Unicode and its discontents

Jeremy G. Kahn
jgk@u.washington.edu
Machine Translation reading group
5 May 2008

Overall outline

- Character encodings: Back to grammar school
  - Vocabulary and history lessons
- Chinese encodings
  - Survey
  - Challenges for MT
- Unicode

Character encodings: why should *linguistics* care?

- Linguistics: often spoken language
- Writing schemes: linguistically interesting
- Computers don't know “letters” (or “words”, or even “numbers” -- all added by our interpretation)
- Computers know *bits* (or *bytes*)
- If we want to deal with “letters”, we need code to map “letter” (or *grapheme*) to *bytes*
- “the wonderful thing about standards...”

Mappings are important

- Wrong character mapping? corpora are noise
- Corpus as fetched may not be in the format your tools like
- Data interchange formats (e.g. XML, HTML) rely heavily on user (at some level) handling encodings correctly

... you should be able to read the encoding docs, even if you can't write your own converter

Mappings are sociologically neat

- Historical aspects
  - {U,V,W,u,v,w} all cognate with Roman V
  - Chinese and Japanese share *many* characters [*han-zi* or *kanji* respectively], but do not agree about their pronunciation (or meaning...)
  - Greek, Roman, Cyrillic alphabets share common lineages
  - Korean writing (Hangul) designed by linguist philosopher king -- really!
- More-contemporary history in a minute...

Vocab 1

- *Character*
  - "An abstract notion denoting a class of shapes declared to have the same meaning or form"
  - think *emic* ?
- *Glyph*
  - "An instance of a character"
  - e.g. with serifs, ligatures, etc.
  - think *etic*

Character v. Glyph

- Character: description of a class

  `LATIN CAPITAL LETTER A`

  `LATIN SMALL LETTER A`

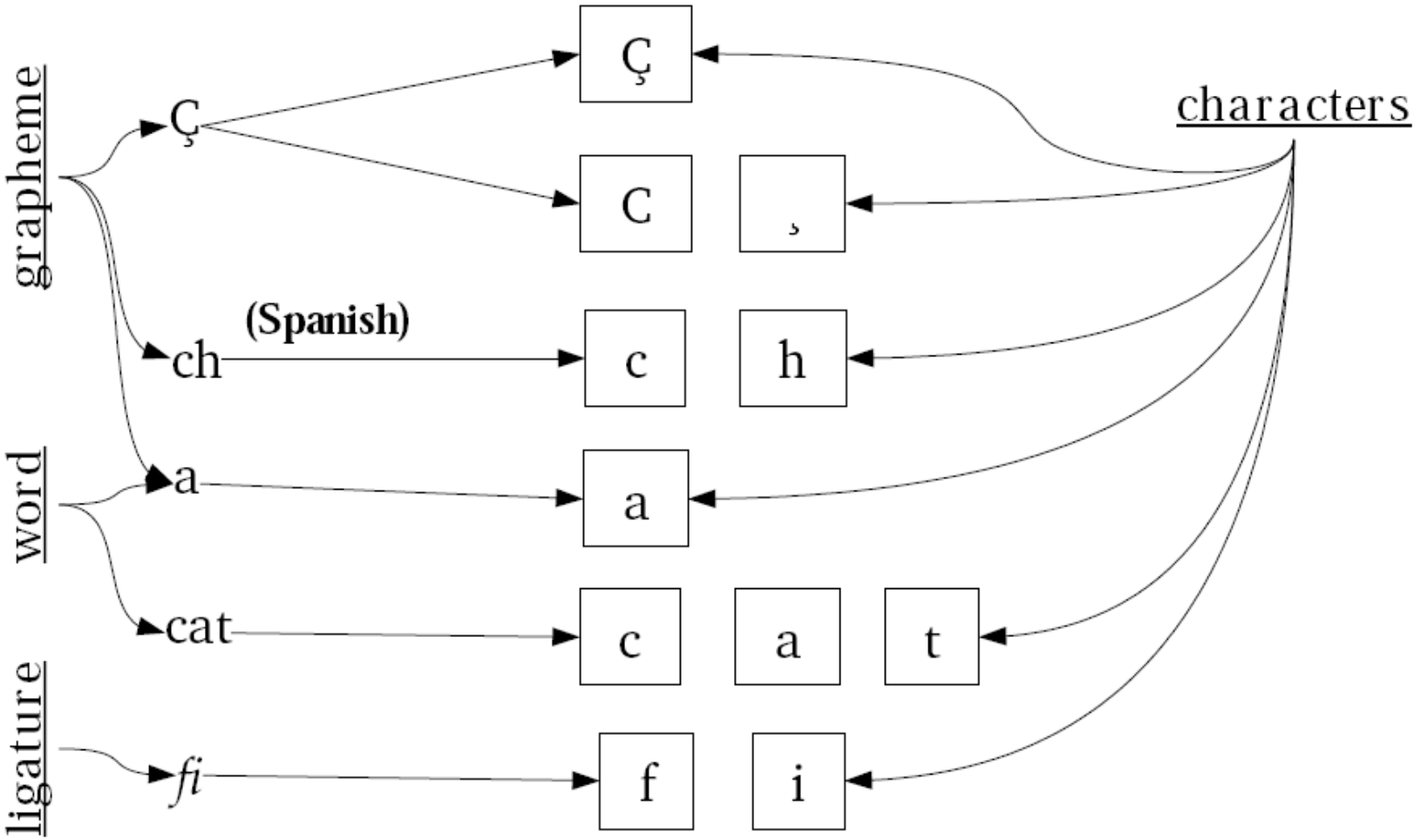
- Glyph

  A 𝒜 𝙰 A

  a a a *a*

- Note that a *character* not quite a *grapheme*
  - "ch" is a grapheme in Spanish, but not English

Character v. Grapheme



Vocabulary 2

- Character set (or “repertoire”)
  - set of unique characters.
- Coded character set
  - A character repertoire, plus a unique non-negative integer associated with each.
- Code point (or “position”)
  - the integer associated with a character in a CCS.

