A review of First-Order Logic

Using KIF

Knowledge Interchange Format

- KIF ~ First order logic theory
- An interlingua for encoded knowledge
  - Takes translation among n systems from O(n^2) to O(n)
- Common language for reusable knowledge
  - Implementation independent semantics
  - Highly expressive - can represent knowledge in typical application KBs.
  - Translatable - into and out of typical application languages
  - Human readable - good for publishing reference models and ontologies.
- Current specification at http://logic.stanford.edu/

Know. Base in Lang1
KIF <-> Lang1 Translator
Sys 1
Know. Base in Lang2
KIF <-> Lang2 Translator
Sys 2
Know. Base in KIF
Library
KIF
Know. Base in Lang3
KIF <-> Lang3 Translator
Sys 3

KIF Syntax and Semantics

- Extended version of first order predicate logic
- Simple list based linear ASCII syntax, e.g.,
  (forall x (=> (P x) (Q x)))
  (exists ?person (mother mary ?person))
  (=> (apple x) (red x))
  (<= (father x ?y) (and (child ?x ?y) (male ?x)))
- Model theoretic semantics
- KIF includes an axiomatic specification of large function and relation vocabulary and a vocabulary for numbers, sets, and lists

KR Language Components

- A logical formalism
  - Syntax for wffs
  - Vocabulary of logical symbols
  - Interpretation semantics for the logical symbols
  - E.g., (=> (Person ?x) (= (Gender (Mother ?x)) Female)))
- An ontology
  - Vocabulary of non-logical symbols
    - Relations, functions, constants
  - Axioms restricting the interpretations of the symbols
    - E.g., (=> (Person ?x) (= (Gender (Mother ?x)) Female)))
- A proof theory
  - Specification of the reasoning steps that are logically sound
    - E.g., (=> S1 S2) and S1 entails S2
Conceptualization

- Universe of discourse
  - Set of objects about which knowledge is being expressed
- Object
  - Concrete: Clyde, my car
  - Abstract: Justice, 2
  - Primitive: Resister
  - Composite: Electric circuit
  - Fictional: Sherlock Holmes

Blocks World

- Objects: a, b, c, d, e, table

Relations and Functions

- Relation
  - Set of finite lists of objects
    - E.g., Parent: (Richard Earl) (Richard Polly) (Debbie Don) ...
  - Mapping: <list of objects> → <truth value>
- Function
  - Relation that associates a unique nth element with a given n-1 elements
    - E.g., +: (1 3 4) (17 23 40) (2 7 10 12 31) ...
  - Referred to as (arg1, arg2, ..., argk, value)
  - Mapping: <list of objects> → <object>

Blocks World

- Objects: a, b, c, d, e, table
- Relations
  - Above: [(a b) (a c) (b c) (d e)]
  - Clear: [(a) (d)]
  - Table: [(c) (e)]
- Functions
  - On: [(a b) (b c) (d e)]
Predicate Calculus - KIF

- Knowledge Base - Collection of sentences
- Sentence - Expression denoting a statement
- Term - Expression denoting an object
- Objects always in the conceptualization
  - Words
  - Complex numbers
  - All finite lists of objects
  - All sets of objects
  - ^ (bottom)

Declarative Semantics

- Interpretation -
  - <object constant> => <object>
  - <logical constant> => <truth value>
  - <relation constant> => (<tuple of objects> → <truth value>)
  - <function constant> => (<tuple of objects> → <object>)
- Variable assignment -
  - <individual variable> => <object>
  - <sequence variable> => <finite sequence of objects>
- Semantic value - <term> => <object>
  - Defined in terms of an interpretation and variable assignment
- Truth value - <sentence> => {true, false}
  - Defined in terms of an interpretation and variable assignment
- Version of a variable assignment
  - V' is a version of a variable assignment V with respect to variables var1,...,varn if and only if V' agrees with V on all variables except for var1,...,varn.

