First–Order Languages without Equality

A first–order language without equality $\mathcal{L}$ will consist of

- a set $\mathcal{F}$ of **function symbols** $f, g, h, \cdots$ with associated arities;

- a set $\mathcal{R}$ of **relation symbols** $r, r_1, r_2, \cdots$ with associated arities;

- a set $\mathcal{C}$ of **constant symbols** $c, d, e \cdots$;

- a set X of **variables** $x, y, z, \cdots$.

Each relation symbol r has a positive integer, called its **arity**, assigned to it.

If the number is n, we say r is **n–ary**.

For small n we use the same special names that we use for function symbols:

unary, **binary**, **ternary**.

The set $\mathcal{L} = \mathcal{R} \cup \mathcal{F} \cup \mathcal{C}$ is called a **first–order language**.

$\{+, \cdot, <, -, 0, 1\}$ would be a natural choice of first-order language when working with the integers.

Interpretations and Structures

The obvious interpretation of a relation symbol is as a **relation** on a set.

If A is a set and n is a positive integer, then an **n–ary relation** r on A is a subset of A^n,

that is, r consists of a collection of **n–tuples** $(a_1, \ldots, a_n)$ of elements of A.

An **interpretation** I of the first–order language $\mathcal{L}$ on a set S is a mapping with domain $\mathcal{L}$ such that

- $I(c)$ is an **element of** S for each constant symbol c in $\mathcal{C}$;

- $I(f)$ is an n**–ary function on** S for each n–ary function symbol f in $\mathcal{F}$;

- $I(r)$ is an n**–ary relation on** S for each n–ary relation symbol r in $\mathcal{R}$.

An $\mathcal{L}$-**structure** $\mathbf{S}$ is a pair (S, I), where I is an interpretation of $\mathcal{L}$ on S.

Preferred notation

We prefer to write

$c^{\mathbf{S}}$	(or just c)	for	$I(c)$
$f^{\mathbf{S}}$	(or just f)	for	$I(f)$
$r^{\mathbf{S}}$	(or just r)	for	$I(r)$
	$(S, \mathcal{F}, \mathcal{R}, \mathcal{C})$	for	(S, I)

Example

The structure $(R, +, \cdot, <, 0, 1)$, the reals with addition, multiplication, less than, and two specified constants has:

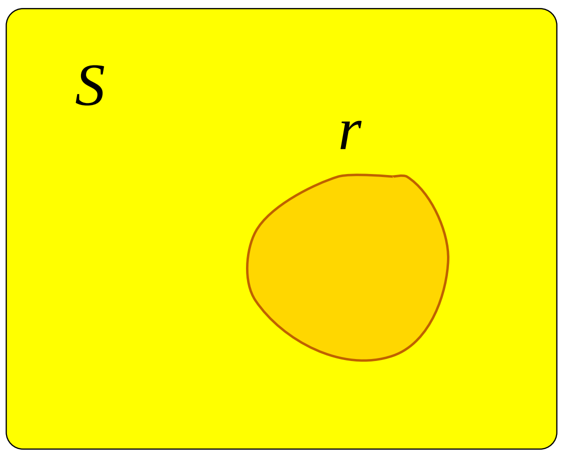
$$\mathcal{F} = \{+, \cdot\} \qquad \mathcal{R} = \{<\} \qquad \mathcal{C} = \{0, 1\}.$$

If $r \in \mathcal{R}$ is a **unary** predicate symbol,

then in any $\mathcal{L}$–structure $\mathbf{S}$,

the relation $r^{\mathbf{S}}$ is a subset of S.

We can picture this as:



If $\mathcal{L}$ consists of a single **binary** relation symbol r,

then we call an $\mathcal{L}$–structure a **directed graph**.

A small finite directed graph can be conveniently described in three different ways:

• **List the ordered pairs** in the relation r.