Some character sets and CCS's

- Digits [10] {0,1,2,3,4,5,6,7,8,9}
- English alphabet [26] {a,b,c,...,w,x,y,z}
- (“7-bit” ASCII) [128] *handles English & TTYs*
- Jōyō kanji [1,945] *Japanese newspapers*
- Latin-1 [256] *Most of western Europe – no TTYs*
- Latin-2 [256] *Most of eastern Europe – no TTYs*
- Unicode [20K+] *most of the world (?)*

What's the problem?

- Defining a CCS isn't enough.
  - Old concerns: space, bandwidth
  - rendering ... compare to teletype control
- We want to decide how to take these integers to bits.

Vocabulary 3

- Octet (sometimes *byte*)
  - eight bits
  - why not “byte”?
- *Encoding* (sometimes “character encoding”)
  - any algorithmic scheme that maps a series of code points from a CCS to a series of octets.
  - encodings vary in their utility and lossiness.
    - most obvious form: list of matches.

PITA vocabulary

- *Code page* is ambiguous.
  - choice of character encoding
  - byte orientation
  - what prefix encoding
- Let's not use this word.

Multiple masters

Character encodings, in their various incarnations, have included information on:

- Rendering information
- Character text flow (breaking, directionality)
- Glyph variants of the same character
- Character variants of the same glyph

- Handling all these is a Herculean labor

History of character encodings

- ASCII
  - both an *encoding* and a *repertoire*
  - handles Am. E. well, but not ¢, £
- ASCII repertoire has 128 code points, thus ASCII encoding can be done in 7 bits. (one per octet, with a dead bit.)
  - 0x00 to 0x20 dedicated to TTY *control characters*
- *We'll probably never see the back of ASCII:*

  QWERTY is dead, long live QWERTY

Signs of trouble with ASCII

- Alphabetization is annoying
  - (all capitals are octet-ordered before lowercase)
- Some early adopters doubted the utility of lowercase (!)
- Arguments about what punctuation to include
- whiny furriners

First-born: Latin-1

- Expands to 256 codepoints
  - a.k.a. ISO-8859-1
  - a.k.a. “8-bit ASCII” (though not PC)
- Western Europe is happy – à, é, etc included

Latin-1 as a CCS

- Integers 0 to 127 same as ASCII CCS
- 128-255 (0x80-0xFF) adds western-European characters:

¡ (Spanish)	0xA1
£ (British English)	0xA3
Ä (German)	0xC4
Þ (Icelandic)	0xDE
ê (French)	0xEA

Latin-1 (and ASCII) as encodings

- Not very hard:
  - 8 bits can represent a non-negative integer < 256
  - one octet per character
- Thus: map *code point* to a single binary octet.
- Order of octets represents order of characters.

Latin-1 troubles

"And all the firstborn in the land of Egypt shall die"

Exodus 11:5

- Alphabetization troubles are even worse than before: all accented characters sort *after* non-accented ones
- Extensibility and inclusion:
  - NATO is happy, but Eastern bloc not (no č, for example)
  - Not to mention Cyrillic, Arabic, Hebrew, Greek
- Where is there room for *détente*? (or is it *detente*?)

Many kingdoms

- Latin-2 (ISO-8859-2) “East” European
  - Polish, Czech, etc + ASCII
- Latin-3 (ISO-8859-3) “South” European
  - Maltese, Turkish, Esperanto + ASCII
- Latin-4 (ISO-8859-4) “North” European
  - Estonian, Baltic, Lithuanian, Lappish + ASCII
- Latin-5 (ISO-8859-***9***)
  - Latin-1 minus Icelandic plus Turkish
- Latin-6 (ISO-8859-10)
  - Latvian + Nordic + ASCII

Gunfight at dawn

“There's not enough room in this octet for the both of us.”

- No way to support Turkish and Icelandic at once
- Not to mention Hebrew (ISO-8859-8) and Arabic (ISO-8859-6)
  - > 256 characters needed
  - ISO-8859-* creates collisions in upper code points

Additional problems with non-Roman alphabets

- Lots of new characters
- Mostly non-overlapping with US-English
- Mostly non-overlapping with each-other
- Complex ligating behaviors
- Arabic and Hebrew have a handedness problem
  - .od su fo tser eht ebyam rO
  - (strictly speaking, this is a layout problem, not a character-encoding one.)