Constants, Individual Variables, Function Terms

- Constant - Word
  - E.g., Fred, Block-A, Justice
  - SIV(<constant>) = I(<constant>)
- Individual Variable - Word beginning with “?”
  - E.g., ?x, ?The-Murderer
  - SIV(<individual variable>) = V(<individual variable>)
- Function Term
  - <function constant> <term>* [sequence variable]
    - E.g., (plus 2 3) (Father-Of Richard)
    - SIV((fn term1 ... termmn)) = I(fn)[SIV(term1) ... SIV(termmn)]
    - SIV((fn term1 ... termmn @var)) =
      - I(fn)[SIV(term1) ... SIV(termmn) | V(@var)]

List Terms and Set Terms

- List Term
  - (listof <term>* [<seqvar>])
    - E.g., (listof A B C) (listof A ?second @rest)
    - SIV((listof term1 ... termmn)) = SIV(term1), ..., SIV(termnn)
    - SIV((listof term1 ... termmn @var)) =
      - SIV(term1), ..., SIV(termnn) | V(@var)
- Set Term
  - (setof <term>* [<seqvar>])
    - E.g., (setof A B C) (setof A ?X @Z)
    - SIV((setof term1 ... termmn)) = SIV(term1), ..., SIV(termnn)
    - SIV((setof term1 ... termmn @var)) =
      - SIV(term1), ..., SIV(termnn) U {x | (Si) x = SIV(nth(@var i))}
Logical Terms

- (if <sentence> <term> [<term>])
  - E.g., (if (Above A B) A B)
  - SIV((if sent term)) =
    - SIV(term) when TIV(sent) = true
    - ^ otherwise
  - SIV((if sent term1 term2)) =
    - SIV(term1) when TIV(sent) = true
    - SIV(term2) otherwise

- (cond (<sentence> <term>) … (<sentence> <term>))
  - E.g., (cond ((Above A B) A) ((Above B A) B))
  - SIV((cond (sent1 term1) … (sentn termn))) =
    - SIV(term1) when TIV(sent1) = true
    - …
    - SIV(termn) when TIV(sentn) = true
    - ^ otherwise

Quantified Terms

- Set Forming Term - (setofall <term> <sentence>)
  - E.g., (setofall ?block (Above ?block A))
  - SIV((setofall term sent)) =
    - SIV(term) when TIV(sent) = true
    - ^ otherwise

- Designator - (the <term> <sentence>)
  - E.g., (the ?block (Above ?block A))
  - SIV((the term sent)) =
    - SIV'(term) when
      - V' is a version of V wrt the variables in term, and
      - TIV'(sent) = true, and
      - SIV''(term) = SIV'(term)
    - ^ otherwise

Logical Constants, Equations, Inequalities

- Logical constant
  - Tiv(constant) = I(constant)
  - Tiv(true) = true
  - Tiv(false) = false

- Equations - (= <term> <term>)
  - E.g., (= (Father Richard) Earl) (= A B)
  - TIV((= term1 term2)) =
    - true when SIV(term1) and SIV(term2) are the same object
    - false otherwise

- Inequalities - (/= <term> <term>)
  - E.g., (/= (Father Richard) (Father Bob)) (= A B)
  - TIV((/= term1 term2)) = TIV((not (= term1 term2)))

Relational Sentences

- (<relation constant> <term>* [<sequence variable>])
  - E.g., (Parent Richard Earl) (Clear A) (Set-Partition Set1 @Sets)
  - TIV((rel term1 … termn)) =
    - true when I(rel)[SIV(term1), …, SIV(termn)] is true
    - false otherwise

- (<function constant> <term>* <term>)
  - E.g., (Father Richard Earl) (Plus 2 5 7)
  - TIV((fun arg1 … argn val)) =
    - true when I(fun)[SIV(val)] is true
    - false otherwise
Logical Sentences: **not, and, or**

- **Negation** - (not <sentence>)
  - E.g., (not (On A D)) (not (On B B))
  - TIV((not sent)) =
    - true when TIV(sent) is false
    - false otherwise

- **Conjunction** - (and <sentence>*)
  - E.g., (and (On A B) (On B C))
  - TIV((and sent1 … sentn)) =
    - true when TIV(senti) is true for all i=1,…,n
    - false otherwise

- **Disjunction** - (or <sentence>*)
  - E.g., (or (On A D) (On A B))
  - TIV((or sent1 … sentn)) =
    - true when TIV(senti) is true for some i=1,…,n
    - false otherwise

---

Universally Quantified Sentences

- **(forall <individual variable> <sentence>)**
  - E.g., (forall ?b1 (not (On ?b1 ?b1)))
  - TIV((forall ?var sent)) =
    - true when TIV'(sent) = true
    - for all versions V' of V with respect to variable ?var
    - false otherwise

- **(forall <individual variable>* <sentence>)**
  - E.g., (forall (?b1 ?b2) (=> (On ?b1 ?b2) (Above ?b1 ?b2)))
  - TIV((forall ?var1 … ?varn sent)) =
    - true when TIV'(sent) = true
    - for all versions V' of V with respect to ?var1 … ?varn
    - false otherwise

