Equational Logic

A **language** $\mathcal{L}$ of **algebras** (or **algebraic structures**) consists of

- a set $\mathcal{F}$ of **function symbols** $f, g, h, \cdots$
- a set $\mathcal{C}$ of **constant symbols** $c, d, e, \cdots$
- a set X of **variables** $x, y, z \cdots$.

Each function symbol has an **arity** to indicate how many arguments it takes.

If the symbol takes n arguments we say it is n-**ary**.

$n =$	1	2	3
	unary	**binary**	**ternary**

Example

The **language $\mathcal{L}_{BA}$ of Boolean algebras** has

$$\mathcal{F} = \{\vee, \wedge, '\} \qquad \mathcal{C} = \{0, 1\}$$

$\vee$ and $\wedge$ are binary function symbols,

$'$ is a unary function symbol.

symbol	name
$\vee$	**join**
$\wedge$	**meet**
$'$	**complement**

The constants are just called by the usual names **zero** and **one**.

The **meaning** of the symbols

Given a set A:

- Function symbols are interpreted as **functions** on the set.

- Constant symbols are interpreted as **elements** of the set.

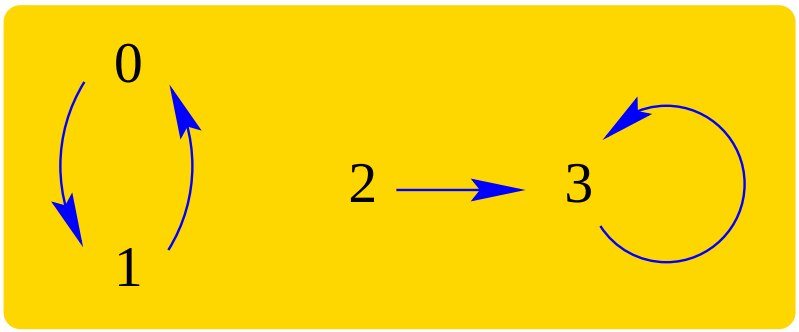
A function f that maps n-tuples of elements of A to A is called an n-**ary** function on A.

We write $f : A^n \to A$ to say f is n-ary on A.

We can describe a single unary function on a small set either with a table, e.g.,

	f
0	1
1	0
2	3
3	3

or with a directed graph representation:



Cayley Tables

We can also describe small binary functions on a set A using a table, called a **Cayley table**.

To describe the integers mod 4, with the binary operation of multiplication mod 4, we have the Cayley table:

$\cdot$	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	0	2
3	0	3	2	1

Function Tables

We can also describe functions on a small set A using a table that is similar to the truth tables used to describe the connectives.

To describe the ternary function $f(x, y, z) = 1 + xyz$ on the integers modulo 2 we could use

x	y	z	f
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

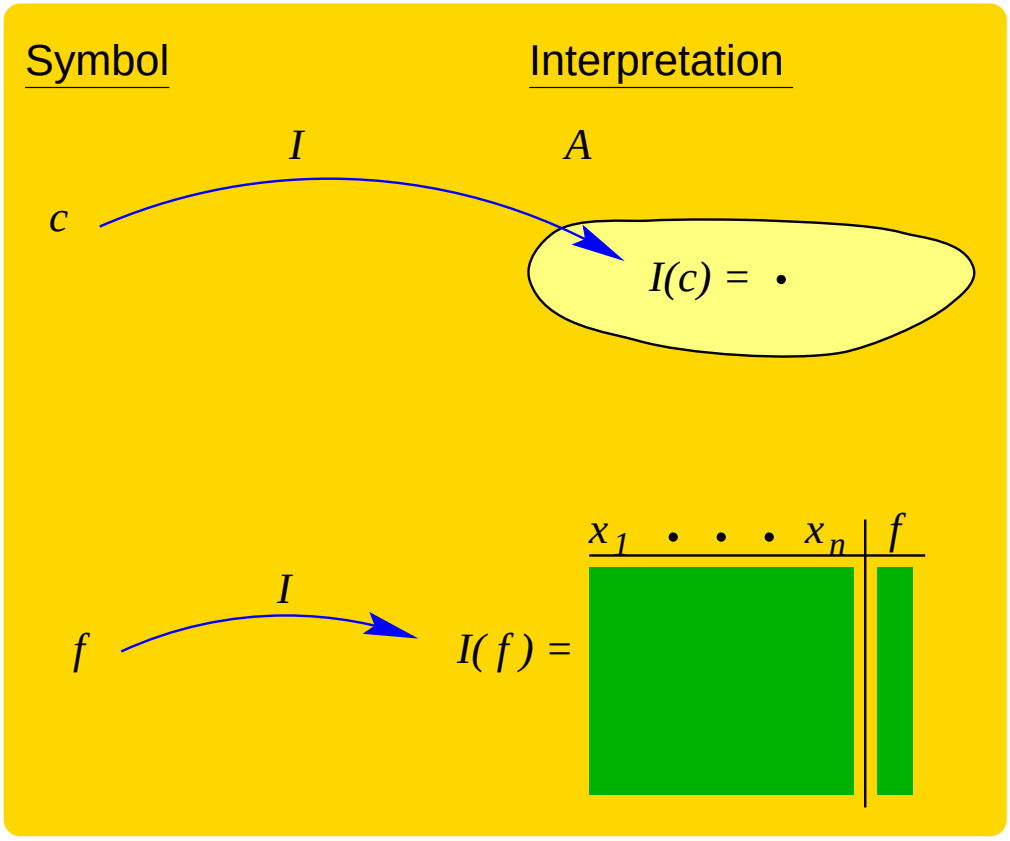
Interpretations

An **interpretation** I of the language $\mathcal{L}$ on a nonempty set A assigns to each symbol from $\mathcal{L}$ a function or constant as follows:

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

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

Visualizing an interpretation I on a set A:



Algebras

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

Given an algebra $\mathbf{A}$:

the interpretations of the constant symbols are called the **constants** of the algebra

the interpretations of the function symbols are called the **fundamental operations** of the algebra.

Notation

We prefer to write

$$\begin{array}{ll} c^{\mathbf{A}} & \text{for } I(c) \\ f^{\mathbf{A}} & \text{for } I(f), \end{array}$$

or even simpler,

$$\begin{array}{ll} c & \text{for } I(c) \\ f & \text{for } I(f), \end{array}$$

Also, rather than writing (A, I) we prefer to write

$$(A, \mathcal{F}, \mathcal{C})$$

or to simply list the symbols in $\mathcal{F}$ and $\mathcal{C}$, e.g., the algebra

$$(R, +, \cdot, 0, 1),$$

Example

Let $\mathcal{L} = \mathcal{L}_{BA} = \{\vee, \wedge, ', 0, 1\}$

Let $Su(U)$ be the collection of all subsets of a given set U (U is called the **universe**).

Interpret join as **union** ($\cup$)
meet as **intersection** ($\cap$)
complement as **complement** ($'$) in U
0 as the **empty set** ($\varnothing$)
1 as the **universe** (U).

Then $\mathbf{Su}(U) = (Su(U), \cup, \cap, ', \varnothing, U)$

is **the Boolean algebra of subsets of** U.

Example

Let $\mathcal{L} = \mathcal{L}_{BA} = \{\vee, \wedge, ', 0, 1\}$

Let $A = \{0, 1\}$, and let the function symbols be interpreted as follows:

$\vee$	0	1
0	0	1
1	1	1

$\wedge$	0	1
0	0	0
1	0	1

	$'$
0	1
1	0

and $0, 1$ are interpreted in the obvious manner (as $0, 1$).

This is the best known of all the Boolean algebras.

Terms are used to make the sides of equations.

Examples of terms using familiar infix notation for the language $\{+, \cdot, -, 0, 1\}$:

$0 \qquad 1 \qquad x \qquad y$

$-0 \qquad -1 \qquad -x \qquad -y$

$1 + 0 \qquad x \cdot y \qquad -(-x) \qquad x + 1$

$x \cdot (y + z) \qquad (-x) \cdot (-y) \qquad 1 + (0 + 1)$

Examples

of terms using familiar infix notation for the language of Boolean algebra $\{\vee, \wedge, ', 0, 1\}$:

$0 \qquad 1 \qquad x \qquad y$

$0' \qquad 1' \qquad x' \qquad y'$

$1 \vee 0 \qquad x \wedge y \qquad x'' \qquad x \vee 1$

$x \wedge (y \vee z) \qquad (x') \wedge (y') \qquad 1 \vee (0 \vee 1)$

Examples

of terms using **prefix** notation.

If f is a unary function symbol:

x fx, ffx are terms.

If c is a constant symbol then

c fc ffc are terms.

If g is a binary function symbol then

gcx gyy $ggfzcgcx$ are terms.

If h is a ternary function then

$hxyz$ $hccx$ $hfgxcgxcggxyfc$ are terms.

Terms

The $\mathcal{L}$-**terms** over X are (inductively) defined by the following:

- A **variable** $\boxed{x}$ in X is an $\mathcal{L}$–term.
- A **constant symbol** $\boxed{c}$ in $\mathcal{C}$ is an $\mathcal{L}$–term.
- If $t_1, \ldots, t_n$ are $\mathcal{L}$–terms

  and f is an n–ary function symbol in $\mathcal{F}$,

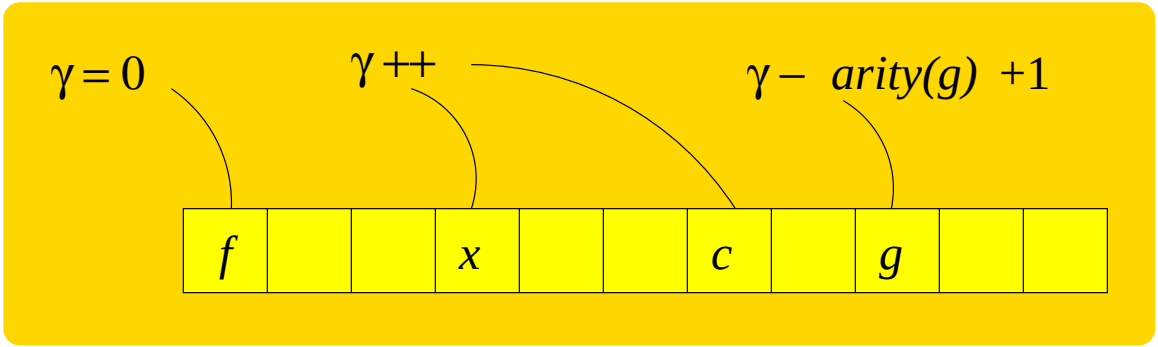
  then $\boxed{ft_1 \cdots t_n}$ is an $\mathcal{L}$–term.