The ~~elephant~~ dragon in the room

- Chinese
  - Big-5 (Taiwan)
  - GB (PRC)
- Korean
  - Johab
  - Wan-Sung
- Japanese
  - Shift-JIS, others

- Size:
  - Chinese & Japanese: thousands of characters
  - Korean: at least 1300 more
- Ordering
  - natural ordering is complex taxonomy at best

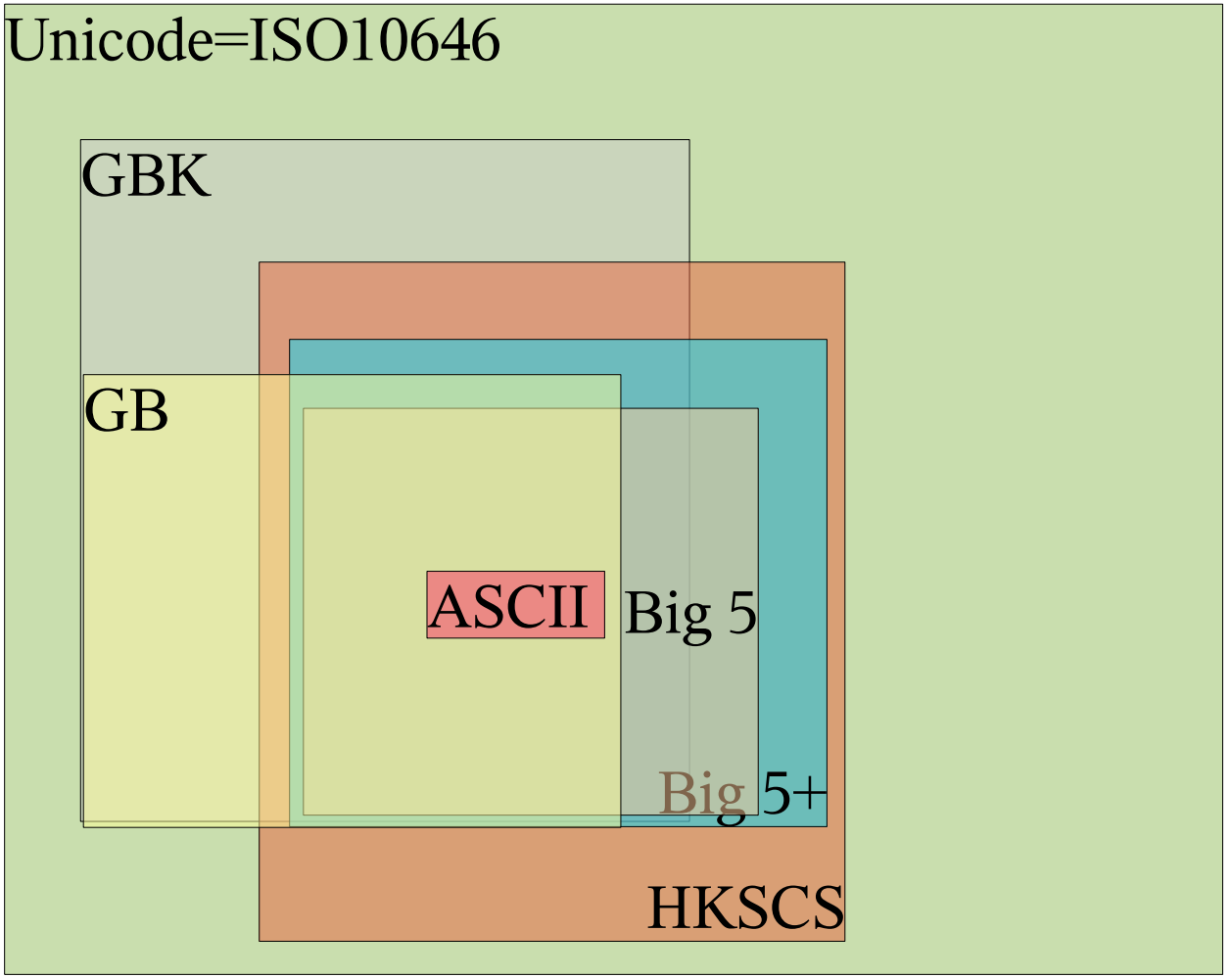
Taming the dragons

- Wider characters: more octets/character
  - Space-hungry. Endian.
  - incompatible with ISO-8859-tuned code
- Shift-encodings: control character indicates when to interpret as wide-characters.
  - difficult to randomly-access
- Prefix-encoding: common characters get shorter-encodings
  - how to define “common”? plus problems above

Chinese coded character sets

- GB: Guójiā Biāozhǔn Mǎ (国家标准码)
  - “National Standards encoding”
  - PRC [mainland], Singapore
  - GB < GBK [Kuòzhǎn “extended] < GB18030 ≈ Unicode
    - More on that later
- Big5
  - The Big 5 (might be): Acer, MiTAC, JiaJia, Zero One, FIC
  - Big5 < Big5+ < Unicode

Chinese coded character sets



No, it's not pretty.

