Grammatical Framework and Chinese

Appendix to the Chinese edition of the book "Grammatical Framework. Programming with Multilingual Grammars" (by Aarne Ranta, CSLI, Stanford, 2011)

by Aarne Ranta

Preface

One of the most pleasant surprises in my life was in December 2011, when I received a mail from Professor Yan Tian 田艳 of Shanghai Jiao Tong University, proposing to translate the GF book to Chinese. Wow, I thought, this would almost double the potential audience of my book! Less than a year later, I got the opportunity to visit Professor Tian and her colleagues in Shanghai. She then showed me a complete manuscript of the translation, of which I unfortunately understood next to nothing. But we had long sessions discussing the translation of the terminology, which convinced me that the Chinese version was being produced with great care. We also had a tutorial course and technical discussions, testing and assessing the newly released Chinese resource grammar, and looking at future project opportunities. Professor Tian suggested that, in order to make the book genuinely interesting to the Chinese audience, we should add an Appendix that shows what GF means for Chinese. This is what the present document attempts to be. It gives Chinese examples and code as a commentary to the main book, thus following its order of presentation. It is probably hard to read without access to the book, which I apologize.

I am grateful to Professors Yan Tian and Yinglin Wang 王英林, as well as Peng Li 李鹏, for their hospitality in Shanghai, to Professor Jyrki Nummenmaa from Tampere for creating the contacts leading to this collaboration and for joining the activities in Shanghai, to Dr Chen Peng 陈鹏 from Beijing for also joining the discussions in Shanghai and contributing to the Chinese resource grammar, and to Dr Qiao Haiyan 乔海燕 from Guangzhou, who wrote the first Chinese GF code - the Numerals module, which is now used in the resource grammar - back in 1999. The most demanding part of the work with the Chinese resource grammar was made by Zhuo Linqiqige 卓琳其其格 as a part of a Masters course at the University of Gothenburg.

Gothenburg, November 2012

Aarne Ranta 阿尔内•兰塔

To Chapter 1.

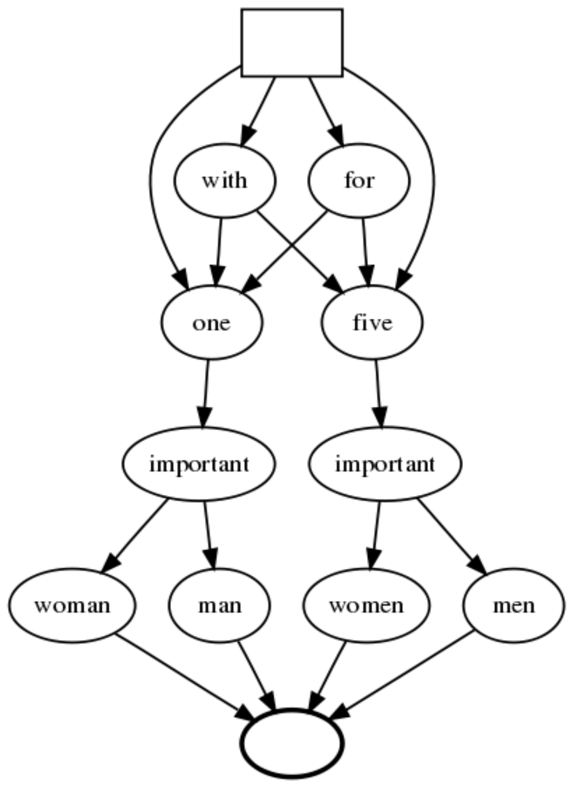
To Section 1.3. An example of long-distance dependencies in Chinese is the use of **classifiers** attached to counting words and determiners. If you are for instance counting people, the classifier is 个 (*ge*), whereas for cats, it is 只 (*zhi*). Thus *five men* is 五个男人 ("five *ge* man") whereas *five cats* is 五只猫 ("five *zhi* cat"); there is no plural inflection of nouns in Chinese. The long-distance dependencies follow from the fact that adjectival attributes are placed between the classifier and the noun. In the following series of noun phrases,

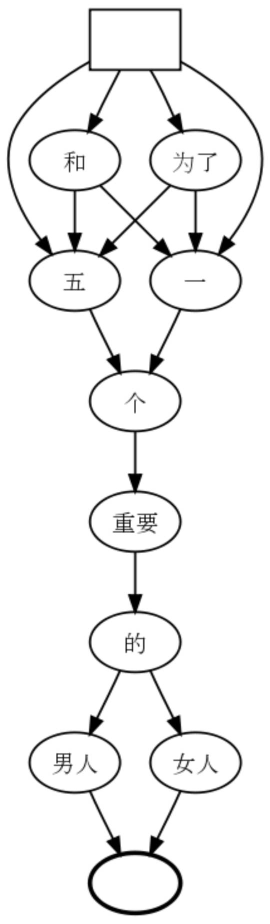
five cats 五只猫
five black cats 五只黑猫
five very small black cats 五只非常小的黑猫

the distance between the classifier and the noun increases, and it becomes increasingly difficult for a statistical model to select the correct classifier.