A simple example with $S = \{a, b, c\}$ is

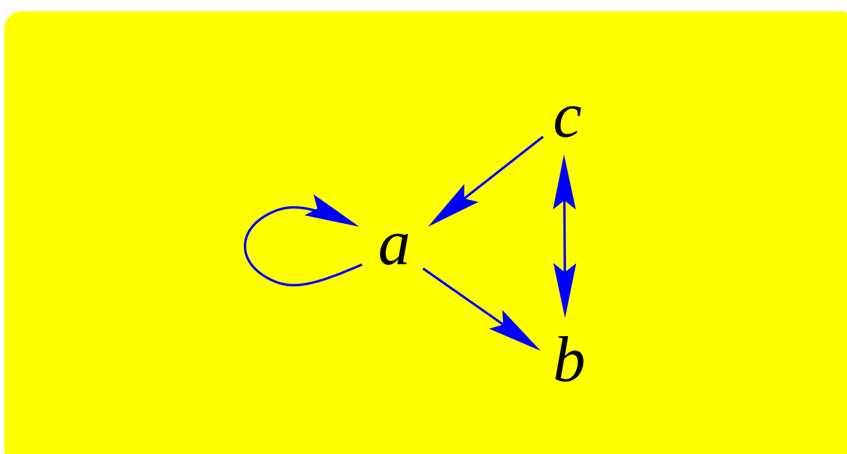
$$r = \{(a, a),\ (a, b),\ (b, c),\ (c, b),\ (c, a)\}.$$

• **Use a table**. For the same example we have

r	a	b	c
a	1	1	0
b	0	0	1
c	1	1	0

(**An entry of 1** in the table indicates **a pair is in the relation**.)

• **Draw a picture**. Again, using the same example:



Example

An interpretation of a language on a small set can be conveniently given by tables.

Let $\mathcal{L} = \{+, <\}$

where $+$ and $<$ are binary.

The following tables give **an interpretation** of $\mathcal{L}$ on the two–element set $S = \{a, b\}$:

$+$	a	b
a	a	b
b	b	a

$<$	a	b
a	0	1
b	0	0

A clause in the predicate logic uses **atomic formulas** instead of propositional variables.

- An **atomic formula** A is an expression

$$rt_1 \cdots t_n \ ,$$

where the t_i are terms, and r is an n–ary relation symbol.

Examples of atomic formulas:

$$x < y \qquad (x + y) < (x \cdot y) \qquad rfxgy0$$

where r and g are binary, f is unary.

Literals

- A **literal** is either

  an atomic formula A

  or a negated atomic formula $\neg$A

Examples of literals

$$x < y \qquad \neg\,((x + y) < (y \cdot z)) \qquad \neg\, rfxgxy$$

An atomic formula is a **positive** literal.

A negated atomic formula is a **negative** literal.

Clauses

- A **clause** C is a finite set of literals

$$\{\mathsf{L}_1, \ldots, \mathsf{L}_n\}\ .$$

We also use the notation

$$\mathsf{L}_1 \vee \cdots \vee \mathsf{L}_n\ .$$

Examples of clauses:

$$\{\neg\,(x < y),\ \neg\,(y < z),\ \neg\,(x < z)\}$$

$$\{rxx,\ rxg1y,\ \neg\,rfxgyz\}$$

The **parsing algorithm** for atomic formulas.

Example

r a binary relation symbol

f a unary function symbol

g a binary function symbol

c a constant symbol

Is $rgxfyfc$ an atomic formula?

If so find the two subterms t_1, t_2 such that $rt_1t_2 = rgxfyfc$.

i	0	1	2	3	4	5	6
s_i	r	g	x	f	y	f	c
γ_i	0	-1	0	0	1	1	2
		(			)	(	)

Semantics

Given a first-order structure $\mathbf{S}$ which tuples of elements $a_1, \ldots, a_n$ make a literal $\mathsf{L}(x_1, \ldots, x_n)$ true?

If $\vec{a}$ is such a tuple for the literal L we say

- $\mathsf{L}(\vec{a})$ **holds** (**is true**) in $\mathbf{S}$

- $\mathbf{S}$ **satisfies** (**models**) $\mathsf{L}(\vec{a})$

and write $\mathbf{S} \models \mathsf{L}(\vec{a})$.

