Lecture II - part I
the logic framework FO(.): extensions

June 7, 2016

The view of knowledge representation:

- formalizing "information"

The view of knowledge representation language:

- a formal language,
  - formal syntax
  - formal semantics
- a theory of informal semantics:
  - a general theory of what information is expressed by formal expressions of the logic

If so, a logic is a formal, exact scientific theory of the informal meaning of the language constructs that it contains.

- E.g., $\wedge, \vee, \neg, \forall, \exists, \Rightarrow, \Leftrightarrow$ in FO.

The plan:

- building logics with language constructs derived from the language used in formal science and mathematics.
- Formal possible world semantics:
  - Structures are abstractions of potential states of affairs.
  - A mathematical theory formalizing the value of formal expressions in structures
  - Models = abstractions of possible states of affairs
  - Non-models = abstractions of impossible states of affairs.
- Simulates the methods of formal science (confer Newtons gravitation theory)

FO as a foundation

- FO is about a small of connectives, essential for KR:

$$\land, \lor, \neg, \forall, \exists, \Leftrightarrow, \Rightarrow$$

- A formal possible world semantics correctly formalizing the informal semantics of connectives.

Intermezzo: material implication

- Compare:
  - If you succeed for all courses, then you pass.
    - Proposed formalization:

$$(\forall c\ \mathit{Succ}(c)) \Rightarrow \mathit{Pass}$$

  - There is a course such that, if you succeed for it, you pass
    - Proposed formalization:

$$\exists c(\mathit{Succ}(c) \Rightarrow \mathit{Pass})$$

- Are the NL statements equivalent? No.
- Are the proposed formalizations logically equivalent? Yes!

The answer

- What we really expressed is:

$$\exists c(\neg Succ(c) \vee Pass)$$

- Next, let us reformulate by introducing "students"

$$\exists c \forall st(Succ(st, c) \Rightarrow Pass(st))$$

Now, the existential quantifier cannot move to the body.

$$\not\Leftrightarrow \forall st(\forall c\ Succ(st, c) \Rightarrow Pass(st))$$

- In the propositional formula, we are missing a modal operator.
  - There exists a course c, such that in every state of affairs, if one succeeds for c, one passes.

$$\exists c \Box(Succ(c) \Rightarrow Pass)$$

This is strict implication, another conditional studied by Lewis 1915

- We believe that FO is fundamental for KR because of its connectives.
- However, it does not suffice for knowledge representation.

Typed FO: FO(Types)

- In standard FO, there is a unique base type where all quantification, functions and predicates range over.
- Just like in other languages, types in FO can often be useful
  - (1) to improve the precision of the modeling and
  - (2) reduce the amount of human errors.

  E.g., in the graph colouring, there are two natural types: vertices and colors. We would like to define the colouring function as a function from vertices to colors so that it is not defined on colors.
- It is straightforward to extend FO with types.

FO(Types)

- The IDP system supports typed logic with subsorts.
- IDP performs type derivation for variables. That is, it "guesses" the type of variables from the positions in which it occurs.

Example

Graph-colouring:

- Types `Vertex` and `Col`
- Symbols `G(Vertex,Vertex), Colour(Vertex):Col`
- A well-typed axiom
  `! x y : (G(x,y) => Colour(x)~=Colour(y))`
  Variables `x, y` are derived to be of type `Vertex`.
- Explicit typing:
  `! x[Vertex] y[Vertex] : (G(x,y) => Colour(x)~=Colour(y))`
- Badly typed on variable `y`:
  `! x y : (G(x,y) => Colour(x)=y)`

FO(Types,Arit)

- FO(Types) is a basis for adding interpreted types.
- FO(Types,Arit) is obtained by adding the integer numbers:
  - type symbol *int*
  - numerals: strings $0, 12, -5, \dots$
  - arithmetic operator symbols $+, \times, \hat{}\,(\text{power}), <, \leq, >, \geq, \dots$
- Their value in every structure $\mathfrak{A}$ is the obvious one.
  - E.g., the value $12^{\mathfrak{A}}$ of numeral 12 in $\mathfrak{A}$ is the number 12.
  - E.g., the value $<^{\mathfrak{A}}$ of symbol $<$ in $\mathfrak{A}$ is the standard strict order relation $<$ on integers, that is, $\{(n, m) \in \mathbb{Z}^2 \mid n < m\}$.

Overloading! Notice the difference between a symbol and a value.

