▼ **11.0 - Strutture algebriche in informatica**

▼ **11.1 - Astrazione e Generalizzazione**

Astrazione

> L'**astrazione** è il tipico processo del pensiero scientifico che permette di definire enti, concetti o procedure matematiche, estraendo e isolando caratteristiche comuni a più oggetti e trascurandone altre.

Esempio: per astrarre il concetto di sedia si può estrarre la proprietà riguardante lo scopo, in quanto tutte le sedie hanno lo scopo di far sedere una persona, e trascurare il materiale con il quale è stata fatta, poichè non tutte le sedie sono fatte dello stesso materiale.

L'**algebra astratta** introduce le strutture algebriche per astrarre e classificare le operazioni che sono possibili su determinate classi di oggetti.

Generalizzazione

> La **generalizzazione** consiste nel prendere una definizione che si applica a certi casi e definirla in un nuovo modo tale per cui si applichi anche ad altri casi oltre ai precedenti.

Astrazione e generalizzazione in informatica

Un esempio di astrazione in informatica consiste nell'utilizzo di **interfacce** per nascondere come una struttura dati è stata implementata ma accentuarne il suo utilizzo, mostrandone solo i metodi che si possono utilizzare su di essa.

Esempio:

- Implementazione concreta di una lista di interi:

```
struct node {
  int item,
  node* next;
}
```

- Concetto astratto:

```
node* empty();
insert(int n, node* l);
remove(int n, node* l);
```

Esistono molti **benefici** nell'attuare astrazione e generalizzazione in informatica, eccoli elencati:

- **Riuso**: le buone generalizzazioni si applicano moltissime volte.
- **Chiarezza**: le generalizzazioni catturano concetti di alto livello, introducendo nome comprensibili.
- **Decoupling**: tramite astrazione e generalizzazione è possibile fare in modo che la correttezza di una parte non dipenda dall'implementazione di un'altra.

- **Correttezza**: l'astrazione consente di non dover utilizzare la tecnica del cut&paste, la quale può portare facilmente con sè problemi di correttezza ed errori, i quali devono poi essere corretti in tutte le copie.

I **linguaggi di programmazione** Turing completi, ovvero la maggior parte di quelli in circolazione, presentano la possibilità di effettuare le stesse operazioni e risolvere gli stessi problemi. Per questo motivo la scelta del linguaggio di programmazione da usare per un determinato scopo non sta nel fatto che un linguaggio può risolvere un determinato problema e un altro no, ma piuttosto nel livello di astrazione e generalizzazione che questo presenta. Ad esempio un linguaggio come c o c++ è un linguaggio poco astratto e generalizzato, in quanto consente di programmare ad un livello molto basso, mentre altri linguaggi di programmazione come Haskell presentano un livello di astrazione più elevato.

Dalla generalizzazione alle strutture algebriche in matematica

(Dimostrazioni dei teoremi presentati nelle slide del 29/11)

Esempio di generalizzazione

Avendo questi **due teoremi di partenza**:

- **Teorema T1**: $\forall e \in N.((\forall x.x + e = x) \Rightarrow e = 0)$

  Definizione: e è un **elemento neutro a destra** per $\circ$ sse $\forall x.x \circ e = x$.

- **Teorema T2**: $\forall e \in N.((\forall x.e * x = x) \Rightarrow e = 1)$

  Definizione: e è un **elemento neutro a sinistra** per $\circ$ sse $\forall x.e \circ x = x$.

Vista la loro similitudine, è possibile trovare una **generalizzazione** comune:

Teorema G: $\forall A.\forall \circ : A \times A \to A.\forall e_r, e_l \in A.(\forall x.e_l \circ x = x) \wedge (\forall x.x \circ e_r = x) \Rightarrow e_l = e_r$

- **G** è **T2** nel caso particolare: $A = N, \circ = *, e_r = 1$

Definizione: e è un **elemento neutro** per $\circ$ sse è neutro sia a sinistra che a destra.