(For clauses C we have parallel concepts.)

The set of tuples from $\mathbf{S}$ that make $\mathsf{L}(x_1, \ldots, x_n)$ true

form an **n-ary relation** that we call $\mathsf{L}^{\mathbf{S}}$.

The set of tuples from $\mathbf{S}$ that make $\mathsf{C}(x_1, \ldots, x_n)$ true

form an **n-ary relation** that we call $\mathsf{C}^{\mathbf{S}}$.

Example

Let $\mathbf{S}$ be given by the tables:

f	a	b
a	a	a
b	a	b

r	a	b
a	0	1
b	0	0

Let $\mathsf{L}_1 = rfxyfxx$, $\mathsf{L}_2 = \neg rfxyx$, $\mathsf{C} = \{\mathsf{L}_1, \mathsf{L}_2\}$.

A combined table for $\mathsf{L}_1, \mathsf{L}_2, \mathsf{C}$ is

				L_1		L_2	C
x	y	fxy	fxx	$rfxyfxx$	$rfxyx$	$\neg rfxyx$	$\{rfxyfxx, \neg rfxyx\}$
a	a	a	a	0	0	1	1
a	b	a	a	0	0	1	1
b	a	a	b	1	1	0	1
b	b	b	b	0	0	1	1

Satisfiability

$$\boxed{\mathbf{S} \ \models \ \mathsf{L}(x_1, \ldots, x_n)}$$

if for **every** $\vec{a}$ from S we have $\mathsf{L}(\vec{a})$ holds in $\mathbf{S}$.

$$\boxed{\mathbf{S} \ \models \ \mathsf{C}(x_1, \ldots, x_n)}$$

if for **every** choice of $\vec{a}$ from S we have $\mathsf{C}(\vec{a})$ holds in $\mathbf{S}$.

For $\mathcal{S}$ a set of clauses, we say

$$\boxed{\mathbf{S} \ \models \ \mathcal{S}}$$

provided $\mathbf{S}$ satisfies every clause C in $\mathcal{S}$.

We say Sat($\mathcal{S}$), or $\mathcal{S}$ **is satisfiable**, if there is a structure $\mathbf{S}$ such that $\mathbf{S} \models \mathcal{S}$.

If this is not the case, we say $\neg\,$Sat($\mathcal{S}$), meaning $\mathcal{S}$ is **not satisfiable**.

Predicate clause logic, like propositional clause logic, revolves around the study of

$$\boxed{\textbf{not satisfiable}}$$

.

Example

Given two **unary** relation symbols r_1, r_2,

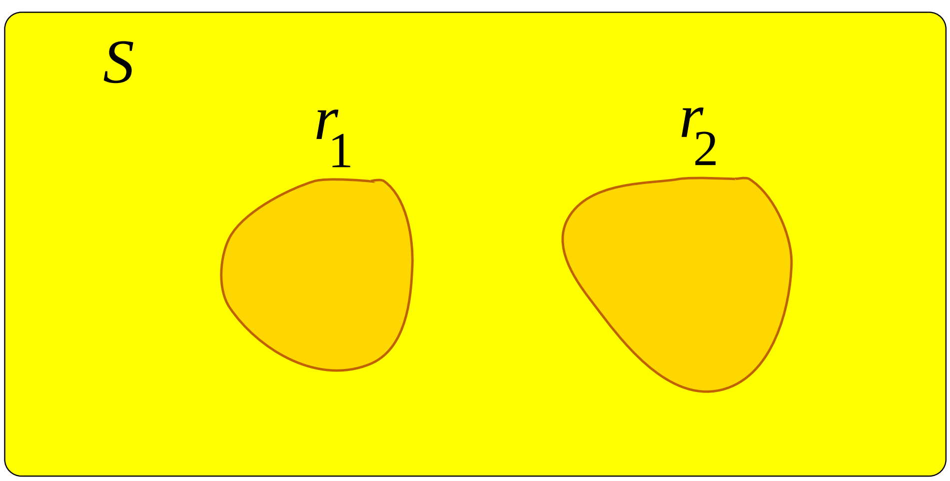
$$\{\neg r_1 x,\ \neg r_2 x\}$$

is satisfied by a structure $\mathbf{S}$ iff

for $a \in S$ either $\neg r_1 a$ or $\neg r_2 a$ holds,

and this is the case iff the sets r_1 and r_2 are **disjoint**, that is, $r_1 \cap r_2 = \emptyset$.