---

Existentially Quantified Sentences

- **(exists <individual variable> <sentence>)**
  - E.g., (exists ?b (or (On ?b1 table) (exists ?b2 (On ?b1 ?b2))))
  - TIV((exists ?var sent)) =
    - true when TIV'(sent) = true
    - for some version V' of V with respect to variable ?var
    - false otherwise

- **(exists <individual variable>* <sentence>)**
  - E.g., (exists (?b1 ?b2) (and (On ?b1 ?b2) (Above ?b1 ?b2)))
  - TIV((exists ?var1 … ?varn sent)) =
    - true when TIV'(sent) = true
    - for some version V' of V with respect to ?var1 … ?varn
    - false otherwise

E.g., (or (On ?b1 table) (exists ?b2 (On ?b1 ?b2)))
An Example: Digital Circuit C1

Russell and Norvig, Figure 8.1

Domain Conceptualization

- Objects
  - Circuits
  - Terminals
  - Signals
  - Gates
  - Gate types
  - Signal values
- Relations
  - Connected: (<terminal> <terminal>)
- Functions
  - Type: <gate> → <gate type>
  - In: (<index> <gate>) → <input terminal>
  - Out: (<index> <gate>) → <output terminal>
  - Signal: <terminal> → <signal value>

Electronic Circuit Domain Theory

- Connected terminals have the same signal
  
  \[ (\Rightarrow (\text{Connected} ?t1 ?t2) = (\text{Signal} ?t1) = (\text{Signal} ?t2)) \]

- Signal at terminal is either on or off
  
  \[ (\text{or} = (\text{Signal} ?t) \text{On}) = (\text{Signal} ?t) \text{Off}) \]
  
  \[ (\text{or} = (\text{Signal} ?t) \text{On}) = (\text{Signal} ?t) \text{Off}) \]
  
  \[ (\text{not} = (\text{On} \text{Off})) \]

- Connected is commutative
  
  \[ (\Leftrightarrow (\text{Connected} ?t1 ?t2) = (\text{Connected} ?t2 ?t1)) \]

OR and AND Gates

- OR gate’s output is on when any of its inputs are on
  
  \[ (\Rightarrow = (\text{Type} ?g) \text{OR}) \]
  
  \[ (\Leftrightarrow = (\text{Signal} (\text{Out} 1 ?g)) \text{On}) \]
  
  \[ (\exists ?i = (\text{Signal} (\text{In} ?i ?g)) \text{On}) \]

- AND gate’s output is off when any of its inputs are off
  
  \[ (\Rightarrow = (\text{Type} ?g) \text{AND}) \]
  
  \[ (\Leftrightarrow = (\text{Signal} (\text{Out} 1 ?g)) \text{Off}) \]
  
  \[ (\exists ?i = (\text{Signal} (\text{In} ?i ?g)) \text{Off}) \]
**XOR and NOT Gates**

- **XOR gate**’s output is on when its inputs are different
  
  \[
  \Rightarrow \; (= \; (\text{Type} \; ?g) \; \text{XOR}) \]
  
  \[
  \Leftrightarrow \; (= \; (\text{Signal} \; (\text{Out} \; 1 \; ?g)) \; \text{On})
  \]
  
  \[
  (\text{not} \; (= \; (\text{Signal} \; (\text{In} \; 1 \; ?g)) \; (\text{Signal} \; (\text{In} \; 2 \; ?g)))\))
  \]

- **NOT gate**’s output is different from its inputs
  
  \[
  \Rightarrow \; (= \; (\text{Type} \; ?g) \; \text{NOT})
  \]
  
  \[
  (\text{not} \; (= \; (\text{Signal} \; (\text{Out} \; 1 \; ?g)) \; (\text{Signal} \; (\text{In} \; 1 \; ?g))))\))
  \]

**Circuit C1 Representation**

- **Gates**
  
  \[
  (= \; (\text{Type} \; X1) \; \text{XOR}) \quad (= \; (\text{Type} \; X2) \; \text{XOR})
  \]
  
  \[
  (= \; (\text{Type} \; A1) \; \text{AND}) \quad (= \; (\text{Type} \; A2) \; \text{AND})
  \]
  
  \[
  (= \; (\text{Type} \; O1) \; \text{OR})
  \]