To Section 1.4. TODO benchmark for Chinese parsing performance. Preliminary experiments suggest that Chinese is about twice as fast to parse as English, which makes it the fastest language in the Resource Grammar Library.

To Section 1.5. Figure 5, p. 15, shows a noun phrase with prepositions in English and German, illustrating differences in morphological variation between languages. Putting the Chinese and English pictures side by side shows the fact that Chinese has no variation in case and number. What makes the Chinese picture higher is the presence of the classifier 个 (*ge*) and the particle 的 (*de*), which is needed for longer than monosyllabic adjectives.





To Chapter 2.

The Chinese food grammar can be correctly defined in the BNF format, and also with a string-based GF grammar. Both grammars are shown below. They are due to Ulysses (TODO full name), an SJTU student who wrote them in a live demo during the GF tutorial in Shanghai.
Here is the BNF grammar, `foodsChi.cf`

```
Pred.       Comment ::= Item "是" Quality
This.       Item    ::= "这个" Kind
That.       Item    ::= "那个" Kind
Mod.        Kind    ::= Quality Kind
Wine.       Kind    ::= "酒"
Cheese.     Kind    ::= "奶酪"
Fish.       Kind    ::= "鱼"
Very.       Quality ::= "非常" Quality
```

```
Fresh.     Quality ::= "新鲜的"
Warm.      Quality ::= "温热的"
Italian.   Quality ::= "意大利式的"
Expensive. Quality ::= "昂贵的"
Delicious. Quality ::= "美味的"
Boring.    Quality ::= "难吃的"
```

The GF grammar is a straightforward variant:

```
concrete FoodChi of Food = {
flags coding = utf8 ;
lincat
    Comment, Item, Kind, Quality = Str ;
lin
    Pred item quality = item ++ "是" ++ quality ;
    This kind = "这 个" ++ kind ;
    That kind = "那 个" ++ kind ;
    Mod quality kind = quality ++ kind ;
    Wine  = "酒" ;
    Cheese  = "奶 酪" ;
    Fish  = "鱼" ;
    Very quality = "非 常" ++ quality ;
    Fresh  = "新 鲜 的" ;
    Warm  = "温 热 的" ;
    Italian  = "意 大 利 式 的" ;
    Expensive  = "昂 贵 的" ;
    Delicious  = "美 味 的" ;
    Boring  = "难 吃 的" ;
}
```

To properly render the food comments in Chinese involves a couple of hacks, which might not work in a full-scale grammar such as the resource grammar. First, the classifier 个 (*ge*) happens to work for all nouns in this grammar, and can hence be safely attached to the determiners in this grammar. Secondly, the particle 的 (*de*) is here attached to the adjectives, which are polysyllabic. In the resource grammar (Chapter 9), both elements are controlled by rules sensitive to the syntactic context.

To Section 2.9. Chinese is known for having reduplication. For instance, adjectives can be reduplicated to weaken their effect: 小小的 (*xiǎo xiǎo de* “small small *de*”, “a bit small”). Another pattern is AABB to intensify two-syllable adjectives of the form AB. An example is 漂亮 (*piao liang*, “beautiful”), giving 漂漂亮亮 (*piao piao liang liang,* “very beautiful”). The latter kind of reduplication is not yet possible with string-based GF but also requires the use of discontinuous constituents, to be introduced in Chapter 3.

Exercise. Extend the Food grammar with a category of one-syllable adjectives that can be used in the category `Quality` either alone or as reduplicated. The reduplication is somewhat similar to the `Very` constructor, but it cannot be iterated many times. Notice that the reduplication makes the adjective two-syllabic, which means that the particle 的 (*de*) must be appended. Define some adjectives in this class and give the corresponding English rules, saying for instance "this fish is a bit small".