È una generalizzazione in quanto:

- **G** è **T1** nel caso particolare: $A = N, \circ = +, e_l = 0$

Da G possiamo scoprire **nuovi concetti** e comprendere meglio i precedenti, ad esempio possiamo arrivare ai seguenti teoremi:

- T**eorema H**: se $\circ$ è un'operazione commutativa, allora se ha un elemento neutro a destra o a sinistra allora tale elemento è neutro.
- Teorema: l'elemento neutro, se esiste, è unico.
- Corollario (per H): l'elemento neutro di un'operazione commutativa, se esiste, è unico.

Per questo motivo G è una **generalizzazione informativa**, in quanto utile. Se una teoria è interessante, vi sono molti teoremi dimostrabili a partire da essi.

Definizione di struttura algebrica

> Una **struttura algebrica** è una tupla formata da uno o più **insiemi**, zero o più **elementi**, zero o più **funzioni** su tali insiemi (chiamate operazioni), e zero o più **assiomi** che devono essere soddisfatti.

Esempi:

- Un **left unital magma** è una tripla $(A, \circ, e)$ tale che $\circ : A \times A \to A$ e e è un elemento neutro a sinistra per $\circ$.
- Un **right unital magma** è una tripla $(A, \circ, e)$ tale che $\circ : A \times A \to A$ e e è un elemento neutro a destra per $\circ$.
- Un **unital magma** è una tripla $(A, \circ, e)$ tale che $\circ : A \times A \to A$ e e è sia un left unital magma che un right unital magma.

Morfismo di strutture algebriche

> Un **morfismo** fra due strutture algebriche dello stesso tipo è una funzione che mappa gli elementi della prima in quelli della seconda rispettandone le proprietà.

Esempio:

- Siano $(A, \circ, a)$ e $(B, \circ, b)$ due left unital magma. Un morfismo da $(A, \circ, a)$ a $(B, \circ, b)$ è una funzione $f \in B^A$ tale che:
  - $f(a) = b$
  - $\forall x, y. f(x \circ y) = f(x) \bullet f(y)$

  Esempio:
  - $(N, +, 0)$ e $(B, *, 1)$ sono due left unital magma. La funzione $f(n) = 2^n$ è un morfismo dal primo al secondo in quanto:
    - $2^0 = 1$
    - $\forall x, y. 2^{x+y} = 2^x * 2^y$

> Un **isomorfismo** di due strutture algebriche è un morfismo dalla prima alla seconda che sia una funzione biettiva e la cui inversa sia un morfismo.

Due strutture algebriche sono **isomorfe** se c'è almeno un isomorfismo fra di esse.

Esempio:

- $(R_0^+, +, 0)$ e $(R^+, *, 1)$ sono isomorfe come testimoniato dal morfismo $e^{\cdot}$ e dal suo morfismo inverso $log_{\cdot}$.

Dalla generalizzazione alle strutture algebriche in informatica

(I seguenti esempi verranno implementati in Haskell)

(Dimostrazioni dei teoremi presentati nelle slide del 29/11)

Esempio di generalizzazione

Problema 1: data una lista di interi, restituirne la somma.

```
sum [] = 0
sum (n:l) = n + sum l
```

Notiamo inoltre che il problema soddisfa la seguente proprietà (ci servirà poi per verificare che anche la generalizzazione la soddisferà):

```
sum (l1 ++ l2) = sum l1 + sum l2
```

(++ indica la concatenazione tra due liste)

Problema 2: data una lista di booleani, restituire la loro congiunzione.

```
conj [] = True
conj (b:l) = b && conj l
```

Questi due problemi presentano due soluzione simili ma non uguali.

