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:

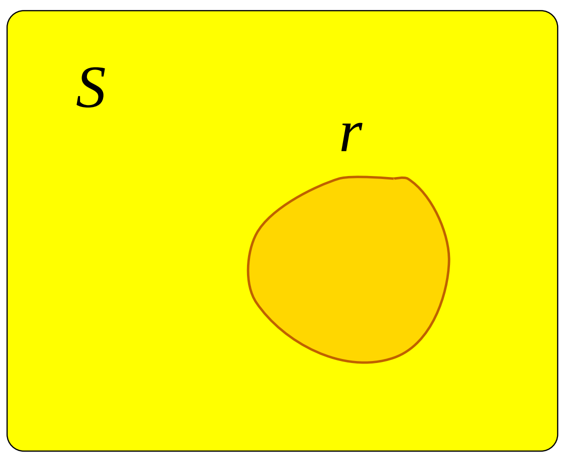
$$\mathcal{F} = \{+, \cdot\} \qquad \mathcal{R} = \{<\} \qquad \mathcal{C} = \{0, 1\}.$$

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

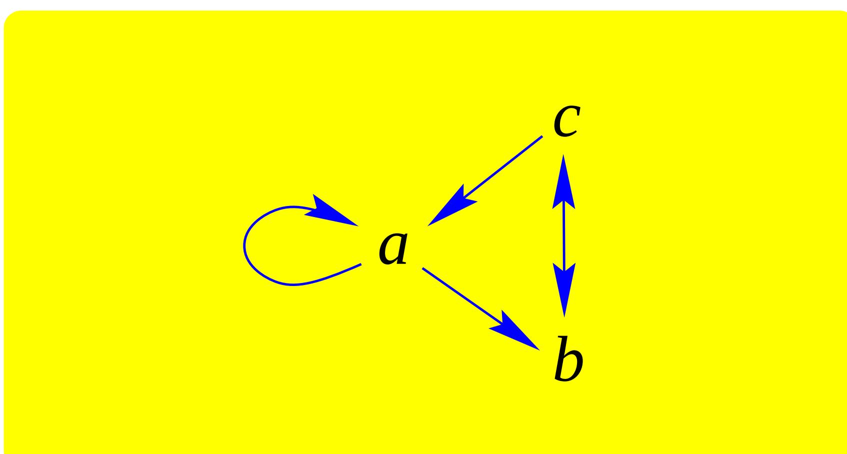
$$r = \{(a, a),\ (a, b),\ (b, c),\ (c, b),\ (c, a)\}.$$

• **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,

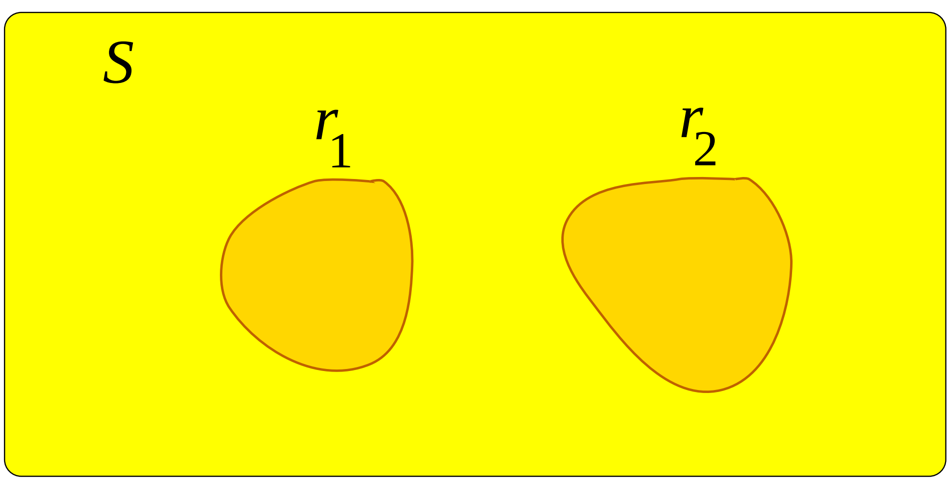
$$\{\neg r_1 x,\ \neg r_2 x\}$$

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, $r_1 \cap r_2 = \varnothing$.

