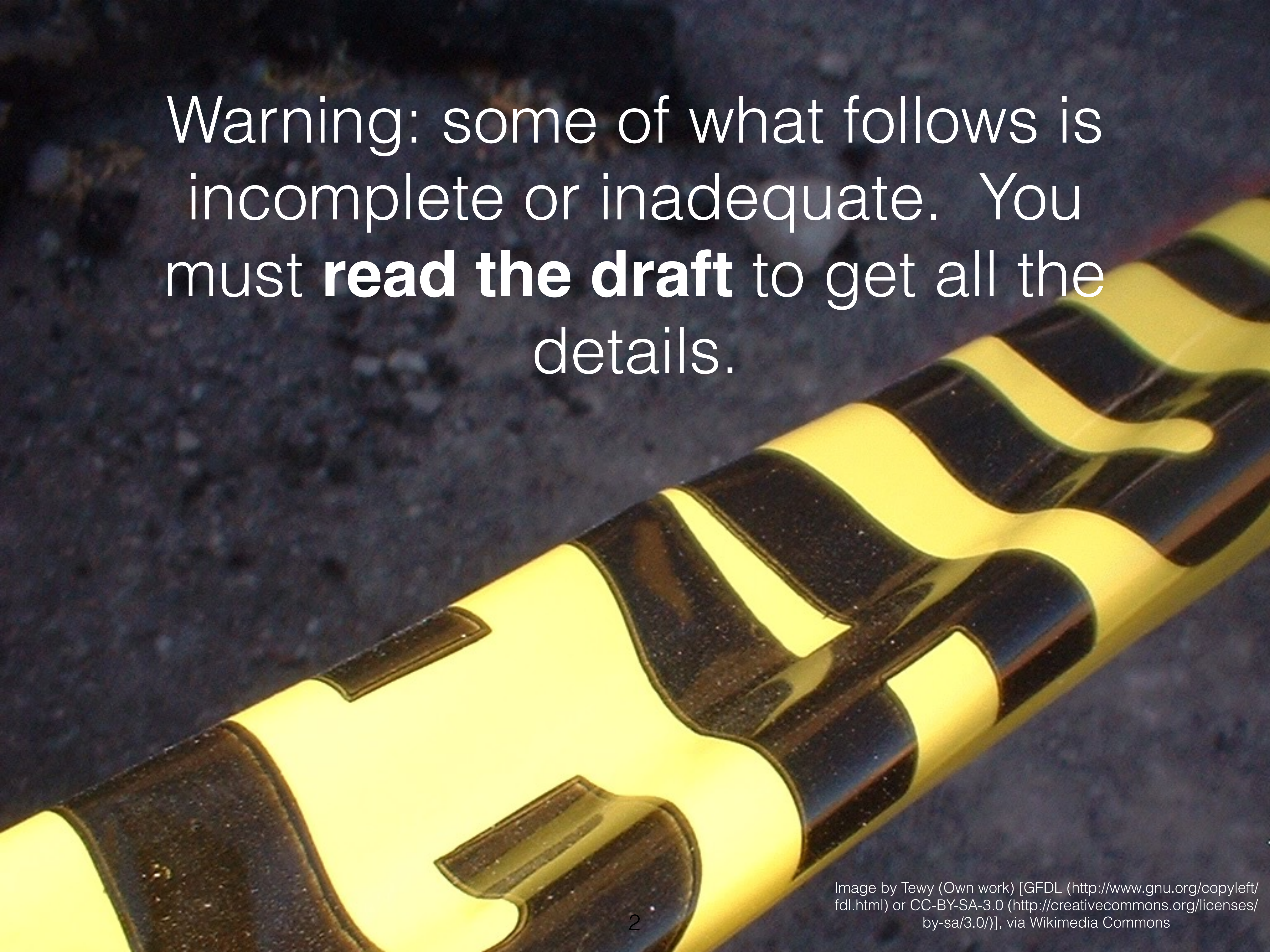
LUCID problem

IETF 92 — Dallas, TX
March 2015

Asmus Freytag & Andrew Sullivan



Warning: some of what follows is incomplete or inadequate. You must **read the draft** to get all the details.