To Section 2.15. Lexing is a notorious problem in Chinese, because words are written together without spaces in between. As the grammar examples show, a word can have one, two, three, or up to five characters, and words are mostly much shorter than in English in terms of characters. In the BNF grammar, each word is a sequence of characters without spaces in between, so that for instance *this expensive wine is delicious* comes out as

这个   昂贵的   酒   是   美味的

(replacing each space by three to make them clearer). The normal orthography is

这个昂贵的酒是美味的

This is easy to produce by the unlexer `-unchars`. But to restore the word boundaries in parsing is more difficult - in fact, it would require a lexicon and a parser, and still leave room for different choices. In the GF grammar above, we have solved the problem by introducing spaces *everywhere*, that is, even inside the words. Thus the sentence comes out as

这 个 昂 贵 的 酒 是 美 味 的

The word segmentation problem now gets a solution as a *parsing* problem, because each character is a distinct token. To parse an unspaced string, you can use the GF command

```
> put_string -chars "这个昂贵的酒是美味的" | parse
```

To Section 2.16. We have used Chinese characters directly in all grammars, with UTF-8 encoding. Creating a transliteration for over 8,000 characters would not be easy. For instance, the use of the phonetic Pinyin system would not do, because many Pinyin words - even if the tone is present - map to several characters. Chinese characters are nowadays very well supported by editors, shells, and web browsers, so this is not an issue. We have used simplified Chinese throughout the work.

To Chapter 3

Morphology in Chinese is non-existent, whereas classifiers and question formation as nice examples of discontinuous constituents, which do require records. Also some adjective and adverb constructions require parameters, as will be better shown in Chapter 9.

Incidentally, the `FoodsChi.gf` grammar *could* be written by just adding some rules to `FoodChi.gf`, without changing the linearization types. However, we can make it more structured and scalable by separating the classifiers from nouns, and the 的 (*de*) particle from adjectives.

Thus we use a record with two strings for `Kind`, with a string s for the noun and c for its classifier:

```
lincat Kind = {s,c : Str} ;     -- c is the classifier
```

Similarly, we use a record with two strings for `Quality`, with a string s for the noun and d for the particle, which can be empty:

```
lincat Quality = {s,p : Str} ; -- p is the particle 的 de, or empty
```

The common cases with classifier 个 (*ge*) and long adjectives with 的 (*de*) have designated oper's:

```
oper
  geKind : Str -> {s,c : Str} = \s -> {s = s ; c = "个"} ;
  longQuality : Str -> {s,p : Str} = \s -> {s = s ; p = "的"} ;
```

The predication and modification rules have to put the particle in the right place. Modification also has to inherit the classifier of the base noun:

```
lin
  Pred item quality = item ++ "是" ++ quality.s ++ quality.p ;
  Mod quality kind = {s = quality.s ++ quality.p ++ kind.s ; c = kind.c}
;
```

Classifiers are used in singular determiners, while plural determiners omit them:

```
    This kind = "这" ++ kind.c ++ kind.s ;
    These kind = "这" ++ "些" ++ kind.s ;
```

Here is the complete grammar written in the way just specified:

```
concrete FoodsChi of Foods = {
flags coding = utf8 ;
lincat
    Comment, Item = Str ;
    Kind = {s,c : Str} ;     -- c is the classifier
    Quality = {s,p : Str} ; -- p is the particle 的 de, or empty
lin
    Pred item quality = item ++ "是" ++ quality.s ++ quality.p ;
    This kind = "这" ++ kind.c ++ kind.s ;
    That kind = "那" ++ kind.c ++ kind.s ;
    These kind = "这" ++ "些" ++ kind.s ;
    Those kind = "那" ++ "些" ++ kind.s ;
    Mod quality kind = {
      s = quality.s ++ quality.p ++ kind.s ;
      c = kind.c
      } ;
    Wine  = geKind "酒" ;
    Pizza = geKind "比 萨 饼" ;
    Cheese  = geKind "奶 酪" ;
    Fish  = geKind "鱼" ;
    Very quality = longQuality ("非 常" ++ quality.s) ;
    Fresh  = longQuality "新 鲜" ;
    Warm  = longQuality "温 热" ;
    Italian  = longQuality "意 大 利 式" ;
    Expensive  = longQuality "昂 贵" ;
    Delicious  = longQuality "美 味" ;
    Boring  = longQuality "难 吃" ;
oper
    geKind : Str -> {s,c : Str} = \s ->
      {s = s ; c = "个"} ;
    longQuality : Str -> {s,p : Str} = \s ->
      {s = s ; p = "的"} ;
}
```

Exercise. Add some nouns with other classifiers than *ge*, as well as one-syllable adjectives, to the grammar. Implement them for both English and Chinese, and make sure the translations come out correct.