We can picture this situation as follows:



Example

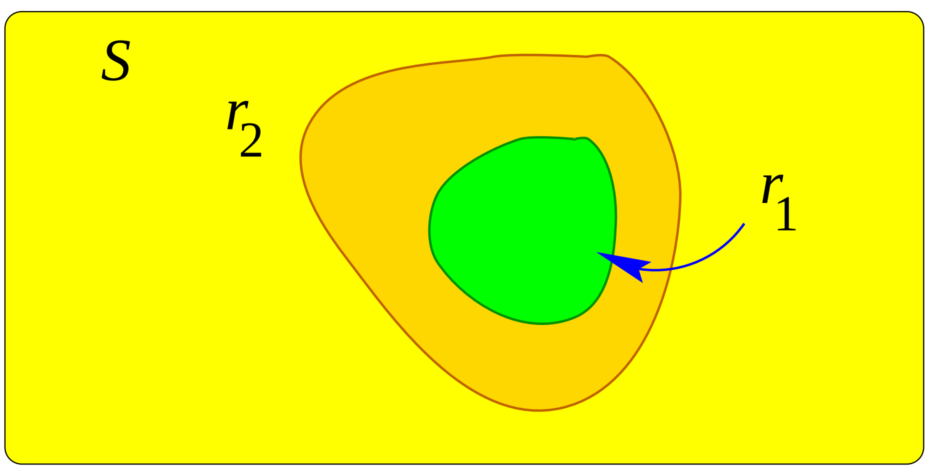
Given two **unary** relation symbols r_1, r_2,

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

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 $\boxed{\{\neg\, rxy,\ ryx\}}$

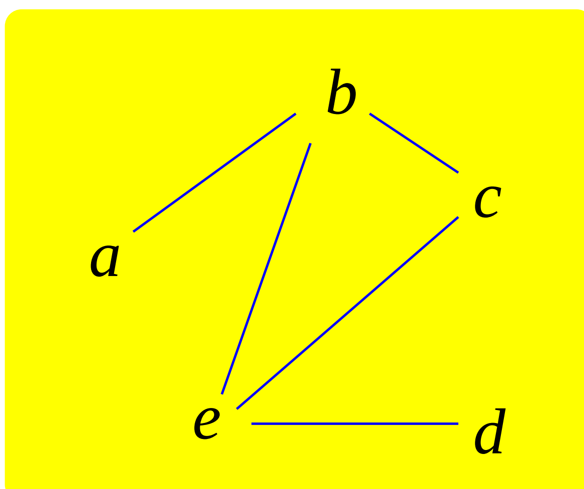
  iff the binary relation r is **symmetric**.

• $\mathbf{S}$ 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
ra	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

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

Example

Consider the clause $\mathsf{C} = \{\neg\, rxfx, \neg\, rfxy\}$.

For

$$\sigma = \begin{pmatrix} x \leftarrow gxfz \\ y \leftarrow fgyx \end{pmatrix},$$

we have $\sigma\mathsf{C} = \{\neg\, rgxfzfgxfz, \neg\, rfgxfzfgyx\}$.

Substitution Theorem

If $\mathbf{S}$ is a structure and C is a clause, then for any substitution σ,

$$\mathbf{S} \models \mathsf{C} \quad \text{implies} \quad \mathbf{S} \models \sigma\mathsf{C}.$$

The **complement** $\overline{\mathsf{L}}$ of a literal L is defined much as it was in the propositional calculus, namely, we convert an atomic formula A to a negated atomic formula $\neg$A, and vice versa.

Resolution for clauses looks similar to that for propositional logic except that **we can first use a substitution**.

If σ is applied to a clause $\mathsf{C} = \{\mathsf{L}_1, \ldots, \mathsf{L}_k\}$, the resulting clause is $\sigma\mathsf{C} = \{\sigma\mathsf{L}_1, \ldots, \sigma\mathsf{L}_k\}$.