Issues in encoding Chinese

- Simplified vs. traditional characters
- Legacy Chinese encodings incorporate layout information
  - Fullwidth vs. halfwidth characters
    - Mostly layout distinctions, but not entirely
  - Variants for vertical vs horizontal layout
    - A “dash” character in vertical layout?
- All the issues of Latin-1

21st century digital world: **Unicode**

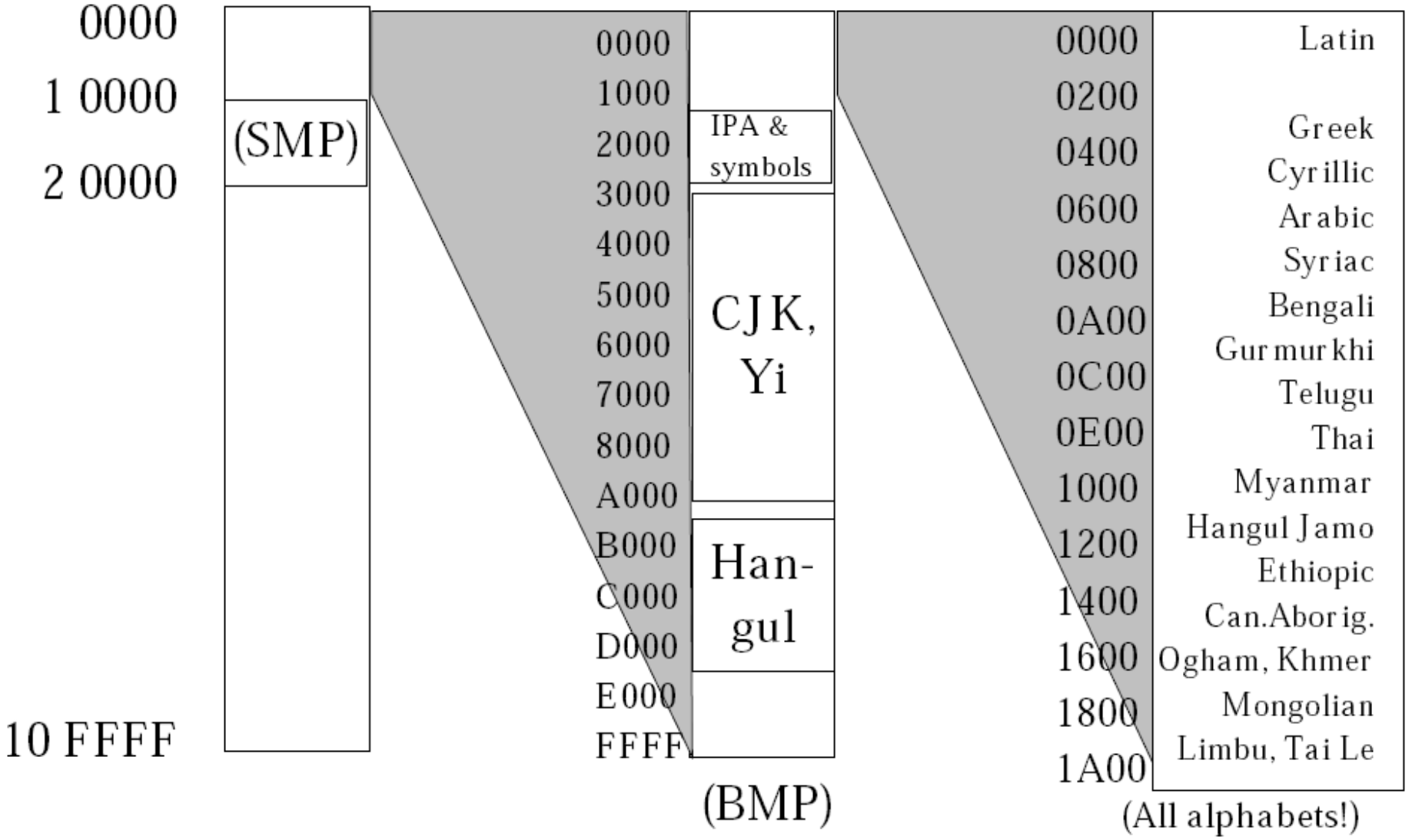
a multilingual solution

- Roughly, a *coded character set* with no upper bound on the integer.
- Aims:
  - character-sequence encoding (not rendering)
  - “logical” order (handedness)
  - unification (don't list characters twice).
  - convertibility (from other, smaller CCS).
- potential conflicts among aims