A Parsing Algorithm

(for terms in prefix form)

Define γ on the symbols of a string $s = fs_1 \cdots s_n$ by:

- γ at the first symbol f is 0.
- Increase γ by one on variables and constants.
- Decrease γ by $arity - 1$ on the remaining function symbols.



Then s is a term **iff**

the value of γ is always less than arity(f)
except

at the last symbol, where γ has the value
arity(f).

If s is a term then $s = ft_1 \cdots t_k$

where $k =$ arity(f).

The end of t_i is the first symbol where γ is
i.

Example

Let $\mathcal{L} = \{f, g, c\}$ with

f unary g binary c a constant.

Determine if $s = ggcxfz$ is a term.

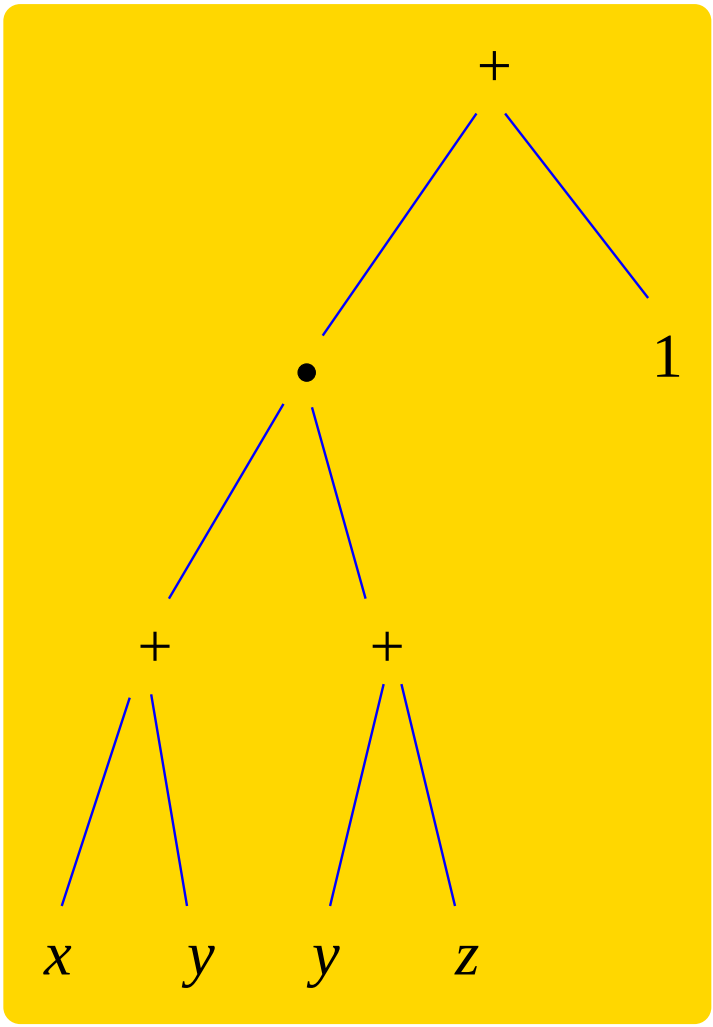
If so find the subterms t_1, t_2 such that $gt_1t_2 = ggcxfz$.

Here is the computation of γ:

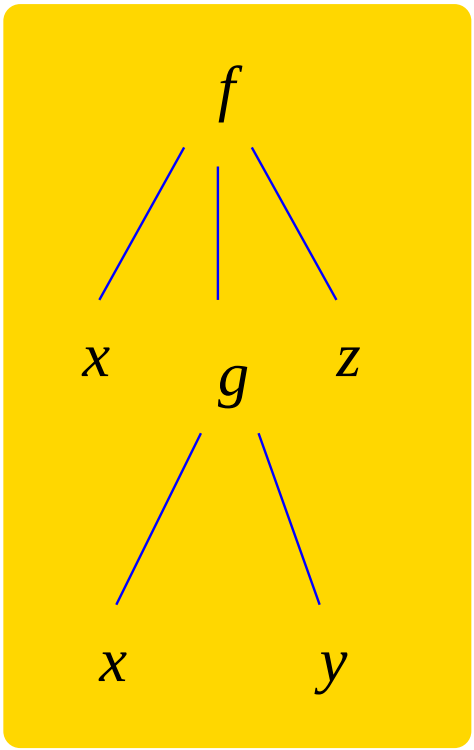
i	0	1	2	3	4	5
s_i	g	g	c	x	f	z
γ_i	0	-1	0	1	1	2

Examples of the **tree** of a term:

$((x+y)\bullet(y+z))+1$



$fxgxyz$ (f is ternary, g binary)



Looking at the tree of a term we see that it is built up in stages called **subterms**.

Example

Using infix notation, the subterms of

$$((x + y) \cdot (y + z)) + 1$$

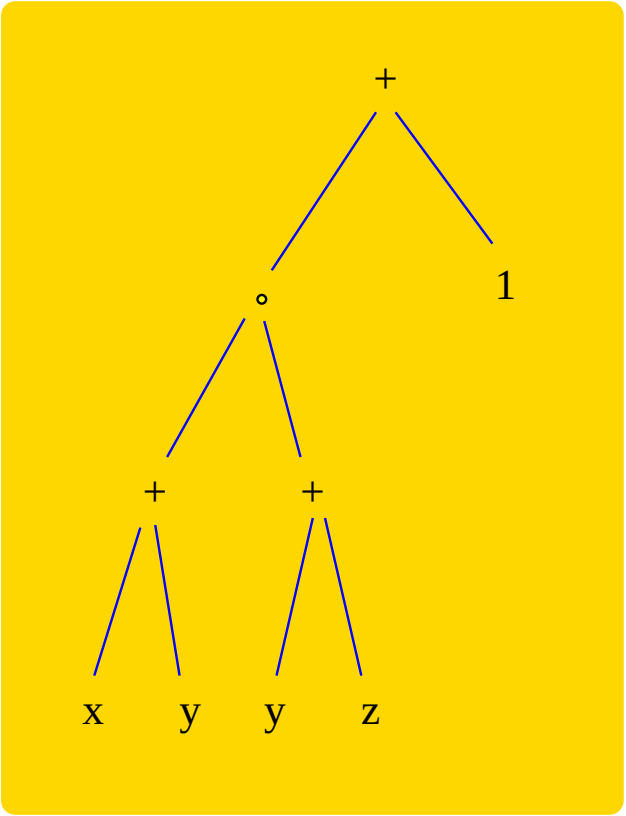
are

$\boxed{x}$ $\boxed{y}$ $\boxed{z}$ $\boxed{1}$

$\boxed{x + y}$ $\boxed{y + z}$

$\boxed{(x + y) \cdot (y + z)}$

$\boxed{((x + y) \cdot (y + z)) + 1}$



Example

Using prefix notation, the subterms of

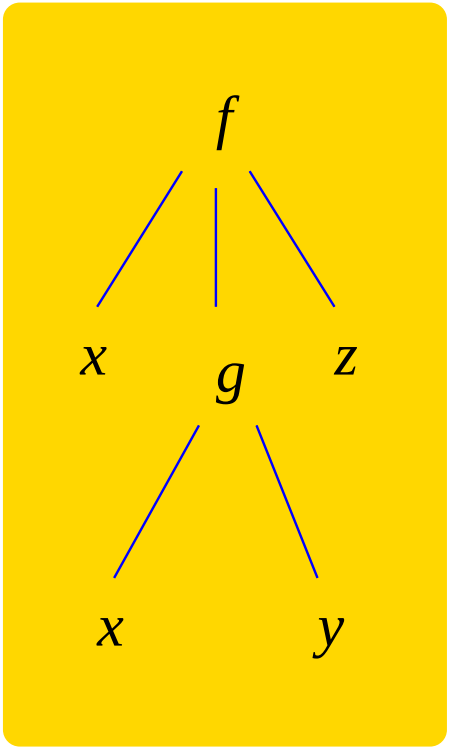
$$fxgxyz \ ,$$

where f is ternary and g is binary, are

$\boxed{x}$ $\boxed{y}$ $\boxed{z}$

$\boxed{gxy}$

$\boxed{fxgxyz}$



Subterms

The **subterms** of a term t are defined inductively:

- The only subterm of a variable $\boxed{x}$ is the variable x itself.

- The only subterm of a constant symbol $\boxed{c}$ is the symbol c itself.

- The subterms of $\boxed{ft_1 \cdots t_n}$ are

$ft_1 \cdots t_n$ itself and all the subterms of the t_i, for $1 \leq i \leq n$.

Term Functions

We interpret terms in an algebra as functions:

terms $t(x_1, \ldots, x_n)$ define **functions** $t^{\mathbf{A}} : A^n \Longrightarrow A$

Example

Using the usual language for the natural numbers, consider the term $t(x, y, z)$ given by

$$(x \cdot (y + 1)) + z$$

The corresponding term function $t^{\mathbf{N}} : N^3 \Longrightarrow N$ maps the triple $(1, 0, 2)$ to 3 as

$$t^{\mathbf{N}}(1, 0, 2) \;=\; (1 \cdot (0 + 1)) + 2 \;=\; 3$$

Term functions $t^{\mathbf{A}}$ for terms $t(x_1, \ldots, x_n)$ are the functions on the algebra $\mathbf{A}$ defined inductively by the following:

- If t is $\boxed{x_i}$ then $t^{\mathbf{A}}(a_1, \ldots, a_n) = a_i$.
- If t is $\boxed{c \in \mathcal{C}}$ then $t^{\mathbf{A}}(a_1, \ldots, a_n) = c^{\mathbf{A}}$.
- If t is $\boxed{ft_1 \cdots t_k}$ then $t^{\mathbf{A}} = f^{\mathbf{A}}(t_1^{\mathbf{A}}, \ldots, t_k^{\mathbf{A}})$.

Evaluation Tables

Example

Let $\mathbf{A}$ be the two–element Boolean algebra on $\{0,1\}$, and let $t(x,y,z)$ be the term $x \vee (y \wedge z')$.

Then the term function $t^{\mathbf{A}}$ can be described by:

x	y	z	t
1	1	1	1
1	1	0	1
1	0	1	1
1	0	0	1
0	1	1	0
0	1	0	1
0	0	1	0
0	0	0	0

or, in more detail

x	y	z	z'	$y \wedge z'$	$\overbrace{x \vee (y \wedge z')}^{t}$
1	1	1	0	0	1
1	1	0	1	1	1
1	0	1	0	0	1
1	0	0	1	0	1
0	1	1	0	0	0
0	1	0	1	1	1
0	0	1	0	0	0
0	0	0	1	0	0

Equations

An $\mathcal{L}$-**equation** is an expression of the form

$$s \approx t$$

where s and t are $\mathcal{L}$–terms.