As a result of substitution, several literals may collapse into a single literal.

Let $\mathsf{C} = \{rxy,\ rxz,\ \neg\, rzx\}$.

Applying the substitution

$$\sigma = \begin{pmatrix} x \leftarrow w \\ z \leftarrow y \end{pmatrix}$$

yields the clause

$$\sigma\mathsf{C} = \{rwy,\ rwy,\ \neg\, ryw\} = \{rwy,\ \neg\, ryw\}.$$

This gives an example where a substitution collapses three literals into two literals.

An **opp–unifier** (**opposite unifier**) of a pair of clauses C'', D'' is a pair of substitutions σ_1, σ_2 such that

$$\begin{aligned} \sigma_1 C'' &= \{L\} \\ \sigma_2 D'' &= \{\overline{L}\}, \end{aligned}$$

where L is a literal.

This says **all** the literals in C'' become L under the substitution σ_1,

and **all** the literals in D'' become $\overline{L}$ under σ_2.

If an opp–unifier exists, then we say the clauses are **opp–unifiable**.

Example

$$\begin{aligned} \mathsf{C}'' &= \{rxfz,\, rxffy\} \\ \mathsf{D}'' &= \{\neg rf0ffx\} \end{aligned}$$

The pair σ_1, σ_2 given by

$$\sigma_1 = \begin{pmatrix} x \leftarrow f0 \\ z \leftarrow fy \end{pmatrix} \quad \text{and} \quad \sigma_2 = (x \leftarrow y)$$

gives an opp–unifier of $\mathsf{C}'', \mathsf{D}''$; indeed,

$$\begin{aligned} \sigma_1 \mathsf{C}'' &= \{rf0ffy\} \\ \sigma_2 \mathsf{D}'' &= \{\neg rf0ffy\} \end{aligned}$$

Resolution

For $\mathsf{C} = \mathsf{C}' \cup \mathsf{C}''$ and $\mathsf{D} = \mathsf{D}' \cup \mathsf{D}''$:

Resolution of Clauses
$$\frac{\mathsf{C}' \cup \mathsf{C}'',\ \mathsf{D}' \cup \mathsf{D}''}{\sigma_1 \mathsf{C}' \cup \sigma_2 \mathsf{D}'},$$ where σ_1, σ_2 is an opp–unifier of $\mathsf{C}'', \mathsf{D}''$, i.e., $$\sigma_1 \mathsf{C}'' = \{\mathsf{L}\}$$ $$\sigma_2 \mathsf{D}'' = \{\overline{\mathsf{L}}\},$$ with L a literal and $\overline{\mathsf{L}}$ its complement.

A **derivation** of a clause C from a set $\mathcal{S}$ of clauses by resolution is a sequence of clauses

$$\mathsf{C}_1, \ldots, \mathsf{C}_n$$

such that each C_i is either

- a member of $\mathcal{S}$, or
- results from applying resolution to two previous clauses in the sequence,

and the last clause C_n is the clause C.

We write $\mathcal{S} \vdash \mathsf{C}$ (read: C is **derivable from** $\mathcal{S}$) if there is such a derivation.

Theorem [J.A. Robinson 1965]

Soundness and Completeness of Resolution

A set $\mathcal{S}$ of clauses is **not satisfiable** iff

there is a derivation of the empty clause by resolution.

Example

1.	$\{ra\}$	given
2.	$\{\neg\, rx,\ rfx\}$	given
3.	$\{\neg\, rffx\}$	given
4.	$\{\neg\, rfx\}$	resolution 2,3
5.	$\{\neg\, rx\}$	resolution 2,4
6.	$\{\ \}$	resolution 1,5.