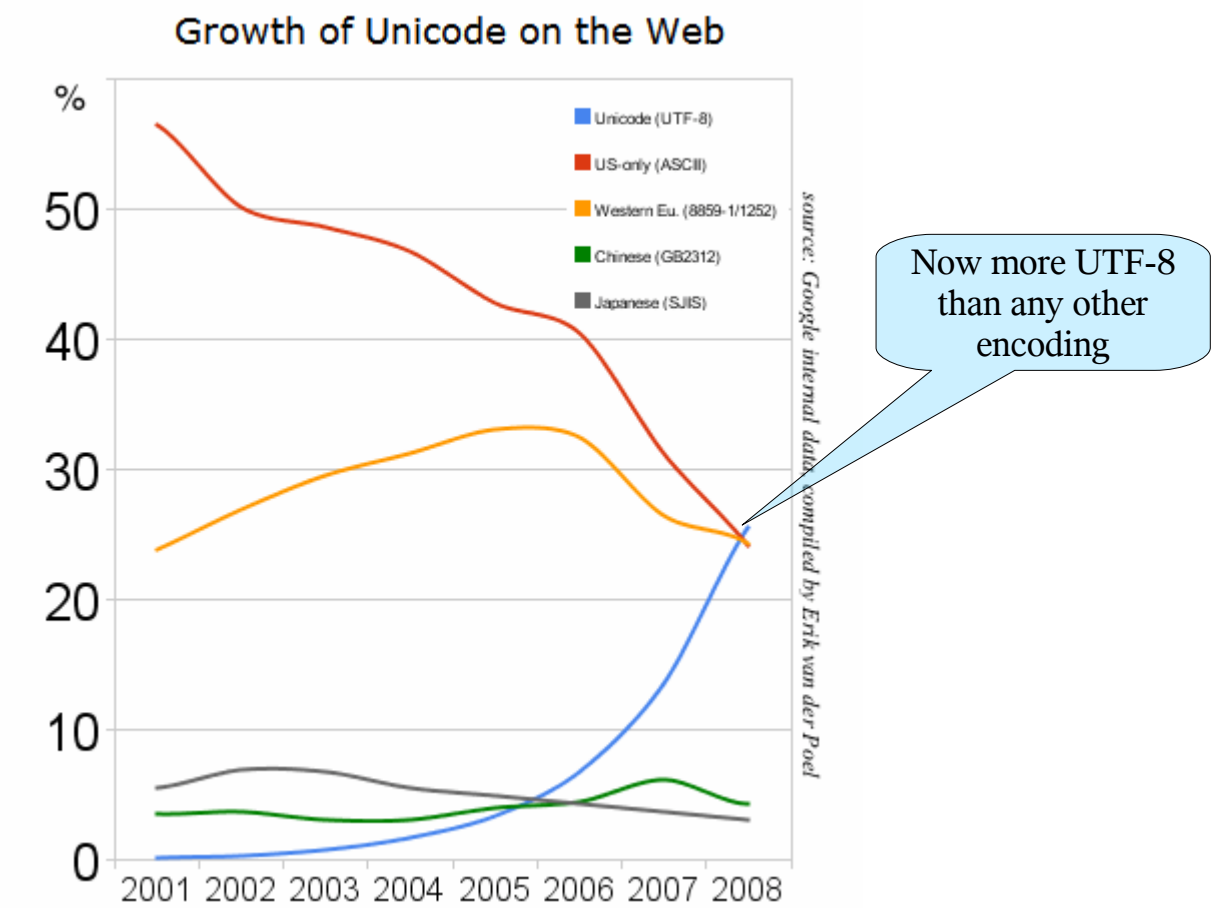
Unicode

- Defines > 1M code points.
- Only looking at Basic Multilingual Plane
  - first 65,536 (this number should look familiar)...

Unicode is *big*



Unicode is popular



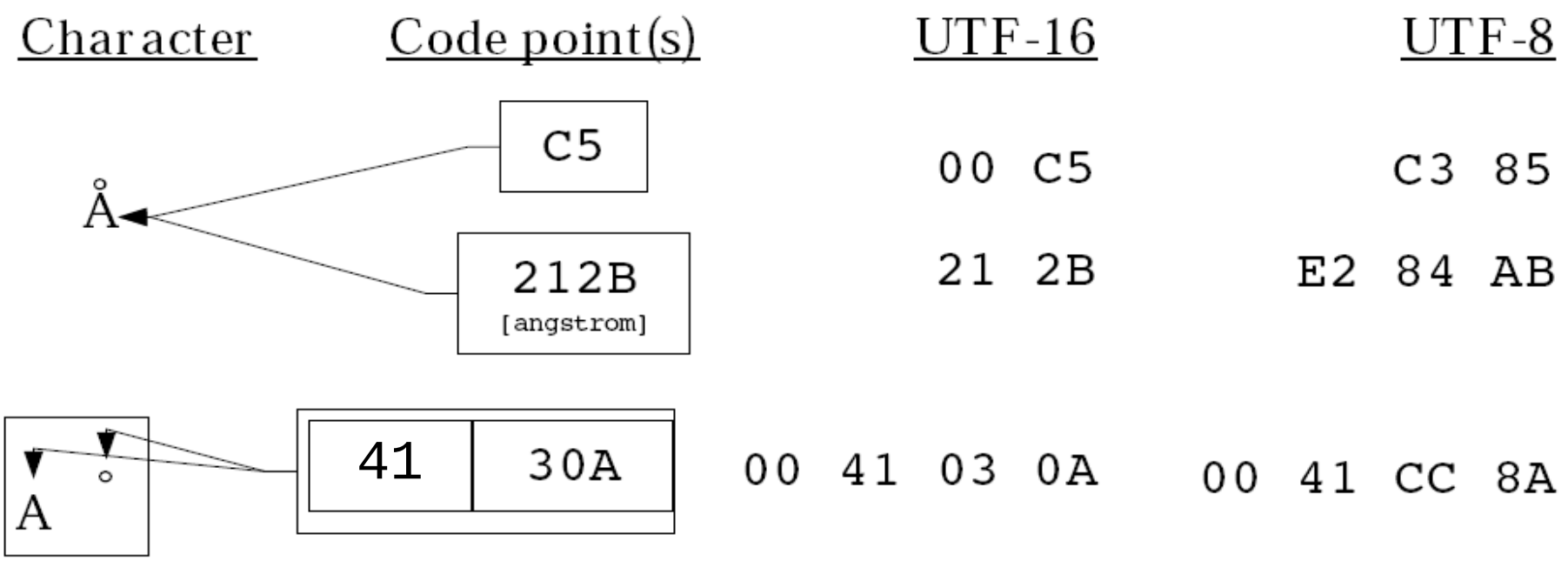
On the Google blog today:
http://googleblog.blogspot.com/2008/05/moving-to-unicode-51.html

