A review of
First-Order Logic

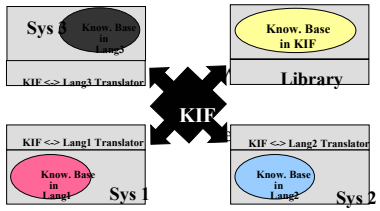
Using KIF

Knowledge Interchange Format

Material adapted from
Professor Richard Fikes
Stanford University

Knowledge Interchange Format

- KIF ~ First order logic theory
- An **interlingua** for encoded knowledge
  - Takes translation among n 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/

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)))
(exisits ?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

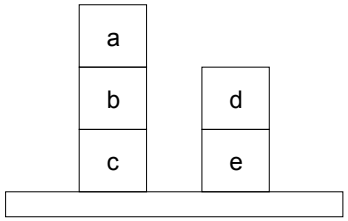
- A logical formalism
  - Syntax for wffs
  - Vocabulary of logical symbols
  - Interpretation semantics for the logical symbols
  - E.g., (=> (Person ?x) (= (Gender (Mother ?x)) Female)))
- An ontology
  - Vocabulary of non-logical symbols
    - Relations, functions, constants
  - Axioms restricting 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

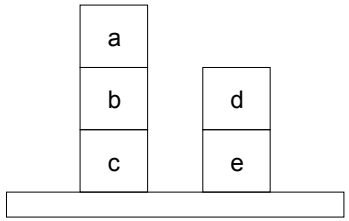


Relations and Functions

- 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 … termn)) = I(fn)[SIV(term1) … SIV(termn)]
  - SIV((fn term1 … termn @var)) =
  - I(fn)[SIV(term1) … SIV(termn) | 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 … termn)) = <SIV(term1), …, SIV(termn)>
  - SIV((listof term1 … termn @var)) =
  - <SIV(term1), …, SIV(termn) | V(@var)>
- Set Term
  - (setof <term>* [<seqvar>])
    - E.g., (setof A B C) (setof A ?X @Z)
  - SIV((setof term1 … termn)) = {SIV(term1), …, SIV(termn)}
  - SIV((setof term1 … termn @var)) =
  - {SIV(term1), … ,SIV(termn)} U {x | ($i) 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) | TIV'(sent) = true}

  for all versions V' of V wrt the variables in term
- 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)

    for all versions V'' of V such that TIV''(sent) = true

    ^ 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
  - TIV((rel term1 … termn @var)) =
    - true when I(rel)[SIV (term1), …, SIV (termn) | SIV (@var)] 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(arg1), …, SIV (argn)] = SIV(val)
    - 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

Logical Sentences: => <= <=>

- Implication - (=> <sentence>* <sentence>)
  - E.g., (=> (On A B) (On B C))
  - TIV((=> ante1 … anten conse)) =
    - true when:
      - TIV(antei) is false for some i=1,…,n; or
      - TIV (conse) is true
    - false otherwise
      - E.g., (=> (On A D) (On D D))
  - TIV((=> a1 … an c)) = TIV ((or (not a1) … (not an) c))
- Implication - (<= <sentence> <sentence>*)
- Equivalence - (<=> <sentence> <sentence>)
  - TIV ((sent1 <=> sent2)) =
    - true when TIV (sent1) = TIV (sent2)
    - false otherwise
  - TIV((<=> s1 s2)) = TIV((and (=> s1 s2) (=> s2 s1)))

Universally Quantified Sentences

- (forall <individual variable> <sentence>)

  E.g, (forall ?b (not (On ?b ?b)))

  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

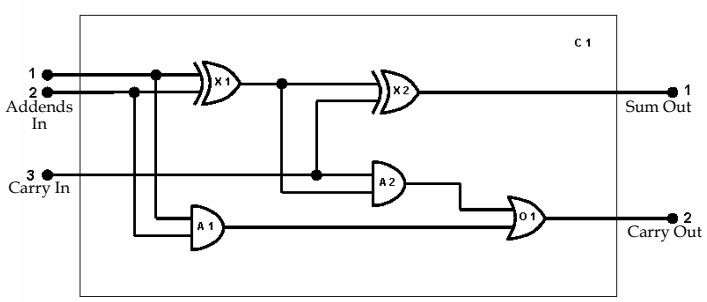
  for all versions V′ of V with respect to ?var1 … ?varn

  false otherwise

Existentially Quantified Sentences

- (exists <individual variable> <sentence>)
  - E.g, (forall ?b1 (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 ?A) (Above ?A ?b2)))
  - TIV ((exists (?var1 … ?varn) sent)) =
    - true when TIV′(sent) = true
    - for some version V′ of V with respect to ?var1 … ?varn
    - false otherwise
- forall not in the scope of an exists may be omitted
  - 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

  (=> (Connected ?t1 ?t2) (= (Signal ?t1) (Signal ?t2)))
- Signal at terminal is either on or off

  (or (= (Signal ?t) On) (= (Signal ?t) Off))

  (or (Signal ?t On) (Signal ?t Off)))

  (not (= On Off))
- Connected is commutative

  (<=> (Connected ?t1 ?t2) (Connected ?t2 ?t1))

OR and AND Gates

- OR gate's output is on when any of its inputs are on

  (=> (= (Type ?g) OR)

  (<=> (= (Signal (Out 1 ?g)) On)

  (exists ?i (= (Signal (In ?i ?g)) On)))
- AND gate's output is off when any of its inputs are off

  (=> (= (Type ?g) AND)

  (<=> (= (Signal (Out 1 ?g)) Off)

  (exists ?i (= (Signal (In ?i ?g)) Off)))

XOR and NOT Gates

- XOR gate's output is on when its inputs are different

```
(=> (= (Type ?g) XOR)
    (<=> (= (Signal (Out 1 ?g)) On)
         (not (= (Signal (In 1 ?g) (Signal (In 2 ?g))))))
```

- NOT gate's output is different from its inputs

```
(=> (= (Type ?g) NOT)
    (not (= (Signal (Out 1 ?g)) (Signal (In 1 ?g)))))
```


Circuit C1 Representation

- Gates

```
(= (Type X1) XOR)      (= (Type X2) XOR)
(= (Type A1) AND)      (= (Type A2) AND)
(= (Type O1) OR)
```

- Connections

```
(Connected (Out 1 X1) (In 1 X2))  (Connected (In 1 C1) (In 1 X1))
(Connected (Out 1 X1) (In 2 A2))  (Connected (In 1 C1) (In 1 A1))
(Connected (Out 1 A2) (In 1 O1))  (Connected (In 2 C1) (In 2 X1))
(Connected (Out 1 A1) (In 2 O1))  (Connected (In 2 C1) (In 2 A1))
(Connected (Out 1 X2) (Out 1 C1))      (Connected (In 3 C1) (In 2 X2))
(Connected (Out 1 O1) (Out 2 C1))      (Connected (In 3 C1) (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

```
(=> (believes John '(material moon bleucheese))
(=> (believes john ?p) (believes mary ?p))
```

or using the *listof* operator

```
(=> (believes John (listof 'material ?x ?y))
    (believes Lisa (listof 'material ?x ?y))
```

- Vocabulary is available for "evaluating" an expression

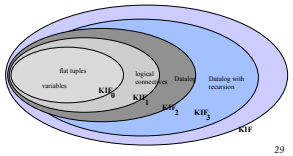
```
(= (denotation (listof 'F ?x ?y)) (F ?x ?y) )
(=> (sentence ?p) (true (listof '=> ?p ?p)))
```


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.