FO(Types,Arit)

- Now we can use integer arithmetic in our theories.
  - E.g., Fermats last theorem is now formalized as follows:

$$\neg\exists x \exists y \exists z \exists n (n > 2 \land x \hat{} n + y \hat{} n = z \hat{} n)$$

  (True, but it took 300years to prove.)
- We can do the same for other types of numbers $\mathbb{N}, \mathbb{Q}, \mathbb{R}$.
- IDP currently only supports finitely bounded arithmetic. Every numerical symbol has to be declared to be an element of a finite subtype of integers.

IDP example

```
vocabulary V{
   type someIntegers isa int
   C1 :  someIntegers
   C2 :  someIntegers
}
theory T: V{
   C1 + C1 = C2.
}
structure S:V{
   someIntegers = {1..5}
   }
```

- a subtype declaration using `isa`
- A bounded integer type declaration `{1..5}`

Extending FO to FO(Agg)

Syntax:

- If $\mathbf{x}$ is a tuple of variable symbols, φ a formula, then $\{\mathbf{x} : \varphi\}$ is a set expression
- If s is a set expression and Agg is an aggregate symbol (e.g., $\#, Sum, Min, \dots$) then $Agg(s)$ is a term (called an aggregate term)

Semantics:

- $\{\mathbf{x} : \varphi\}^{\mathfrak{A}} = \{\bar{d} \mid \mathfrak{A}[\mathbf{x} : \bar{d}] \models \varphi\}$
- $(Agg(s))^{\mathfrak{A}} = Agg^{\mathfrak{A}}(s^{\mathfrak{A}})$

(That is what it takes to extend FO with aggregates.)

In IDP, aggregates are represented in a slightly different syntax:

- number of elements of P
  ```
  #{x,y:  P(x,y)}.
  ```
- sum of x+y, for all (x,y)∈ P
  ```
  sum{x,y:  P(x,y) :  x+y}.
  ```
- minimum of set { x : Q(x)& R(x)}
  ```
  min{x:  Q(x)& R(x) :  x}.
  ```
- maximum :
  ```
  max{x:  Q(x)& R(x) :  x}.
  ```
- Nesting is allowed, as in:
  ```
  Pnest= sum{x[num]:x=#{y:Q(x,y)} :  x }.
  ```

`http://dtai.cs.kuleuven.be/krr/idp-ide/?present=Agg`

Homework 1: Experiment with different input/output.

- Compute value aggregate expressions from values for P, R.
- Compute values of P, R from value aggregates expressions.
- Compute minimal structure satisfying aggregates expressions.

Definitions in KR

- Axioms and definitions are the basic building blocks of scientific and mathematical theories.
- Also in KR and formal modelling, we need to define new symbols in terms of the existing ones.
- Some definitions are inductive/recursive.
- In general, inductive definitions cannot be expressed in FO (in databases: Aho& Ullman 79)
- Therefore, the last extension of FO that we consider here, is a language construct for definitions.

In general, inductive definitions cannot be expressed in FO.

Proof idea:

- Take a proposition that can be expressed using inductive definitions

  There is a path from vertex A to vertex B in graph G

- Show that the proposition cannot be expressed in FO.

- The class of models of theory T is denoted $\mathcal{M}od(T)$.
- A theory T over Σ expresses a class of Σ-structures $\mathcal{C}$ if $\mathcal{M}od(T) = \mathcal{C}$.

The compactness theorem of FO

The compactness theorem is historically the first technique to prove inexpressibility in FO.

Compactness Theorem of FO

An infinite FO theory Ψ is unsatisfiable iff at least one finite subset is unsatisfiable.

(There exists much more powerful techniques; e.g., Ehrenfeucht-Fraïssé Games.)

Unreachability can be infinitely expressed

"There is no path from A to B in graph G"

- $\Sigma = \{A, B, G/2\}$. Every Σ-structure specifies a graph interpreting G and vertices interpreting A, B.
- The proposition is expressed by the theory $T_{UnR} =$

$$\begin{array}{l} \neg G(A, B), \\ \neg\exists x_1(G(A, x_1) \wedge G(x_2, B)), \\ \ldots \\ \neg\exists x_1 \ldots \exists x_n(G(A, x_1) \wedge \cdots \wedge G(x_n, B)), \\ \ldots \end{array}$$

Proof. The propositions express: there is no path of length 1, of length 2, of length 3, ... A structure in which there is no path from A to B satisfies each of these propositions, and vice versa, a structure that satisfies each of these propositions cannot have a path from A to B since paths have finite length.

