```
(defun the-थचऊँ -old-بطهص-fn (aגզ១d)
   (if (eql aגզ១d 2) 2 (* aגզ១d (the-थचऊँ -old-بطهص-fn (1-
aגզ១d)))))
```

Unicode 4.0 In Common Lisp

Adoption of Unicode In CLforJava

Jerry Boetje
ILC 2005
boetjeg@cofc.edu

College of Charleston
School of Sciences and Mathematics

ASCII Legacy

- In the beginning (1983), there was
  - ASCII (universally recognized)
  - Everything else - mostly 8-bit encodings
    - ISO-8859-x
    - Code Pages (IBM PC)
    - JIS and some Chinese encodings (16 bit)
- Couldn’t mix encodings
  - Doc in Hebrew, Kanji, and Serbo-Croation

Lisp Response

- Agree on a subset of ASCII that works everywhere (standard char)
- Add font and bits attributes to characters (later dropped)
- Fuzzy distinction between types of chars
- Non-portable method for specifying file encoding
- Define functions that would work with ASCII

Pretty Good For Its Time

The Rest of the World's Response

- Define a uniform encoding for all characters on Earth
- Deal with the hard issues
  - Collation
  - Line breaks
  - Equivalence
  - Composition
  - etc.

Unicode

20 Years Later

- Globalization requires speaking all languages
- Many vendor-specific solutions
- Unicode version 4 has answers to many of the issues evoked by Common Lisp - and then some
- It’s time to formally integrate Unicode into the Common Lisp Standard
- But it’s not going to be easy!

Unicode 4 in Brief

Nature of Characters

- It’s not enough to assign a number to a char
- Characters are no longer atomic
  - A run of chars may be equivalent to one char
- Some provide information but not content
  - Direction
  - Formatting

Nature of Characters

- Never confuse the encoding with an ordering
  - Collation is entirely context-dependent
  - Does ‘o’ come before, after, or the same as ‘ö’
    - Different if your German or Swedish
- Chars have a rich set of properties
  - Simple - digit?, whitespace?
  - Complex - composition, direction, mirrored?

Encoding

- Number assignments are called ‘code points’
- Range **`#x0000`** to **`#x10FFFF`** (21 bits)
- ASCII range is the same in Unicode
- Chars grouped into named ‘blocks’
  - E.g. Tamil, Arabic, Number Forms

Composition / Normalization

- Some chars are composed of others
  - E.g. ‘Ä’ decomposes to ‘A’ and ‘’
- 2 chars are equivalent iff their decomposed, binary forms are identical
- But some chars are really “the same” even if they’re different
  - E.g. some Katakana full and half-width chars
- **There are 2 definitions of equivalence**
  - <u>Canonical</u> and <u>Compatibility</u>

Collation

- Context-dependent (locales)
- Unicode defines a table-driven mechanism
  - Very configurable (originally from IBM)
  - Specifically not required
  - Other mechanisms ok if equivalent results
    - Sun/Java uses a rule-based system

Bi-directional Algorithm

- Unicode specifies algorithm to handle nested changes in direction (R to L, L to R)
- Locale-dependent
- Very important with mixed languages
- Impacts the printer
  - Characters not printed in memory order
  - Some characters are mirrored

Line Break Algorithm

- Unicode specifies algorithm to determine possible line breaks
- Handles the <cr>, <lf>, <crlf> problem
- Locale-dependent
- Very important with mixed languages
- Impacts the pretty printer

Implies Pervasive Changes to Several Lisp Components

CLforJava Implementation

CLforJava Project

- Capstone software engineering course
  - Multi-semester undergraduate project
  - Gives students a “real world” experience
- New, original implementation of Common Lisp
  - Written in Java and Lisp
- See “Common Lisp for Java: A New Implementatoin Intertwined with Java”
  Wed 11am

Character Types

- CL standard defines
  - Standard-Char - 96 ASCII chars
  - Base-char, Extended-char - up to the impl
- CLforJava defines
  - Standard-Char - same as standard
  - Base-char - Unicode definition of base character
    - Can’t be composed with char to the left
  - Extended-char - all the rest

Character Naming

- Official names - LATIN SMALL LETTER A
- Unofficial names - a
- Lispified names - LATIN-SMALL-LETTER-A
- **`#\a`**, **`#\|LATIN SMALL LETTER A|`**, **`#\LATIN-SMALL-LETTER-A`**
- Lisp names - RETURN, LINEFEED

