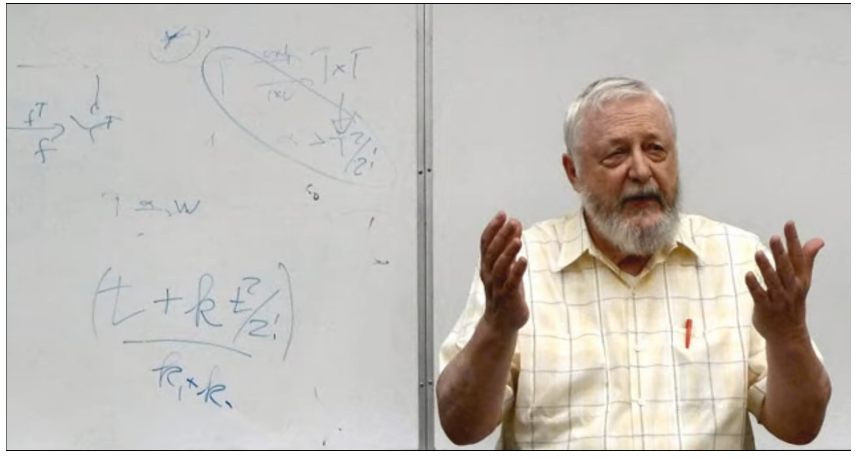
Semantica funtoriale delle teorie algebriche



(William Lawvere • 1937 — 2023)

Fosco Loregian

13 dicembre 2025

Gruppi

Un 'gruppo_1' è un insieme non vuoto G, dotato di operazioni

$$m : G \times G \to G \qquad i : G \to G \qquad e : G$$

che soddisfano i seguenti assiomi:

(notazione: $m(x, y) = x.y$ è la *moltiplicazione* e $i(x) = \bar{x}$ l'*inverso*)

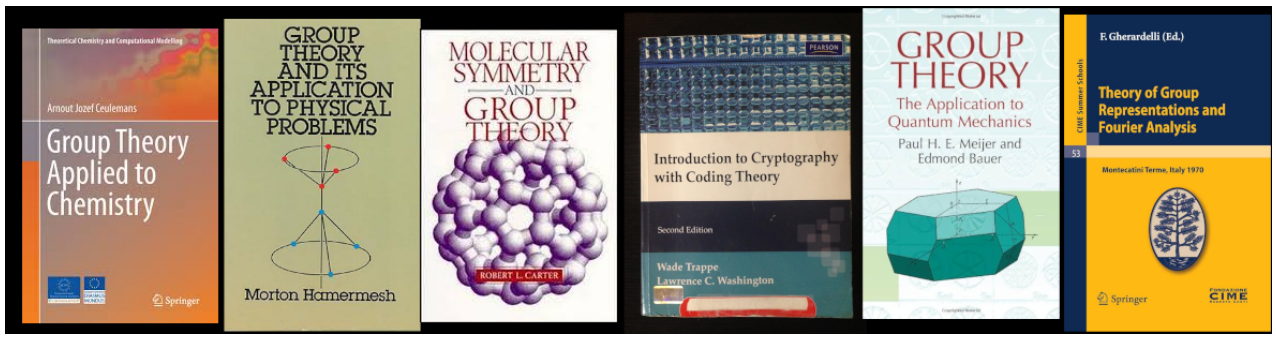
- m è associativa: $x.(y.z) = (x.y).z$ per ogni $x, y, z \in G$;
- m è unitale: $x.e = x = e.x$ per ogni $x \in G$;
- ogni elemento $x \in G$ ha un inverso rispetto a m: $x.\bar{x} = e = \bar{x}.x$.

Un gruppo_1 è *commutativo* se $x.y = y.x$ per ogni $x, y \in G$.

Gruppi

Vi sono diverse ragioni per studiare la teoria dei gruppi: mediante essi si può

- definire e classificare le simmetrie nei sistemi fisici (particelle elementari) e chimici (gruppi cristallografici);
- capire quali equazioni sono risolvibili in radicali (perché solo fino al grado 4?) (thm di Abel-Ruffini + teoria di Galois);
- parlare di vari protocolli crittografici (Diffie–Hellman, RSA, etc.);
- studiare l'elaborazione di immagini e la teoria dei codici;
- ...



Gruppi

Un 'gruppo_2' è un insieme non vuoto G, con una operazione binaria $-/-$ detta '*taglio*', $(x, y) \mapsto x/y$ soddisfacente l'assioma

$$\forall x, y, z \,:\, x/\Big(\Big(((x/x)/y)/z\Big)/\Big(((x/x)/x)/z\Big)\Big) = y.$$

Un 'gruppo_2' è *commutativo* se

$$\forall x, y, z \,:\, x/\big((y/z)/(y/x)\big) = z$$

Teorema

Le nozioni di gruppo$_1$ e di gruppo$_2$ sono equivalenti.

- Dato un gruppo$_1$, si ottiene un gruppo$_2$ definendo

$$x/y := m(x, i(y)) = x.\bar{y}$$

- Dato un gruppo$_2$, si ottiene un gruppo$_1$ definendo

$$i(x) := ??? \qquad m(x, y) := ??? \qquad e := ???$$

Per mostrare la prima implicazione, va visto che con la definizione data di taglio, vale l'equazione

$$x.\overline{(\bar{y}.(\bar{z}.\overline{(\bar{x}.\bar{z})}))} = y$$

per ogni $x, y, z \in G$.
Per vederlo: (si ricordi che $\overline{\bar{a}.\bar{b}} = b.a$, e $\bar{\bar{x}} = x$)

$$\begin{aligned}
x.\overline{(\bar{y}.(\bar{z}.\overline{(\bar{x}.\bar{z})}))} &= x.\overline{(\bar{y}.(\bar{z}.(z.x)))} \\
&= x.\overline{(\bar{y}.((\bar{z}.z).x))} \\
&= x.\overline{(\bar{y}.(e.x))} \\
&= x.\overline{(\bar{y}.x)} \\
&= x.(\bar{x}.y) \\
&= (x.\bar{x}).y \\
&= e.y = y
\end{aligned}$$

Definire le operazioni di gruppo$_1$ in termini del solo taglio è più complesso:

Lemma

Sia G un gruppo$_2$. Per ogni $a, b \in G$, $a/a = b/b$.