For example: $x + y \approx y + x$.

We think of this as saying
for all x and y, $x + y = y + x$.

$=$ simply means "identical to".

$\approx$ expresses equality in formulas of our logics.

The Basic Question: When does an equation follows from other equations?

From

$$x + y \approx y + x$$

we can conclude

$$x + (y + z) \approx (z + y) + x$$

but not

$$x + x \approx x + y$$

Two vantage points:

- (**semantic**) a notion of when an equation is true in an algebra;

- (**syntactic**) axioms and rules of inference for deriving equational consequences of equations.

When you see the word "syntatic" think **strings of symbols**.

Axioms are strings of symbols.

Rules of Inference tell you how to take strings of symbols and make new strings of symbols.

The semantic and syntatic notions of equational logic are equivalent!

This equivalence is the **soundness** and **completness** of equational logic.

Soundness means that one can only derive equations that correspond to valid arguments.

Completeness means that one can derive all equations that correspond to valid arguments.

Semantics

Suppose $\mathcal{L}$ is a language of algebras.

• For $\mathbf{A}$ an $\mathcal{L}$–algebra and $s \approx t$ an $\mathcal{L}$–equation we write

$$\mathbf{A} \models s \approx t$$

and say

$\mathbf{A}$ **satisfies** $s \approx t$ or $s \approx t$ **holds** in $\mathbf{A}$ if $s^{\mathbf{A}} = t^{\mathbf{A}}$,
that is, s and t define the same term function on A.

• If $\mathbf{A}$ is an $\mathcal{L}$–algebra and $\mathcal{S}$ is a set of $\mathcal{L}$–equations, then we write

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

and say

$$\mathbf{A} \textbf{ satisfies } \mathcal{S} \quad \text{or} \quad \mathcal{S} \textbf{ holds in } \mathbf{A}$$

if $\mathbf{A} \models s \approx t$ for every equation $s \approx t$ in $\mathcal{S}$.

• If $\mathcal{S}$ is a set of $\mathcal{L}$–equations and $s \approx t$ is an $\mathcal{L}$–equation, then we write

$$\mathcal{S} \models s \approx t$$

and say

$s \approx t$ is a **consequence** of $\mathcal{S}$ or

$s \approx t$ **follows from** $\mathcal{S}$

if $\mathbf{A} \models s \approx t$ whenever $\mathbf{A} \models \mathcal{S}$.

For **satisfies** we also use the word **models**, and we say $\mathbf{A}$ is a **model** of an equation or a set of equations.

Laws of Binary Algebras

A binary algebra $\mathbf{A} = (A, \cdot)$ is **associative** if it satisfies the associative law

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

In such case it is also called a **semigroup**.

It is **commutative** if it satisfies the commutative law

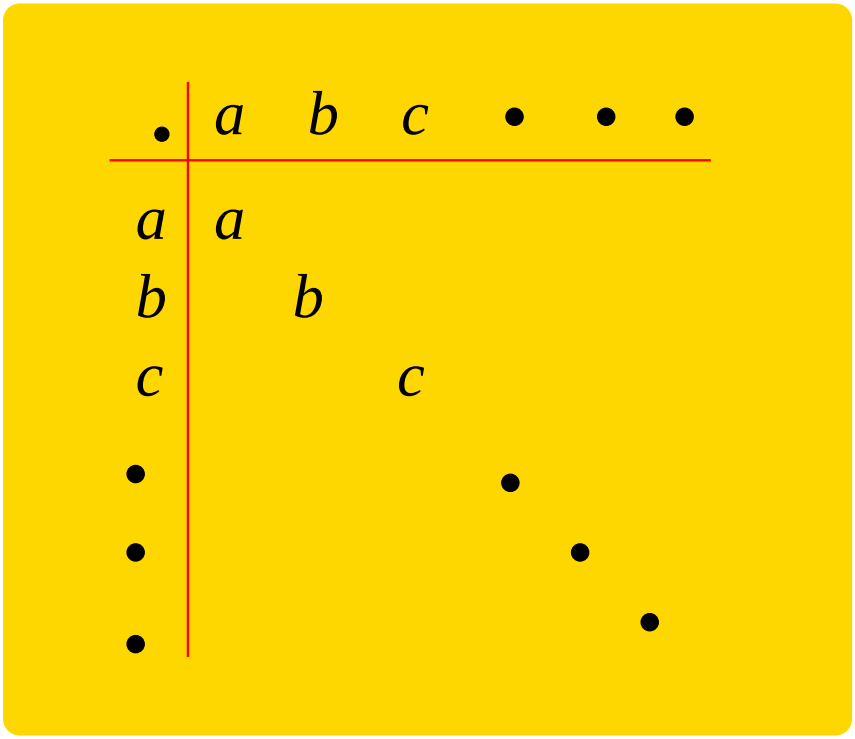
$$x \cdot y \approx y \cdot x$$

And it is **idempotent** if it satisfies the idempotent law

$$x \cdot x \approx x.$$

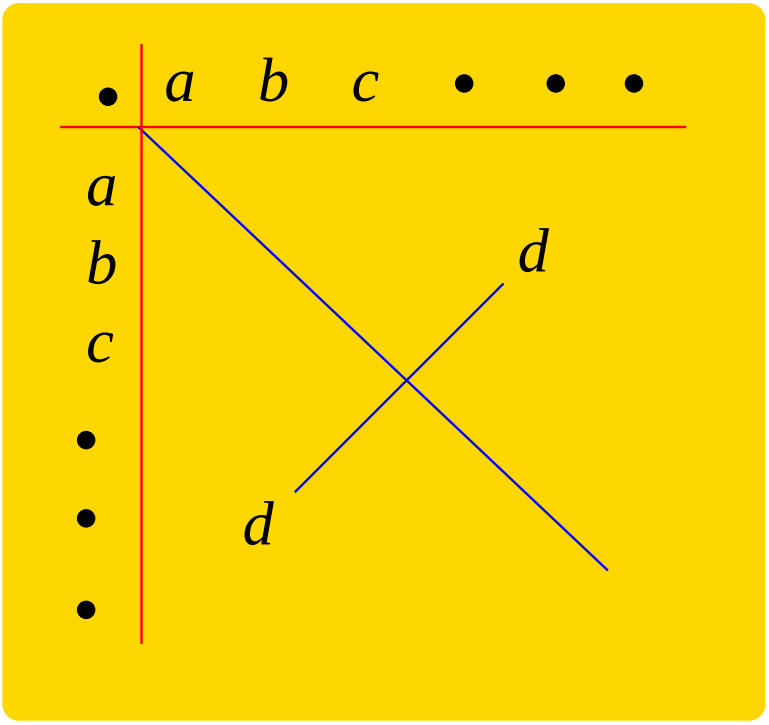
The **idempotent law** is easy to check:

one looks down the main diagonal of the table of the operation to see that $x \cdot x$ always has the value x.



The **commutative law** is also easy to check:

look at the table of the operation to see if it is **symmetric** about the main diagonal.



Example

The binary algebra $\mathbf{A}$ given by

$\cdot$	a	b
a	a	a
b	a	b

is idempotent, commutative, and associative:

$$\begin{aligned} \mathbf{A} &\models x \cdot x \approx x \\ \mathbf{A} &\models x \cdot y \approx y \cdot x \\ \mathbf{A} &\models x \cdot (y \cdot z) \approx (x \cdot y) \cdot z. \end{aligned}$$

The first two properties follow by the preceding remarks and diagrams.

To check the associative law we construct an evaluation table for the terms $x \cdot (y \cdot z)$ and $(x \cdot y) \cdot z$ to show the term functions are equal:

From the binary algebra given by

$\cdot$	a	b
a	a	a
b	a	b

we have the following table:

	x	y	z	$y \cdot z$	$\overbrace{x \cdot (y \cdot z)}^{s}$	$x \cdot y$	$\overbrace{(x \cdot y) \cdot z}^{t}$
1.	a	a	a	a	a	a	a
2.	a	a	b	a	a	a	a
3.	a	b	a	a	a	a	a
4.	a	b	b	b	a	a	a
5.	b	a	a	a	a	a	a
6.	b	a	b	a	a	a	a
7.	b	b	a	a	a	b	a
8.	b	b	b	b	b	b	b

Since the columns for $x \cdot (y \cdot z)$ and $(x \cdot y) \cdot z$ are the same, the associative law holds.

Example

The binary algebra $\mathbf{A}$ given by

$\cdot$	a	b
a	b	a
b	b	b

is not idempotent nor commutative nor associative.

To see the failure of the associative law we use an evaluation table:

	x	y	z	$y \cdot z$	$\overbrace{x \cdot (y \cdot z)}^{s}$	$x \cdot y$	$\overbrace{(x \cdot y) \cdot z}^{t}$
1.	a	a	a	b	a	b	b
2.	a	a	b	a	b	b	b
3.	a	b	a	b	a	a	b
4.	a	b	b	b	a	a	a
5.	b	a	a	b	b	b	b
6.	b	a	b	a	b	b	b
7.	b	b	a	b	b	b	b
8.	b	b	b	b	b	b	b

Indeed, associativity fails precisely on lines 1 and 3.

However, this binary algebra

$\cdot$	a	b
a	b	a
b	b	b

does satisfy an interesting equation, namely,

$$\mathbf{A} \models (x \cdot y) \cdot z \approx (x \cdot z) \cdot y.$$

We can see this by using an evaluation table:

x	y	z	$x \cdot y$	$\overbrace{(x \cdot y) \cdot z}^{s}$	$x \cdot z$	$\overbrace{(x \cdot z) \cdot y}^{t}$
a	a	a	b	b	b	b
a	a	b	b	b	a	b
a	b	a	a	b	b	b
a	b	b	a	a	a	a
b	a	a	b	b	b	b
b	a	b	b	b	b	b
b	b	a	b	b	b	b
b	b	b	b	b	b	b

Many interesting classes of algebras are defined by equations.

The text gives examples of:

- Rings
- Boolean Algebras
- Semigroups
- Monoids
- Groups

You will NOT be required to memorize the equations defining these classes. They are for reference only.

Rings

The language $\mathcal{L}_{\mathcal{R}}$ is $\{+, \cdot, -, 0, 1\}$.