Image by Tewy (Own work) [GFDL (http://www.gnu.org/copyleft/fdl.html) or CC-BY-SA-3.0 (http://creativecommons.org/licenses/by-sa/3.0/)], via Wikimedia Commons

Unicode has lots of abstract characters.

You can (sometimes) encode **the same** abstract character in more than one way.

Some things that look the same are not in fact the same abstract character.

When you have an identifier, you need **exactly one** way to encode it, or it won’t match.

“Inclusion mechanism” of i3 (**inclusion-based identifier internationalization**) tries to make this mostly work.

Inclusion

1. Start with an empty list
2. Look at character **properties**
3. If they’re good for identifiers (in the IETF sense), they’re in
4. Otherwise, they remain out

Code points have **properties**, which allow you to make decisions about them. One of those is whether the code point (or sequence) is in Normalization Form C.

Normalization forms are designed to make two ways of encoding **the same abstract character** “be the same” for some meaning of “same” (i.e. compatibly or canonically).

Normalization **does not** make characters that look the same be the same:

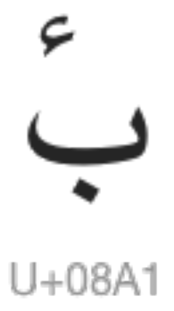
A (U+0041) ≠ А (U+0410)

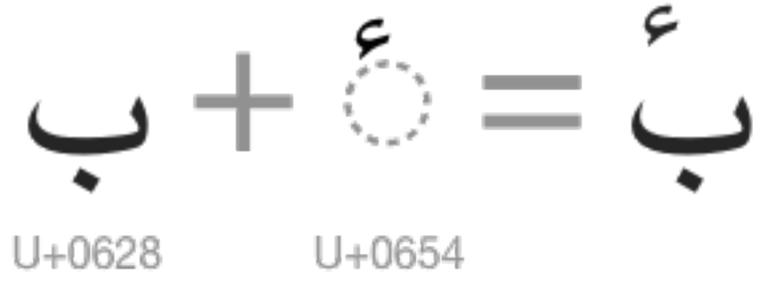
Homoglyphs can sometimes be distinguished other ways (e.g. script property).

Operational conventions can help (especially registration conventions).

So what’s the problem?

This is what caused the realization:





So *what’s* the problem?

- Turns out there’s a lot of that:
- ځ (ARABIC LETTER HAH WITH HAMZA ABOVE)
- أ (ARABIC LETTER ALEF WITH HAMZA ABOVE)
- ৡ (BENGALI LETTER VOCALIC LL)

So what is the problem?

Some cases (usually) don’t look perfectly the same:

- ƚ (U+019A) vs. l̵ (\u’006C’\u’0335’)
- ø (U+00F8) vs. o̷ (\u’006F’\u’0037’)
    - If I type **o̷** in bold, it doesn’t seem better. “Now you have two problems.”

But the problem?

What about digraphs?

- ʦ (U+02A6) vs ts (\u’0074’\u’0073’)
- ǉ (U+01C9) vs lj (\u’006C’\u’006A’)

Note that the first of these is ok under most forms of i3, and the second is DISALLOWED.

So, just combining marks? No!

- க (U+0B95, Ka) vs ௧ (U+0BE7, digit 1)
- ٣ (U+0663, digit 3) vs. ۳ (U+06F3, digit 3)
- 口 (U+53E3, “mouth, gate”) vs. 囗 (U+56D7, “proud, upright)

This is **not** just confusability, homoglyphs, and so on. No tech fix for human perception.

For Unicode, if two code points or code point sequences do not normalize to one another, then they’re **not the same abstract character**.

Making different characters match

- Canonical normalization
  - For sure really the same abstract character
- Compatibility normalization
  - Improve matching of related but maybe different abstract characters
- Mapping
  - Thumb-on-scale forcing to make things match

Directions

This is only allowed if we think we ought to solve stuff!

1. Find them, disallow new, cope with old
2. Disallow some combining sequences
3. Do nothing/just warn
4. Get a (or >1?) new Unicode property
5. Create NFI