Anzitutto, si dimostra che

$$((y/z)/(((x/x)/x)/z)) = ((y/w)/(((x/x)/x)/w))$$

per ogni $z, w \in G$, cosicché l'operazione $K_{x,y}(z)$ definita da $((y/z)/(((x/x)/x)/z))$ è costante in z. Poi si osserva che l'assioma di taglio implica che ogni elemento $a \in G$ si scrive nella forma $((x/x)/x)/z$ per qualche z; allora, dati $a, b \in G$ si ha

$$\exists z : a = ((x/x)/x)/z \qquad \exists z' : b = ((x/x)/x)/z'$$

e usando ripetutamente l'assioma di taglio, si mostra che

$$a/a = \Big(((x/x)/x)/z\Big)/\Big(((x/x)/x)/z\Big) = \Big(((x/x)/x)/z'\Big)/\Big(((x/x)/x)/z'\Big) = b/b.$$

$\text{gruppo}_2 \rightarrow \text{gruppo}_1$

- Il valore comune $e = x/x$ per un qualsiasi $x \in G$ in un gruppo_2 è l'identità rispetto all'operazione

$$x.y := x/(e/y) = x/((x/x)/y) = x/((y/y)/y)$$

- L'inverso rispetto a . è definito da $\bar{x} = e/x = (x/x)/x$.

Quindi dato un gruppo_2, si ottiene un gruppo_1 definendo

$$i(x) := (x/x)/x \qquad m(x,y) := x/((x/x)/y) \qquad e := x/x$$

$\text{gruppo}_2 \to \text{gruppo}_1$

Vanno ora mostrati i fatti seguenti:

- (assoc)

$$\forall xyz : (x/((x/x)/y))/(((x/((x/x)/y))/(x/((x/x)/y)))/z) = x/((x/x)/(y/((y/y)/z)))$$

- (unitL)

$$\forall x : (x/x)/(((x/x)/(x/x))/x) = x$$

- (unitR)

$$\forall x : x/((x/x)/(x/x)) = x$$

- (invL)

$$\forall x : ((x/x)/x)/((((x/x)/x)/((x/x)/x))/x) = x/x$$

- (invR)