- **Connections**
  
  \[
  (\text{Connected} \; (\text{Out} \; 1 \; X1) \; (\text{In} \; 1 \; X2)) \quad (\text{Connected} \; (\text{In} \; 1 \; C1) \; (\text{In} \; 1 \; X1))
  \]
  
  \[
  (\text{Connected} \; (\text{Out} \; 1 \; X1) \; (\text{In} \; 2 \; A2)) \quad (\text{Connected} \; (\text{In} \; 1 \; C1) \; (\text{In} \; 1 \; A1))
  \]
  
  \[
  (\text{Connected} \; (\text{Out} \; 1 \; A2) \; (\text{In} \; 1 \; O1)) \quad (\text{Connected} \; (\text{In} \; 2 \; C1) \; (\text{In} \; 2 \; X1))
  \]
  
  \[
  (\text{Connected} \; (\text{Out} \; 1 \; A1) \; (\text{In} \; 2 \; O1)) \quad (\text{Connected} \; (\text{In} \; 2 \; C1) \; (\text{In} \; 2 \; A1))
  \]
  
  \[
  (\text{Connected} \; (\text{Out} \; 1 \; X2) \; (\text{Out} \; 1 \; C1)) \quad (\text{Connected} \; (\text{Out} \; 1 \; O1) \; (\text{Out} \; 2 \; C1))
  \]
  
  \[
  (\text{Connected} \; (\text{Out} \; 1 \; A2) \; (\text{Out} \; 1 \; C1)) \quad (\text{Connected} \; (\text{Out} \; 3 \; C1) \; (\text{In} \; 2 \; X2))
  \]
  
  \[
  (\text{Connected} \; (\text{Out} \; 2 \; O1) \; (\text{Out} \; 2 \; C1)) \quad (\text{Connected} \; (\text{In} \; 3 \; C1) \; (\text{In} \; 1 \; A2))
  \]

**Knowledge About Knowledge**

- **KIF represents knowledge about knowledge** by allowing expressions to be treated as objects in the universe of discourse

- **KIF expressions** are lists and can be referred to using the `quote` operator
  
  \[
  \Rightarrow \; (\text{believes} \; \text{John} \; \text{material moon bleuchese})
  \]
  
  \[
  \Rightarrow \; (\text{believes} \; \text{john} \; ?p) \; (\text{believes} \; \text{mary} \; ?p)
  \]

  or using the `listof` operator
  
  \[
  \Rightarrow \; (\text{believes} \; \text{John} \; (\text{listof} \; \text{material} \; ?x \; ?y))
  \]
  
  \[
  (\text{believes} \; \text{Lisa} \; (\text{listof} \; \text{material} \; ?x \; ?y))
  \]

- **Vocabulary is available for evaluating an expression**
  
  \[
  (= \; (\text{denotation} \; (\text{listof} \; \text{F} \; ?x \; ?y)) \; (\text{F} \; ?x \; ?y))
  \]
  
  \[
  \Rightarrow \; (\text{sentence} \; ?p) \; (\text{true} \; (\text{listof} \; \Rightarrow \; ?p \; ?p))
  \]

**Big KIF and Little KIF**

- **That KIF is highly expressive language** is a desirable feature; but there are disadvantages.
  - complicates job of building fully conforming systems.
  - resulting systems tend to be “heavyweight”

- **KIF has “conformance categories”** representing dimensions of conformance and specifying alternatives within that dimension.

- **A “conformance profile”** is a selection of alternatives from each conformance category.

- System builders decide upon and adhere to a conformance profile sensible for their applications.
Conformance Categories and Profiles

- **Conformance Categories**
  - **logical form:** {atomic, conjunctive, positive, logical, rule-based, quantified}
  - **recursion:** yes/no
  - **terms:** {constants, variables, complex terms}
  - **relational variables:** yes/no

- **Common Conformance Profiles might be**
  - Databases (ground atomic assertions & conjunctive forms)
  - Datalog
  - Relational logic
  - First order logic
  - Second order logic

KIF vs ANSI KIF

- **KIF is the object of an ANSI Ad Hoc standardization group (X3T2)**
- **ANSI KIF is somewhat different from previous specs**
  - No non-monotonic rules
  - Allow for possible (future) higher order extensions
  - Defines a standard infix format for presenting KIF

KIF Software

- Several KIF based reasoners in LISP are available from Stanford (e.g., EPILOG).
- IBM’s ABE (Agent Building Environment) & RAISE reasoning engine use KIF as their external language.
- Stanford’s Ontolingua uses KIF as its internal language.
- Translators (partial) exist for a number of other KR languages, including LOOM, Classic, CLIPS, Prolog,...
- Parsers for KIF exist which take KIF strings into C++ or Java objects.