Possiamo dunque tentare di trovare una generalizzazione utile partendo da sum: data una lista $[x_1, \ldots, x_n]$, un valore e e un'operazione op, restituisce $op(x1, op(x2, \ldots op(x_n, e) \ldots))$.

```
foldr op e [] = e
foldr op e (n:l) = op n (foldr op e l)
```

Ritroviamo dunque i problemi sum e conj come istanze:

```
sum = foldr (+) 0
conj = foldr (&&) True
```

Inoltre è facilmente dimostrabile che anche la generalizzazione soddisfa le stesse proprietà dei problemi iniziali:

```
foldr op e (l1 ++ l2) = op (foldr op e l) (foldr op e l2)
```

Strutture algebriche in programmazione

Foldr dunque lavora su una **struttura algebrica** (A, op, e), dove A è un tipo di dato, op un'operazione binaria su A consigliata essere associativa, e e un elemento consigliato essere l'elemento neutro dell'operazione.

Ogni linguaggio di programmazione ha i suoi meccanismi per dichiarare tali strutture algebriche.

Ad esempio Haskell ha le **type classes**, e presenta anche già in maniera predefinita la struttura che ci serve in questo caso, ovvero il **monoide**: una tripla $(A, \circ, e)$ tale che $\circ$ è associativa e $(A, \circ, e)$ è un unital magma, ovvero e è l'elemento neutro di $\circ$. Possiamo dunque riscrivere la funzione foldr restringendoci alla type class Monoid:

```
foldr [] = e
foldr (n:l) = op n (foldr l)
```

In questo modo Haskell capisce da solo che per questa funzione deve usare il tipo Monoid, in quanto va a guardare tra le type class definite (Monoid è definita di default). A questo punto è possibile richiamare la funzione sulla lista [1, 2, 3] semplicemente scrivendo **foldr [1, 2, 3]** piuttosto che scrivere **foldr (+) 0 [1, 2, 3]**, in quanto comprende autonomamente l'operazione più adatta è il + (in quanto è associativa per i numeri interi) e l'elemento neutro è lo 0, aumentando dunque la leggibilità e la comprensione del codice.

A questo punto, come abbiamo visto per il caso della matematica, una volta che abbiamo introdotto la struttura algebrica (sotto forma di type class) possiamo costruirci una libreria sopra.

Morfismi in programmazione

Come abbiamo visto per la matematica, è possibile applicare i morfismi anche in informatica, avendo così la possibilità di passare da un'istanza di una struttura algebrica che soddisfi una certa proprietà ad un'altra che soddisfi le stessa proprietà.

Esempio:

- Partiamo da una struttura algebrica molto generica che chiamiamo **foldable** e costituita in questo modo (a, b, e, op), dove a e b sono insiemi, $e \in b$ e op è una funzione $(a \times b) \to b$ (da questa struttura algebrica è possibile definire liste, alberi, array ecc.).

  Un morfismo di foldable da (a, b, e, op) a (a', b', e', op') è una coppia di funzioni $f_a : a \to a'$ e $f_b : b \to b'$ tali che:

  - $f_b(e) = e$'
  - $f_b(op(x, l)) = op'(f_a(x), f_b(l))$

  Definiamo le liste con elementi dell'insieme a come un'istanza di foldable $(a, [a], [], :)$, dove $:$ è la concatenazione tra un elemento di a e una lista di a.

  Applicando il morfismo alle liste con le funzioni $a' = a$ e $f_a(x) = x$ otteniamo:

  - $f_b([]) = e'$
  - $f_b(x : l) = op'(x, f_b(l))$

  Notiamo che f_b è esattamente la **foldr**! Mettendo in pratica quanto visto dunque si riesce ad arrivare a scrivere un morfismo foldr super-generico che data una qualunque istanza di foldable ne sintetizza un'altra combinando ogni elemento della prima con certe funzioni passate dall'utente.

▼ **11.2 - Strutture algebriche interessanti**

Monoidi e gruppi

> $(\mathbb{A}, \circ)$, con $\mathbb{A}$ un insieme e $\circ$ un'operazione binaria del tipo $\mathbb{A} \times \mathbb{A} \to \mathbb{A}$, è un **magma**.