Reachability cannot be expressed

Theorem

"There is a path from A to B in graph G" is not expressible in FO.

Proof.

- Consider the class $\mathcal{C}_R$ of Σ-structures $\mathfrak{A}$ with a path from $A^{\mathfrak{A}}$ to $B^{\mathfrak{A}}$ in graph $G^{\mathfrak{A}}$.
- Assume $\mathcal{C}_R$ is expressible by FO theory T_R. Then the infinite theory $T_R \cup T_{UnR}$ is unsatisfiable.
- Hence, it has an unsatisfiable finite subset Ω.
- Let n be the largest path length forbidden by Ω. Every structure with a shortest path from A to B that is strictly longer than n satisfies Ω.
- Contradiction

- Reachability, transitive closure are important concepts in many applications, but they cannot be expressed in FO.
- Adding inductive definitions to FO will solve this problem.

It will be expressed by the FO(ID) theory using an auxiliary predicate R:

$$\left\{ \begin{array}{l} R(A). \\ \forall x (R(x) \leftarrow \exists y (R(y) \land G(y,x))) \end{array} \right\}$$
$$R(B)$$

Some observations of inductive definitions in mathematics

A prototypical monotone inductive definition

The definition of the reachability relation/transitive closure.

The reachability relation R_G of a graph G is defined inductively:
- $(x, y) \in R_G$ if $(x, y) \in G$;
- $(x, y) \in R_G$ if for some vertex z,

$$(x, z), (z, y) \in R_G.$$

A prototypical inductive definition over an induction order

The satisfaction relation $\models$ of propositional logic.

Let Σ be a vocabulary, $\mathfrak{A}$ a Σ-structure, φ a formula over Σ.
We define $\mathfrak{A} \models \varphi$ by induction on the structure of φ :
- $\mathfrak{A} \models q$ if q is a propositional variable and $q \in \mathfrak{A}$;
- $\mathfrak{A} \models \alpha \wedge \beta$ if $\mathfrak{A} \models \alpha$ and $\mathfrak{A} \models \beta$;
- $\mathfrak{A} \models \alpha \vee \beta$ if $\mathfrak{A} \models \alpha$ or $\mathfrak{A} \models \beta$;
- $\mathfrak{A} \models \neg\alpha$ if $\mathfrak{A} \not\models \alpha$.

The induction order:

- the subformula order

Informal inductive definitions (ID's)

- Definitions in mathematics belong to "informal language"
  - serve to define formal objects
  - but are not themselves formal objects
  - We call them informal definitions
- Informal language, but of a special kind:
  - of mathematical precision
  - broadly used
  - broadly understood
- An ideal topic for formal empirical study.

Some common properties

The transitive closure T_G of a graph G is defined inductively:
- $(x, y) \in T_G$ if $(x, y) \in G$;
- $(x, y) \in T_G$ if for some vertex z,

$$(x, z), (z, y) \in T_G.$$

We define $\mathfrak{A} \models \varphi$ by induction on structure of φ :
- $\mathfrak{A} \models q$ if $q \in \mathfrak{A}$;
- $\mathfrak{A} \models \alpha \wedge \beta$ if $\mathfrak{A} \models \alpha$ and $\mathfrak{A} \models \beta$;
- $\mathfrak{A} \models \alpha \vee \beta$ if $\mathfrak{A} \models \alpha$ or $\mathfrak{A} \models \beta$;
- $\mathfrak{A} \models \neg\alpha$ if $\mathfrak{A} \not\models \alpha$

- Linguistically, a set of conditionals
- Semantically, two principles: a structure is a possible world if
  - Constructively, the defined set is obtained by iterated rule application.
  - Non-constructively, the defined set is the least set closed under rule application.
- These two principles coincide – Tarski!

Right?

Properties of informal ID's

- The two principles coincide for monotone ID's
- But not in general for ordered ID's.

We define $\mathfrak{A} \models \varphi$ by induction on structure of φ :
- ...
- $\mathfrak{A} \models \neg\alpha$ if $\mathfrak{A} \not\models \alpha$
(i.e., if not $\mathfrak{A} \models \alpha$);

This definition is nonmonotone, has infinitely many minimal sets closed under its rules, and no least set.

The constructive principle is the fundamental one.

- Starting at the empty set
- Iterated rule-application
  - respecting the induction order
- Until the set is saturated.

This is the common basic intuition that we all share and that is formalized in our theory.

