if($opts{"תו"})` or `my $plural=$opts{"רבים"}`

While `wolig` needs hints to inflect some nouns correctly, about 90 percent of the nouns do not need any hint other than the "ע" saying this is a noun. This is because most of the time in Hebrew the plural form can be guessed just by looking at the last letter (or letters) of the singular. For example:

מלך	מלכ**ים**
מלכ**ה**	מלכ**ות**
כותר**ת**	כותר**ות**
כמ**ות**	כמו**יות**
אחר**יות**	אין צורת רבים

The choice of plural form is not the only type of hint that might be needed, unfortunately. Hebrew has many complications when inflecting nouns with vowels, like vowels being changed or disappearing completely, and a few of these complications remain also for niqqud-less spelling[5]. Other words have irregularities in some of their inflections. It is beyond the scope of this article to get into all these problems and cover all the cases, so instead we will just show some examples without justification (each input line is followed by the list of inflections generated from it):

```
חנית ע,שמור__ת
  חנית חנית– חניתי חניתנו חניתך חניתך חניתכם חניתכן חניתו חניתה חניתן חניתם
  חניתות חניתות– חניתותיי חניתותינו חניתותיך חניתותייך חניתותיכם חניתותיכן חניתותיו
  חניתותיה חניתותיהן חניתותיהם
כותל ע,אבד__ו
  כותל כותל– כותלי כותלנו כותלך כותלך כותלכם כותלכן כותלו כותלה כותלן כותלם
  כתלים כותלי– כתליי כתלינו כתליך כתלייך כותליכם כותליכן כתליו כתליה כותליהן כותליהם
עיפרון ע,ות,אבד__י
  עיפרון עפרון– עפרוני עפרוננו עפרונך עפרונך עפרונכם עפרונכן עפרונו עפרונה עפרונן עפרונם
  עפרונות עפרונות– עפרונותיי עפרונותינו עפרונותיך עפרונותייך עפרונותיכם עפרונותיכן
  עפרונותיו עפרונותיה עפרונותיהן עפרונותיהם
רובה ע,ים,סגול__ה
  רובה רובה– רובהו רובי רובנו רובך רובך רובכם רובכן רובו רובה רובן רובם
  רובים רובי– רוביי רובינו רוביך רובייך רוביכם רוביכן רוביו רוביה רוביהן רוביהם
אח ע,מיוחד__אח
  אח אחי– אחי אחינו אחיך אחיך אחיכם אחיכן אחיו אחיה אחיהן אחיהם
```

[5]luckily for us, in some cases the Academia's official niqqud-less spelling rules save us from dealing with certain potential exceptions. For example, the plural of עז (goat) is עזים, not עיזים with an extra yod.

```
  אחים אחי– אחיי אחינו אחיך אחייך אחיכם אחיכן אחיו אחיה אחיהן אחיהם
שן ע,יים,מיוחד__שן
  שן שן– שיני שיננו שינך שינך שינכם שינכן שינו שינה שינן שינם
  שיניים שיני– שיניי שינינו שיניך שינייך שיניכם שיניכן שיניו שיניה שיניהן שיניהם
מגור ע,אין__יחיד
  מגורים מגורי– מגוריי מגורינו מגוריך מגורייך מגוריכם מגוריכן מגוריו מגוריה מגוריהן מגוריהם
אחור ע,אין__נטיות__יחיד,יים
  אחור אחוריים אחורי– אחוריי אחורינו אחוריך אחורייך אחוריכם אחוריכן אחוריו אחוריה אחוריהן אחוריהם
סתו ע,נפרד=סתיו,נסמך=סתיו
  סתיו סתיו– סתווי סתוונו סתווך סתווך סתווכם סתווכן סתוו סתווה סתוון סתוום
  סתווים סתווי– סתוויי סתווינו סתוויך סתווייך סתוויכם סתוויכן סתוויו סתוויה סתוויהן סתוויהם
גברת ע,רבים=גברות,נסמכים=גבירות
  גברת גברת– גברתי גברתנו גברתך גברתך גברתכם גברתכן גברתו גברתה גברתן גברתם
  גברות גבירות– גבירותיי גבירותינו גבירותיך גבירותייך גבירותיכם גבירותיכן גבירותיו גבירותיה
  גבירותיהן גבירותיהם
```

In addition to dealing with, or guessing, the various hints, and generating the correct inflections based on them, the `wolig` program also needs to deal with the correct rules of niqqud-less spelling. For example, the correct construct-state (נסמך) form of קריה is קריית־, because the rules say that a consonant yod is usually doubled, but not next to a vowel letter (in this case, the ה).