> $(\mathbb{A}, \circ)$ è un **semigruppo** se $\circ$ è associativa.

> $(\mathbb{A}, \circ, e)$ è un **monoide** se è un semigruppo e e è l'elemento neutro di $\circ$.

> $(\mathbb{A}, \circ, e, \cdot^{-1})$, con $\cdot^{-1}$ una generica operazione unaria, è un **gruppo** se $(\mathbb{A}, \circ, e)$ è un monoide e $\forall x \in \mathbb{A}.x \circ x^{-1} = e = x^{-1} \circ x$.

Ognuna di queste strutture algebriche viene detta **abeliana** se $\circ$ è commutativa, ovvero $x \circ y = y \circ x$.

Anelli

Gli **anelli** nascono dal bisogno di descrivere insiemi su cui agiscono due operazioni differenti che interagiscono "bene" tra di loro.

> $(\mathbb{A}, +, 0, *)$, dove $+$ e $*$ sono due operazioni binarie, è un **semianello** se $(\mathbb{A}, +, 0)$ è un monoide abeliano, $(\mathbb{A}, *, 1)$ è un monoide e valgono le seguenti proprietà:
>
> - $\forall x . x * 0 = 0 = 0 * x$.
> - $\forall x, y . (x + y) * z = xz + yz$.

> $(\mathbb{A}, +, 0, \cdot^{-1}, *)$ è un **anello** se $(\mathbb{A}, +, 0, *)$ è un semianello e $(\mathbb{A}, +, 0, \cdot^{-1})$ è un gruppo abeliano.

▼ **11.3 - Costruzioni dell'algebra universale**

Sottoinsiemi chiusi e sottostrutture algebriche

Insieme chiuso

> Sia $(\mathbb{A}, \circ)$ un semigruppo e $\mathbb{B} \subseteq \mathbb{A}$. $\mathbb{B}$ si dice **chiuso** rispetto a $\circ$ sse $\forall x, y \in \mathbb{B} . x \circ y \in \mathbb{B}$.

Esempio: l'insieme $\mathbb{P}$ dei numeri pari è chiuso rispetto alla somma, mentre l'insieme $\mathbb{D}$ dei numeri dispari non lo è.

Sottostrutture algebriche

> Data una struttura algebrica che si applica ad un insieme $\mathbb{A}$ e un $\mathbb{B} \subseteq \mathbb{A}$, $\mathbb{B}$ forma una **sottostruttura algebrica** della prima se tutte le operazioni sono chiuse rispetto a $\mathbb{B}$ e tutte le costanti della struttura appartengono a $\mathbb{B}$.

Esempio: considera l'insieme $\mathbb{P}$ dei numeri naturali pari. $\mathbb{P}$ è il sostegno di un sottosemigruppo di $(\mathbb{N}, *)$, di un sottomonoide di $(\mathbb{N}, +, 0)$, di un sottosemianello di $(\mathbb{N}, +, 0, *)$, ma non forma un sottomonoide di $(\mathbb{N}, *, 1)$ perchè $1 \notin \mathbb{P}$.

Intersezione e unione di sottostrutture algebriche

> Data una struttura di cui $\mathbb{B}$ e $\mathbb{C}$ sono sottostrutture, allora anche $\mathbb{B} \cap \mathbb{C}$ lo è.

(Dimostrazione slide 20 del pacco di slide "slides_alg2_Fabio").

> L'unione di sottostrutture non è in generale una sottostruttura.

Prodotto cartesiano di strutture algebriche

> Date due strutture algebriche A e B rispettivamente di sostegno $\mathbb{A}$ e $\mathbb{B}$, il loro **prodotto cartesiano** è una struttura algebrica dello stesso tipo e presenta le seguenti proprietà:
>
> - il **sostegno** è $\mathbb{A} \times \mathbb{B}$.