$$\forall x : x/((x/x)/((x/x)/x)) = x/x$$

Le dimostrazioni (in MaCE4/PROVER):

```
1  ============================== PROOF =================================
2
3  % Proof 1 at 0.00 (+ 0.00) seconds.
4  % Length of proof is 15.
5  % Level of proof is 3.
6  % Maximum clause weight is 17.000.
7  % Given clauses 12.
8
9  1 x / ((((x / x) / y) / z) / (((x / x) / x) / z)) = y # label(non_clause) # label(goal).  [goal].
10 2 (x @ y) @ z = x @ (y @ z).  [assumption].
11 3 x @ e = x.  [assumption].
12 4 e @ x = x.  [assumption].
13 5 x @ x' = e.  [assumption].
14 6 x' @ x = e.  [assumption].
15 7 x / y = x @ y'.  [assumption].
16 8 c1 / ((((c1 / c1) / c2) / c3) / (((c1 / c1) / c1) / c3)) != c2.  [deny(1)].
17 9 c1 @ (c2' @ (c3' @ (c1' @ c3')'))' != c2.  [copy(8),rewrite([7(4),5(5),7(4),4(5),7(5),7(9),5(10),7(9),4(10),7(10),7(12),2(13),7(14)])].
18 10 x @ (x' @ y) = y.  [para(5(a,1),2(a,1,1)),rewrite([4(2)]),flip(a)].
19 11 x @ (y @ (x @ y)') = e.  [para(5(a,1),2(a,1)),flip(a)].
20 13 x' @ (x @ y) = y.  [para(6(a,1),2(a,1,1)),rewrite([4(2)]),flip(a)].
21 15 x'' = x.  [para(5(a,1),10(a,1,2)),rewrite([3(2)]),flip(a)].
22 19 x @ (y @ x)' = y'.  [para(11(a,1),13(a,1,2)),rewrite([3(3)]),flip(a)].
23 20 $F.  [back_rewrite(9),rewrite([19(12),15(6),19(7),15(3)]),xx(a)].
24
25 ============================== end of proof ==========================
26
```

```
1  ============================== PROOF =================================
2
3  % Proof 2 at 0.01 (+ 0.00) seconds.
4  % Length of proof is 17.
5  % Level of proof is 8.
6  % Maximum clause weight is 51.000.
7  % Given clauses 11.
8
9  2 (x / x) @ x = x # label(non_clause) # label(goal).  [goal].
10 3 x / ((((x / x) / y) / z) / (((x / x) / x) / z)) = y.  [assumption].
11 4 x @ y = x / ((x / x) / y).  [assumption].
12 8 (c2 / c2) @ c2 != c2.  [deny(2)].
13 9 (c2 / c2) / (((c2 / c2) / (c2 / c2)) / c2) != c2.  [copy(8),rewrite([4(5)])].
14 10 ((((x / x) / (x / x)) / y) / z) / ((((x / x) / (x / x)) / (x / x)) / z) = x / ((y / u) / (((x / x) / x) / u)).  [para(3(a,1),3(a,1,2,1,1)),flip(a)].
15 11 x / (y / (((x / x) / x) / ((((((x / x) / z) / ((x / x) / z)) / y) / u) / (((((x / x) / z) / ((x / x) / z)) / ((x / x) / z)) / u)))) = z.  [para(3(a,1),3(a,1,2,1))].
16 17 ((x / x) / (x / x)) / ((((((x / x) / (x / x)) / ((x / x) / (x / x))) / y) / ((((x / x) / (x / x)) / (x / x)) / ((x / x) / (x / x)))) / (x / x)) = y.  [para(10(a,1),3(a,1,2,2)),rewrite([3(30)])].
17 18 (x / x) / (x / ((y / z) / (((x / x) / x) / z))) = y.  [para(10(a,1),3(a,1,2))].
18 22 ((((x / x) / (x / x)) / ((x / x) / y)) / z) / ((((x / x) / (x / x)) / (x / x)) / z) = y.  [para(10(a,2),3(a,1))].
19 38 (x / x) / (x / (y / (((x / x) / x) / ((((z / z) / y) / u) / (((z / z) / z) / u))))) = z.  [para(3(a,1),18(a,1,2,2,1))].
20 78 (x / x) / (x / (y / y)) = (x / x) / x.  [para(3(a,1),38(a,1,2,2,2))].
21 112 ((x / x) / (x / x)) / ((((((x / x) / (x / x)) / (x / x)) / y) / ((((x / x) / (x / x)) / (x / x)) / ((x / x) / (x / x)))) / (x / x)) = y.  [back_rewrite(17),rewrite([78(10)])].
22 114 x / (y / y) = x.  [para(3(a,1),11(a,1,2,2))].
23 152 (x / x) / ((x / x) / y) = y.  [back_rewrite(112),rewrite([114(3),114(4),114(4),114(6),114(6),114(7),114(6),114(5),114(5)])].
24 201 (x / y) / ((z / z) / y) = x.  [back_rewrite(22),rewrite([114(3),152(4),114(4),114(4)])].
25 202 $F.  [resolve(201,a,9,a)].
26
27 ============================== end of proof ==========================
```

```
═══════════════════════════════ PROOF ═══════════════════════════════

% Proof 1 at 0.01 (+ 0.00) seconds.
% Length of proof is 18.
% Level of proof is 8.
% Maximum clause weight is 51.000.
% Given clauses 11.

2 x' @ x = x / x # label(non_clause) # label(goal).  [goal].
3 x / ((((x / x) / y) / z) / (((x / x) / x) / z)) = y.  [assumption].
4 x @ y = x / ((x / x) / y).  [assumption].
5 x' = (x / x) / x.  [assumption].
8 c2 / c2 ≠ c2' @ c2.  [deny(2)].
9 ((c2 / c2) / c2) / ((((c2 / c2) / c2) / ((c2 / c2) / c2)) / c2) ≠ c2 / c2.  [copy(8),rewrite([5(5),4(10)]),flip(a)].
10 ((((x / x) / (x / x)) / y) / z) / ((((x / x) / (x / x)) / (x / x)) / z) = x / ((y / u) / (((x / x) / x) / u)).  [para(3(a,1),3(a,1,2,1,1)),flip(a)].
11 x / (y / (((x / x) / x) / ((((((x / x) / z) / ((x / x) / z)) / y) / u) / (((((x / x) / z) / ((x / x) / z)) / ((x / x) / z)) / u)))) = z.  [para(3(a,1),3(a,1,2,1))].
17 ((x / x) / (x / x)) / ((((((x / x) / (x / x)) / ((x / x) / (x / x))) / y) / ((((x / x) / (x / x)) / (x / x)) / ((x / x) / (x / x)))) / (x / x)) = y.  [para(10(a,1),3(a,1,2,2)),rewrite([3(30)])].
18 (x / x) / (x / ((y / z) / (((x / x) / x) / z))) = y.  [para(10(a,1),3(a,1,2))].
22 ((((x / x) / (x / x)) / ((x / x) / y)) / z) / ((((x / x) / (x / x)) / (x / x)) / z) = y.  [para(10(a,2),3(a,1))].
38 (x / x) / (x / (y / (((x / x) / x) / ((((z / z) / y) / u) / (((z / z) / z) / u))))) = z.  [para(3(a,1),18(a,1,2,2,1))].
78 (x / x) / (x / (y / y)) = (x / x) / x.  [para(3(a,1),38(a,1,2,2,2))].
112 ((x / x) / (x / x)) / ((((((x / x) / (x / x)) / (x / x)) / y) / ((((x / x) / (x / x)) / (x / x)) / ((x / x) / (x / x)))) / (x / x)) = y.  [back_rewrite(17),rewrite([78(10)])].
114 x / (y / y) = x.  [para(3(a,1),11(a,1,2,2))].
151 (x / x) / ((x / x) / y) = y.  [back_rewrite(112),rewrite([114(3),114(4),114(4),114(6),114(6),114(7),114(6),114(5),114(5)])].
200 (x / y) / ((z / z) / y) = x.  [back_rewrite(22),rewrite([114(3),151(4),114(4),114(4)])].
201 $F.  [resolve(200,a,9,a)].

═══════════════════════════ end of proof ═══════════════════════════

═══════════════════════════════ PROOF ═══════════════════════════════

% Proof 2 at 0.02 (+ 0.00) seconds.
% Length of proof is 16.
% Level of proof is 7.
% Maximum clause weight is 51.000.
% Given clauses 11.

1 x @ x' = x / x # label(non_clause) # label(goal).  [goal].
3 x / ((((x / x) / y) / z) / (((x / x) / x) / z)) = y.  [assumption].
4 x @ y = x / ((x / x) / y).  [assumption].
5 x' = (x / x) / x.  [assumption].
6 c1 / c1 ≠ c1 @ c1'.  [deny(1)].
7 c1 / ((c1 / c1) / ((c1 / c1) / c1)) ≠ c1 / c1.  [copy(6),rewrite([5(6),4(10)]),flip(a)].
10 ((((x / x) / (x / x)) / y) / z) / ((((x / x) / (x / x)) / (x / x)) / z) = x / ((y / u) / (((x / x) / x) / u)).  [para(3(a,1),3(a,1,2,1,1)),flip(a)].
11 x / (y / (((x / x) / x) / ((((((x / x) / z) / ((x / x) / z)) / y) / u) / (((((x / x) / z) / ((x / x) / z)) / ((x / x) / z)) / u)))) = z.  [para(3(a,1),3(a,1,2,1))].
17 ((x / x) / (x / x)) / ((((((x / x) / (x / x)) / ((x / x) / (x / x))) / y) / ((((x / x) / (x / x)) / (x / x)) / ((x / x) / (x / x)))) / (x / x)) = y.  [para(10(a,1),3(a,1,2,2)),rewrite([3(30)])].
18 (x / x) / (x / ((y / z) / (((x / x) / x) / z))) = y.  [para(10(a,1),3(a,1,2))].
38 (x / x) / (x / (y / (((x / x) / x) / ((((z / z) / y) / u) / (((z / z) / z) / u))))) = z.  [para(3(a,1),18(a,1,2,2,1))].
78 (x / x) / (x / (y / y)) = (x / x) / x.  [para(3(a,1),38(a,1,2,2,2))].
112 ((x / x) / (x / x)) / ((((((x / x) / (x / x)) / (x / x)) / y) / ((((x / x) / (x / x)) / (x / x)) / ((x / x) / (x / x)))) / (x / x)) = y.  [back_rewrite(17),rewrite([78(10)])].
114 x / (y / y) = x.  [para(3(a,1),11(a,1,2,2))].
151 (x / x) / ((x / x) / y) = y.  [back_rewrite(112),rewrite([114(3),114(4),114(4),114(6),114(6),114(7),114(6),114(5),114(5)])].
209 $F.  [back_rewrite(7),rewrite([151(10)]),xx(a)].

═══════════════════════════ end of proof ═══════════════════════════
```

```
1  ========================== PROOF =========================
2
3  % Proof 1 at 0.03 (+ 0.00) seconds.
4  % Length of proof is 40.
5  % Level of proof is 17.
6  % Maximum clause weight is 63.000.
7  % Given clauses 39.
8
9  1 x @ (y @ z) = (x @ y) @ z # label(non_clause) # label(goal).  [goal].
10 2 x / ((((x / x) / y) / z) / (((x / x) / x) / z)) = y.  [assumption].
11 3 x @ y = x / ((x / x) / y).  [assumption].
12 4 x' = (x / x) / x.  [assumption].