Han Unification

- Intent (unify graphemes)
- Resistance (especially from Japan)
- Redundant encodings (for compatibility)
  - Raises need for normalization
- Political choice: retain traditional/simplified split
  - More possible needs for normalization

Unicode splits some differences

- Unicode is really only a coded character set
- Character encoding is switchable



Normalizing Unicode

- There's more than one way to code many characters
  - Is Á:
    - U+00c1 (Á)
    - or U+0041(A)+ U+0301 (combining accent)?
- Unicode includes a normalization standard
  - Perl & Python & others have libraries for this
  - Doesn't cope with other normalization issues
    - Angstrom Å U+212B is not equivalent to U+0041 + U+030A combining ring
    - Some of these problems are worse for Chinese

Unicode Transformation Format (UTF)

UTF-16

- sometimes mislabeled “Unicode”, mostly by Windows
- encodes each *character* in the BMP into two octets
- Character semantics very simple, but space-inefficient & breaks mono-octet-character-semantics code

UTF-8

- *variable* number of octets to encode the BMP
- Prefix mapping approach, skewed towards ASCII
- ASCII-character-set is encoded in 1 octet!

State of the art

- most modern programming languages support UTF-8 internally now
  - Perl, Python, probably others use these internally
- C/C++ still often needs to specify what kind of character to expect – remember, “string” isn't a primitive!
- Browsers often get it right, now, as does emacs (!) -- this can make it a challenge to find out what's not working
- Linux/OS-X lifesaver tool: *iconv* .

Issues for Chinese MT

- Detect input encoding
  - Mostly GB, Big5
  - Encode::Guess (tool)
- Collapse “non-semantic” distinctions
  - Encode::HanConvert
    - Caveat: using Unicode table information
  - Convert::EastAsianWidth

    `FULLWIDTH LATIN CAPITAL A` ≈ `LATIN CAPITAL A`

Further issues for Chinese MT

- When do fullwidth vs. halfwidth distinctions matter?
- When do simplified vs. traditional distinctions matter?

- Guess: not much, at current state of art.
  - More wins in reducing data sparsity
  - Compare to case normalization in European translation