To Chapter 4

To Section 4.4. With pattern-matching on strings, it is easy to decide whether an adjective has one or more syllables. Assuming the linearization type of the FoodsChi grammar (previous Chapter), we can define the following “smart paradigm”:

```
    oper mkAdj : Str -> {s,p : Str} = \s -> case s of {
      ? => {s = word s ; p = []} ;
      _ => {s = word s ; p = "的"}
      } ;
```

As another example of string matching, here is a helper function that enables the grammarian to postpone the decision on whether to put spaces inside words:

```
oper
  bword : Str -> Str -> Str = \x,y -> x ++ y ;
  -- change to x + y to treat words as single tokens

  word : Str -> Str = \s -> case s of {
      x@? + y@? + z@? + u@? => bword x (bword y (bword z u)) ;
      x@? + y@? + z@? => bword x (bword y z) ;
      x@? + y@? => bword x y ;
      _ => s
      } ;
```

The function `word` thus converts a string of characters (up to length 4) to a string of tokens. But this behaviour can be changed by just changing the operator ++ in `bword` to +. We have used this to produce a Pinyin version of the grammar, where syllables inside words are written without spaces.

To Section 4.7. Overloaded paradigms use the one-argument function to select the most common inflection patterns. Even though Chinese doesn't have inflection, the selection of the classifier is a similar task, where 个 is the most common choice. Thus `mkN` looks as follows:

```
oper mkN = overload {
    mkN : (man : Str) -> N        = \n -> lin N (mkNoun n "个") ;
    mkN : (man : Str) -> Str -> N = \n,c -> lin N (mkNoun n c)
    } ;
```

To Chapter 5

The Chinese resource grammar API is of course the same as in the other languages. The lexicon instance and functor instantiation thus work as expected:

```
--# -path=.:alltenses

concrete FoodsRChi of Foods = FoodsI with
  (Syntax = SyntaxChi),
  (LexFoods = LexFoodsChi) ;


instance LexFoodsChi of LexFoods =
    open SyntaxChi, ParadigmsChi in {
  flags coding = utf8 ;
  oper
    wine_N  = mkN "酒" ;
    pizza_N = mkN "比萨饼" ;
    cheese_N = mkN "奶酪" ;
    fish_N = mkN "鱼" ;
    fresh_A = mkA "新鲜" ;
    warm_A = mkA "温热" ;
    italian_A = mkA "意大利式" ;
    expensive_A = mkA "昂贵" ;
    delicious_A = mkA "美味" ;
    boring_A = mkA "难吃" ;
}
```

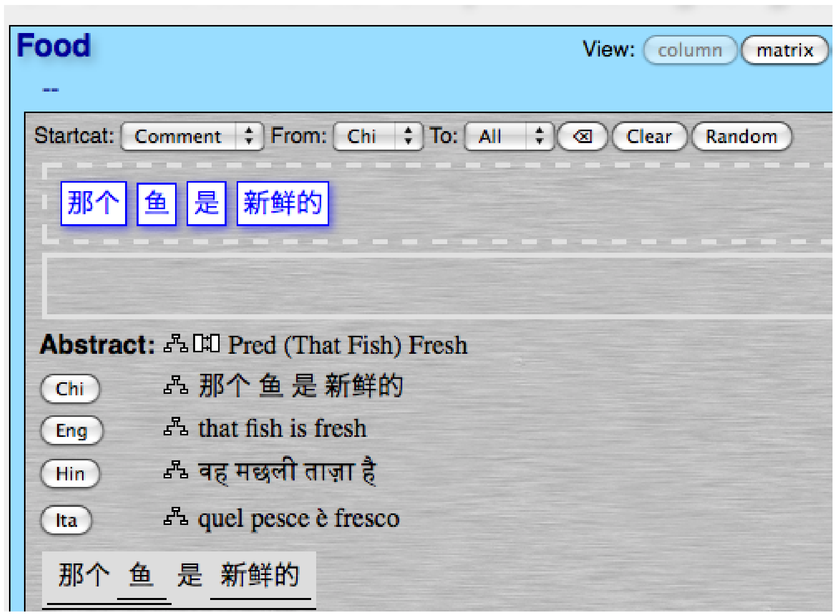
To Section 5.17. Notice that we use `alltenses` in the path for Chinese. The reason is that there is no `present` version of the Chinese grammar, and there is very little need for it. However, Chinese has a notion of **aspect**, which is expressed by different particles. A table similar to Figure 44 shows the aspect (and polarity) variation of the Chinese clause, according to the resource grammar:

```
> i alltenses/LangEng.gfo alltenses/LangChi.gfo
> p -lang=Eng -cat=Cl “I sleep” | l -lang=Chi -table -unchars
Pos APlain   : 我睡
Pos APerf    : 我睡了
Pos ADurStat : 我睡着
Pos ADurProg : 我在睡
Pos AExper   : 我睡过
Neg APlain   : 我没睡
Neg APerf    : 我不睡了
Neg ADurStat : 我不睡
Neg ADurProg : 我没在睡
Neg AExper   : 我没睡过
```

More about aspects will be said in Chapter 10.