13 5 (c1 @ c2) @ c3 != c1 @ (c2 @ c3).  [deny(1)].
14 6 (c1 / ((c1 / c1) / c2)) / (((c1 / ((c1 / c1) / c2)) / (c1 / ((c1 / c1) / c2))) / c3) != c1 / ((c1 / c1) / (c2 / ((c2 / c2) / c3))).  [copy(5),rewrite([3(3),3(9),3(29),3(34)])].
15 7 ((((x / x) / (x / x)) / y) / z) / ((((x / x) / (x / x)) / (x / x)) / z) = x / ((y / u) / (((x / x) / x) / u)).  [para(2(a,1),2(a,1,2,1,1)),flip(a)].
16 8 x / (y / (((x / x) / x) / (((((x / x) / z) / ((x / x) / z)) / y) / u) / ((((x / x) / z) / ((x / x) / z)) / ((x / x) / z)) / ((x / x) / z)) / u)))) = z.  [para(2(a,1),2(a,1,2,1))].
17 14 ((x / x) / (x / x)) / ((((((x / x) / (x / x)) / ((x / x) / (x / x))) / y) / ((((x / x) / (x / x)) / (x / x)) / ((x / x) / (x / x)))) / (x / x)) = y.  [para(7(a,1),2(a,1,2,2)),rewrite([2(30)])].
18 15 (x / x) / (x / ((y / z) / (((x / x) / x) / z))) = y.  [para(7(a,1),2(a,1,2))].
19 19 ((((x / x) / (x / x)) / ((x / x) / y)) / z) / ((((x / x) / (x / x)) / (x / x)) / z) = y.  [para(7(a,2),2(a,1))].
20 28 (x / x) / (((((x / x) / (x / x)) / ((x / x) / (x / x))) / ((x / x) / (x / x))) / ((((x / x) / (x / x)) / (x / x)) / y)) = (x / x) / ((y / z) / ((((x / x) / (x / x)) / (x / x)) / z)).  [para(7(a,1),7(a,1,1)),rewrite([2(9)])].
21 35 (x / x) / (x / (y / (((x / x) / x) / ((((z / z) / y) / u) / (((z / z) / z) / u))))) = z.  [para(2(a,1),15(a,1,2,2,1))].
22 38 x / ((((y / y) / (y / y)) / (y / y)) / (((y / y) / (y / y)) / ((x / z) / (((((y / y) / (y / y)) / ((y / y) / (y / y))) / ((y / y) / (y / y))) / ((y / y) / (y / y)))) / ((y / y) / (y / y)))) = y / y.  [para(15(a,1),7(a,1,1)),rewrite([2(34)])].
23 69 (((x / x) / (x / x)) / ((x / x) / y)) / z = (u / u) / (u / (y / (((u / u) / u) / ((((x / x) / (x / x)) / (x / x)) / z)))).  [para(19(a,1),15(a,1,2,2,1)),flip(a)].
24 75 (x / x) / (x / (y / y)) = (x / x) / x.  [para(2(a,1),35(a,1,2,2,2))].
25 104 x / ((((y / y) / (y / y)) / (y / y)) / (((y / y) / (y / y)) / ((x / z) / (((((y / y) / (y / y)) / (y / y)) / ((y / y) / (y / y))) / z)))) = y / y.  [back_rewrite(38),rewrite([75(16)])].
26 106 (x / x) / (((((x / x) / (x / x)) / (x / x)) / ((x / x) / (x / x))) / ((((x / x) / (x / x)) / (x / x)) / y)) = (x / x) / (((y / z) / ((((x / x) / (x / x)) / (x / x)) / z)).  [back_rewrite(28),rewrite([75(8)])].
27 109 ((x / x) / (x / x)) / ((((((x / x) / (x / x)) / (x / x)) / y) / ((((x / x) / (x / x)) / (x / x)) / ((x / x) / (x / x)))) / (x / x)) = y.  [back_rewrite(14),rewrite([75(10)])].
28 111 x / (y / y) = x.  [para(2(a,1),8(a,1,2,2))].
29 137 (x / y) / ((z / z) / y) = (z / z) / (((((z / z) / x) / ((z / z) / x)) / u) / (((((z / z) / x) / ((z / z) / x)) / ((z / z) / x)) / u)).  [para(8(a,1),19(a,1,1,1)),rewrite([111(4),111(4),111(7),111(8),111(7),111(8),111(7),111(8),111(9),111(8),111(10),111(11),111(10),111(12),111(14),111(15),111(14),111(16),111(17),111(16),111(19),111(20),111(19)])].
30 148 (x / x) / ((x / x) / y) = y.  [back_rewrite(109),rewrite([111(3),111(4),111(4),111(6),111(6),111(7),111(6),111(5),111(5)])].
31 151 (x / x) / ((y / z) / ((x / x) / z)) = (x / x) / y.  [back_rewrite(106),rewrite([111(4),111(4),111(5),111(4),111(5),111(5),148(5),111(7),111(7)]),flip(a)].
32 152 x / x = y / y.  [back_rewrite(104),rewrite([111(3),111(3),111(4),111(6),111(6),111(7),111(6),151(7),148(4)])].
33 172 (x / x) / (x / (y / (((x / x) / x) / ((z / z) / u)))) = y / u.  [back_rewrite(69),rewrite([111(3),148(4),111(7),111(7)]),flip(a)].
34 197 (x / y) / ((z / z) / y) = x.  [back_rewrite(19),rewrite([111(3),148(4),111(4),111(4)])].
35 223 (x / x) / (((x / x) / y) / (z / y)) = z.  [back_rewrite(137),rewrite([197(4),197(6),197(8),148(7)]),flip(a)].
36 263 (x / x) / y = (y / y) / y.  [para(152(a,1),4(a,2,1)),rewrite([4(1)]),flip(a)].
37 264 (c1 / ((x / x) / c2)) / ((c3 / c3) / c3) != c1 / ((c1 / c1) / (c2 / ((c2 / c2) / c3))).  [para(152(a,1),6(a,1,1,2,1)),rewrite([263(22)])].
38 270 (x / x) / ((y / y) / z) = z.  [para(152(a,1),148(a,1,1))].
39 282 (c1 / ((x / x) / c2)) / ((y / y) / c3) != c1 / ((c1 / c1) / (c2 / ((c2 / c2) / c3))).  [para(152(a,1),264(a,1,2,1))].
40 285 (c1 / ((x / x) / c2)) / ((y / y) / c3) != c1 / ((z / z) / (c2 / ((c2 / c2) / c3))).  [para(152(a,1),282(a,2,2,1))].
41 287 (c1 / ((x / x) / c2)) / ((y / y) / c3) != c1 / ((z / z) / (c2 / ((u / u) / c3))).  [para(152(a,1),285(a,2,2,2,1))].
42 291 (x / (((y / y) / z) / (u / z))) / u = x.  [para(223(a,1),197(a,1,2))].
43 293 (x / x) / (y / z) = z / y.  [para(197(a,1),223(a,1,2,2)),rewrite([270(5)])].
44 325 (c1 / ((x / x) / c2)) / ((y / y) / c3) != c1 / (((z / z) / c3) / c2).  [para(293(a,1),287(a,2,2))].
45 368 (x / x) / (x / (y / (((x / x) / x) / (z / u)))) = y / (u / z).  [para(293(a,1),172(a,1,2,2,2,2))].
46 375 (x / x) / (((y / y) / c3) / (c1 / ((z / z) / c2))) != c1 / (((u / u) / c3) / c2).  [para(293(a,2),325(a,1))].
47 404 (x / x) / (((y / y) / c3) / (c1 / (((z / z) / u) / (c2 / u)))) != c1 / (((w / w) / c3) / c2).  [para(291(a,1),375(a,1,2,2,2))].
48 405 $F.  [resolve(404,a,368,a)].
49
50 ======================== end of proof ====================
51
```

```
1  ========================== PROOF =========================
2
3  % Proof 1 at 0.01 (+ 0.00) seconds.
4  % Length of proof is 19.
5  % Level of proof is 9.
6  % Maximum clause weight is 63.000.
7  % Given clauses 8.
8
9  1 x / x = y / y # label(non_clause) # label(goal).  [goal].
10 2 x / ((((x / x) / y) / z) / (((x / x) / x) / z)) = y.  [assumption].
11 3 c2 / c2 ≠ c1 / c1.  [deny(1)].
12 4 ((((x / x) / (x / x)) / y) / z) / ((((x / x) / (x / x)) / (x / x)) / z) = x / ((y / u) / (((x / x) / x) / u)).  [para(2(a,1),2(a,1,2,1,1)),flip(a)].
13 5 x / (y / (((x / x) / x) / ((((((x / x) / z) / ((x / x) / z)) / y) / u) / (((((x / x) / z) / ((x / x) / z)) / ((x / x) / z)) / u)))) = z.  [para(2(a,1),2(a,1,2,1))].
14 11 ((x / x) / (x / x)) / ((((((x / x) / (x / x)) / ((x / x) / (x / x))) / y) / ((((x / x) / (x / x)) / (x / x)) / ((x / x) / (x / x)))) / (x / x)) = y.  [para(4(a,1),2(a,1,2,2)),rewrite([2(30)])].
15 12 (x / x) / (x / ((y / z) / (((x / x) / x) / z))) = y.  [para(4(a,1),2(a,1,2))].
16 21 (x / x) / (((((x / x) / (x / x)) / ((x / x) / (x / x))) / ((x / x) / (x / x))) / ((((x / x) / (x / x)) / (x / x)) / y)) = (x / x) / ((y / z) / ((((x / x) / (x / x)) / (x / x)) / z)).  [para(4(a,1),4(a,1,1)),rewrite([2(9)])].
17 28 (x / x) / (x / (y / (((x / x) / x) / ((((z / z) / y) / u) / (((z / z) / z) / u))))) = z.  [para(2(a,1),12(a,1,2,2,1))].
18 31 x / ((((y / y) / (y / y)) / (y / y)) / (((y / y) / (y / y)) / ((x / z) / (((((y / y) / (y / y)) / ((y / y) / (y / y))) / ((y / y) / (y / y))) / z)))) = y / y.  [para(12(a,1),4(a,1,1)),rewrite([2(34)])].
19 67 (x / x) / (x / (y / y)) = (x / x) / x.  [para(2(a,1),28(a,1,2,2,2))].
20 96 x / ((((y / y) / (y / y)) / (y / y)) / (((y / y) / (y / y)) / ((x / z) / (((((y / y) / (y / y)) / (y / y)) / ((y / y) / (y / y))) / z)))) = y / y.  [back_rewrite(31),rewrite([67(16)])].
21 98 (x / x) / (((((x / x) / (x / x)) / (x / x)) / ((x / x) / (x / x))) / ((((x / x) / (x / x)) / (x / x)) / y)) = (x / x) / ((y / z) / ((((x / x) / (x / x)) / (x / x)) / z)).  [back_rewrite(21),rewrite([67(8)])].
22 100 ((x / x) / (x / x)) / ((((((x / x) / (x / x)) / (x / x)) / y) / ((((x / x) / (x / x)) / (x / x)) / ((x / x) / (x / x)))) / (x / x)) = y.  [back_rewrite(11),rewrite([67(10)])].
23 102 x / (y / y) = x.  [para(2(a,1),5(a,1,2,2))].
24 137 (x / x) / ((x / x) / y) = y.  [back_rewrite(100),rewrite([102(3),102(4),102(4),102(6),102(6),102(7),102(6),102(5),102(5)])].
25 139 (x / x) / ((y / z) / ((x / x) / z)) = (x / x) / y.  [back_rewrite(98),rewrite([102(4),102(4),102(5),102(4),102(5),102(5),137(5),102(7),102(7)]),flip(a)].
26 140 x / x = y / y.  [back_rewrite(96),rewrite([102(3),102(3),102(4),102(6),102(6),102(7),102(6),139(7),137(4)])].
27 141 $F.  [resolve(140,a,3,a)].
28
29 ======================== end of proof ====================
30
```