$\mathcal{R}$ is the following set of equations in this language:

R1.	$x + 0$	$\approx$	x	additive identity
R2.	$x + (-x)$	$\approx$	0	additive inverse
R3.	$x + y$	$\approx$	$y + x$	$+$ is commutative
R4.	$x + (y + z)$	$\approx$	$(x + y) + z$	$+$ is associative
R5.	$x \cdot 1$	$\approx$	x	right mult. identity
R6.	$1 \cdot x$	$\approx$	x	left mult. identity
R7.	$x \cdot (y \cdot z)$	$\approx$	$(x \cdot y) \cdot z$	$\cdot$ is associative
R8.	$x \cdot (y + z)$	$\approx$	$(x \cdot y) + (x \cdot z)$	left distributive
R9.	$(x + y) \cdot z$	$\approx$	$(x \cdot z) + (y \cdot z)$	right distributive.

All algebras $\mathbf{R} = (R, +, \cdot, -, 0, 1)$ that satisfy $\mathcal{R}$ are called **rings**.

$\mathcal{R}$ is called a **set of axioms** or a **set of defining equations** for rings.

Boolean Algebras

We choose $\mathcal{BA}$ to be the following set of equations in the language $\mathcal{L}_{\mathcal{BA}}$:

B1.	$x \vee y$	$\approx$	$y \vee x$	commutative
B2.	$x \wedge y$	$\approx$	$y \wedge x$	commutative
B3.	$x \vee (y \vee z)$	$\approx$	$(x \vee y) \vee z$	associative
B4.	$x \wedge (y \wedge z)$	$\approx$	$(x \wedge y) \wedge z$	associative
B5.	$x \wedge (x \vee y)$	$\approx$	x	absorption
B6.	$x \vee (x \wedge y)$	$\approx$	x	absorption
B7.	$x \wedge (y \vee z)$	$\approx$	$(x \wedge y) \vee (x \wedge z)$	distributive
B8.	$x \vee x'$	$\approx$	1	
B9.	$x \wedge x'$	$\approx$	0	
B10.	$x \vee 1$	$\approx$	1	
B11.	$x \wedge 0$	$\approx$	0.	

All algebras $\mathbf{B} = (B, \vee, \wedge, ', 0, 1)$ that satisfy $\mathcal{BA}$ are called **Boolean algebras**.

$\mathcal{BA}$ is called a **set of axioms** or a **set of defining equations** for Boolean algebra.

Semigroups

The language $\mathcal{L}_{\mathcal{SG}} = \{\cdot\}$ consists of a single binary operation.

$\mathcal{SG}$ has only one equation, the associative law:

$$\text{SG1}: \quad (x \cdot y) \cdot z \approx x \cdot (y \cdot z).$$

Models of $\mathcal{SG}$ are called **semigroups**.

$\mathcal{SG}$ **axiomatizes** or **defines** the class of semigroups.

Monoids

$\mathcal{L}_{\mathcal{M}}$ is $\{\cdot, 1\}$, where $\cdot$ is binary and 1 is a constant symbol.

$\mathcal{M}$ consists of:

$$\begin{aligned} &\text{MO1}: \quad x \cdot 1 \approx x \\ &\text{MO2}: \quad 1 \cdot x \approx x \\ &\text{MO3}: \quad (x \cdot y) \cdot z \approx x \cdot (y \cdot z)\ . \end{aligned}$$

Any algebra $\mathbf{A}$ that satisfies $\mathcal{M}$ is called a **monoid**.

$\mathcal{M}$ is a set of **axioms** or **defining equations** for monoids.

Groups

$\mathcal{L}_{\mathcal{G}} = \{\cdot, ^{-1}, 1\}$, where $\cdot$ is binary, $^{-1}$ is unary, and 1 is a constant symbol.

$\mathcal{G}$ is the set of equations:

$$\begin{array}{lrcl} \text{G1}: & x \cdot 1 & \approx & x \\ \text{G2}: & x \cdot x^{-1} & \approx & 1 \\ \text{G3}: & (x \cdot y) \cdot z & \approx & x \cdot (y \cdot z). \end{array}$$

There is also an **additive notation** for groups, namely, the language is $\{+, -, 0\}$ and is usually reserved for groups that are **commutative**:

$$\begin{array}{lrcl} \text{G1}': & x + 0 & \approx & x \\ \text{G2}': & x + (-x) & \approx & 0 \\ \text{G3}': & (x + y) + z & \approx & x + (y + z) \\ \text{G4}': & x + y & \approx & y + x. \end{array}$$

Three Very Basic Properties of Equations

($\approx$ behaves like an **equivalence relation**)

- $\mathbf{A} \models s \approx s$
- $\mathbf{A} \models s \approx t$ implies $\mathbf{A} \models t \approx s$
- $\mathbf{A} \models s_1 \approx s_2$ and $\mathbf{A} \models s_2 \approx s_3$

  implies $\mathbf{A} \models s_1 \approx s_3$.

Similar results hold for **consequences**.
(Recall the definition of $\mathcal{S} \models s \approx t$ from slide III.33, namely that any algebra that satisfies $\mathcal{S}$ will also satisfy $s \approx t$.)

- $\mathcal{S} \models s \approx s$
- $\mathcal{S} \models s \approx t$ implies $\mathcal{S} \models t \approx s$
- $\mathcal{S} \models s_1 \approx s_2$ and $\mathcal{S} \models s_2 \approx s_3$

  implies $\mathcal{S} \models s_1 \approx s_3$.

Arguments that are Valid in $\mathbf{A}$

Given an algebra $\mathbf{A}$, an argument

$$s_1 \approx t_1,\ \cdots,\ s_n \approx t_n \quad \therefore s \approx t,$$

is **valid in A** (or **correct in A**) provided either

- some equation $s_i \approx t_i$ does not hold in $\mathbf{A}$, or
- $s \approx t$ holds in $\mathbf{A}$,

i.e., if all the premisses hold in $\mathbf{A}$, then so does the conclusion.

Let $\mathbf{A}$ be the two–element Boolean algebra.
To check the validity of the argument

$$x \vee y \approx y \vee x, \quad x \wedge (y \vee x) \approx x \quad \therefore x \vee x' \approx x \wedge x'$$

we form an evaluation table:

		$\boxed{s_1}$	$\boxed{t_1}$	$\boxed{s_2}$	$\boxed{t_2}$	$\boxed{s}$	$\boxed{t}$
x	y	$x \vee y$	$y \vee x$	$x \wedge (y \vee x)$	x	$x \vee x'$	$x \wedge x'$
0	0	0	0	0	0	1	0
0	1	1	1	0	0	1	0
1	0	1	1	1	1	1	0
1	1	1	1	1	1	1	0

Thus this is not a valid argument in $\mathbf{A}$.

Valid Equational Arguments

An argument

$$s_1 \approx t_1,\ \cdots,\ s_n \approx t_n \quad \therefore s \approx t,$$

is **valid** (or **correct**) provided it is valid in **every** algebra $\mathbf{A}$.

Of course this becomes impossible to check using evaluation tables because there are an infinite number of algebras to examine.

So how do we ever verify that an equational argument is valid?

There are two ways. One is to use a proof system that we will soon study.

The other is to study abstract algebra to learn special methods that can aid in a semantic analysis of validity.

Refuting an Equational Argument

An equational argument

$$s_1 \approx t_1,\ \cdots,\ s_n \approx t_n \quad \therefore s \approx t$$

is **not valid** iff one can find an algebra $\mathbf{A}$ such that

- all the premisses hold in $\mathbf{A}$,
- but the conclusion does not hold.

Such an $\mathbf{A}$ is called a **counterexample** to the argument.

One-Element Algebras

We can **never** use a one–element algebra to refute an equational argument because in a one–element algebra **all** equations are true

(there is only one value for the term functions to take).

So to find a counterexample to an equational argument one must look to algebras with more than one element.

Example

Show that the argument

$$x \cdot y \approx y \cdot x \quad \therefore x \cdot x \approx x$$

is not valid.

We need to find a binary algebra that is commutative but not idempotent.

The following two–element binary algebra does the job:

$\cdot$	a	b
a	a	a
b	a	a

Example

Show that the argument

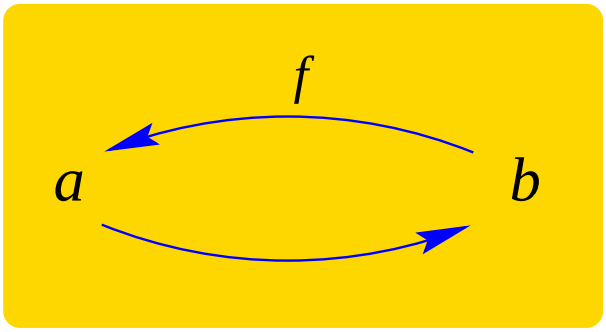
$$fffx \approx fx \quad \therefore ffx \approx fx$$

is not valid.

The following two–element monounary algebra provides a counterexample:

	f
a	b
b	a

Such examples we usually discover by drawing pictures such as the following:



What is Substitution?

The word **substitution** encompasses two quite distinct kinds of substitution.

One is for **uniform substitution for variables**. For example from

$$x + y \approx y + x \qquad (*)$$

we can deduce

$$(x \cdot y) + (z + z) \approx (z + z) + (x \cdot y)$$

by substituting in (*) as follows:

the term $x \cdot y$ for every occurrence of x

the term $z + z$ for every occurrence of y.

The other use of substitution is to **substitute one term for another**.

For example from

$$x + y \;\approx\; y + x$$

we can also deduce

$$(u \cdot (\underline{x + y})) + (z + z) \;\approx\; (u \cdot (\underline{y + x})) + (z + z)$$

Notice that we **replaced** the underlined term on the left with the underlined term on the right.

Since these two kinds of substitution are so different we will use the word **substitution** only for the first kind, that is, for the uniform substitution for variables; and call the second kind of substitution **replacement**.

Substitution

Given a term

$$s(x_1, \ldots, x_n)$$

and terms $t_1, \ldots, t_n$, the expression

$$s(t_1, \ldots, t_n)$$

denotes the result of **simultaneously** substituting t_i for x_i in s.

The notation we use for the **substitution** is

$$\begin{pmatrix} x_1 \leftarrow t_1 \\ \vdots \\ x_n \leftarrow t_n \end{pmatrix}.$$

Example

Let $s(x, y)$ be $x + y$.

