Dasher and Unicode

Chris Ball

cjb@mrao.cam.ac.uk

June 6, 2005

Motivations for this talk

- Internationalisation is important
- There are pitfalls
- Dasher seems to have got it right – over 100 languages!
- We will tell you all our dirty little secrets
  - (Well, our professional ones)

The problem

- Historically, there have been hundreds of ways for representing characters as numbers – ASCII, EBCDIC, Shift-JIS, ...
- Internationalised software would have to detect and support all of these encodings
- Unicode tries to be a single solution for internationalisation
- Contains glyphs for over 100,000 characters
- Each character is defined by a "code-point" in hexadecimal
- E.g.:
  - U+221E = ∞
  - U+00E9 = é

Example codepoints

	304	305	306	307	308	309
0		ぐ 3050	だ 3060	ば 3070	む 3080	み 3090
1	ぁ 3041	け 3051	ち 3061	ぱ 3071	め 3081	ゑ 3091
2	あ 3042	げ 3052	ぢ 3062	ひ 3072	も 3082	を 3092
3	い 3043	こ 3053	っ 3063	び 3073	ゃ 3083	ん 3093
4	い 3044	ご 3054	つ 3064	ぴ 3074	や 3084	ゔ 3094
5	う 3045	さ 3055	づ 3065	ふ 3075	ゅ 3085	か 3095
6	う 3046	ざ 3056	て 3066	ぶ 3076	ゆ 3086	け 3096

Representation on disk

- In its simplest encoding, Unicode needs two (or even four) bytes per character
- UTF-8 is a "variable-width" encoding, $1 \leq \textit{bytes} < 6$
- ASCII is valid UTF-8
- When writing Roman text, UTF-8 uses one byte per character

Representation on disk

- In UTF-8, the high bit denotes whether there are subsequent bytes
  - $01000001 = 65 = A$, leading zero says only one byte
- When you need a multiple-byte character:
  - The two high bits are set to (11): begin a multi-byte character
  - The two high bits are set to (10): continue that character
- This makes it possible not to "waste" bytes on Roman text

Internationalised Dasher

Dasher defines a language by:

- An alphabet file
- A training text
- A colour scheme (optional)

Alphabet file

- Lists the valid characters for a language
- Organises the characters into “groups”
- Tells Dasher where to find the training text
- May specify colour scheme, writing orientation

Alphabet file example: English

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE alphabets SYSTEM "alphabet.dtd">
<?xml-stylesheet type="text/xsl" href="alphabet.xsl"?>
<alphabets>
<alphabet name="English alphabet - limited punctuation">
<orientation type="LR"/>
<palette>European/Asian</palette>
<train>training_english_GB.txt</train>
<group name="Lower case Latin letters" b="0">
<s d="a" t="a"/>
<s d="b" t="b"/>
<s d="c" t="c"/>
...
</group>
<group name="Upper case Latin letters" b="111">
<s d="A" t="A"/>
<s d="B" t="B"/>
<s d="C" t="C"/>
...
</group>
<group name="Punctuation" b="112">
<s d="!" t="!"/>
<s d="," t=","/>
<s d="." t="."/>
...
</group>
</alphabet>
</alphabets>
```

Alphabet file example: Japanese