The way `wolig` deals with niqqud-less spelling rules is simple, and yet very powerful. The idea is that when outputting each word, `wolig` runs on it a postprocessing function `outword`. In addition to correcting final forms of letters in the middle of the word, `outword` takes special (non-Hebrew) characters, like a "y" signifying a consonant yod, and converts them to the correct Hebrew letters (in this case, a single or double yod) according to the rules. The Academia's spelling rules were directly converted to the form of Perl substitutions, and part of the rule for "y" looks like this:

```
$word =~ s/(?<=[^ויy])y(?=[^הויy]|$)/יי/go;
$word =~ s/y/י/go;                          # otherwise, just one yod.
```

This is just a small part of the rules in `outword` — the full rules deal with with the characters Y, y, h, w, i, e, a, and are heavily commented to explain what they mean, and where these rules come from, so we will not discuss more of them here.

But where do these special characters y, w, etc., come from? The user may use them when entering input base-words to make sure that `wolig` knows which yods and waws are consonants and which are vowels, but in most cases this is unnecessary: the user can input the words in the standard niqqud-less spelling, and a function `inword` is run on these words to try to guess where these special characters are appropriate. To return to the example above, when the user enters the nouns קריה, `inword` changes it into קרyה. Later, the normal inflection algorithm produces the construct-state form קרyת־, and when `outword` is run on that, קריית־ is output, just like we wanted.

`wolig` can also inflect adjectives, marked by the ת flag. These also have an interesting selection of possible hints, which we would not be able to cover here. They are explained in the beginning of the `wolig.dat` input file, and also in the heavily commented code.

4 Lessons learned

Our word-list based approach to Hebrew spell-checking proved viable. Writing the word-list generators in Perl proved an excellent idea — it it allowed us to concentrate more on ideas and techniques, and less on programming details, and its powerful regular expressions and other constructs allowed the code to remain simple and straightforward.

The decision to write the actual front-end spell-checker, `hspell` in Perl, is more questionable. On one hand it allowed us to very quickly (just a few hours for the first version!) create something that worked, and actually worked surprisingly well. On the other hand, the current Perl implementation is a memory hog and very slow to start (reading the word lists into a huge hash table), and so is in the process of being rewritten in C, and using a different search algorithm (radix-tree instead of a hash table). Even this may not be unnecessary, when we write code that does affix-compression on our generated word lists and give it to existing, general-purpose, spell-checkers.

References

[1] עוזי אורנן. *נטיית הפועל בתרשימי זרימה*. אקדמון, תש״ם.

[2] שושנה בהט ומרדכי מישור. *מילון ההווה*. 1995.

[3] אברהם אבן־שושן. *המלון העברי המרכז*. 2000.

[4] דוד ילין. *דקדוק הלשון העברית*. ראובן מס, 1942.

[5] שאול ברקלי. *לוח הפעלים השלם*. ראובן מס, 1953.

[6] שאול ברקלי. *לוח השמות השלם*. ראובן מס, 1960.

[7] Choueka et al. Rav milim. `http://www.ravmilim.co.il/`.

[8] Goeff Kuennig et al. International ispell. `http://fmg-www.cs.ucla.edu/fmg-members/geoff/ispell.html`.

[9] Kevin Atkinson et al. Gnu aspell. `http://aspell.sourceforge.net/`.

[10] Linus Torvalds et al. The Linux kernel. `http://www.kernel.org/`.

[11] Nadav Har'El. Ivrix project — call for participation. `http://www.ivrix.org.il/announcements/1.html`, June 1999.

[12] Richard Stallman. Initial announcement — Gnu project. `http://www.fsf.org/gnu/initial-announcement.html`, January 1984.