Scone Knowledge Base for “Read the Web” Project

Scott E. Fahlman
Research Professor, LTI & CSD
sef@cs.cmu.edu

What is a Knowledge Base?

- A symbolic knowledge base (KB) system is like a database, except:
  - Designed to hold less regular patterns.
    - Entities in a type hierarchy. Multiple inheritance and exceptions.
    - Statements about these entities.
    - User-defined roles (slots) and relations.
    - Type restrictions on slots.
  - Support for simple inference.
    - Inheritance (respecting cancellation).
    - Type checking (splits).
    - Simple rules.
    - For more complex inference, attach and trigger Lisp functions.
  - Support for search.
    - Find all known entities that have some set of features.

Scone

- Scone is a software knowledge-base system with emphasis on performance, scalability, and ease of use.
  - Engine in Common Lisp, runs as a server.
  - Run it file-to-file, interactive, or use Java “gateway” by Ben Lambert.
  - Should be able to handle a few million entities and statements on a high-end workstation.
  - We have an AMD-64 Linux server with 8GB, will send out info on “rtw-guest” account.
- Scone is now under development.
  - Engine, some initial KBs, Programmer level manual now exist.
  - Has been used some, but this course will be a big test.
  - Will soon be released as open-source.
- Currently, knowledge is added in a specialized entry language (Lisp).
- Work is under way to parse simple English sentences into Scone.

Role of Scone in RtW

- Place to store the information your systems extract.
- Initial Scone ontology establishes a common vocabulary for entities and propositions.
- Simple type-checking of entities and slot-fillers.

Advanced:

- More complex sanity checking.
- Background knowledge of all kinds.

Elements and Names

- Each individual, type, and statement in Scone is an “element”.
  - Unique name surrounded by curly braces. {Scott Fahlman}
  - This represents the *concept*.
- “English” names are Lisp strings: “S. E. Fahlman” or “Scott”.
  - Could be a multi-word phrase like “Read-the-Web course”.
  - Normally preserve case, but ignore it during matching.
  - Syntax tags (optional).
- An element may be associated with several names.
- A name may be associated with several elements, so it may be necessary to disambiguate – perhaps just ask the user, or perhaps be more clever.
- Someday we will support multiple natural languages and sub-languages.
  - Unicode strings with labeled word-meaning links.
  - Sub-languages like “techy English”.

Splits

- A type or instance can have multiple immediate superiors. Inherits from all of them.
- Create split-sets to indicate that N types are disjoint – no members in common.
- `(new-split-types {person}`
  `'({male} {female}))`
- `(new-split-types {person}`
  `'({child} {adult}))`
- `(new-intersection-type {boy}`
  `'({child} {male}))`
- These splits are efficiently checked every time a new element is created.
- Also checked by `can-x-be-a-y?` function.

Example: Type Hierarchy

- **;;; Create the “room” type.**
- (add-type {room} {place})

- **;;; Every room has a size.**
- (add-indv-role {room size} {room} {area measure})

- **;;; Rooms come in various flavors.**
- (add-split-subtypes {room}
  ‘({classroom} {office} {lab}))
- (add-split-subtypes {room}
  ‘({window room} {non-window room}))

- **;;; Create an individual room.**
- (add-indv {Wean 8214} {office})
- (add-is-a {Wean 8214} {window room})
- (the-x-of-y-is-z {room size} {Wean 8214}
  (add-measure 180 {square foot}))

Special User-Defined “Add” Functions

- (add-person “Elvis Aaron Presley”
  :birthday “January 1 1935”
  :profession {entertainer}
  :death-date “6/32/77”
  :shoe-size 10.5
  :shoe-size-certainty .85
  :nicknames ‘(“the King” “tubby”))
- This can parse subfields for dates, etc., then adds multiple elements to Scone, checking for contradictions.
- Adds all common forms of the name: “Elvis”, “E. A. Presley”, “Elvis A. Presley”, plus nicknames, all referring to {elvis aaron presley} concept (indv element in Scone).

Example: User-Defined Relation

- **;;; Create the "adjacent to" relation.**
- (add-relation {adjacent to}
  :a-instance-of {place}
  :b-instance-of {place}
  :symmetric t)

- **;;; Make some statements.**
- (add-statement {Wean 8212} {adjacent to} {Wean 8214})
- (add-statement {Wean 8214} {adjacent to} {Wean 8216})

- **;;; Some queries:**
- (statement-true? {Wean 8214} {adjacent to} {Wean 8212}) => T
- (statement-true? {Wean 8212} {adjacent to} {Wean 8214}) => T
- (statement-true? {Wean 8212} {adjacent to} {Wean 8216}) => NIL
- (show-rel {Wean 8214} {adjacent to})
- {Wean 8212}
- {Wean 8216}

Example: Queries

- (is-x-a-y? {Wean 8214} {office}) => T
- (is-x-a-y? {Wean 8214} {room}) => T
- (can-x-be-a-y? {Wean 8214} {lab}) => NIL
- (show-the-x-of-y {room size} {Wean 8214})
- {180 square foot}
- (show-all-instances {classroom})
- {Wean 5409}
- {NSH 1304}
- … and 120 more.

Contexts

- Scone has an efficient, lightweight way to create distinct mini-worlds and to reason within them.
- Every entity in Scone exists in a certain *context*, and every statement is true in a certain context.
- Contexts form a hierarchy, so it is easy to clone one, then make a few specific changes.
  - {Harry Potter World} is like {England today}, but with some specific changes.
- Can create a hypothetical context, reason about it, then destroy or abandon it.
- This mechanism is good for representing “X said…”, “X believes…”, “X wants a world in which…”, etc.
- Also good for representing changes in the world due to some event: before and after contexts.

Interacting with Scone (Ben Lambert)

- You can interact with Scone
  - Interactively, through the Lisp read/eval/print loop, …or…
  - through a TCP/IP socket.
- Scone is single-user (for now), so each group will need to start its own server.
- More details on starting up a server will come soon.
- We have an example class in Java for talking to Scone via TCP/IP (SconeClient.java)

Interacting with Scone (cont.)

- Once you are connected, Scone will send a “[PROMPT]” to tell you it’s ready to receive a command.
- You can then send a command to Scone terminated by a newline (and no other newline characters)
  - E.g. “(new-indv {Tom Mitchell} {person})\n”
- Scone will always send a reply. Occasionally it’s more than one line, so keep reading until you get another “[PROMPT]”
- When you’re done send: “(disconnect)”

Example interaction with Scone

<TCP/IP Connection established>

Server: [PROMPT]

Client: (new-indv {Tom Mitchell} {person})

Server: {common:Tom Mitchell}

Server: [PROMPT]

Client: (new-indv {Elvis Presley} {person})

Server: {common:Elvis Presley}

Server: [PROMPT]

Client: (disconnect)

<Connection closed>