Ciascuno di questi fatti si può dimostrare (con enorme tedio) usando ripetutamente l'assioma di taglio e vari altri lemmi preliminari.
Questa maniera di dimostrare le cose però ha molti svantaggi:

- E' molto verbosa (è facile sbagliare, nel fare queste dimostrazioni...);
- E' del tutto ad hoc (che si fa, se una terza nozione di gruppo$_3$ viene introdotta?);

Fa sospettare che sia indispensabile provare fatti di teoria dei gruppi in termini di una certa sintassi, rigida e difficile da cambiare. Questo è falso.

C'è differenza tra ciò di cui la matematica parla, e come implementa ciò di cui parla.

Come chiarire la situazione?

C'è bisogno di un altro approccio,
più intrinseco ed elegante...

Consideriamo un gruppo$_1$ (l'assioma di taglio di un gruppo$_2$ è... scomodo).
La presentazione di un gruppo$_1$ si può astrarre e presentare come un *oggetto* $\mathtt{G}$ e

$$m : \mathtt{G}^2 \to \mathtt{G}^1 \qquad i : \mathtt{G}^1 \to \mathtt{G}^1 \qquad e : \mathtt{G}^0 \to \mathtt{G}^1$$

dove $\mathtt{G}$ è una stenografia per un 'simbolo segnaposto', e $\mathtt{G}^n$ è la ripetizione di n di questi simboli

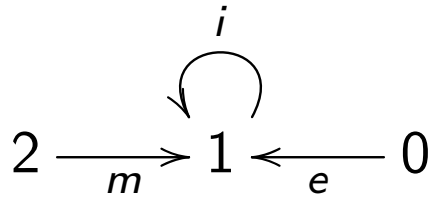
$$\mathtt{G}, \quad \mathtt{GG} = \mathtt{G}^2, \quad \mathtt{GGG} = \mathtt{G}^3, \ldots$$

(da pensare come astrazione del prodotto $\mathtt{G}^0 = \{\bullet\}$ e $\mathtt{G}^{1+n} := \mathtt{G} \times \mathtt{G}^n$).