To Chapter 7

The next picture is a Chinese version of Figure 52, showing an editing session with the Chinese Food grammar.



那个 鱼 是 新鲜的

Abstract: Pred (That Fish) Fresh

Chi 那个 鱼 是 新鲜的

Eng that fish is fresh

Hin वह मछली ताज़ा है

Ita quel pesce è fresco

那个 鱼 是 新鲜的

To Chapter 9

The Chinese miniature resource grammar originates in the work of Zhuo Linqiqige. She built a version extended with adverbs, questions, and aspects, which required more complex linearization types than the ones shown here. The complete code of the miniature grammar is given in the Appendix A section below.

To Section 9.3. Here are the linearization types of the key phrasal and lexical categories in Chinese, as needed in the miniature resource grammar.

```
  lincat
    Cl  = {s : Bool => Str} ;
    NP  = {s : Str} ;
    VP  = {verb : Verb ; compl : Str} ;
    AP  = {s : Str ; monoSyl : Bool} ;
    CN  = {s : Str ; c : Str} ;
    Det = {s : Str ; n : Number} ;

    N   = {s : Str ; c : Str} ;
    A   = {s : Str ; monoSyl : Bool} ;
    V   = {s : Str} ;
    V2  = {s : Str} ;
```

Very few features are used: in fact, even the `monoSyl` feature of adjective could be replaced as a discontinuous particle, as we did in the Foods grammar in Chapter 3. The classifier field, c, of common nouns could in principle be replaced by a parameter, but this parameter would have hundreds of values. Determiners have a number feature, which is used for selecting the classifier (see below).

To Section 9.4. Here is the linearization rule for predication:

```
  lin PredVP np vp = {s = \\p => np.s ++ neg p ++ vp.verb.s ++ vp.compl} ;
```

It defines the order subject-negation-verb-complement. There is no agreement between the subject and the verb. The negation is put in place with the operation `neg`, which returns `"不"` for the negative polarity. In the full resource grammar, clauses in Chinese also depend on aspect, the negation word depends on the verb and the aspect, and verb phrases have more fields, e.g. for adverbs.

To Section 9.5. Here is the linearization rule for complementation:

```
  lin ComplV2 v2 np = {verb = v2 ; compl = np.s} ;
```

The reason to keep VP discontinuous at all is that material is sometimes attached to the position before the complement. But this is only exploited in the full resource grammar.

To Section 9.6. Here is the linearization rule for determination:

```
    lin DetCN det cn = case det.n of {
            Sg => {s = det.s ++ cn.c ++ cn.s} ;
            Pl => {s = det.s ++ "些" ++ cn.s}
      } ;
```

Thus the inherent number of the determiner is needed to decide whether the inherent classifier of the noun is overridden.

To Section 9.7. Here is the linearization rule for modification:

```
    lin ModCN ap cn = case ap.monoSyl of {
            True => {s = ap.s ++ cn.s ; c = cn.c} ;
            False => {s = ap.s ++ "的" ++ cn.s ; c = cn.c}
            } ;
```

In Section 3, we saw that a particle field would actually be nicer to use than the `monoSyl` parameter, since it doesn’t require case analysis.

To Section 9.8. Lexical insertion is done with simple identities, except for verbs, where the complement field is initialized as the empty string:

```
  lin UseV v = {verb = v ; compl = []} ;
```

To Section 9.15. Coordination in Chinese is interesting, because the words for *and* and *or* depend on what kind of phrases are combined. This can be regulated by the following parameter type:

```
  param
    SForm = Phr PosType | Sent;
    PosType = APhrase | NPhrase | VPhrase ;
```

The type SForm is hierarchical, to reflect the fact that *and* has more variation than *or*:

```
  lin
    and_Conj = {s = table {
                    Phr NPhrase => "和" ;
                    Phr APhrase => "而" ;
                    Phr VPhrase => "又" ;
                    Sent =>  []
                    }
                } ;
    or_Conj  = {s = table {
                    Phr _ => "或" ;
                    Sent => word "还是"
                    }
                } ;
```

The coordination rules select relevant values of the SForm parameter:

```
  lin
    ConjNP co x y = {s = x.s ++ co.s ! Phr NPhrase ++ y.s} ;
    ConjS  co x y = {s = x.s ++ co.s ! Sent ++ y.s} ;
```

The other values come out usable in the full-scale resource grammar.

