ECE 566

Representing Knowledge via Frames(structured objects)

Frames aggregate several related predicate calculus expressions into larger structures (sometimes called objects) that are identified as important objects in the domain.

Structure carries some of the representational and computational burden. Certain operations, which would have otherwise been performed by explicit rule application, can be performed in an automatic way.

Suppose we want to represent the following:

1. John gave Mary the book.
2. John is a programmer.
3. Mary is a lawyer.
4. John's address is 37 Maple Street

The following would be a predicate logic representation:

$$\begin{aligned}
&\text{Give}(\text{John}, \text{Mary}, \text{Book}) \wedge \\
&\text{Occupation}(\text{John}, \text{Programmer}) \wedge \\
&\text{Occupation}(\text{Mary}, \text{Lawyer}) \wedge \\
&\text{Address}(\text{John}, 37\text{-Maple-St})
\end{aligned}$$

Individual constant symbols refer to six entities, John, Mary, Book, Programmer, lawyer, 37-Maple-St, in this small database.

As the database grows (we add other entities, and other information about these same entities), it would be helpful to gather all of the facts about a given entity into a single group.

Ex: Facts associated with John:

John

Give (John, Mary, Book)

Occupation (John, Programmer)

Address (John, 37-Maple-St)

Similarly for Mary:

Mary

Give (John, Mary, Book)

Occupation (Mary, Lawyer)

There is nothing wrong to have a fact (such as giving) associated with two different entities.

In most object-centered representations only binary predicates are used to express facts about objects => we have to convert n-ary predicates in (n+1) binary predicates.

Ex: We need to convert the three-argument predicate Give (John, Mary, Book) into one involving binary predicates.

We create a class of events called Giving_Events, of which the action of giving the book to Mary by John is a member of (or element of). All Giving_Events have a recipient, an object, and a giver. Then, Give (John, Mary, Book) can be converted to:

$$\begin{aligned}&(\exists ?x)[\mathrm{EL}(?x,\mathrm{Giving\_Events}) \wedge \mathrm{Giver}(?x,\mathrm{John})\\ &\qquad \wedge \mathrm{Recip}(?x,\mathrm{Mary}) \wedge \mathrm{Obj}(?x,\mathrm{Book})]\end{aligned}$$

EL denotes set membership.

Skolemizing the existential variable in the above formula, we get:

$$\begin{aligned} & \mathrm{EL}(\mathrm{G1}, \mathrm{Giving\_Events}) \wedge \mathrm{Giver}(\mathrm{G1}, \mathrm{John}) \\ & \quad \wedge \mathrm{Recip}(\mathrm{G1}, \mathrm{Mary}) \wedge \mathrm{Obj}(\mathrm{G1}, \mathrm{Book}) \end{aligned}$$

where G1 is a particular name for our giving event.

=> We have converted the three-argument Give predicate into conjunction of four binary ones.