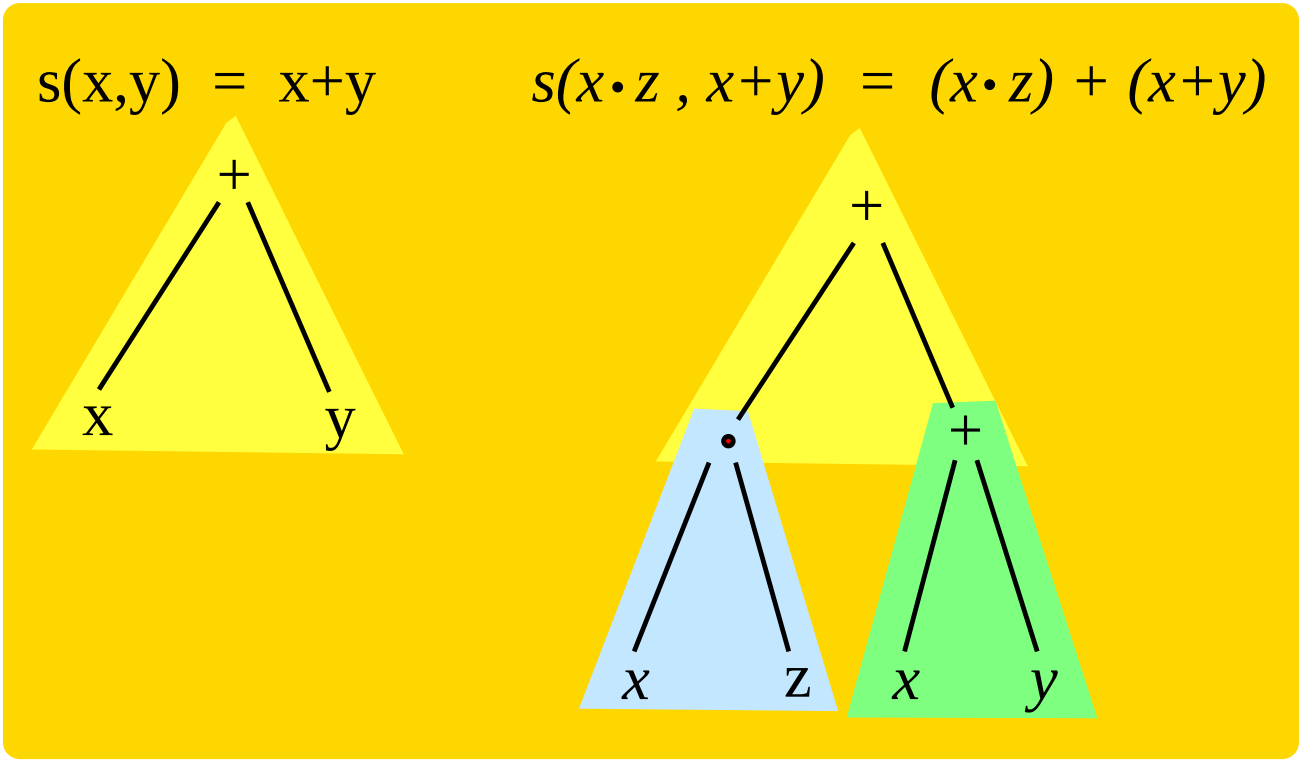
Applying the substitution

$$\begin{pmatrix} x & \leftarrow x \cdot & z \\ y & \leftarrow & x + y \end{pmatrix}$$

to $s(x, y)$ gives $s(x \cdot z, x + y)$ which is

$$(x \cdot z) + (x + y)$$

We can illustrate the substitution procedure with a picture.



As you can see, substitution makes the tree grow downwards from the bottom.

Substitution can only **increase** the tree size.

Substitution Theorems

$$\mathbf{A} \models p(x_1, \ldots, x_n) \approx q(x_1, \ldots, x_n)$$

implies

$$\mathbf{A} \models p(t_1, \ldots, t_n) \approx q(t_1, \ldots, t_n).$$

$$\mathcal{S} \models p(x_1, \ldots, x_n) \approx q(x_1, \ldots, x_n)$$

implies

$$\mathcal{S} \models p(t_1, \ldots, t_n) \approx q(t_1, \ldots, t_n).$$

Replacement

Starting with a term $p(\cdots s \cdots)$ with an occurrence of a subterm s in it,

we **replace** s with another term t and obtain $p(\cdots t \cdots)$.

Example

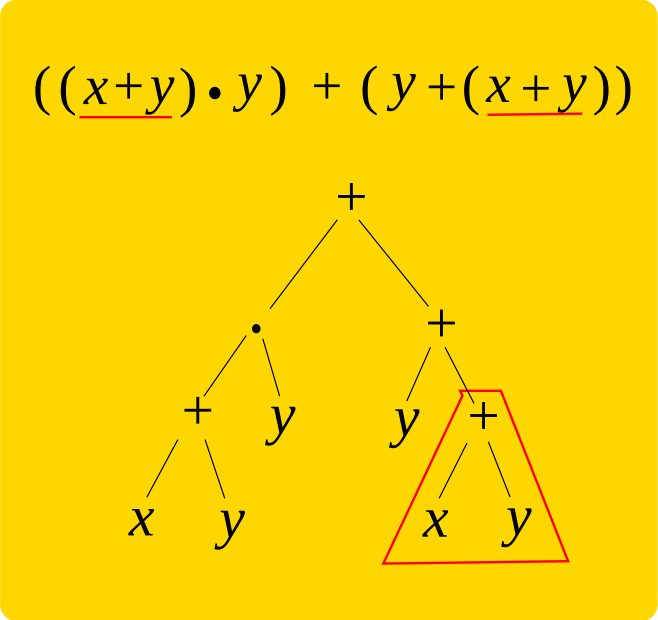
Replacing the second occurrence (from the left) of $x + y$ in

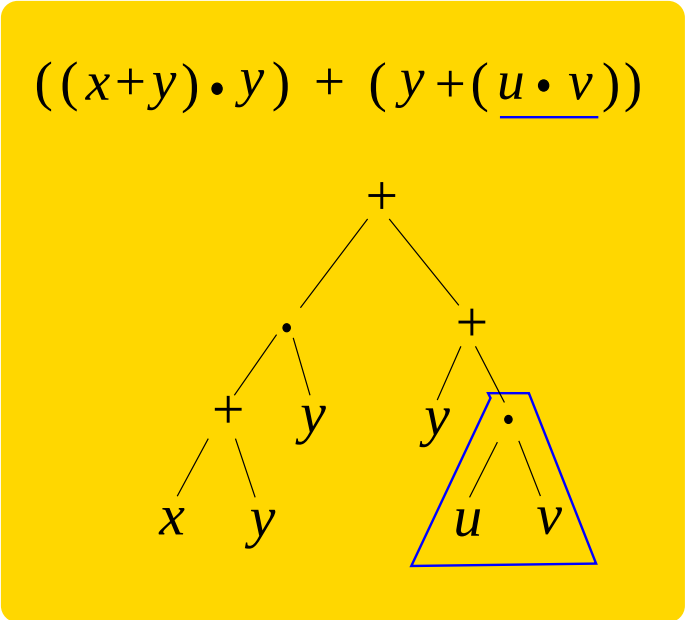
$$((\underbrace{x + y}_{first}) \cdot y) \ + \ (y + (\underbrace{x + y}_{second}))$$

with the term $u \cdot v$ gives the term

$$((x + y) \cdot y) \ + \ (y + (\underline{u \cdot v}))$$

We can visualize this by the use of trees:





Replacement has the effect of replacing one part of the tree.

The replacement part can be larger or smaller than the original.

Replacement can **increase** or **decrease** the size of the tree.

Replacement Theorems

$$\mathbf{A} \models s \approx t$$

implies

$$\mathbf{A} \models p(\cdots s \cdots) \approx p(\cdots t \cdots).$$

$$\mathcal{S} \models s \approx t$$

implies

$$\mathcal{S} \models p(\cdots s \cdots) \approx p(\cdots t \cdots).$$

The Syntactic Viewpoint

Birkhoff's Rules

Rule	Name	Example
$\dfrac{}{s \approx s}$	Reflexive	$\dfrac{}{x + y \approx x + y}$
$\dfrac{s \approx t}{t \approx s}$	Symmetric	$\dfrac{x \approx x \cdot x}{x \cdot x \approx x}$
$\dfrac{r \approx s,\ s \approx t}{r \approx t}$	Transitive	$\dfrac{x \approx x \cdot x,\ x \cdot x \approx 1}{x \approx 1}$
$\dfrac{r(\vec{x}) \approx s(\vec{x})}{r(\vec{t}) \approx s(\vec{t})}$	Substitution	$\dfrac{x \cdot 1 \approx x}{(x \cdot x) \cdot 1 \approx x \cdot x}$
$\dfrac{s \approx t}{r(\cdots s \cdots) \approx r(\cdots t \cdots)}$	Replacement	$\dfrac{x \cdot x \approx x}{(x \cdot x) + x \approx x + x}$

Derivation of Equations

A **derivation** of an equation $s \approx t$ from $\mathcal{S}$ is a sequence

$$s_1 \approx t_1, \ \ldots, \ s_n \approx t_n$$

of equations such that each equation is either

(1) a member of $\mathcal{S}$, or

(2) is the result of applying a rule of inference to previous members of the sequence,

and the last equation is $s \approx t$.

We write $\mathcal{S} \vdash s \approx t$ if such a derivation exists.

Example

A derivation to witness

$$fffx \approx fy \vdash fx \approx fy$$

is given by

1.	$fffx \approx fy$	given
2.	$fffx \approx fx$	subs 1
3.	$fx \approx fffx$	symm 2
4.	$fx \approx fy$	trans 1,3

Example

Show $fgx \approx x, \quad ggx \approx x \vdash fx \approx gx.$

1.	fgx	$\approx$	x	given
2.	ggx	$\approx$	x	given
3.	$fggx$	$\approx$	gx	subs 1
4.	$fggx$	$\approx$	fx	repl 2
5.	fx	$\approx$	$fggx$	symm 4
6.	fx	$\approx$	gx	trans 3,5

Example Show $\mathcal{R} \vdash x \cdot 0 \approx 0$.