We can picture this situation as follows:



Example

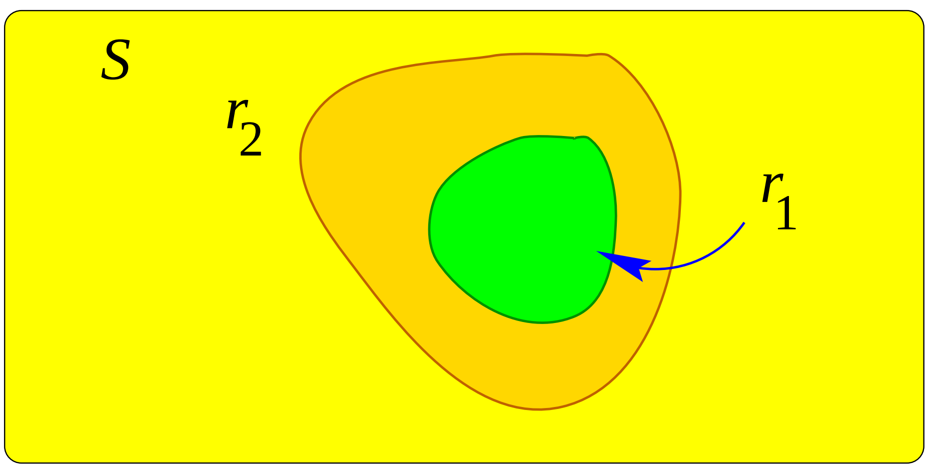
Given two **unary** relation symbols r_1, r_2,

$$\{\neg r_1 x,\ r_2 x\}$$

is satisfied by a structure $\mathbf{S}$ iff

the set r_1 is a **subset of** r_2.

We can picture this situation as follows:



Example

Let $\mathbf{S}$ be a directed graph, with $\mathcal{L} = \{r\}$.

- $\mathbf{S}$ will satisfy the clause $\boxed{\{rxx\}}$

  iff the binary relation r is **reflexive**.

- $\mathbf{S}$ will satisfy the clause $\boxed{\{\neg\, rxx\}}$

  iff the binary relation r is **irreflexive**.

- $\mathbf{S}$ will satisfy the clause $\boxed{\{\neg\, rxy,\ ryx\}}$

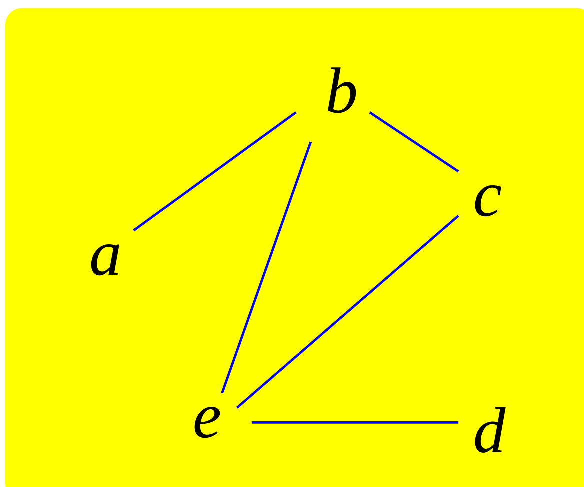
  iff the binary relation r is **symmetric**.

• $\mathbf{S}$ will satisfy the clause

$\boxed{\{\neg\, rxy,\ \neg\, ryz,\ rxz\}}$ iff the binary relation r is **transitive**

• A **graph** is an irreflexive, symmetric directed graph.