In modo ancora più conciso, le operazioni che definiscono un gruppo$_1$ sono della forma

$$m : 2 \to 1 \qquad i : 1 \to 1 \qquad e : 0 \to 1$$

dove il riferimento a $\mathbb{G}^n$ è stato rimosso e si scrive solo 'n' in luogo di $\mathbb{G}^n$.
Meglio ancora:



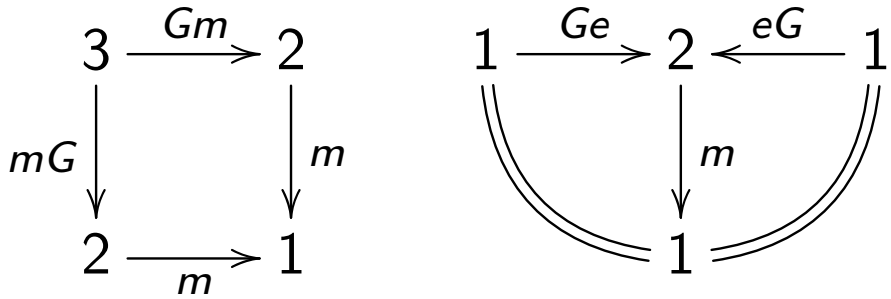
$Gm : 1{+}2 \to 1{+}1 \qquad mG : 2{+}1 \to 1{+}1 \qquad eG, Ge : 0{+}1, 1{+}0 \to 1{+}1$

si interpretano come:

$$G \times m : G \times G \times G \to G \times G, \ldots \, etc$$

Questa si dice una *segnatura* per la teoria dei gruppi, rappresentante una categoria $Th(Grp_1)$, contenente almeno questi oggetti e frecce.

Le condizioni da imporre sulla categoria sono quelle che definiscono un gruppo$_1$:



Almeno...

Quanti (e quali) altri oggetti e frecce ci sono in questa categoria $Th(Grp_1)$? Solo m, i, e?

Beh, no: ad esempio, esistono anche

$$5 \xrightarrow{GmGG} 4 \xrightarrow{GGm} 3 \xrightarrow{Gm} 2 \xrightarrow{m} 1$$

$$2 \xrightarrow{eeGeG} 5 \xrightarrow{GGmG} 4 \xrightarrow{GmG} 3 \xrightarrow{Gm} 2 \xrightarrow{m} 1$$

che esprimono rispettivamente le moltiplicazioni iterate

$$x.((y.z).(w.t)) \qquad e.((e.(x.e)).y)$$

Si noti però che tutte le composizioni

$$x.(y.((z.w).t)), (x.(y.z)).(w.t), (((x.y).z).w).t \ldots$$

sono uguali a $x.(y.(z.(w.t)))$.

Si noti anche che tutte le composizioni

$$e.((e.(x.e)).y), e.((e.(x.e)).y), (e.(e.x)).(e.y), \ldots$$

sono uguali a $x.y$.

$$\begin{aligned} e.((e.(x.e)).y) &= (e.(x.e)).y \\ &= (x.e).y \\ &= x.y \end{aligned}$$

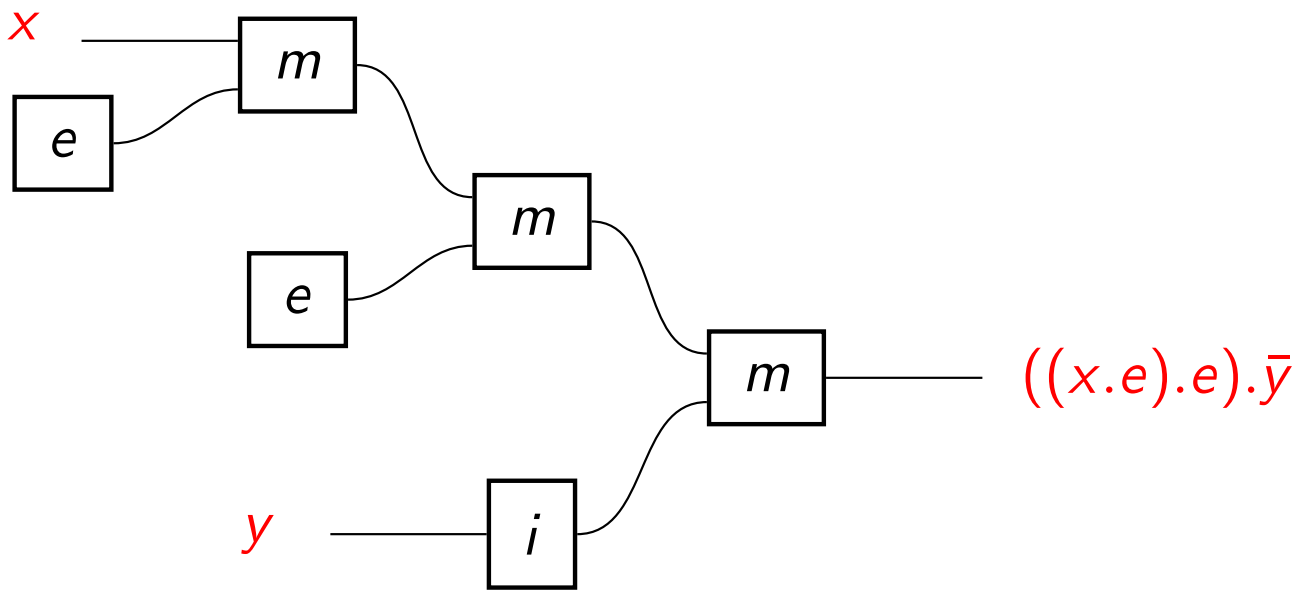
Definizione

Una teoria algebrica (a una sorte) consiste di una categoria $\mathcal{T}$ i cui oggetti sono potenze iterate $\mathtt{X}^n$ di un unico $\mathtt{X}$,