Step 4:

$$\begin{cases} \sigma_1 = (x \leftarrow fx) & \text{applied to (2)} \\ \sigma_2 = (x \leftarrow x) & \text{applied to (3).} \end{cases}$$

Step 5: $$\begin{cases} \sigma_1 = (x \leftarrow x) & \text{applied to (2)} \\ \sigma_2 = (x \leftarrow x) & \text{applied to (4).} \end{cases}$$

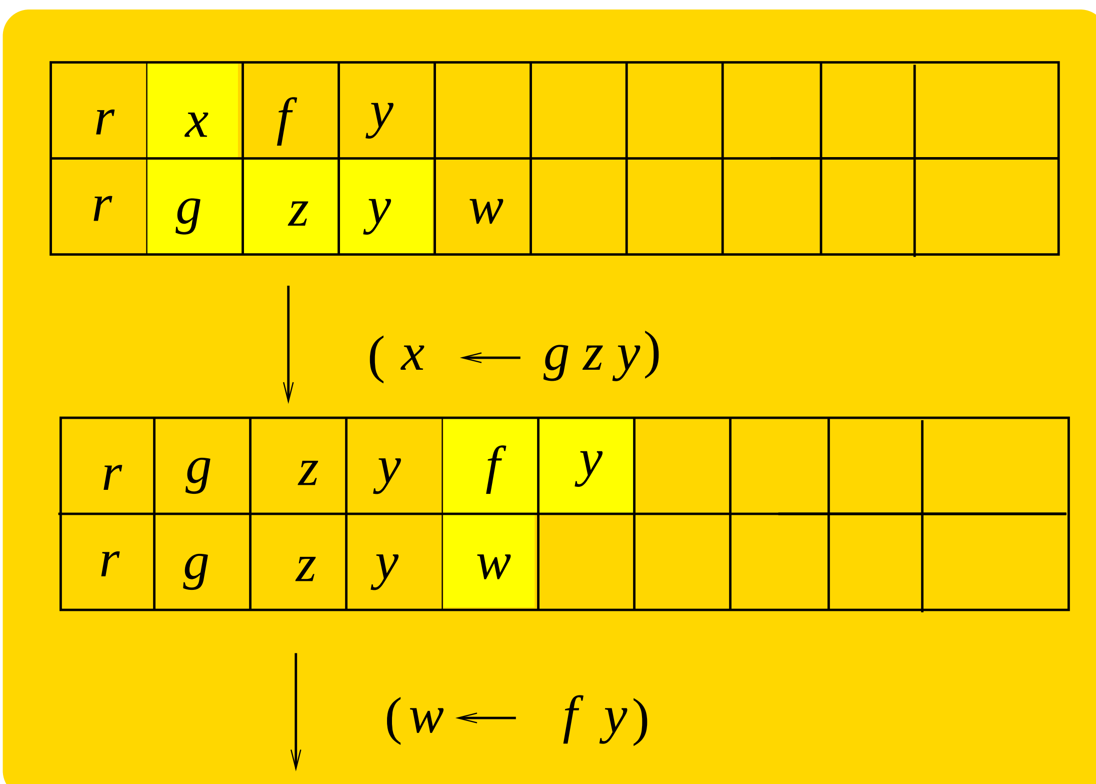
Step 6: $$\begin{cases} \sigma_1 = (x \leftarrow x) & \text{applied to (1)} \\ \sigma_2 = (x \leftarrow a) & \text{applied to (5).} \end{cases}$$

How to Find σ_1 and σ_2

First: Unification of pairs of literals

Use the same algorithms as for pairs of terms.

Example Consider $rxfy$ and $rgzyw$:



r	g	z	y	f	y				
r	g	z	y	f	y				

$$\mu = \begin{pmatrix} x \longleftarrow g\,z\,y \\ w \longleftarrow f\,y \end{pmatrix}$$

The literals are unifiable, and the most general unifier is

$$\mu = \begin{pmatrix} x \leftarrow gzy \\ w \leftarrow fy \end{pmatrix}.$$

Example

Apply the unification algorithm to the following three literals, where r is binary and f is unary:

$$r f f x f y$$

$$r f y f f f z$$

$$r f f f z f f x$$

r	f	f	x	f	y				
r	f	y	f	f	f	z			
r	f	f	f	z	f	f	x		

$$\Big\downarrow \quad (y \longleftarrow f x)$$

r	f	f	x	f	f	x			
r	f	f	x	f	f	f	z		
r	f	f	f	z	f	f	x		

$$\Big\downarrow \quad (x \longleftarrow fz)$$

r	f	f	f	z	f	f	f	z	
r	f	f	f	z	f	f	f	z	
r	f	f	f	z	f	f	f	z	

$$\Big\downarrow$$

$$\mu = \begin{pmatrix} x \longleftarrow fz \\ y \longleftarrow ffz \end{pmatrix}$$

Thus the three literals are unifiable, and the most general unifier is μ.