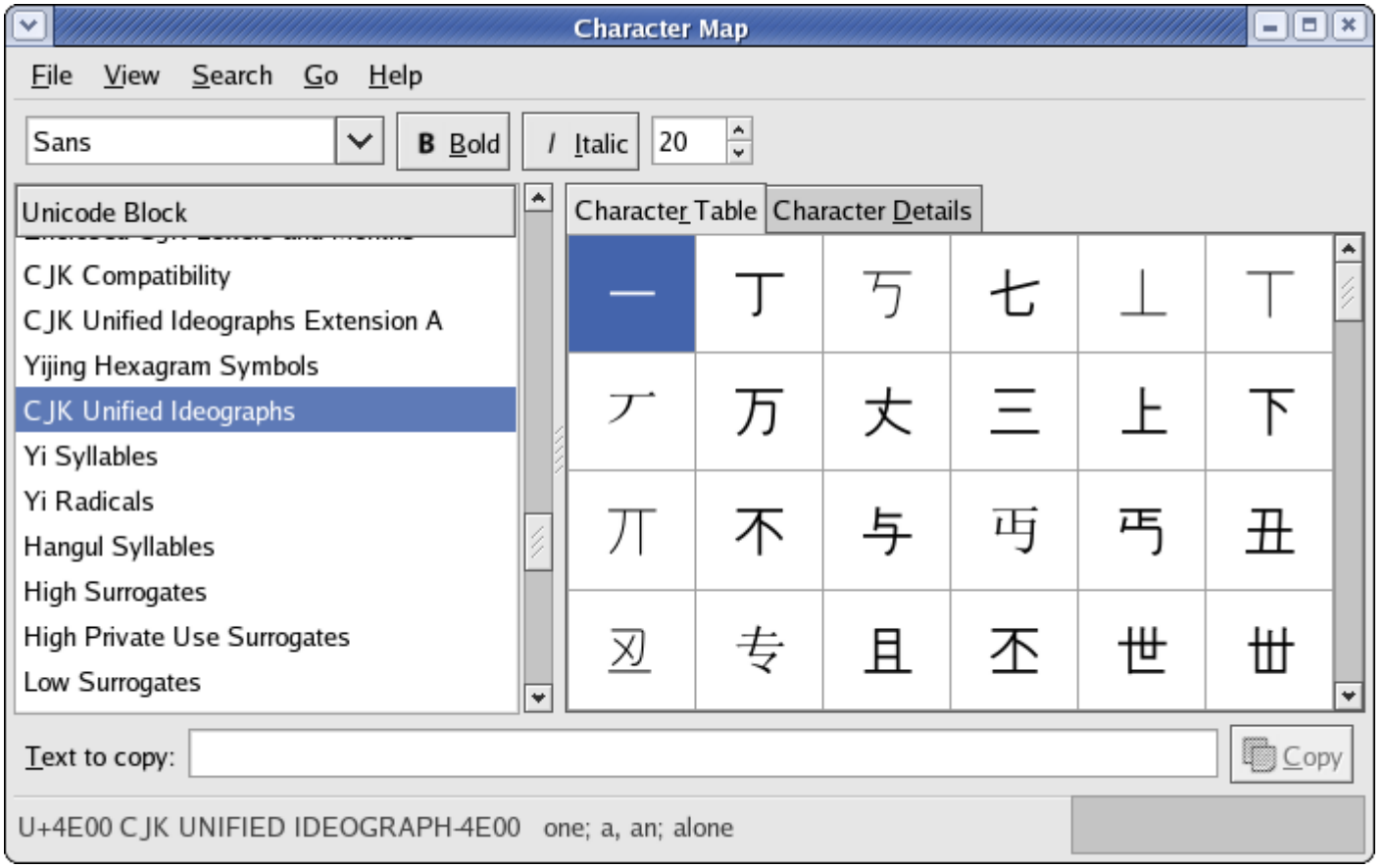
UTF-8 isn't magic

(but it's pretty close)

- Backwards-“compatible” with ASCII CCS, encoding
- Won't break Latin-1 compatible code
- Round trip from Big5, GB: okay
- Inter-coding GB-UTF8-Big5 (or reverse)
  - okay [with some “semantic” normalization]
- Shared CCS with GB18030
  - But different encoding

Sanity tools

- *iconv* and Perl modules as mentioned above
- *gucharmap* if you use Gnome



Perl code points:

```
$ perl -le 'print chr(0x41)'
A
```

(you can do it backwards too: set terminal to UTF8, and:)

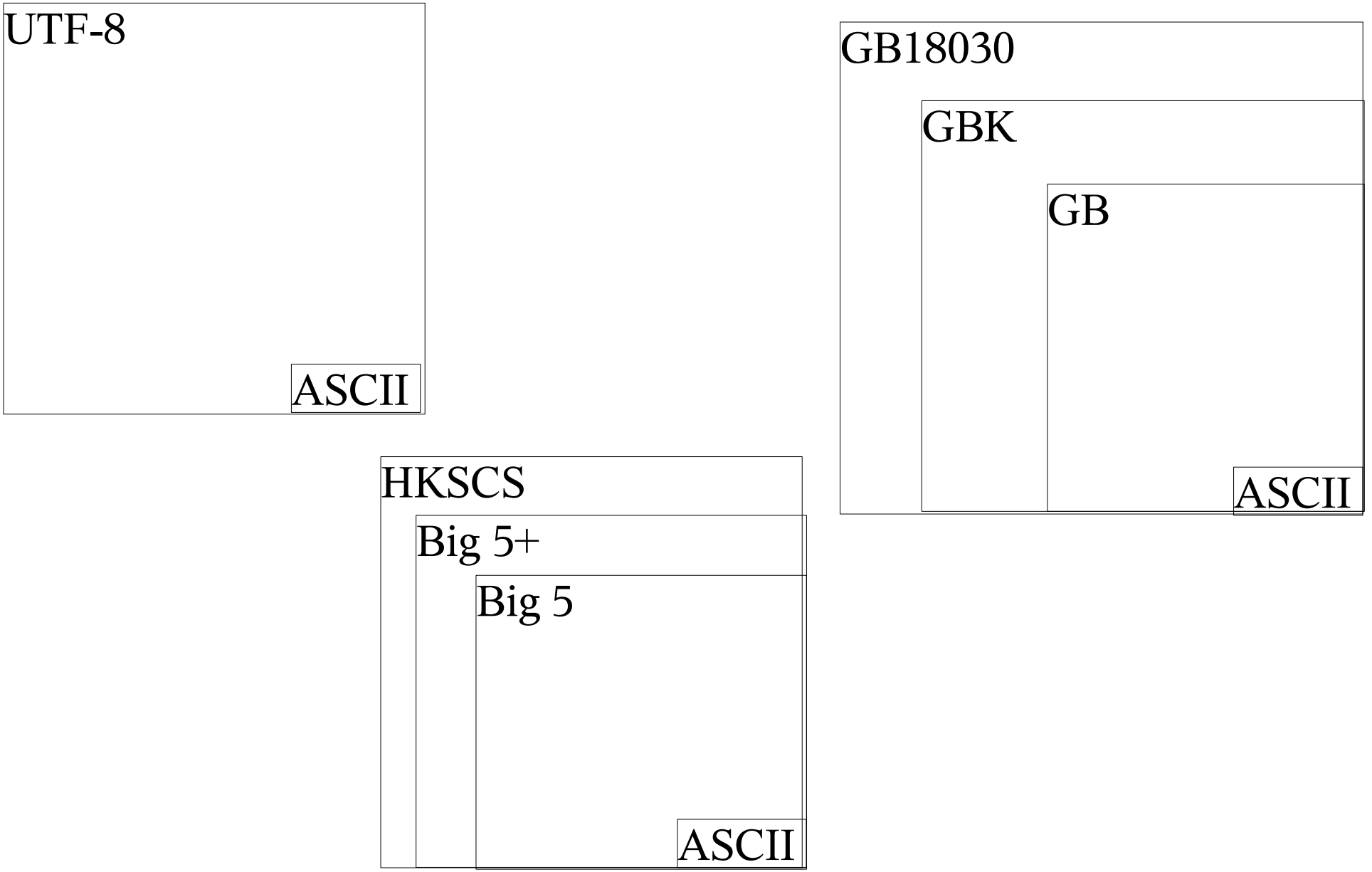
```
$ perl -Mutf8 -e 'printf "0x%x\n",
  ord("毛")'
0x6bdb
```