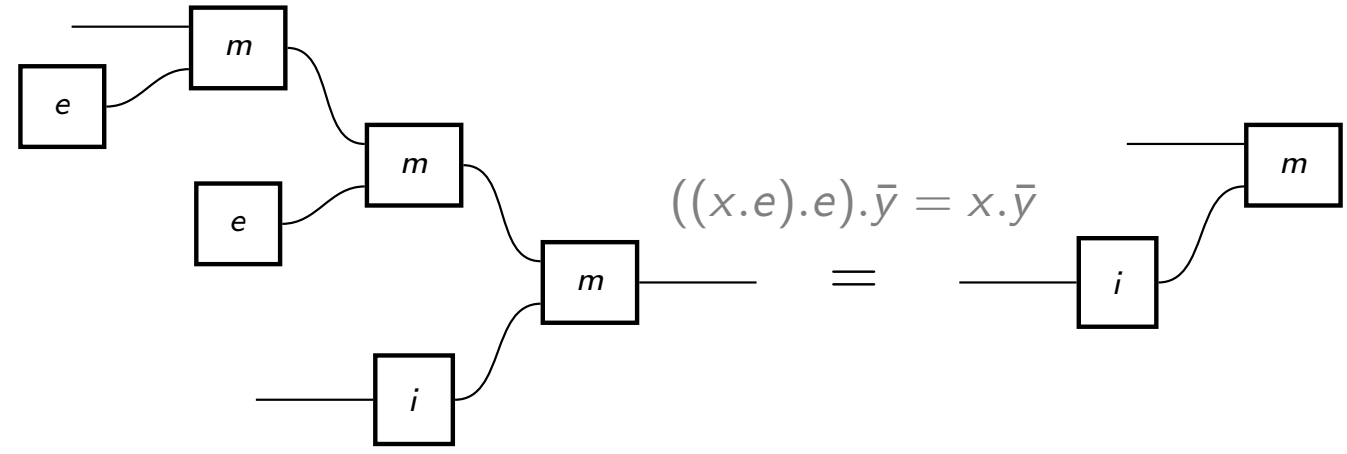
$$\mathtt{X}^0, \quad \mathtt{X}^1, \quad \mathtt{X}^2, \quad \ldots \quad \mathtt{X}^n, \quad \mathtt{X}^{n+1} \ldots$$

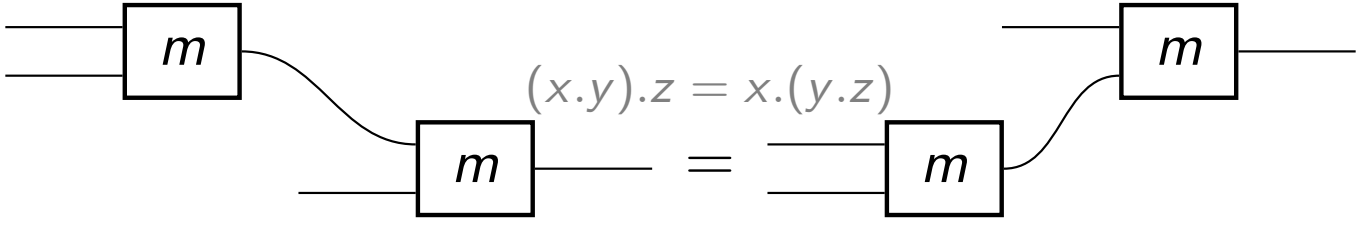
e le frecce sono da interpretare come operazioni algebriche.
Data una presentazione $\Omega = \{f_\omega : [n_\omega] \to [1]\}$ per una classe di strutture algebriche, esiste una teoria algebrica ad essa associata, dove

- gli oggetti sono i numeri naturali $0, 1, 2, \ldots$;
- esiste una freccia $f_\omega : [n_\omega] \to [1]$ per ogni operazione nella presentazione;
- dalle operazioni elementari i termini si costruiscono induttivamente come alberi


x
m
e
m
e
m
((x.e).e).ȳ
y
i

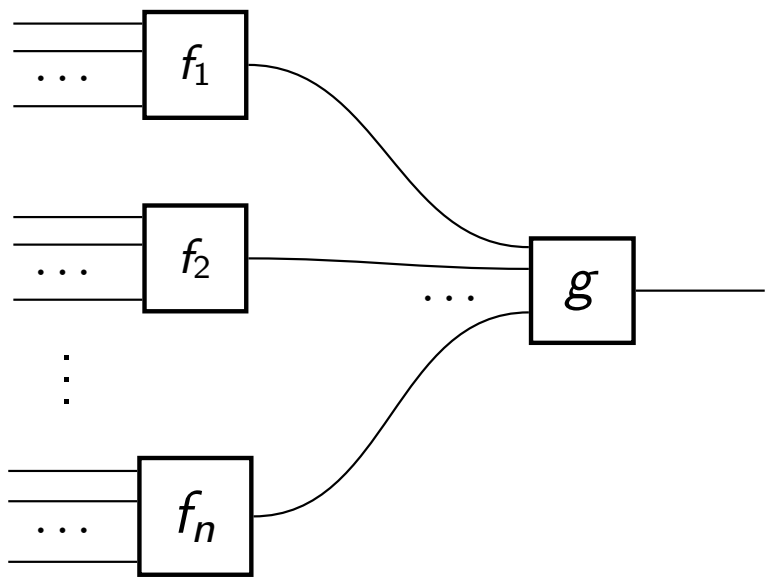
Alcuni alberi sono uguali ad altri...





Le frecce in $\mathcal{T}$ della forma $[m] \to [n]$ sono n-uple di termini $(f_1, \ldots, f_n)$, dove ciascun $f_i : [m] \to [1]$ è una operazione m-aria della teoria.
La composizione di $f : [m] \to [n]$ con $g : [n] \to [k]$ è la freccia $f ; g : [m] \to [k]$ definita, se $f = (f_1, \ldots, f_n)$ e $g = (g_1, \ldots, g_k)$, da

$$f ; g = \Big(g_1[f_1, f_2, \ldots, f_n], g_2[f_1, f_2, \ldots, f_n], \ldots, g_k[f_1, f_2, \ldots, f_n] \Big)$$



Un modello di una teoria algebrica $\mathcal{T}$ consiste di un funtore

$$M : \mathcal{T} \longrightarrow \mathbf{Set}$$

che 'preserva i prodotti'.

I modelli formano la categoria $\mathbf{Mod}(\mathcal{T}, \mathbf{Set})$ i cui morfismi sono le trasformazioni naturali tra funtori.

In altre parole, M associa

- ad ogni oggetto $[n] \in \mathcal{T}$, un insieme $M(n) = M \times M \times \cdots \times M$ (n volte, se $M := M(1)$), e
- ad ogni freccia $f : n \to m$ della teoria algebrica un'applicazione $M(f) = (f_1, \ldots, f_m) : M^n = M(n) \to M(m) = M^m$.

M è quindi determinato dalla sua azione su un generatore, $M(1)$, e sulle operazioni algebriche definienti la teoria algebrica.

Realizza le operazioni prescritte dalla teoria in un modello concreto (il 'supporto' M).

Questa maniera di presentare le teorie algebriche è elegante e concisa, permette di definire le operazioni in modo induttivo, senza dover fare riferimento a un insieme concreto (cosa di cui si occupa la specifica di un modello).