1.	$x + 0$	$\approx$	x	given (R1)
2.	$0 + 0$	$\approx$	0	subs 1
3.	$x \cdot (0 + 0)$	$\approx$	$x \cdot 0$	repl 2
4.	$x \cdot (y + z)$	$\approx$	$x \cdot y + x \cdot z$	given (R8)
5.	$x \cdot (0 + 0)$	$\approx$	$x \cdot 0 + x \cdot 0$	subs 4
6.	$x \cdot 0 + x \cdot 0$	$\approx$	$x \cdot (0 + 0)$	symm 5
7.	$x \cdot 0 + x \cdot 0$	$\approx$	$x \cdot 0$	trans 3,6
8.	$(x \cdot 0 + x \cdot 0) + (-(x \cdot 0))$	$\approx$	$x \cdot 0 + (-(x \cdot 0))$	repl 7
9.	$x + (-x)$	$\approx$	0	given (R2)
10.	$x \cdot 0 + (-(x \cdot 0))$	$\approx$	0	subs 9
11.	$(x \cdot 0 + x \cdot 0) + (-(x \cdot 0))$	$\approx$	0	trans 8,10
12.	$x + (y + z)$	$\approx$	$(x + y) + z$	given (R4)
13.	$x \cdot 0 + (x \cdot 0 + (-(x \cdot 0)))$	$\approx$	$(x \cdot 0 + x \cdot 0) + (-(x \cdot 0))$	subs 12
14.	$x \cdot 0 + (x \cdot 0 + (-(x \cdot 0)))$	$\approx$	0	trans 11,13
15.	$x \cdot 0 + (-(x \cdot 0))$	$\approx$	0	subs 9
16.	$x \cdot 0 + (x \cdot 0 + (-(x \cdot 0)))$	$\approx$	$(x \cdot 0) + 0$	repl 15
17.	$x \cdot 0 + 0$	$\approx$	$x \cdot 0$	subs 1
18.	$x \cdot 0 + (x \cdot 0 + (-(x \cdot 0)))$	$\approx$	$x \cdot 0$	trans 16,17
19.	$x \cdot 0$	$\approx$	$x \cdot 0 + (x \cdot 0 + (-(x \cdot 0)))$	symm 18
20.	$x \cdot 0$	$\approx$	0	trans 14,19

Soundness and Completeness

Birkhoff's Rules are

- **sound**: $\mathcal{S} \vdash s \approx t$ implies $\mathcal{S} \models s \approx t$
- **complete**: $\mathcal{S} \models s \approx t$ implies $\mathcal{S} \vdash s \approx t$

Propositional logic has many competing proof systems.

In equational logic we prefer Birkhoff's rules.

Determining Validity

In the propositional logic we have (often very slow) methods to check the validity of an argument.

Equational logic has **no such algorithm**!

This does not mean that we have not yet found an algorithm, but rather that **it simply does not exist**.

Summary of Our Methods for Analyzing Arguments

- An equational argument

$$s_1 \approx t_1,\ \cdots,\ s_n \approx t_n \quad \therefore s \approx t$$

is **valid** iff there is a **derivation** of the conclusion from the premisses, that is, iff

$$s_1 \approx t_1,\ \cdots,\ s_n \approx t_n \ \vdash\ s \approx t.$$

Thus we look for a derivation to show an argument is valid.

- An equational argument

$$s_1 \approx t_1,\ \cdots,\ s_n \approx t_n \quad \therefore s \approx t$$

is **invalid** iff there is a **counterexample** $\mathbf{A}$, that is, iff some $\mathbf{A}$ makes the premisses true and the conclusion false.

Thus we can show an argument is not valid by finding a counterexample.

Unification

One of the most popular and powerful tools of automated theorem proving is an algorithm to find most general unifiers.

Let $s(x_1, \ldots, x_n)$ and $s'(x_1, \ldots, x_n)$ be two terms. A **unifier** of s and s' is a substitution

$$\begin{pmatrix} x_1 \leftarrow t_1 \\ \vdots \\ x_n \leftarrow t_n \end{pmatrix}$$

such that

$$s(t_1, \ldots, t_n) \;=\; s'(t_1, \ldots, t_n).$$

After the substitution has been carried out, the two terms have become the same term.

If a unifier can be found for s and t, we say they **can be unified**, or they **are unifiable**.

Example

Let

$$\begin{aligned} s(x,y) &= (x+y)\cdot x \\ t(x,y) &= (y+x)\cdot y. \end{aligned}$$

There are many unifiers of s and t, e.g.,

$$\begin{pmatrix} x \leftarrow 1 \\ y \leftarrow 1 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} x \leftarrow u\cdot u \\ y \leftarrow u\cdot u \end{pmatrix}.$$

However the substitution

$$\begin{pmatrix} x \leftarrow y \\ y \leftarrow y \end{pmatrix}$$

is a unifier that is **more general** than the preceding two examples.

Most General Unifiers

If two terms s, t have a unifier μ such that every unifier of s, t is an instance of μ then we say μ is a **most general unifier** of s, t.

Theorem (Robinson, 1965)

- If two terms are unifiable then they have a most general unifier.

- The most general unifier of two terms is (essentially) unique.

- There is an algorithm to determine if two terms are unifiable, and if so the algorithm produces the most general unifier.

Critical Subterms of s, t

are the first subterms where s, t disagree.

Example For the terms

$$\begin{aligned} s &= (x \cdot y) + ((x + y) \cdot x) \\ t &= (x \cdot y) + ((y \cdot (x + y)) \cdot y) \end{aligned}$$

the critical subterms are

$$\begin{aligned} s' &= x + y \\ t' &= y \cdot (x + y), \end{aligned}$$

$s =$	+	∘	x	y	∘	+	x	y	x			
$t =$	+	∘	x	y	∘	∘	y	+	x	y	y	

Critical Subterm Condition (CSC)

The **critical subterm condition** (CSC) is satisfied by s and t if their critical subterms s' and t' consist of:

- a variable, say x, and

- another term, say r, that has no occurrence of x in it.

Unification Algorithm:

let μ be the identity substitution

WHILE $s \neq t$

find the critical subterms s', t'

if the CSC fails, return NOT UNIFIABLE

else with $\{s', t'\} = \{x, r\}$

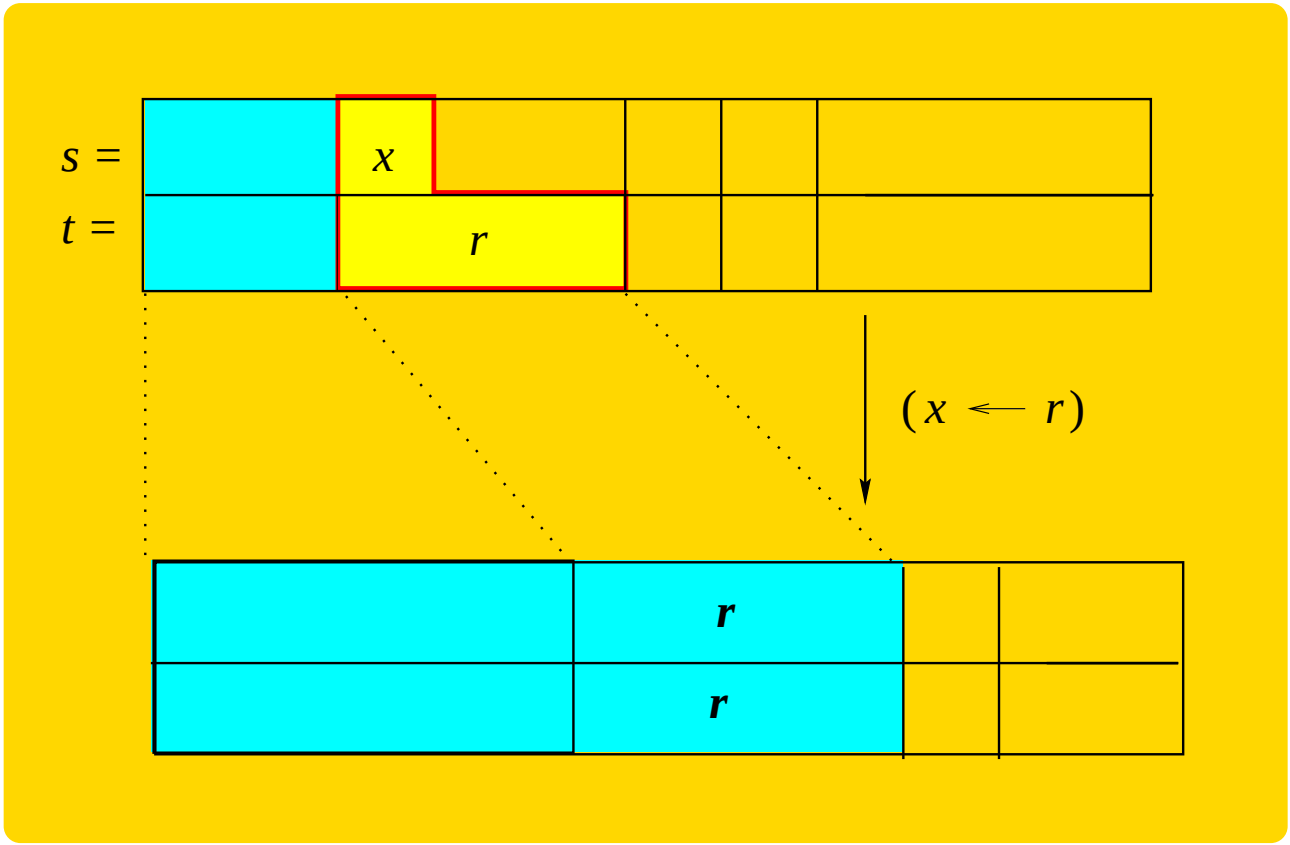
apply $(x \leftarrow r)$ to both s and t