Most General Opp-Unifiers

If two clauses C'' and D'' are opp-unifiable then it is possible to find **most general opp-unifiers**.

Example

For r a *binary* relation symbol and f a *unary* function symbol let

$$\begin{aligned}\mathsf{C}''(x,y,z) &= \{rfxfy,\ rzy,\ rfyfz\}\\ \mathsf{D}''(x,y,z) &= \{\neg rxfy,\ \neg rfyx\}.\end{aligned}$$

We want to analyze the unifiability of the five literals

$$rfxfy \quad rzy \quad rfyfz \quad rufv \quad rfvu.$$

Applying the unification algorithm for literals:

r	f	x	f	y					
r	z	y							
r	f	y	f	z					
r	u	f	v						
r	f	v	u						

$$\Big\downarrow \; (\, z \leftarrow f\, y \,)$$

r	f	x	f	y					
r	f	y	y						
r	f	y	f	f	y				
r	u	f	v						
r	f	v	u						

$$\Big\downarrow \; (\, u \leftarrow f\, v \,)$$

r	f	x	f	y					
r	f	y	y						
r	f	y	f	f	y				
r	f	v	f	v					
r	f	v	f	v					

$$\big\downarrow \quad (\, v \longleftarrow y \,)$$

r	f	x	f	y					
r	f	y	y						
r	f	y	f	f	y				
r	f	y	f	y					
r	f	y	f	y					

$$\big\downarrow \quad (\, x \longleftarrow y \,)$$

r	f	y	f	y					
r	f	y	y						
r	f	y	f	f	y				
r	f	y	f	y					
r	f	y	f	y					

NOT OPP-UNIFIABLE

Thus the clauses C'', D'' are not opp–unifiable.

Another Example

Now let

$$\begin{aligned} \mathsf{C}''(x,y,z) &= \{rfxfy,\ rfyfz\} \\ \mathsf{D}''(x,y,z) &= \{\neg\, rxfy,\ \neg\, rfyx\}. \end{aligned}$$

We want to analyze the unifiability of the four literals:

$$rfxfy \quad rfyfz \quad rufv \quad rfvu.$$

Applying the unification algorithm for literals:

r	f	x	f	y					
r	f	y	f	z					
r	u	f	v						
r	f	v	u						

$$\downarrow \quad (\, u \leftarrow f\,y \,)$$

r	f	x	f	y					
r	f	y	f	z					
r	f	y	f	v					
r	f	v	f	y					

$$\downarrow \quad (\, v \leftarrow y \,)$$

r	f	x	f	y					
r	f	y	f	z					
r	f	y	f	y					
r	f	y	f	y					

$$\Big\downarrow \; (\, y \leftarrow x \,)$$

r	f	x	f	x					
r	f	x	f	z					
r	f	x	f	x					
r	f	x	f	x					

$$\Big\downarrow \; (\, z \leftarrow x \,)$$

r	f	x	f	x					
r	f	x	f	x					
r	f	x	f	x					
r	f	x	f	x					

$$\mu = (z \leftarrow x)\,(y \leftarrow x)\,(v \leftarrow y)(u \leftarrow fy)$$

$$= \begin{pmatrix} y \leftarrow x \\ z \leftarrow x \\ u \leftarrow fx \\ v \leftarrow x \end{pmatrix} \quad \text{so} \quad \boxed{\begin{array}{l} \mu_1 = \begin{pmatrix} y \leftarrow x \\ z \leftarrow x \end{pmatrix} \\ \\ \mu_2 = \begin{pmatrix} x \leftarrow fx \\ y \leftarrow x \end{pmatrix} \end{array}}$$

Thus the clauses C'', D'' are opp–unifiable, and the most general opp–unifier is given by μ_1, μ_2.