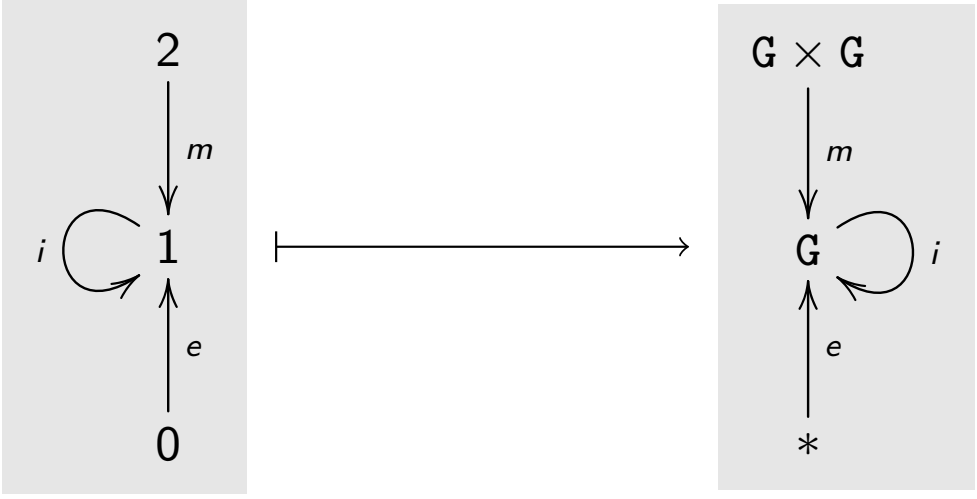
- La teoria è un oggetto puramente sintattico, costruito in maniera indipendente da questa o quella interpretazione concreta;
- La semantica, lungi dall'essere un concetto vago, è a sua volta un oggetto matematico ben preciso, su cui la sintassi viene 'disegnata' nei vari modi ammissibili.

- le teorie-categorie di gruppo_1 e gruppo_2 possono essere molto diverse...
- ...e tuttavia avere 'gli stessi' modelli, nel senso che esiste un'equivalenza tra $\mathbf{Mod}(Th(\text{gruppo}_1), \mathbf{Set})$ e $\mathbf{Mod}(Th(\text{gruppo}_2), \mathbf{Set})$.

Questo è un modo molto generale di interpretare la sintassi e la semantica e stabilisce tra di loro una dualità ('aggiunzione'):

- la sintassi è una classe **Syn** di categorie (piccole) strutturate;
- la semantica è una classe **Sem** di categorie (grandi) che sono modelli/funtori.

La semantica funtoriale è la parte della teoria delle categorie che studia questo tipo di dualità.



$$\mathbf{Syn} \underset{\heartsuit}{\overset{Mod}{\rightleftarrows}} \mathbf{Sem}$$

Dalla sintassi alla semantica si passa prendendo i modelli di una teoria.
Dalla semantica alla sintassi si passa prendendo il 'cuore' della categoria dei modelli $Mod(\mathcal{T}, \mathbf{Set}) = \{\mathcal{T} \to \mathbf{Set}\}_{\times}$.

Dentro $Mod(\mathcal{T}, \mathbf{Set})$ esistono i modelli perfetti (definizione: non importante al momento), con la proprietà che

Teorema

La sotto-categoria $\mathcal{P}erf(\mathcal{T})$ dei modelli perfetti di $Mod(\mathcal{T}, \mathbf{Set})$ è equivalente alla teoria algebrica $\mathcal{T}$.

Per chi sa...

La categoria dei modelli perfetti è un completamento dell'opposto della categoria dei modelli liberi su un insieme finito di generatori.

Interdefinibilità

'Estrarre' la teoria algebrica dalla presentazione gruppo_2 costruisce un modello equivalente per la teoria algebrica di gruppo_1, e viceversa.

(Grazie alla dimostrazione che abbiamo già fatto: costruire un'equivalenza tra le categorie $Th(Grp_1)$ e $Th(Grp_2)$ significa trovare il modo di definire le operazioni di gruppo_1 in termini di quelle di gruppo_2 e viceversa, in un modo che preservi la costruzione e composizione di termini).

Dunque le teorie algebriche di gruppo_1 e gruppo_2 sono equivalenti.

Abelianità

La teoria algebrica dei gruppi abeliani ammette una presentazione molto concisa ed elegante:

- gli oggetti sono i numeri naturali $[0], [1], [2], \ldots$;
- le frecce $n \to m$ sono matrici $m \times n$ (n colonne, m righe) i cui elementi sono numeri interi: la composizione

$$3 \xrightarrow{\begin{bmatrix} 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix}} 2 \xrightarrow{\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}} 3 \quad = \quad 3 \xrightarrow{\begin{bmatrix} 27 & 30 & 33 \\ 61 & 68 & 75 \\ 95 & 106 & 117 \end{bmatrix}} 3$$

è letteralmente il prodotto di matrici.

Teorema / esercizio per i temerari

La teoria algebrica dei gruppi abeliani Ab_1 (assiomi di gruppo$_1$ + $x.y = y.x$) è interdefinibile con la teoria algebrica Ab_2 dei gruppi abeliani specificata dalla teoria equazionale

$$\left\{ 0 \xrightarrow{\ t\ } 1 \xleftarrow{\ \text{-}/\text{-}\ } 2 \;\middle|\; \begin{array}{c} \forall xyz : x/((((x/x)/y)/z)/(((x/x)/x)/z))=y \\ \forall xyz : x/((y/z)/(y/x))=z \end{array} \right\}$$

Dualità

Quella tra teorie algebriche e modelli è un importante esempio delle dualità tra sintassi e semantica, che si possono trovare in molti altri contesti matematici... giusto alcuni nomi per chi vuole usare Google:

- dualità di Stone
- dualità di Gel'fand-Grothendieck
- dualità di Galois-Grothendieck
- dualità di Isbell
- dualità di Pontrjagin
- dualità di Lawvere
- ...

Grothendieck è l'unico ad avere non una ma ben due dualità a lui dedicate...

Bibliografia minima

La teoria della semantica funtoriale è vastissima; per chi volesse approfondire, ecco qualche riferimento iniziale:

- Lawvere, F. William; "Functorial semantics of algebraic theories." Proceedings of the National Academy of Sciences 50.5 (1963): 869-872.
- Linton, Fred E.J; "Some aspects of equational categories." Proceedings of the Conference on Categorical Algebra: La Jolla 1965. Berlin, Heidelberg: Springer Berlin Heidelberg, 1966.
- Adámek, Jiří, Jiří Rosický, and Vitale, Enrico; "Algebraic theories": a categorical introduction to general algebra. Vol. 184. Cambridge University Press, 2010.
- Hyland, J. Martin E; "Classical lambda calculus in modern dress." Mathematical Structures in Computer Science 27.5 (2017): 762-781.
- Adámek, J., Lawvere, F. and Rosický, J. "On the duality between varieties and algebraic theories." Algebra univers. 49, 35–49 (2003).

`https://www.math.union.edu/~niefiels/13conference/Web/`

Alcuni volti