To Chapter 10

To Section 10.2. After the completion of the book, the Resource Grammar Library has grown to 26 languages, of which Chinese is the latest one. It is not the first East-Asian language, but Thai and Japanese were written before it. The Thai grammar was used for bootstrapping Chinese. The languages share some important features - the absence of inflection and the use of classifiers - while they differ quite a lot in word order.

Here is a summary of the code size (in lines of GF code) and effort (in person months) for Chinese and some other new languages, computed in the same way as in Figure 77 on p. 223. The "morpho" part includes the definitions of lexical categories and lexicon building functions. In the table, it comprises just Paradigms modules, as well as Morpho and Irreg (in case of Dutch). Parts of Res could also be counted there, in particular in Japanese. The English figures are from the book.

language	syntax	morpho	lexicon	total	months	start
Chinese	966	118	562	1646	1	2012
Dutch	1846	1024	487	3357	2	2009
English	1025	772	506	2303	6	2001
Japanese	3435	65	470	3970	5	2011
Thai	1877	95	473	2445	2	2007

A general trend in resource grammar development is that the required effort goes down. This is due to the stability of the abstract syntax (where the English effort includes coping with several changes) and to the availability of closely related grammars to bootstrap from.

To Section 10.3. The Chinese grammar was bootstrapped from Thai, following closely the recommended workflow (steps 1-8 on p. 224), except for Step 7. Thus we did not comment out the contents of the modules, but started with a Thai grammar that was just called Chinese. The next step was to comment out the contents of the lexicon file contents (LexiconChi and StructuralChi) and to replace all in-lined strings in rules (such as the copula) with either their Chinese equivalents (if easily found in grammar books or in the mini resource) or a default dummy string.

In the next stage, the workflow was lexicon-driven. We first populated the Swadesh list words with the Chinese equivalents found in the Wikipedia. Then we used various other freely available word lists to find more words. Lexicon extraction was easy for content words, because we only needed to know the part of speech - with the exception of the classifier of nouns, which were all initialized to 个 (*ge*).

With an almost complete lexicon, it was possible to generate Chinese sentences and assess their correctness. At this point, it became obvious that the Chinese word order is radically different from Thai. Even though clauses follow the SVO pattern in both languages, the order inside noun phrases is completely different, and so is the placement of adverbs.

At this point, it was appropriate to change to the syntax-driven work flow and follow a miniature resource implementation, extended with some categories and functions. This implementation was work by a Chinese student in connection with a course. It had taken her three weeks to write, while the Thai-based lexicon driven baseline took just one working day. Merging the two took half a working day, but after that, the grammar was in a very good shape: it was complete with respect to the API, and most syntactic rules were correct. The rest of the work was mostly fixing bugs in individual rules not present in the miniature resource, such as relative clauses and passives.

As a conclusion, a mixed lexicon-syntax-based workflow can be recommended for many projects that bootstrap a language from an already existing one. For many languages, the lexicon part of it would of course also involve substantial work in implementing the morphology. If the linearization types are too far apart, bootstrapping the syntax could then also be far more complex than porting Thai to Chinese.

To Section 10.4. The full resource grammar extends the miniature resource grammar with many more categories and functions, which makes it necessary to even make some of the old ones more general. Most notably, this affects the clause category. In the full resource grammar, the linearization type is

```
lincat Cl =
  {s : Polarity => Aspect => Str ; np: Str ; vp: Polarity => Aspect => Str}
```

The addition of an aspect feature is obvious: it corresponds to the variable tense in English and other Western languages. But the `np` and `vp` fields are new. Their purpose is to make it possible to form questions with interrogative adverbials. For instance, *where does he walk* can be expressed by placing the word *where* between the subject and the verb phrase:

他	哪里	走
he	*where*	*walk*

In fact, the s field could be omitted altogether, because it can always be constructed from the other fields. The implementation uses the s field mainly to make it easier to test the linearizations of clauses and their different forms. (This very phrase can moreover be expressed by placing *where* to the end.)

Another peculiar category in adverbs, Adv. In Chinese, the place of an adverb in a sentence depends on whether it is an adverb of manner (after the verb) or time or place (before the verb).