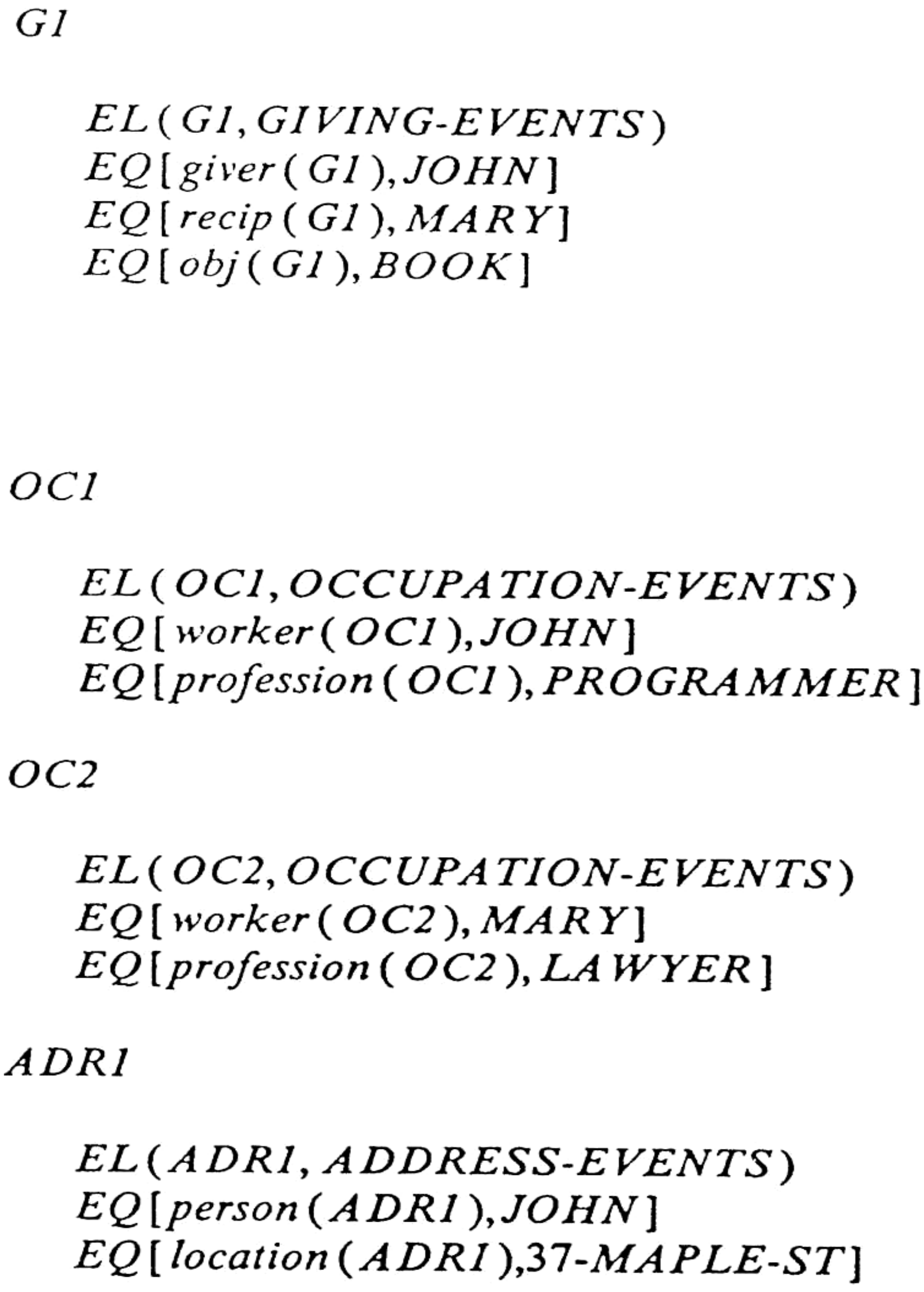
Using only binary predicates has advantages, most important is modularity.

Most object-centered representations use 3 generic predicates to represent all entities. These predicates are:

EL: Element of a set
SS: Something is subset of another set
EQ: An attribute is equal to a value

Instead of other binary predicates, then functions are used.

Ex: The original set of expressions can be represented as:



G1

EL(G1, GIVING-EVENTS)
EQ[giver(G1), JOHN]
EQ[recip(G1), MARY]
EQ[obj(G1), BOOK]

OC1

EL(OC1, OCCUPATION-EVENTS)
EQ[worker(OC1), JOHN]
EQ[profession(OC1), PROGRAMMER]

OC2

EL(OC2, OCCUPATION-EVENTS)
EQ[worker(OC2), MARY]
EQ[profession(OC2), LAWYER]

ADR1

EL(ADR1, ADDRESS-EVENTS)
EQ[person(ADR1), JOHN]
EQ[location(ADR1), 37-MAPLE-ST]

Note: The above "frames" share a common structure. First an EL predicate to describe which set the frame is a member of, (or this can be SS, if the unit itself is a set). Another term by which this predicate is sometimes referred to is instance-of. Second the values of different functions of the object.

Instead of EQ [giver(G1), John], we can simply use "giver: John". Then we get:

G1
Element-of: Giving_events
giver: John
recip: Mary
obj: Book

Constructs like "giver: John" are called slots, where giver is the slot-name, and John is the slot-value.