An outline of the theory

- A series of definitions from first principles (Def. 3-16) formalizing our intuitions and conventions on
  - monotone ID's, ordered ID's, iterated ID's
  - the induction order
  - the induction process
  - the defined sets
- Proof that the defined set coincides with the well-founded model from logic programming.

Syntax of formal definitions

A definition Δ is a set of definitional rules:

$$\forall \mathbf{x}(P(\mathbf{t}) \leftarrow \varphi)$$

where φ is a FO-formula and $\mathbf{t}$ a tuple of terms

- Defined symbols $\mathit{def}(\Delta)$ of Δ : predicates in the head.
- Parameter symbols $\mathit{pars}(\Delta)$ of Δ: all other symbols appearing in it.
- The definitional implication $\leftarrow$ (is not material implication!!).

A context structure $\mathcal{O}$: a structure interpreting $\mathit{pars}(\Delta)$

The informal semantics of definitions

We say that a formal definition Δ is a faithful specification of an informal definition D under an intended interpretation $\mathcal{I}$ if

- there is a one to one correspondence between rules of Δ and the informal rules of D, such that
- the informal reading of head and bodies of the formal rules corresponds to the head and body of the informal rule.

Here we fall back on the informal semantics of FO.

Example: transitive closure

The reachability relation R of graph G is defined inductively:
- $(x, y) \in R$ if $(x, y) \in G$;
- $(x, y) \in R$ if for some vertex z,

$$(x, z), (z, y) \in R.$$

A proposed formalization:

- Σ= { R/2, G/2 }, with obvious intended interpretation
- The formal definition Δ_R:

$$\left\{ \begin{array}{l} \forall x \forall y (R(x, y) \leftarrow G(x, y)) \\ \forall x \forall y (R(x, y) \leftarrow \exists z (R(x, z) \wedge R(z, y)) \end{array} \right\}$$

- A context structure $\mathcal{O}$ interprets symbol G by a graph

The formal definition Δ_R is a faithfull representation of D_R.

- This claim can be mathematically proven, using the formal definition of FO's satisfaction relation.

We define $\mathfrak{A} \models \varphi$ by induction on the structure . . . :

- . . .
- $\mathfrak{A} \models \neg\alpha$ if $\mathfrak{A} \not\models \alpha$.

$$\Downarrow$$

$$\Delta_{\models} = \left\{ \begin{array}{l} \forall i \forall p (Sat(i,p) \leftarrow Atom(p) \land In(p,i)) \\ \forall i \forall f \forall g (Sat(i, And(f,g)) \leftarrow Sat(i,f) \land Sat(i,g)) \\ \forall i \forall f \forall g (Sat(i, Or(f,g)) \leftarrow Sat(i,f) \lor Sat(i,g)) \\ \forall i \forall f (Sat(i, Not(f)) \leftarrow \neg Sat(i,f)) \end{array} \right\}$$

Defined symbol: $Sat/2$
Parameter symbols: $Atom/1$, $In/2$, function symbols $Not/1, And/2, Or/2,$.

A context $\mathcal{O}$:

- Domain: $\mathit{Structures}(\Sigma) \cup \mathit{Sentences}(\Sigma)$
- Interpretation for parameters:
  - $\mathit{Not}^{\mathcal{O}} = \{\psi \rightarrow \neg\psi | \psi \in \mathit{Sentences}(\Sigma)\}$
  - $\mathit{Atom}^{\mathcal{O}} = \{q \mid q \in \Sigma\}$
  - $\ldots$

- A natural induction $\mathcal{N}$ of Δ in context $\mathcal{O}$:

$$\mathfrak{A}_0 \to \mathfrak{A}_1 \to \ldots \to \mathfrak{A}_\beta$$

  where

  - $\mathfrak{A}_0, \ldots \mathfrak{A}_\beta$ are $\mathit{def}(\Delta)$-structures
  - $\mathfrak{A}_0 = \emptyset$
  - $\mathfrak{A}_{i+1}$ is obtained from $\mathfrak{A}_i$ by applying some rule instances.
  - $\mathfrak{A}_\beta$ is closed under all rules.
  - Sequence may be transfinite.

A natural induction for $\Delta_{\models}$ in $\mathcal{O}_{\Sigma}$

$$\mathfrak{A}_0 \to \mathfrak{A}_1 \to \ldots \to \mathfrak{A}_\beta$$

A sequence of subsets of $\mathit{Structures}(\Sigma) \times \mathit{Sentences}(\Sigma)$

- $\mathfrak{A}_0 = \emptyset$
- $\mathfrak{A}_\beta =$ the satisfaction relation for Σ

- $\mathfrak{A}$ is the structure defined by Δ in $\mathcal{O}$ if $\mathfrak{A}$ is the limit $\mathfrak{A}_\beta$ of a natural induction in $\mathcal{O}$.