To Section 10.6. Pinyin is a standard way of writing Chinese with Latin letters, with or without diacritics for the tones. There are good resources on the web for converting Chinese characters to Pinyin strings and back. The conversion from characters to Pinyin is almost deterministic, whereas the inverse conversion may yield many results, even if the tone is marked. In the resource grammar implementation, the master grammar is in Chinese characters. But the library provides a script for converting all strings in grammar modules to Pinyin. As an example, the rule

```
  lin very_AdA = {s = word "非常"} ;
```

is converted to

```
 lin very_AdA = {s = word “fei1chang2"} ;
```

The conversion is applied to all string literals in the grammar’s source modules. In addition, the operation bword (Section 4.4 above) is changed to using + instead of ++.

To Section 10.9. The core resource grammar for Chinese was exceptionally easy to implement, mostly due to the fixed word order and the lack of morphology. While the core is sufficient for translating other languages to Chinese, it lacks many forms of expression that would be needed in order to *parse* Chinese. At the time of writing this, extending the ExtraChi module has not even started. But some obvious candidates have been mentioned earlier: the reduplication of adjectives (Section 2.9) and the aspect system of Chinese (Section 5.17). Earlier work on similar tasks includes e.g. the LFG grammar of Fang and Holloway King (2007), with publication details in Appendix F below.

To Section 10.10. The lexicon-driven workflow described above used several web resource, of which the most important one was the Swadesh list in the Wiktionary,

  http://en.wiktionary.org/wiki/Appendix:Mandarin_Swadesh_list

As a complement to the lists, Google translate was used to bootstrap the lexicon. The best way to use it for this purpose is to translate small English sentences or phrases, which put the words into context. This helps disambiguate the English words, for instance tell verbs from nouns. It can also help extract the classifiers of the nouns. For instance, the inpute *five cats* yields the answer 五只猫, from which we can extract both the noun 猫 and the classifier 只 and thereby create the lexical entry

```
  lin cat_N = mkN "猫" "只" ;
```

To Appendix A

A.1 Abstract Syntax is the same as in the book.

A.2 Auxiliary resource module

```
resource ResChi = open Prelude in {

flags coding=utf8;

-- parameters

param
    Number = Sg | Pl ;
    SForm = Phr PosType | Sent;
    PosType = APhrase | NPhrase | VPhrase ;

-- parts of speech

oper
  VP = {verb : Verb ; compl : Str} ;
  NP = {s : Str} ;

-- for morphology

  Noun : Type = {s : Str; c : Str} ;
  Adj  : Type = {s : Str; monoSyl: Bool} ;
  Verb : Type = {s : Str} ;

  mkNoun : Str -> Str -> Noun = \s,c -> {s = word s ; c = word c};

  mkAdj : Str -> Adj = \s -> case s of {
    ? => {s = word s ; monoSyl = True} ;
    _ => {s = word s ; monoSyl = False}
    } ;

  copula : Verb = mkVerb "是" ;
  mkVerb : (v : Str) -> Verb = \v ->
    {s = word v} ;

  neg : Bool -> Str = \b -> case b of {True => [] ; False => "不"} ;

-- for structural words

  mkDet : Str -> Number -> {s : Str ; n : Number} = \s,n -> {
    s = word s ;
    n = n
    } ;

  pronNP : (s : Str) -> NP = \s -> {
    s = word s
    } ;

-- Write the characters that constitute a word separately.
-- This enables straightforward tokenization.

  bword : Str -> Str -> Str = \x,y -> x ++ y ;
  -- change to x + y to treat words as single tok ens

  word : Str -> Str = \s -> case s of {
      x@? + y@? + z@? + u@? => bword x (bword y (bword z u)) ;
      x@? + y@? + z@? => bword x (bword y z) ;
      x@? + y@? => bword x y ;
      _ => s
      } ;
}
```