Graphs are drawn without using directed edges, for example



The Herbrand Universe

Given a first-order language $\mathcal{L} = \mathcal{R} \cup \mathcal{F} \cup \mathcal{C}$, the **ground terms** are terms that have no variables in them.

The **Herbrand Universe** $T_{\mathcal{C}}$ for $\mathcal{L}$ is the set of ground terms for the language $\mathcal{L}$.

Example

Suppose our language has a binary function symbol f and two constants $0, 1$. Then the following ground terms will be in the Herbrand universe:

$$0,\ 1,\ f00,\ f01,\ f10,\ f11,\ f0f00,\ \text{etc.}$$

Now we create **the algebra** $\mathbf{T}_{\mathcal{C}}$ on the Herbrand universe $T_{\mathbf{C}}$ as follows:

$$I(c) \ = \ c$$

$$I(f)(t_1, \ldots, t_n) \ = \ f t_1 \cdots t_n$$

The Herbrand universe provides an analog of the two–element algebra in the propositional calculus.

It provides a place to check for satisfiability.

We say that a set of clauses $\mathcal{S}$ is **satisfiable over the Herbrand universe** if

it is possible to interpret the relation symbols on the Herbrand universe in such a way that $\mathcal{S}$ becomes true in this structure.

The basic theorem says that a set of clauses $\mathcal{S}$ is not satisfiable (in any structure) iff

some finite set $\mathcal{G}$ of ground instances of $\mathcal{S}$ is not satisfiable over the Herbrand universe.

To check that

$\boxed{\text{a finite set of ground clauses } \mathcal{G} \text{ is satisfiable over the Herbrand universe}}$

it suffices to check that

$\boxed{\mathcal{G} \text{ is } \textit{propositionally satisfiable}}$

written p-**satisfiable** for short.

To check that $\mathcal{G}$ is p-satisfiable means:

$\boxed{\text{consider all atomic formulas in } \mathcal{G} \text{ to be propositional variables}}$

and then check to see if the propositional clauses are satisfiable.

Example

Consider the set of four **ground clauses**:

$$\{ra\}$$

$$\{\neg\, ra,\ rfa\}$$

$$\{\neg\, rfa,\ rffa\}$$

$$\{\neg\, rffa\}$$

List the atomic formulas in these clauses with simple propositional variable names:

atomic formula	renamed
raa	P
rfa	Q
$rffa$	R

The set of four ground clauses becomes

$$\{P\} \quad \{\neg\, P,\ Q\} \quad \{\neg\, Q,\ R\} \quad \{\neg\, R\}$$

Continuing with this example, we can now show that the set of three clauses

$$\{ra\}$$
$$\{\neg\, rx,\ rfx\}$$
$$\{\neg\, rffx\}$$

is not satisfiable as one has a set of ground instances

$$\{ra\}$$
$$\{\neg\, ra,\ rfa\}$$
$$\{\neg\, rfa,\ rffa\}$$
$$\{\neg\, rffa\}$$

that is easily seen not to be p-satisfiable by the translation into

$$\{P\} \quad \{\neg\, P,\ Q\} \quad \{\neg\, Q,\ R\} \quad \{\neg\, R\}$$

Substitution

Given a substitution $\sigma = \begin{pmatrix} x_1 \leftarrow t_1 \\ \vdots \\ x_n \leftarrow t_n \end{pmatrix}$ and a literal $\mathsf{L}(x_1, \ldots, x_n)$,

we write $\sigma\mathsf{L}$, or $\mathsf{L}(t_1, \ldots, t_n)$, for the result of applying the substitution σ to L.

Given a clause

$$\mathsf{C} = \mathsf{C}(x_1, \ldots, x_n) = \{\mathsf{L}_1, \cdots, \mathsf{L}_k\},$$

we write $\sigma\mathsf{C}$, or $\mathsf{C}(t_1, \ldots t_n)$, for the clause

$$\{\sigma\mathsf{L}_1, \ldots, \sigma\mathsf{L}_k\}.$$