But is this a good definition?

Is this a good definition?

- The process of natural induction is highly non-deterministic.
  - The order of rule application is not fixed.
  - There are many natural inductions.
- Do they converge to the same limit? If not, the defined set would be ambiguous. This would be unacceptable in mathematics.
- Convergence:
  - Guaranteed for monotone ID's (transitive closure)
  - But not for non-monotone ID's (satisfaction relation)
  - This is why the induction order is required!

Example: the satisfaction definition

At the initial stage, when $\mathfrak{A}_o = \emptyset$, all instances of the $\neg$-rule apply:

$$I \models \neg\varphi \text{ if } I \not\models \varphi$$

Applying an arbitrary one is likely an error.

- In $\mathfrak{A}_0 = \emptyset$, we have:
$$\{P\} \not\models P.$$
- So we can apply the $\neg$-rule to derive
$$\{P\} \models \neg P.$$
- This is an error.

A fundamental property of informal definitions

An all important point of the inductive construction process:

All natural inductions of a definition should converge to the same limit.

In mathematics, there are conventions in place that guarantee this property.
We have formalized these.

Definition by induction over a well-founded order

Given context $\mathcal{O}$.

- An ordered definition in $\mathcal{O}$ is a pair $\langle \Delta, \prec \rangle$ with
  - Δ a definition and
  - $\prec$ a strict well-founded order on the set of defined facts.
- A mathematician or formal scientist is not free to take any order: the order has to match the definition rules
  - Rules define entities in terms of strictly smaller entities
  - $\prec$ is a dependency relation of Δ in $\mathcal{O}$.

We define the grue numbers by induction on standard order $\leq$ on *natnrs*:
- n is a grue number if $n+1$ is a grue number

What would you do if you were a reviewer of a paper containing this definition?

- The rule does not match the induction order.
  - an inductive rule should define an object in terms of strictly smaller objects in the induction order
- Reject

Given an ordered inductive definition $\langle \Delta, \prec \rangle$ in context $\mathcal{O}$.

- A natural induction $\mathcal{N}$ respects $\prec$ if rules are applied only if no strictly smaller facts can be derived anymore.

Convergence theorem

Theorem

If $(\Delta, \prec)$ is an ordered definition in context $\mathcal{O}$, then all natural inductions that respect $\prec$ converge.

Moreover the limit is independent of $\prec$.

Some Induction orders $\prec$ that are dependencies of the satisfaction relation:

- subformula relation
- size order on formulas
- depth order on formulas

These are strict well-founded orders that are dependency relations of the rules of the satisfaction relation.

Many well-founded orders $\prec$ are not a dependency relation.

- E.g., the empty order

Using them, no convergence or an unintended limit.

Take-away message:

- The order of a non-monotone informal definition must be chosen with care, to match the rules.
- But any order that matches the rules is ok.

Do we really need this order?

Link with well-founded semantics

Natural inductions versus three-valued constructions

- Natural induction $\mathcal{N}$ = sequence of two-valued structures starting at $\emptyset$
- $\mathcal{N}$ naturally corresponds to a sequence of three-valued structures
  - At each $\mathfrak{A}_i$, we have a three-valued structure $\mathcal{I}_i$.
    - defined atoms true in $\mathfrak{A}_i$ are certainly true,
    - some false atoms are certain to remain false,
    - for some false facts it is unknown if they remain false
- This sequence is a well-founded induction sequence (Denecker& Vennekens 2007)
  - Each well-founded induction sequence constructs the well-founded model.

Theorem

Let $\langle \Delta, \prec \rangle$ be an ordered inductive definition in $\mathcal{O}$.

- *Each natural induction of Δ in $\mathcal{O}$ converges to the well-founded model of Δ in $\mathcal{O}$.*
- *The well-founded model of Δ is two-valued.*

The theory also works for Iterated inductive definitions:

- Mixtures of monotone induction and ordered induction
- Have been studied in mathematical logic : IID (Kleene, Feferman, Martin-Löf, . . .)
- Think of locally stratified logic programs
  - recursion over negation but no loops over negation
- Iterated inductive definitions may have many minimal fixpoints. The induction process that obeys the induction order constructs one of them.
- The well-founded semantics constructs without an explicit given order.