A.3 Concrete syntax

```
concrete GrammarChi of Grammar = open ResChi, Prelude in {

  flags coding = utf8;

  lincat
    S  = {s : Str} ;
    Cl = {s : Bool => Str} ;
    NP = ResChi.NP ;
      -- {s : Str} ;
    VP = ResChi.VP ;
      -- {verb : Verb ; compl : Str} ;
    AP = {s : Str; monoSyl: Bool} ;
    CN = ResChi.Noun ;        -- {s : Str; c : Str} ;
    Det = {s : Str ; n : Number} ;
    N = ResChi.Noun ;         -- {s : Str; c : Str} ;
    A = ResChi.Adj ;          -- {s : Str; monoSyl: Bool} ;
    V = ResChi.Verb;          -- {s : Str}
    V2 = ResChi.Verb ;
    AdA = {s : Str} ;
    Pol = {s : Str ; b : Bool} ;
    Tense = {s : Str} ;
    Conj = {s : SForm => Str} ;

  lin
    UseCl t p cl = {s = t.s ++ p.s ++ cl.s ! p.b} ;

    PredVP np vp = {s = \\p => np.s ++ neg p ++ vp.verb.s ++ vp.compl} ;

    ComplV2 v2 np = {
     verb  = v2 ;
     compl = np.s
     } ;

    UseV v = {
      verb = v ;
      compl = []
      } ;

    DetCN det cn = case det.n of {
            Sg => {s = det.s ++ cn.c ++ cn.s} ;
            Pl => {s = det.s ++ "些" ++ cn.s}
      } ;

    ModCN ap cn = case ap.monoSyl of {
            True => {s = ap.s ++ cn.s ; c = cn.c} ;
            False => {s = ap.s ++ "的" ++ cn.s ; c = cn.c}
            } ;

    CompAP ap = {
      verb = copula ;
      compl = ap.s ++ "的"
      } ;

    AdAP ada ap = {
      s = ada.s ++ ap.s ;
      monoSyl = False
      } ;

    ConjNP co x y = {
      s = x.s ++ co.s ! Phr NPhrase ++ y.s
      } ;

    ConjS  co x y = {s = x.s ++ co.s ! Sent ++ y.s} ;
```

```
    UseN n = n ;
    UseA adj = adj ;


    a_Det = mkDet "一" Sg ;
    every_Det = mkDet "每" Sg ;
    the_Det = mkDet "那" Sg ;

    this_Det = mkDet "这" Sg ;
    these_Det = mkDet "这" Pl ;
    that_Det = mkDet "那" Sg ;
    those_Det = mkDet "那" Pl ;

    i_NP = pronNP "我" ;
    she_NP = pronNP "她" ;
    we_NP = pronNP "我们" ;

    very_AdA = ss (word "非常") ;

    and_Conj = {s = table {
                    Phr NPhrase => "和" ;
                    Phr APhrase => "而" ;
                    Phr VPhrase => "又" ;
                    Sent =>  []
                          }
                } ;

    or_Conj  = {s = table {
                    Phr _ => "或" ;
                    Sent => word "还是"
                          }
                } ;

    Pos  = {s = [] ; b = True} ;
    Neg  = {s = [] ; b = False} ;
    Pres = {s = []} ;
    Perf = {s = []} ;

}
```

A.4 Morphological paradigms API

```
resource ParadigmsChi = GrammarChi [N,A,V] **
  open ResChi, GrammarChi, Prelude in {
```

```
flags coding=utf8;

oper
  mkN = overload {
    mkN : (man : Str) -> N
      = \n -> lin N (mkNoun n "个") ;
    mkN : (man : Str) -> Str -> N
      = \n,c -> lin N (mkNoun n c)
    } ;

  mkA : (small : Str) -> A
      = \a -> lin A (mkAdj a) ;

  mkV : (walk : Str) -> V
      = \s -> lin V (mkVerb s) ;

  mkV2 = overload {
    mkV2 : (love : Str) -> V2
      = \love -> lin V2 (mkVerb love) ;
    mkV2 : (love : V) -> V2
      = \love -> lin V2 love ;
   } ;
}
```

A.5 Test lexicon

```
concrete TestChi of Test = GrammarChi ** open ParadigmsChi in {
flags coding=utf8;
lin
  man_N = mkN "男人" ;
  woman_N = mkN "女人" ;
  house_N = mkN "房子" ;
  tree_N = mkN "树" "棵";
  big_A = mkA "大" ;
  small_A = mkA "小" ;
  green_A = mkA "绿" ;
  walk_V = mkV "走" ;
  arrive_V = mkV "到" ;
  love_V2 = mkV2 "爱" ;
  please_V2 = mkV2 "麻烦" ;
}
```

A.6 Syntax API is the same as in the book, changing Ita to Chi.

To Appendix F

Fang, J. ; King, T. H. An LFG Chinese grammar for machine use. *Proceedings of the GEAF 2007 Workshop*; 2007 July 13-15; Stanford, CA. *An LFG grammar for parsing Chinese developed at Xerox.*

Magistry, P. and K. Gerdes. Paddy Fields: A Topological Description of Chinese Word Order 4th International Conference on Meaning-Text Theory, 2009, Montréal. *An approach similar to GF's discontinuous constituents.*