apply $(x \leftarrow r)$ to μ

ENDWHILE

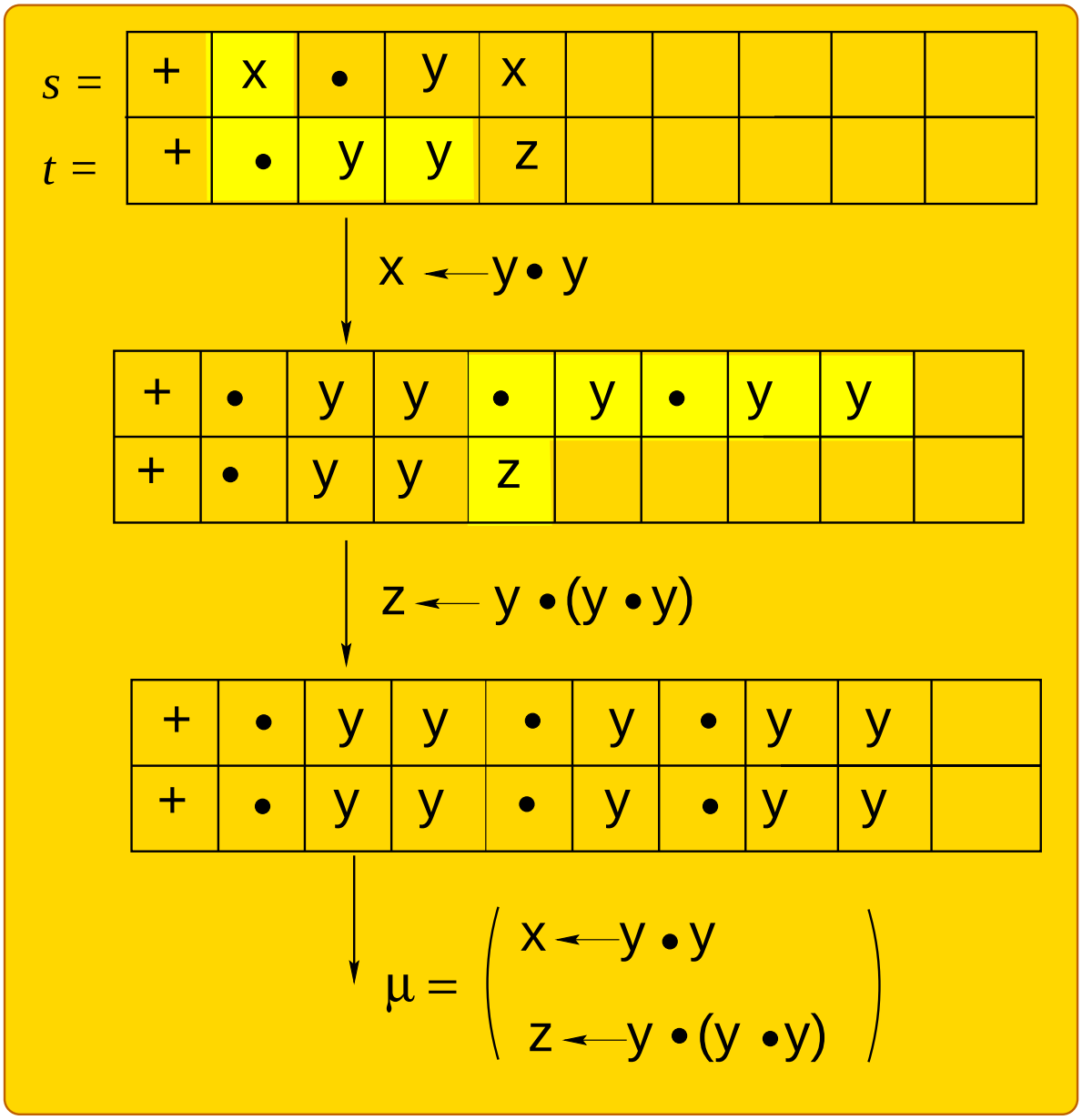
return (μ)

A picture of a possible single step of the unification algorithm when the CSC holds:



In this single step the substitution μ changes to $(x \leftarrow r)\mu$.

Example Apply the unification algorithm to $x + (y \cdot x)$ and $(y \cdot y) + z$:



Composition of Substitutions

Given two substitutions σ, σ', the **composition** of the two is defined by

$$(\sigma'\sigma)(t) \;=\; \sigma'(\sigma(t))$$

Example

$$\sigma \;=\; \begin{pmatrix} x \leftarrow x+y \\ \\ y \leftarrow x \cdot y \end{pmatrix} \quad \text{and} \quad \sigma' \;=\; \begin{pmatrix} x \leftarrow y+x \\ \\ y \leftarrow x+x \end{pmatrix}$$

leads to

$$\sigma'\sigma \;=\; \begin{pmatrix} x \leftarrow (y+x)+(x+x) \\ \\ y \leftarrow (y+x)\cdot(x+x) \end{pmatrix}.$$

Term Rewrite Systems (TRS's)

A **term rewrite (rule)** is an expression $s \longrightarrow t$, sometimes called a **directed equation**, where s and t are terms.

A **term rewrite system**, abbreviated TRS, is a set $\mathcal{R}$ of term rewrite rules.

Example

A simple system $\mathcal{R}$ of **three term rewrite rules used for monoids** is

$$\mathcal{R} = \left\{ \begin{array}{l} (x \cdot y) \cdot z \longrightarrow x \cdot (y \cdot z) \\ 1 \cdot x \longrightarrow x \\ x \cdot 1 \longrightarrow x \end{array} \right\} .$$

Elementary Rewrites

An **elementary rewrite** obtained from $s \longrightarrow t$ is a rewrite of the form

$$r(\cdots \sigma s \cdots) \longrightarrow r(\cdots \sigma t \cdots),$$

where σ is a substitution, r a term, and one occurrence of σs on the left has been replaced by σt on the right.

Example

$$1 \cdot (x \cdot (y \cdot z)) \longrightarrow (x \cdot (y \cdot z))$$

is an elementary rewrite obtained from

$$1 \cdot x \longrightarrow x$$

by using the substitution

$$\Big(x \leftarrow x \cdot (y \cdot z)\Big)$$

If $\mathcal{R}$ is a set of term rewrite rules then

$$s \longrightarrow_{\mathcal{R}} t$$

means $s \longrightarrow t$ is an elementary rewrite of some term rewrite $p \longrightarrow q$ in $\mathcal{R}$.

We write

$$s \longrightarrow_{\mathcal{R}}^{+} t$$

if there is a finite sequence of elementary rewrites such that

$$s = t_1 \longrightarrow_{\mathcal{R}} t_2 \longrightarrow_{\mathcal{R}} \cdots \longrightarrow_{\mathcal{R}} t_n = t.$$

The notation $s \longrightarrow_{\mathcal{R}}^{*} t$ means $s = t$ or $s \longrightarrow_{\mathcal{R}}^{+} t$ holds.

Example

Using the monoid rules $\mathcal{R}$ we have

$$((x \cdot 1) \cdot (1 \cdot y)) \cdot (z \cdot 1) \longrightarrow_{\mathcal{R}}^{+} x \cdot (y \cdot z),$$

since

$$((x \cdot 1) \cdot (1 \cdot y)) \cdot (z \cdot 1) \longrightarrow_{\mathcal{R}} (x \cdot (1 \cdot y)) \cdot (z \cdot 1)$$

$$(x \cdot (1 \cdot y)) \cdot (z \cdot 1) \longrightarrow_{\mathcal{R}} (x \cdot y) \cdot (z \cdot 1)$$

$$(x \cdot y) \cdot (z \cdot 1) \longrightarrow_{\mathcal{R}} (x \cdot y) \cdot z$$

$$(x \cdot y) \cdot z \longrightarrow_{\mathcal{R}} x \cdot (y \cdot z)$$

Terminating TRS's

• A TRS $\mathcal{R}$ is **terminating** if every sequence of elementary rewrites

$$s_1 \longrightarrow_{\mathcal{R}} s_2 \longrightarrow_{\mathcal{R}} \cdots$$

eventually stops.

• A term s is **terminal** if no elementary rewrite $s \longrightarrow_{\mathcal{R}} t$ is possible.

Example

The monoid example $\mathcal{R}$ is terminating.

The terminal terms are either 1 or of the form

$$x \cdot (y \cdot (z \cdots w) \cdots))$$

For example we have the terminating sequence

$$((x \cdot y) \cdot u) \cdot v \longrightarrow_{\mathcal{R}} (x \cdot y) \cdot (u \cdot v) \longrightarrow_{\mathcal{R}} x \cdot (y \cdot (u \cdot v))$$

Two Necessary Conditions for Terminating

- No rule of $\mathcal{R}$ is of the form $x \longrightarrow t$.

For otherwise this could be applied to any term s, so no term would be terminal.

- If $s \longrightarrow t$ is in $\mathcal{R}$ then the variables of t are also variables of s.

For otherwise a substitution into $s \longrightarrow t$ would give an elementary rewrite of the form $s \longrightarrow t'$ that has s as a subterm of t', and this would permit an infinitely long sequence of rewrites.

The Length of a Term

$|t|$ is the **length** of t (the number of symbols in the **prefix** form of t).

$|t|_x$ is the x–**length** of t, the number of times the variable x occurs in t.

Thus for $t = (x \cdot y) + (x \cdot z)$ we have

$$\begin{aligned} |t| &= 7 \\ |t|_x &= 2 \end{aligned}$$

Theorem

Let $\mathcal{R}$ be a term rewrite system with the property that for each $s \longrightarrow t \in \mathcal{R}$ we have

- $|s| > |t|$, and
- $|s|_x \geq |t|_x$ for every variable x.

Then $\mathcal{R}$ is a terminating TRS.

(In this situation all of the elementary rewrites $s \longrightarrow_{\mathcal{R}} t$ are **length-reducing**.)

Applications

The following TRS's are terminating:

- $\mathcal{R} = \{x \cdot x \longrightarrow 0\}$
- $\mathcal{R} = \{x \cdot 1 \longrightarrow x, \quad 1 \cdot x \longrightarrow x\}$
- $\mathcal{R} = \{x \vee x' \longrightarrow x\}$
- $\mathcal{R} = \{x \vee (x \wedge y) \longrightarrow x \wedge y\}$
- $\mathcal{R} = \{x'' \vee x' \longrightarrow x \vee x'\}$
- $\mathcal{R} = \{fggx \longrightarrow gfx, \quad fgx \longrightarrow fx\}$.

Normal Form TRS's

A **normal form** TRS is a **uniquely terminating** TRS.

For such a TRS the terminating form for any given term s is called the **normal form** of s, and written $\mathbf{n_{\mathcal{R}}(s)}$.

A normal form TRS $\mathcal{R}$ is a normal form TRS **for a set $\mathcal{E}$ of equations** if

$$\mathcal{E} \models s \approx t \quad \text{iff} \quad n_{\mathcal{R}}(s) = n_{\mathcal{R}}(t).$$

Example

$\mathcal{R} = \{fffx \longrightarrow x\}$ is a normal form TRS. The normal form of $fffx$ is fx.

Example

$$\mathcal{R} = \left\{ \begin{array}{l} (x \cdot y) \cdot z \longrightarrow x \cdot (y \cdot z) \\ x \cdot 1 \longrightarrow x \\ 1 \cdot x \longrightarrow x \end{array} \right\}$$

is a normal form TRS (for monoids). The normal form of $((x \cdot 1) \cdot (1 \cdot y)) \cdot z$ is $x \cdot (y \cdot z)$.

A Normal Form TRS for Groups

(Knuth and Bendix, 1967)

$\mathcal{E}$	$\mathcal{R}$
$1 \cdot x \approx x$	$1^{-1} \longrightarrow 1$
$x^{-1} \cdot x \approx 1$	$x \cdot 1 \longrightarrow x$
$(x \cdot y) \cdot z \approx x \cdot (y \cdot z)$	$1 \cdot x \longrightarrow x$
	$(x^{-1})^{-1} \longrightarrow x$
	$x^{-1} \cdot x \longrightarrow 1$
	$x \cdot x^{-1} \longrightarrow 1$
	$x^{-1} \cdot (x \cdot y) \longrightarrow y$
	$x \cdot (x^{-1} \cdot y) \longrightarrow y$
	$(x \cdot y)^{-1} \longrightarrow y^{-1} \cdot x^{-1}$
	$(x \cdot y) \cdot z \longrightarrow x \cdot (y \cdot z)$

In this case it is not immediate that the system $\mathcal{R}$ is terminating, much less a normal form TRS.