A formal scientific study of these types of inductive definitions

- A formal syntax to express definitions
- A formal possible world semantics
- An informal semantics
  - We specify a simple criterion to say when an formal definition expresses an informal definition

    a formal definition faithfully represents an informal definition

- The theory is refutable
- Each faithfully representable mathematical definition is an experiment for the theory.
  - Compare the relations defined by the formal semantics with those defined by the informal definition.

Existing work:

- Inductive/recursive definitions were studied in mathematical logic
- Language constructs to express inductive definitions were added to database languages
  - fixpoint expressions

Contribution of this work?

- Improved understanding of certain nonmonotone inductive definitions:
  - Ordered Definitions : definitions with an induction order
  - Iterated inductive definitions
- Connection with logic programming

Exploring conventions about ordered definitions.

We define the grue numbers by induction over the standard order $\leq$ on $\mathbb{N}$:
- 0 is a grue number
- $n + 1$ is a grue number if n is a grue number

- Monotone and a definition over a well-founded order
- It defines ... grue numbers $= \mathbb{N}$

We define the grue numbers by induction on the standard order $\leq$ on $\mathbb{N}$:
- $n+1$ is a grue number if n is a grue number

- It defines ... grue numbers $= \emptyset$
- No base case

We define the grue numbers by induction on the inverse $\geq$ of the standard order on $\mathbb{N}$:
- n is a grue number if $n+1$ is a grue number

What would you do if you were a reviewer of a paper containing this definition?

- The induction order needs to be a well-founded order
- Reject the paper

We define the grue numbers by induction on standard order $\leq$ on *natnrs*:
- n is a grue number if $n + 1$ is a grue number

What would you do if you were a reviewer of a paper containing this definition?

- The rule does not respect the induction order.
  - an inductive rule should define an object in terms of strictly smaller objects in the induction order
- Reject

We define the grue numbers by induction on standard order $\leq$ on *natnrs*:
- 0 is a grue number
- $n+1$ is a grue number if n is not a grue number

What about this definition?

- A simplified version of the satisfaction definition
- The induction process constructs ... the even numbers.

$$\to 0 \to 2 \to 4 \to 6 \to \ldots \to 2n \to \ldots$$

- Infinitely many minimal sets satisfying the two rules

$$0, 1, 3, 5, \ldots, 2n+1, \ldots$$

$$0, 2, 3, 5, \ldots, 2n+1, \ldots$$

$$0, 2, 4, 5, 7, \ldots, 2n+1, \ldots$$

Defining the logic FO(ID)

FO(ID)

- Syntax:

  An FO(ID) theory is a set of FO sentences and definitions.

- Semantics:
  - $\mathfrak{A} \models T$ if $\mathfrak{A} \models \varphi$, for every FO sentence $\varphi \in T$ and $\mathfrak{A}$ is a well-founded model of Δ, for every definition $\Delta \in T$.

Non-inductive definitions: examples

Definition by exhaustive enumeration:

$$\left\{\begin{array}{l} \mathit{Instructor}(\mathit{Ray}, \mathit{CS}230) \leftarrow \\ \mathit{Instructor}(\mathit{Hec}, \mathit{CS}230) \leftarrow \\ \mathit{Instructor}(\mathit{Wal}, \mathit{HD}87) \leftarrow \\ \mathit{Instructor}(\mathit{Mar}, \mathit{HD}88) \leftarrow \end{array}\right\}$$

Non-inductive definition:

$$\left\{\begin{array}{l} \forall x(\mathit{European}(x) \leftarrow \mathit{Albanian}(x)) \\ \forall x(\mathit{European}(x) \leftarrow \mathit{Armenian}(x)) \\ \dots \\ \forall x(\mathit{European}(x) \leftarrow \mathit{Turkish}(x)) \\ \forall x(\mathit{European}(x) \leftarrow \mathit{Ukrainian}(x)) \end{array}\right\}$$

50 cases.

Also many non-inductive definitions consists of independent cases which can be represented by separate rules.

Definition by simultaneous induction

Defining multiple predicates at the same time in the natural numbers:

$$\left\{ \begin{array}{l} \forall x(\mathit{Even}(x) \leftarrow x = 0)) \\ \forall x(\mathit{Odd}(S(x)) \leftarrow \mathit{Even}(x)) \\ \forall x(\mathit{Even}(S(x)) \leftarrow \mathit{Odd}(x)) \end{array} \right\}$$

Here $S/1$ is the successor function, mapping n to n+1.