```
<?xml version="1.0"?>
<!DOCTYPE alphabets SYSTEM "alphabet.dtd">
<?xml-stylesheet type="text/xsl" href="alphabet.xsl"?>
<alphabets>
<alphabet name="Nihongo / Japanese Kana and 7000 Kanji">
<orientation type="UD"/>
<palette>Hiragana</palette>
<train>training_Japanese_JP.txt</train>
<group name="Hiragana" b="0">
<s d="&#x3041;" t="&#x3041;" b="10" />
<s d="&#x3042;" t="&#x3042;" b="10" />
<s d="&#x3043;" t="&#x3043;" b="10" />
...
</group>
<group name="kanji" b="9">
<s d="&#x3303;" t="&#x3303;" />
<s d="&#x3300;" t="&#x3300;" />
<s d="&#x3301;" t="&#x3301;" />
...
</group>
<group name="Punctuation" b="112">
<s d="&#x300C;" t="&#x300C;" note="Asian left single quotation mark" />
<s d="&#x300D;" t="&#x300D;" note="Asian right single quotation mark" />
<s d="&#x300E;" t="&#x300E;" note="Asian left double quotation mark" />
...
</group>
</alphabet>
</alphabets>
```

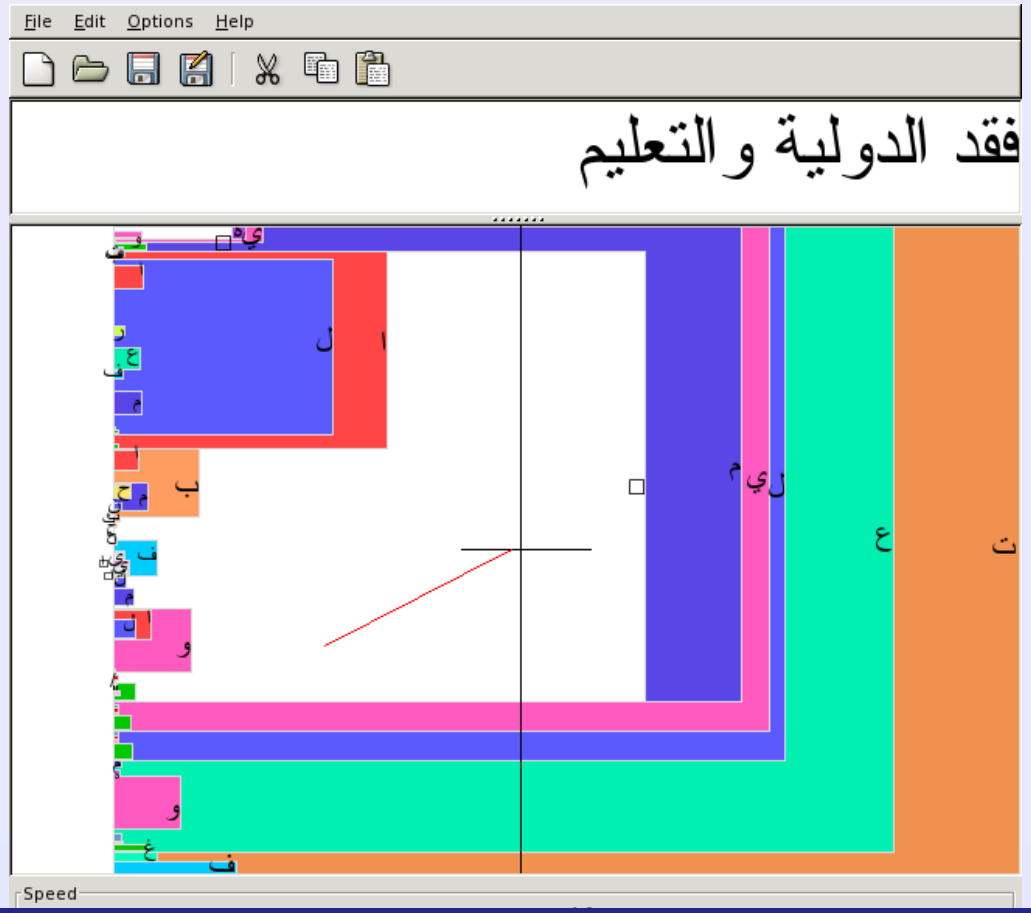
Training text

- A corpus of text, with no other information attached
- When Dasher trains, it will increment the PPM count for the context each symbol appears in
- The encodings in the alphabet file and training text must match!

Normalisation

What about when one character can *alter* the previous character?

- Examples: French (e-acute), Arabic, Hiragana (accents)



Normalisation

- This would be a mess if we had to do it ourselves
- But we don't!
- Unicode contains characters that combine with previous ones

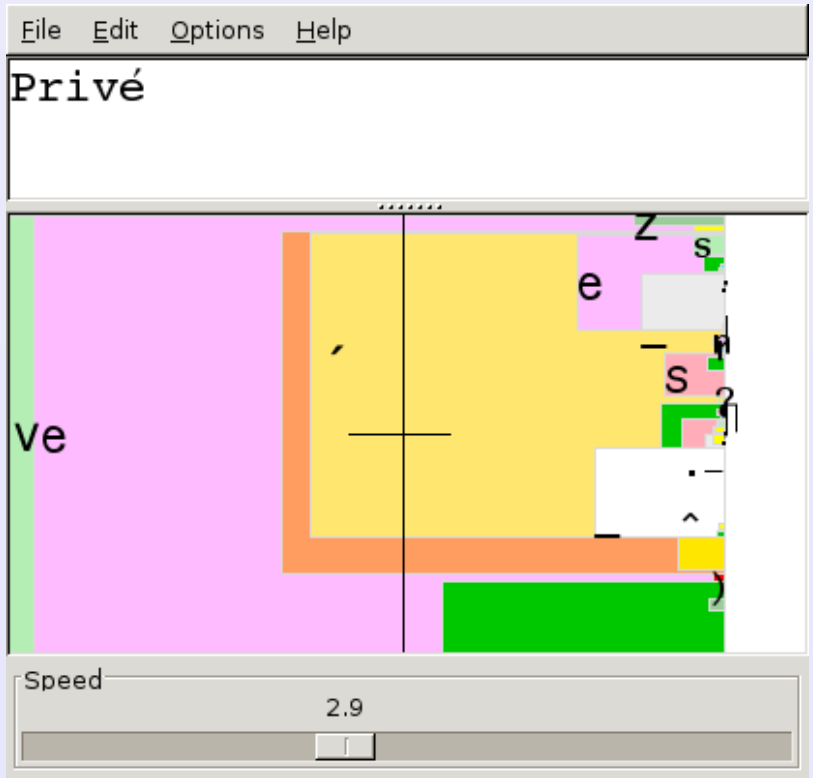
Normalisation

Example:

- U+0065 (E) *followed by*
- U+0301 (Combining acute)

Generates:

- U+00E9 (Latin small letter E with acute)



Normalisation

The two strings both represent *e-acute*, but in different forms.

- `U+0065 U+0301` is in NFD (Normalized Form Decomposition)
- `U+00E9` is in NFC (Normalized Form Composition)

Pitfalls

- "There is no such thing as a plain text file"
  - *Text* = encoding + data
  - Always know your encoding
- XML is useful for character interchange
  - Handles encoding, cross-platform issues for you
- Choose a normalisation form and enforce it throughout

Conclusion: Adding new languages to Dasher

- We need an alphabet file and a training text for the new language
- Both are stored in UTF-8
- Some languages have variants for composed/decomposed alphabets

Thank you!

Questions?