Converting $\mathcal{R}$ into an equational theory

Let $\mathcal{E}(\mathcal{R}) = \{s \approx t : s \longrightarrow t \in \mathcal{R}\}$.

Proposition

If $\mathcal{R}$ is a normal form TRS, then it is a normal form TRS for $\mathcal{E}(\mathcal{R})$.

Thus, for example, once we show that

$$\mathcal{R} = \{(x \cdot y) \cdot z \longrightarrow x \cdot (y \cdot z)\}$$

is a normal form TRS, then it follows that it is a normal form TRS for semigroups.

Critical Pairs

Given $\begin{array}{lcl} s_1 & \longrightarrow & t_1 \\ s_2 & \longrightarrow & t_2. \end{array}$

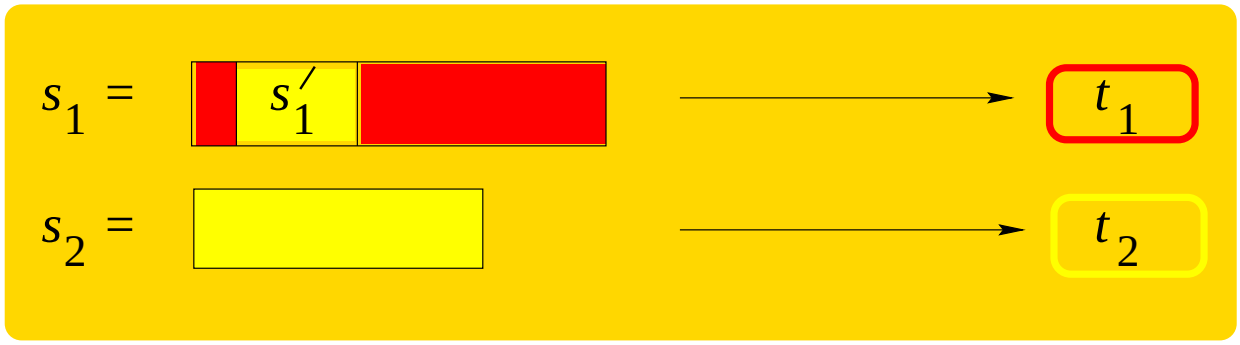
Rename the variables in one of them (if necessary) so that s_1 and s_2 have **no variables in common**.

Choose an occurrence of a **nonvariable subterm** s_1' of s_1 such that s_1', s_2 are unifiable.

Find the most general unifier μ of s_1' and s_2.

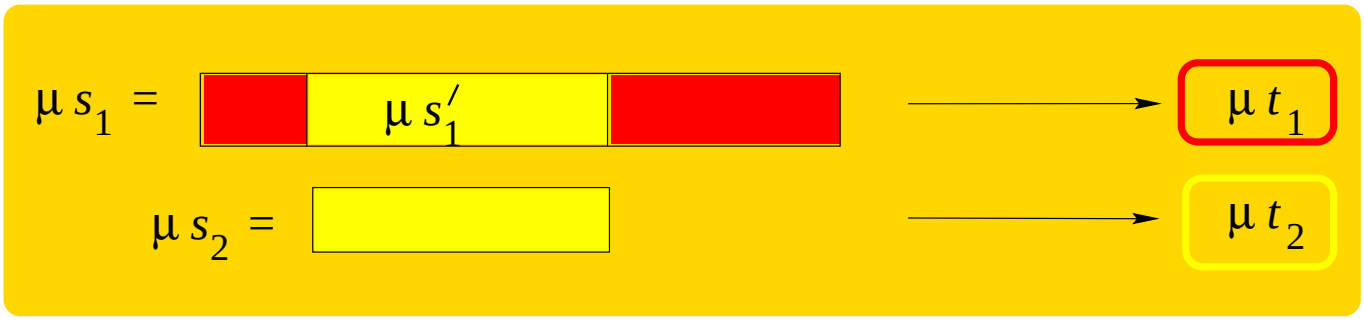
Two rewrites of μs_1 give a critical pair.

Given: Two Rewrite Rules with disjoint variables

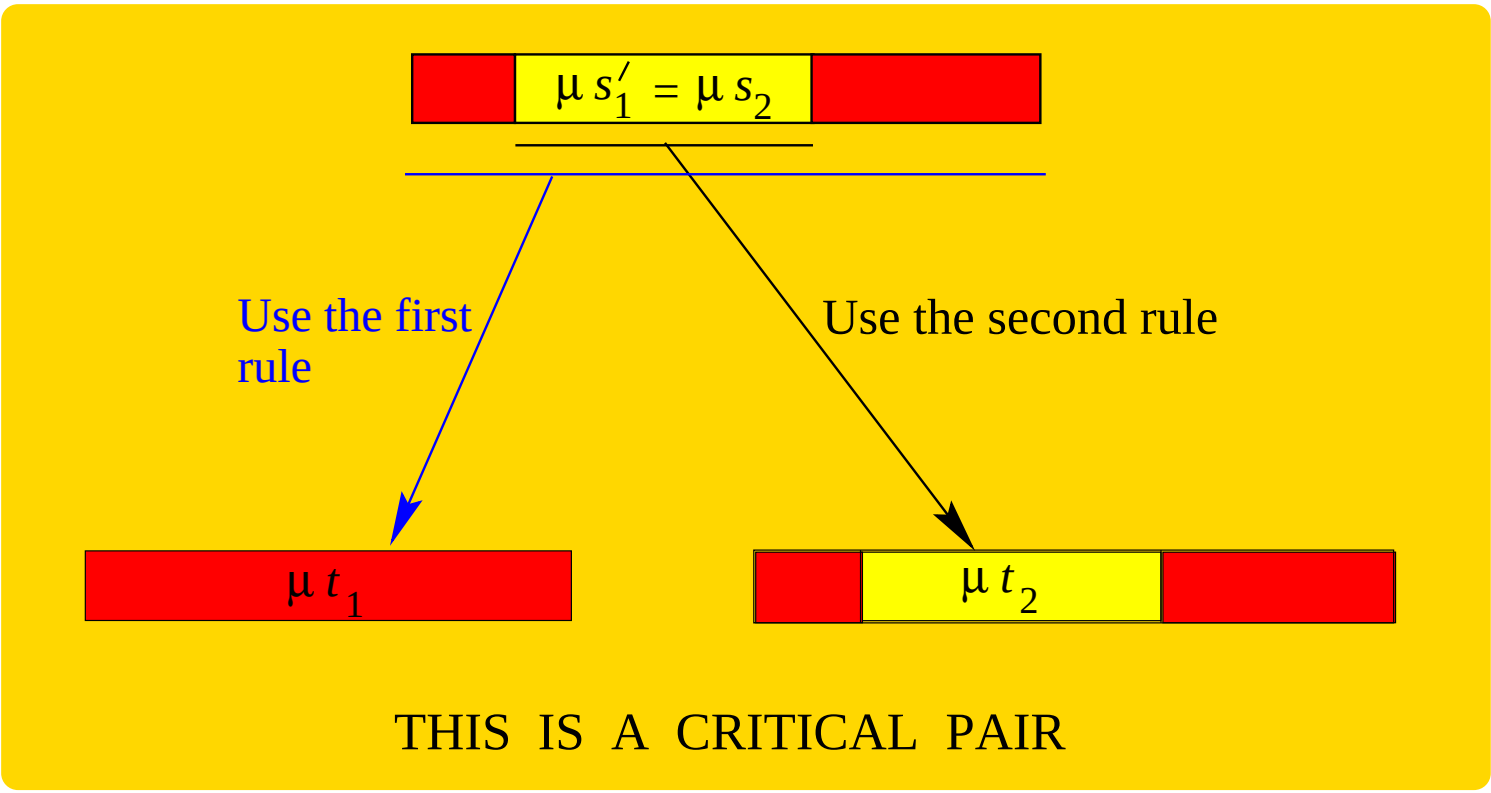


Find μ, the most general unifier of s_1' and s_2 .

Then apply it to both rules.



Creating the Critical Pair



Example: Find the critical pair for

$$\begin{aligned} y + \boxed{(f(x) + y)} &\longrightarrow f(x + y) \\ \boxed{x + g(y)} &\longrightarrow g(x + y) \end{aligned}$$

Changing variables we have

$$\begin{aligned} y + \boxed{(f(x) + y)} &\longrightarrow f(x + y) \\ \boxed{u + g(v)} &\longrightarrow g(u + v) \end{aligned}$$

A most general unifier: $\mu = \begin{pmatrix} u \leftarrow f(x) \\ y \leftarrow g(v) \end{pmatrix}$.

Apply this to the original rewrites:

$$\begin{aligned} g(v) + \boxed{(f(x) + g(v))} &\longrightarrow f(x + g(v)) \\ \boxed{f(x) + g(v)} &\longrightarrow g(f(x) + v). \end{aligned}$$

Thus the critical pair is:

$$(f(x + g(v)) \quad \text{and} \quad g(v) + g(f(x) + v)).$$

A TRS $\mathcal{R}$ is **confluent on critical pairs** if, given any critical pair (s, t), there is a term r such that

$$s \longrightarrow_{\mathcal{R}}^{*} r$$

$$t \longrightarrow_{\mathcal{R}}^{*} r.$$

Critical Pairs Lemma

(Knuth and Bendix)

Suppose $\mathcal{R}$ is a terminating TRS.

Then $\mathcal{R}$ is a normal form TRS iff

$\mathcal{R}$ is confluent on critical pairs.

Corollary

Let $\mathcal{R}$ be a term rewrite system with the property that

for each $s \longrightarrow t \in \mathcal{R}$ we have

- $|s| > |t|$,
- $|s|_x \geq |t|_x$ for every variable x, and
- $\mathcal{R}$ is confluent on critical pairs.

Then $\mathcal{R}$ is a normal form TRS.

Example

Consider $\mathcal{R} = \{fgfx \longrightarrow gx\}$, a terminating TRS.

$$\begin{array}{rcl} fg\boxed{fx} & \longrightarrow & gx \\ \boxed{fgfu} & \longrightarrow & gu \end{array} \quad \text{has} \quad \mu = (x \leftarrow gfu).$$

Applying μ:
$$\begin{array}{rcl} fg\boxed{fgfu} & \longrightarrow & ggfu \\ \boxed{fgfu} & \longrightarrow & gu \end{array}$$

The critical pair is: $(ggfu, \quad fggu)$.

We **do not have confluence for this critical pair**, so the system $\mathcal{R}$ is not a normal form TRS.