Main Theorem
on
Resolution Theorem Proving

(J.A. Robinson 1965)

A set $\mathcal{S}$ of clauses is not satisfiable iff there is a derivation of the empty clause by resolution using only **most general opp-unifiers**.

Handling Equality

We will handle equality ($\approx$) by giving some of its crucial properties stated as clauses, and then proceed to treat it like any other binary relation symbol.

So let $\mathcal{S}$ be a set of clauses in the language $\mathcal{L} = \mathcal{R} \cup \mathcal{F} \cup \mathcal{C}$ with equality.

Let $\equiv$ be a new binary relation symbol.

First, we give axioms for $\equiv$ so it behaves like equality.

It will be convenient to write $s \not\equiv t$ instead of $\neg\,(s \equiv t)$.

Axiomatizing Equality

Let $\mathsf{Ax}_{\equiv}$ be the set of clauses given by

$\{x \equiv x\}$

$\{x \not\equiv y,\ y \equiv x\}$

$\{x \not\equiv y,\ y \not\equiv z,\ x \equiv z\}$

$\{x_1 \not\equiv y_1, \ldots, x_n \not\equiv y_n,\ f\vec{x} \equiv f\vec{y}\}$ $\quad$ f n–ary in $\mathcal{F}$

$\{x_1 \not\equiv y_1, \ldots, x_n \not\equiv y_n,\ \neg\, r\vec{x},\ r\vec{y}\}$ $\quad$ r n–ary in $\mathcal{R}$.

A clause is a **Horn clause** if there is at most one positive literal in the clause.

The clauses in $\mathsf{Ax}_{\equiv}$ are Horn clauses.

Example

For the language $\mathcal{L} = \{f, r\}$,

where f is *unary* and r is *binary*,

we can formulate $\mathsf{Ax}_{\equiv}$ as the five clauses

$$\{x \equiv x\}$$
$$\{x \not\equiv y,\ y \equiv x\}$$
$$\{x \not\equiv y,\ y \not\equiv z,\ x \equiv z\}$$
$$\{x \not\equiv y,\ \ fx \equiv fy\}$$
$$\{x_1 \not\equiv y_1,\ x_2 \not\equiv y_2,\ \neg\, rx_1x_2,\ ry_1y_2\}.$$

To eliminate $\approx$ we introduce the following:

- Given a clause C:

Define the clause $\mathsf{C}_{\equiv}$ to be the result of replacing $\approx$ in C with $\equiv$.

- Given a set $\mathcal{S}$ of clauses:

Define the set $\mathcal{S}_{\equiv}$ of clauses to be the set of $\mathsf{C}_{\equiv}$ for $\mathsf{C} \in \mathcal{S}$.

Theorem

A set $\mathcal{S}$ of clauses is not satisfiable

iff

the set of clauses $\mathcal{S}_{\equiv} \cup \mathsf{Ax}_{\equiv}$ is not satisfiable

iff

there is a derivation of the empty clause by resolution* from $\mathcal{S}_{\equiv} \cup \mathsf{Ax}_{\equiv}$.

*Using only most general opp-unifiers!

Equational Arguments and Clauses

Now we can adapt our work on clauses to study equational arguments.

Theorem

An equational argument

$$\mathcal{S} \;\therefore s \approx t$$

(that is, $\mathcal{S} \models s \approx t$) is valid iff

$$\neg\,\mathsf{Sat}(\ \mathcal{S}\ \cup\ \{\ s(\vec{c})\ \not\approx\ t(\vec{c})\ \}\ ),$$

where $\vec{c}$ is a sequence of constant symbols that do not appear in the original argument.

Example

We will show

$$x \cdot y \;\approx\; x \;\models\; x \cdot (y \cdot z) \;\approx\; (x \cdot y) \cdot z$$

using clause logic.