Homework 2: Verify that we have seen already a mathematical definition in this course showing simultaneous induction: the definition of term and formula. Recall that formulas are defined in term of terms, and we defined aggregate terms (e.g., $\#\{x : \varphi\}$) in terms of formulas.

Induction over aggregates

Express the following inductive definition:

A company A controls company B if the total sum of the shares in company B owned by A or by companies controlled by A is more than 50%.

Using the vocabulary

- $Cont(x, y)$: company x controls company y.
- $OwnsSh(x, y, s)$: company x owns s shares in company y.

$$\left\{ \forall a \forall b (Cont(a, b) \leftarrow Sum\{(s, c) : \begin{array}{l} (c = a \vee Cont(a, c)) \wedge \\ OwnsSh(c, b, s)\} > 0.50) \end{array} \right\}$$

Definitions are not procedures

- It is tempting to see a definition as a procedure, to compute the defined relations from the parameters by iterated rule application. However, this is not the right view.
- A definition states a logical relationship between defined and parameter symbols. It frequently occurs that the defined predicates are known and given, but the parameters are unknown.
- In that case, we obviously cannot "execute" the rules since we have no parameter values to start from.

A case of a definition with unknown parameters

Consider the following theory.

$$\begin{array}{l}\left\{\begin{array}{l} R(A) \leftarrow \\ \forall x(R(x) \leftarrow \exists y(R(y) \wedge G(y,x))) \end{array}\right\} \\ \forall x R(x)\end{array}$$

This theory imposes the constraint on graph G that every element of the domain should be reachable by a finite path from A. There is obviously no way we can execute the definition of R here, since G is not given.

Use IDP at
`http://dtai.cs.kuleuven.be/krr/idp-ide/?present=ReachabilityIsTotal`
to compute all graphs G with domain $\{a, b\}$ and value $A^{\mathfrak{A}} = a$ that satisfy this theory. Next, extend the structure with value $G^{\mathfrak{A}} = \{(a, a), (b, b)\}$ and try again to find models. Does it give what you expected?

Inductive definitions

- are not material implications
- are not expressible as equivalences in FO

Definitions are not sets of implications

Compare definition :

$$\left\{ \begin{array}{l} \forall x(Parent(x,y) \leftarrow Father(x,y)) \\ \forall x(Parent(x,y) \leftarrow Mother(x,y)) \end{array} \right\}$$

with set of material implications:

$$\begin{array}{l} \forall x(Parent(x,y) \Leftarrow Father(x,y)) \\ \forall x(Parent(x,y) \Leftarrow Mother(x,y)) \end{array}$$

With empty $Father, Mother$ relations.

- According to the definition, the $Parent$ relation is empty.
- According to the implications, the $Parent$ relation is arbitrary.

`http://dtai.cs.kuleuven.be/krr/idp-ide/?present=DefImp`

Definitions are not equivalences

Use IDP to compute certain models of the reachability definition and its completion at `http://dtai.cs.kuleuven.be/krr/idp-ide/?present=ReachCompletion`
Verify that in the unexpected (and incorrect) model of the completion, $R(B)$ satisfies the completed definition of R, despite the fact that B is not reachable from A.

About the informal semantics of $\neg$ and $\leftarrow$ in FO(ID).

We define $\mathfrak{A} \models \varphi$ by structural induction:
- $\mathfrak{A} \models P(t_1, \ldots, t_n)$ if $(t_1^{\mathfrak{A}}, \ldots, t_n^{\mathfrak{A}}) \in P^{\mathfrak{A}}$
- ...
- $\mathfrak{A} \models \alpha \wedge \beta$ if $\mathfrak{A} \models \alpha$ and $\mathfrak{A} \models \beta$;
- $\mathfrak{A} \models \neg\alpha$ if $\mathfrak{A} \not\models \alpha$
i.e., if it is not the case that $\mathfrak{A} \models \alpha$
- ...

What is the meaning of not?

What is the meaning of the conditional?

The conditional in inductive definitions: a third kind of conditional.

Some familiar looking examples

$$\left\{ \begin{array}{l} \forall x \forall t (Member(x, .(x, t)) \leftarrow \mathbf{t}) \\ \forall x \forall h \forall t (Member(x, .(h, t)) \leftarrow Member(x, t)) \end{array} \right\}$$

$$\left\{ \begin{array}{l} \forall l (Append(Nil, l, l) \leftarrow \mathbf{t}) \\ \forall h \forall t \forall l \forall t_1 (Append(.(h, t), l, .(h, t_1)) \leftarrow Append(t, l, t_1)) \end{array} \right\}$$

Do these look familiar to you?