Character Naming in Java

- 4 interfaces
  - `lisp.common.type.Character`
    - `lisp.common.type.BaseChar`
      - `lisp.common.type.StandardChar`
    - `lisp.common.type.ExtendedChar`
- Standard chars available as static fields in `StandardChar`
  - `public static final Character a;`
  - `public static final Character slash;`

Loading Character Database

- XML file derived from Unicode database
  - Approx 15,100 chars
  - Contains all names, code points, etc
- Loaded on startup
  - All chars are singleton objects
  - Stored in a hash map by code point, all names
- Factory class is always a lookup

Character I/O Streams

- Lisp character I/O streams extend the Java buffered Reader and Writer classes
- Necessary to specify the input encoding
  - Java system default if not specified
  - No “guessing” function implemented

Other CLs and Unicode

Comparison Table

- 4 Common Lisp Implementations
  - Allegro (Franz), CLisp, LispWorks, CLforJava
- 16 aspects

General		*File Encoding*	*Characters*	*Strings*
Unicode level	Base Char definition	System default	Reader support	Reader support
Comparison algorithm	Printing support	Discovery support	Comparison algorithm	Comparison algorithm
Custom Collation	Locale support	Available encodings	Printing support	Printing support
Char Width				

The Highlights

- Allegro and CLforJava support
  - Unicode 4, Naming, and Collation
- Allegro and LispWorks support encoding discovery
- CLforJava only one to escape Unicode chars in strings
- Each has a different definition of **`base-char`**

Proposal for Unicode in the Common Lisp Standard

> "Someone had to do it."
> - *Michael Palin*

Components of the Proposal

- Characters - type, naming, properties, functions
- Strings - types, encoding, functions
- The Reader - read macros, strings, numbers
- The Printer - characters, strings, direction, line breaks, char width
- Character I/O - types, functions, locales

Characters

Characters - Types

- Retain the current Standard-Char definition
- Retain the current Extended-char definition
  - **`(not base-char)`**
- Redefine Base-Char to conform to the Unicode definition of base character
  - Canonical Combining Class value of **`0`**

Characters - Naming

- Characters accessible via their Unicode name
  - `(name-char “LATIN SMALL LETTER A”) => #\a`
  - `(char-name #\|LATIN SMALL LETTER A|) => “LATIN SMALL LETTER A”`
- Unicode names are also lispified by ‘-’
  - `LATIN-SMALL-LETTER-A`
- Standard-Chars retain their legacy names as well
- Characters have a ‘preferred’ name

Characters - Properties

- Unicode chars have a wealth (49) of properties
  - Digit, whitespace, direction, combining, etc
- Functions, macros, and constants for support
  - char-available-properties =>
    list of all char properties
  - char-properties *char* =>
    property list for the char
  - getf *char indicator* &optional *default* =>
    value of the indicated property
  - maximum-surrogate-code-point
    minimum-surrogate-code-point -
    values of the high/low surrogate code points

Characters - Modified Fns

- Comparison functions conform to the 2 types of equivalence and of decomposition
- **`char=`** and **`char>`** (and similar) compare characters after canonical decomposition
- **`char-equal`** and **`char-greaterp`** (and similar) compare characters after compatibility decomposition. Also, it is case-insensitive.

Characters - Modified Fns

- **`char-code`**, **`char-int`** ***`char`*** => code-point (an integer)
  `code-char` ***`code-point`*** => character at that code point
- **`char-name`** ***`char`*** => returns the preferred name of the character. The preferred name can be changed to another of the char names by **`setf`**.
- **`digit-char-p`** ***`char`*** &optional ***`radix`*** => true if its <u>digit</u> property is true. Radix is honored except for Roman numerals.
- **`alpha-char-p`** ***`char`*** => true if its <u>letter</u> property is true.
- **`graphic-char-p`** ***`char`*** => true if char is not <u>ignorable</u>
- **`code-char-limit`** upper bound for code points for the supported Unicode level (v4 is **`#x10FFFF`**)

Characters - New Fns

- **`char-names char`** => list of names of the char. The first name is the preferred name.
- **`char-compose`** ***`base-char`*** *&rest* ***`extended-chars`***

  => a compatibility composed char

Strings

Strings - Types

- **`base-string`** contains only **`base-char`**s (current)
- Implications of this restriction
  - Does not contain any combining chars
  - Affects alterations of **`base-string`**s and coercion to a **`base-string`**
- Insertion of an **`extended-char`** changes the preceding **`base-char`**
  - Composed on the fly