First translate the argument into the nonsatisfiability of a set of clauses:

$$\neg\,\text{Sat}(\ \{x \cdot y \;\approx\; x\},\quad \{a \cdot (b \cdot c) \;\not\approx\; (a \cdot b) \cdot c\}\ ).$$

We now want to replace the $\approx$ by $\equiv$.

The set $\mathcal{S}_{\equiv} \cup \mathsf{Ax}_{\equiv}$ of clauses is:

$$\{x \cdot y \equiv x\}$$
$$\{a \cdot (b \cdot c) \not\equiv (a \cdot b) \cdot c\}$$
$$\{x \equiv x\}$$
$$\{x \not\equiv y,\ y \equiv x\}$$
$$\{x \not\equiv y,\ y \not\equiv z,\ x \equiv z\}$$
$$\{x_1 \not\equiv y_1,\ x_2 \not\equiv y_2,\ x_1 \cdot x_2 \equiv y_1 \cdot y_2\}.$$

We want to show this set of clauses is not satisfiable.

We can approach this two ways, via ground instances or via resolution.

For the ground instances method consider

Clause	Ground instances
$\{x \cdot y \equiv x\}$	$\{(a \cdot b) \cdot c \equiv a \cdot b\}$
	$\{a \cdot b \equiv a\}$
	$\{a \cdot (b \cdot c) \equiv a\}$
$\{a \cdot (b \cdot c) \not\equiv (a \cdot b) \cdot c\}$	$\{a \cdot (b \cdot c) \not\equiv (a \cdot b) \cdot c\}$
$\{x \not\equiv y,\ y \equiv x\}$	$\{(a \cdot b) \cdot c \not\equiv a,\ a \equiv (a \cdot b) \cdot c\}$
$\{x \not\equiv y,\ y \not\equiv z,\ x \equiv z\}$	$\{(a \cdot b) \cdot c \not\equiv a \cdot b,\ a \cdot b \not\equiv a,\ (a \cdot b) \cdot c \equiv a\}$
	$\{a \cdot (b \cdot c) \not\equiv a,\ a \not\equiv (a \cdot b) \cdot c,$
	$a \cdot (b \cdot c) \equiv (a \cdot b) \cdot c\}$

Rename the (ground) atomic formulas:

$$P: \ (a \cdot b) \cdot c \ \equiv \ a \cdot b$$
$$Q: \ a \cdot b \ \equiv \ a$$
$$R: \ a \cdot (b \cdot c) \ \equiv \ a$$
$$S: \ (a \cdot b) \cdot c \ \equiv \ a$$
$$T: \ a \ \equiv \ (a \cdot b) \cdot c$$
$$U: \ a \cdot (b \cdot c) \ \equiv \ (a \cdot b) \cdot c,$$

The ground instances become

$$\{P\} \quad \{Q\} \quad \{R\} \quad \{\neg U\}$$
$$\{\neg S, \ T\}$$
$$\{\neg P, \ \neg Q, \ S\}$$
$$\{\neg R, \ \neg T, \ U\}.$$

This collection of propositional clauses is easily seen to be unsatisfiable.

And for the direct resolution method:

1.	$\{x \cdot y \ \equiv \ x\}$	given
2.	$\{a \cdot (b \cdot c) \ \not\equiv \ (a \cdot b) \cdot c\}$	given
3.	$\{x \ \not\equiv \ y,\ y \ \equiv \ x\}$	given
4.	$\{x \ \not\equiv \ y,\ y \ \not\equiv \ z,\ x \ \equiv \ z\}$	given
5.	$\{a \cdot (b \cdot c) \ \not\equiv \ y,\ y \ \not\equiv \ (a \cdot b) \cdot c\}$	$2, 4$
6.	$\{a \ \not\equiv \ (a \cdot b) \cdot c\}$	$1, 5$
7.	$\{(a \cdot b) \cdot c \ \not\equiv \ a\}$	$3, 6$
8.	$\{(a \cdot b) \cdot c \ \not\equiv \ y,\ y \ \not\equiv \ a\}$	$4, 7$
9.	$\{a \cdot b \ \not\equiv \ a\}$	$1, 8$
10.	$\{\ \}$	$1, 9.$