Relationship to Prolog

- Prolog (programmation en logique): a declarative programming language.
- Invented by Robert Kowalski (Imperial College London) and Alain Colmerauer (U.Marseille) around 1971-1975.
- Rule based, variables start with capitals, negation as failure `not`.
- A Prolog interpreter is called by posing a query.
- A procedural semantics: topdown execution of queries, induced by inference algorithm called SLDNF resolution (Selective Linear Definite clause resolution with Negation as Failure).
  - Unification, backtracking, negation by failure
- Side-effects (cut !, assert and retract,..) make it impossible to view Prolog as a whole as a declarative (modelling) logic
- A logic program = a Prolog program without side-effects.

An example logic program with negation

```
parent_child(trude, sally).
parent_child(tom, sally).
parent_child(tom, erica).
parent_child(mike, tom).

female(trude).

sibling(X, Y) :- parent_child(Z, X), parent_child(Z, Y).

father_child(X, Y) :- parent_child(X, Y), not female(X).
mother_child(X, Y) :- parent_child(X, Y), female(X).
```

Queries:

```
?- father_child(mike,tom).
True
?- father_child(tom,X).
X=sally
X=erica
?- mother_child(mike,tom)
False
```

The semantical problem of Logic Programming

- a logic program was introduced as a set of (special) material implications.
- This cannot explain Prolog's answers.
- But the answers are so intuitive and also very useful
- What formal semantics can explain these conclusions?
- What informal semantics can explain these conclusion?
- Two main proposals:
  - A logic program as a defaul/autoepistemic theory
    - Michael Gelfond and Vladimir Lifschitz
    - stable semantics
    - basis of ASP
  - A logic program as a definition.
    - Clark, Harel, Schlipf, Denecker et al.

Relationship logic programs - definitions

A proposal:

- The informal semantics of a logic program:
  - UNA of all constant symbols
  - The set of rules is a (parameter-free) definition defining all predicate symbols.
    - If no rule exists for a predicate, it is defined to be the empty relation. E.g. if we delete the unique rule for `female`, female is defined to be empty and every parent is entailed to be a father.
- This view explains the answers to all queries in the example.
- It explains also the desired answers to recursive logic programs.

UNA is needed.
Homework 3: Show that the example logic program on page 87 viewed as a definition in FO(ID) without UNA, does not entail `father_child(mike,tom)`. You need to sketch a structure that satisfies the definition (but not UNA), in which this atom is false.

Relationship logic programs - FO(ID)

- As a modelling language, logic programming is not expressive.
  - a logic program is categorical: has only one model; hence, no incomplete knowledge can be represented in it.
- FO(ID) arose as an integration of extended LP with FO.
- FO(ID) theories extend logic programs
  - No automatic UNA
  - Definitions have Parameters!
  - Multiple definitions
  - FO and FO(.) axioms

E.g., the binary reachability definition has parameter $G/2$.

$$\left\{ \begin{array}{l} \forall x \forall y (R(x,y) \leftarrow G(x,y)) \\ \forall x \forall y \forall z (R(x,y) \leftarrow R(x,z) \wedge R(z,y)) \end{array} \right\}$$

On the other hand, in the following logic program, `G/2` is defined to be the empty relation:

```
R(x,y) :- G(x,y).
R(x,y) :- R(x,z), R(z,y).
```

An alternative view on logic programming

The origins of Answer Set Programming

Answer set programming

- In origin, the stable semantics of logic programs (LP) and extended logic programs (ELP) is not a possible world semantics.
  - Gelfond and Lifschitz
- A stable model = a belief set, the set of literals in one epistemic state of a rational introspective agent whose beliefs are expressed by the program.
- ELP was the basis of ASP.

In origin, ELP was based on logics from common sense reasoning

- embeddings to default logic
- embeddings to autoepistemic logic

Mathematical information versus common sense information

FO($\cdot$): turning FO into a practical KR language

- FO does not suffice for knowledge representation, modelling, specification

$\Rightarrow$ FO(Types,ID,Agg,Arit,FD,Mod,HO,Caus,...)

  - Types
  - (Inductive) Definitions
  - Aggregates
  - Arithmetic
  - Coinductive Definitions
  - Modal operators
  - Higher Order logic
  - Causation
  - ...

The FO($\cdot$) language framework