G1
Element-of: Giving_events
giver: John
recip: Mary
obj: Book

OC2
Element-of: Occupation_events
worker: Mary
profession: Lawyer

OC1
Element-of: Occupation_events
worker: John
profession: Programmer

ADR1
Element-of: Address_events
person: John
location: 37-maple-st

Other entities in our domain might similarly be described by the following units:

JOHN
Element-of: Persons

LAWYER
Element-of: Jobs

BOOK
Element-of: Phys_objs

MARY
Element-of: Persons

37-MAPLE-ST
Element-of: Addresses

PERSONS
Subset-of: Animals

PROGRAMMER
Element-of: Jobs

The value in a slot can also be a function, and not a constant.

Ex: G1

Element-of: Giving_events
giver: John
recip: Mary
obj: Book

G2

Element-of: Giving_events
giver: Bill
recip: recip(G1)
obj: Pen

which says "Bill gave the pen to the person to whom John gave the book".

We can also accommodate quantified variables.

Ex: John gave something to everyone.

$$\begin{aligned}(\forall ?x)(\exists ?y)(\exists ?z)\{&\mathrm{EL}(?y,\mathrm{Giving\_Events})\wedge\\ &\mathrm{EQ}[\mathrm{giver}(?y),\mathrm{John}]\wedge\\ &\mathrm{EQ}[\mathrm{obj}(?y),?z]\wedge\\ &\mathrm{EQ}[\mathrm{recip}(?y),?x]\}\end{aligned}$$

Skolemization replaces variables ?y and ?z by functions of ?x:

g(?x)
Element-of: Giving_events
giver: John
obj: sk(?x)
recip: ?x

The remaining variables are universally quantified.

The scope of universal variables in objects is the entire object.

Object-based knowledge systems, may provide some generic functions like: the_set_of, intersection, union, and complement are also allowed.

"John or Bill bought a Ford or Chevy which was not a convertible" :

B1
Element-of: Buying_Events
buyer: the_set_of (John, Bill)
bought: (element_of intersection
(union (Ford, Chevy),
complement (Convertibles)))

Reasoning with Structured Objects

1. Matching:

Two objects match if and only if the predicate logic formula associated with one of them unifies with the predicate logic formula for the other.

We usually have a goal object that we want to match against a fact object. The goal object matches the fact object if the goal object unifies with some sub-conjunction of the formulas of the fact object.

Ex: Fact object:

M1
Element-of: Marriage_Events
male: John_Jones
female: Mary_Jones

The associated predicate logic formula is:

$$\begin{aligned}&\mathrm{EL}(\mathrm{M1}, \mathrm{Marraige\_Events}) \wedge \\ &\mathrm{EQ}(\mathrm{male}(\mathrm{M1}), \mathrm{John\_Jones}) \wedge \\ &\mathrm{EQ}(\mathrm{female}(\mathrm{M1}), \mathrm{Mary\_Jones})\end{aligned}$$

This fact matches the goal unit:

M1
Element-of: Marriage_Events
male: John_Jones

It doesn’t match :

M1
Element-of: Marriage_Events
male: John_Jones
female: Mary_Jones
duration: 10

Matching Structures with Variables:

Variables that occur in fact structures have implicit universal quantification, variables that occur in goal structures have implicit existential quantification.

Ex: Suppose we want to find out "to whom did John give the book?" We represent this with the following goal frame:

?x
Element_of: Giving_Events
giver: John
recip: ?y
obj: Book

Matching this with the frame G1 yields this substitution: {G1/?x, Mary/?y} which produces the answer.

To match objects containing functional expressions for slot values, we evaluate the functional expressions first whenever possible.

Ex: Suppose we have the query "Did Bill give Mary the pen?" It is represented as :

?x
Element-of: Giving_Events
giver: Bill
recip: Mary
obj: Pen

Suppose the fact units are as we had before:

G1

Element-of: Giving_Events
giver: John
recip: Mary
obj: Book

G2

Element-of: Giving_Events
giver: Bill
recip: recip(G1)
obj : Pen

recip(G1) is first evaluated to Mary, then our goal unit matches G2.