Note that `-Mutf8` tells Perl you're using UTF8 literals (`use utf8` within a script)

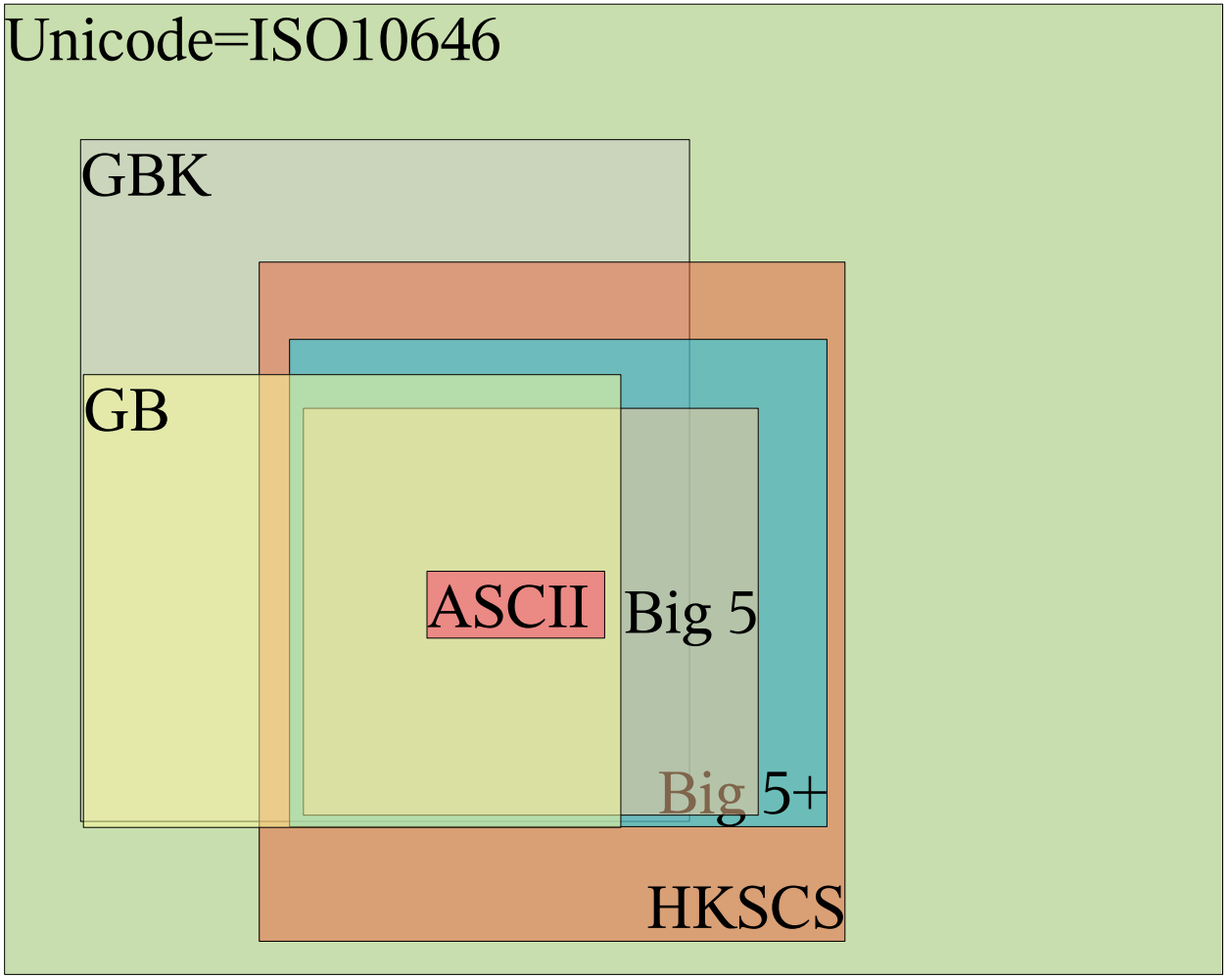
So what's the point?

- We get the encodings wrong all the time
- GB < GBK, for example
  - Extra information in GBK is lost – often deliberately
- Understanding these problems helps us use our tools better.

Revisiting character *encodings*



(For review):
Chinese coded character sets



Nope, still not pretty.

Questions?

- *The Unicode Standard* (The Unicode Consortium)
  - current revision is 5.0 http://www.unicode.org/
- http://czyborra.com/charsets/iso8859.html
- *CJKV Information Processing*, Ken Lunde (aka "The Blowfish Book")
- Wikipedia is (perhaps unsurprisingly) quite rich in this information now (2008).
- ... or ask now!