Strings - Encoding

- Standard does not specify an internal encoding
- It must support all of the updated and new functions
- Common choices would be UTF-8 and UTF-16

Strings -Modified Fns

- String comparision - similar to Character compare
- **`string=`**, **`string<`**, etc use canonical decomposition and either binary or locale-based comparison (Unicode NFC)
- **`string-equal`**, **`string-lessp`**, etc use compatibility decomposition for equivalence or locale-based comparison (Unicode NFKC)
- Implementations may support sort keys (pre-computed comparison key)

Strings - New Fns

- Support for Unicode decomposition and composition algorithms
- **`string-decompose-canonical`** ***string***
  => new string in NFD form
- **`string-decompose-compatible`** ***string***
  => new string in NFKD form
- **`string-compose-canonical`** ***string***
  => new string in NFC if string is in NFD form
  or
  => new string in NFKC if string is in NFKD form

The Reader

Reader - The Basics

- The Reader is always presented with Unicode characters
  - Reader never has to translate
- Affects the stream functions (e.g. **`read-char`**)

Reader - Read Macros

- **`#\`**
  - Supports the Unicode char names and their lispified form
- **`#U`**, **`#U+`**
  - Takes 4 or 6 hex digits representing the code point of the char
- `""` - the string read macro
  - Works as now, but recognizes **`#U`** and **`#U+`** read macros embedded in the string

Reader - Numbers

- Potential numbers
  - Definition includes any character whose ‘digit’ property is true - includes Roman numerals
- Legal integer numbers must come from the same Unicode block
  - E.g. can’t mix European (`1, 2`...) with Devanagari (१, २ ...)
  - Question of hex definition (**`#x१२FF`**)
- Recognizes ratio characters (⅔, ⅘)
  - **8⅔** => **`26/3`**

The Printer

Printer - *Print-Escape*

- Characters
    - If `nil`, the character is sent uninterpreted to the stream
        - Stream encoding may lose information
    - Otherwise, character is printed using `#\` notation

Printer - *Print-Escape*

- Strings
  - If `nil`, the string is composed (NFC or NFKC) and the characters are sent to the output. The printer must honor bi-directional information. This may also require mirroring.
  - Otherwise, the characters are streamed in memory order between “ ”. If the stream encoding supports a char, the char is streamed. If not, the char is escaped using `#U` or `#U+` syntax.
  - Ignorable chars are always passed

Pretty Printer

- All of the behavior for the Printer
- Pretty Printer must also conform to
  - Unicode line break algorithm to determine potential line break locations
  - Char width information
    - Unicode chars may be zero, half, or full width characters - **`format`**

Character I/O

Character I/O - Types

- **`encoding`**
  - A CLOS class that translates between Unicode encoding and some other encoding (e.g ISO-8859-1)
  - An **`encoding`** instance may be passed to the **`open`** function’s **`:external-format`** parameter
  - An **`encoding`** instance is one of the IANA recognized encodings or an implementation-specific encoding
  - **`Encodings`** may be combined in a stream

Character I/O - Modified Fns

- **`open`**
  - **`:external-format`** arg takes an **`encoding`**
    - Current **`*locale*`** provides a default
  - **`:probe`** argument
    - Returns a stream that contains an **`encoding`**
- **`probe-file`**
  - Returns a second value that is the file **`encoding`**
- **`read-char`** returns a valid Unicode character

Character I/O - New Fns

- **`list-encodings`** => returns a list of the encodings supported by this implementation
- **`encoding-name`** ***`encoding`*** => name of the **`encoding`**
- **`stream-encoding`** ***`stream`*** => encoding of the stream

Summary

Unicode
Integration Implications

- Goes beyond just adding some characters
- Pervasive effects in major subsystems
    - Characters, Strings
    - Reader, Printer
    - Character I/O
    - Sorting, comparisons

Unicode Implications

- It’s so complex an issue...
  - Small differences in implementation can disrupt portability
- What to do?
  - Update the Common Lisp standard
- Give it a name - How about...?

Common Lisp 2006

☠ Optimist! ☢

A Demo!

There’s a Discussion Forum

- http://clforjava.cs.cofc.edu/forum/
- Go to the “Dealing with Unicode” board
- There’s even a voting system built in

Q & A