> - le **costanti** $e_{A\times B}$ richieste dal tipo di struttura sono coppie $< e_A, e_B >$ di costanti corrispondenti nelle due strutture.
> - le **operazioni** $\circ_{A\times B}$ richiesti dal tipo di struttura agiscono applicando l'operazione corrispondente sugli elementi della coppia: $< x_1, y_1 > \circ_{A\times B} < x_2, y_2 >=< x_1 \circ_A x_2, y_1 \circ_B y_2 >$.

Dunque è possibile costruire nuove istanze di sottostrutture algebriche usando intersezioni e prodotti cartesiani.

Sottostrutture tramite i morfismi

Morfismi di strutture algebriche

> Date due strutture algebriche dello stesso tipo A di sostegno $\mathbb{A}$ e B di sostegno $\mathbb{B}$, un morfismo da A a B è una funzione $f : A \to B$ tale che:
>
> - per ogni costante e_A di A, $f(e_A) = e_B$.
> - per ogni operazione op_A di A: $\forall x_1, \ldots, x_n . f(op_A(x_1, \ldots, x_n)) = op_B(f(x_1), \ldots, f(x_n))$.

Immagini di morfismi

> Sia f un morfismo da una struttura algebrica A a una struttura algebrica B. $Imm(f)$ è una sottostruttura di B.

Morfismo come osservazione

Un morfismo $f : \mathbb{A} \to \mathbb{B}$, data una qualche struttura algebrica sui sostegni $\mathbb{A}$ e $\mathbb{B}$, può essere pensato come un modo per **osservare** sugli elementi di $\mathbb{A}$ delle **proprietà** $\mathbb{B}$.

Esempio: dato un insieme X, la funzione $|\cdot| = P(X) \to \mathbb{N}$ (cardinalità) osserva per ogni sottoinsieme di Xquanto sia la sua cardinalità, ovvero il suo numero di elementi.

Supponiamo che tali osservazioni siano le uniche che ci interessano in un determinato momento, dunque vogliamo attuare un'astrazione degli elementi di $\mathbb{A}$ che mantenga solo le loro proprietà osservabili $\mathbb{B}$.

> Data un morfismo $f : \mathbb{A} \to \mathbb{B}$, la **relazione di equivalenza** indotta da f, $\sim_f$, è definita come segue: $x_1 \sim_f x_2$ sse $f(x_1) = f(x_2)$.

Esempio: se $f$calcola l'età di una persona allora $\sim_f$ è la relazione "essere coetanei".

A questo punto possiamo introdurre la definizione di proiezione $[\cdot]$, molto importante per arrivare al teorema fondamentale dei morfismi per strutture algebriche:

> Chiamiamo $[\cdot]$: $\mathbb{A} \to \mathbb{A}_{/\sim_f}$ (**proiezione**) la funzione che mappa ogni $x \in \mathbb{A}$ nella sua classe di equivalenza $\sim_f$.

Esempio: se f calcola l'età di una persona allora $\mathbb{A} \to \mathbb{A}_{/\sim_f}$ mappa ogni persona nella classe di equivalenza dei suoi coetanei. In questo modo si ottiene l'insieme quoziente i cui elementi sono le classi di equivalenza "età".

Teorema fondamentale dei morfismi per strutture algebriche

> Per ogni f morfismo da A a B si ha che:
>
> 1. $\mathbb{A}_{/\sim_f}$ è il sostegno di una struttura algebrica dello **stesso tipo**.
> 2. $\mathbb{A}_{/\sim_f}$ è **isomorfo** a $Imm(f)$.

▼ **11.4 - Teoria dei gruppi**

> Un **gruppo** $(\mathbb{A}, \circ, e, \cdot^{-1})$ è un monoide con un elemento aggiuntivo $\cdot^{-1}$ tale che: $\forall x \in \mathbb{A}. x \circ x^{-1} = e = x^{-1} \circ x$. a^{-1} si chiama **elemento opposto** di a.

Teoremi elementari sui gruppi

- L'elemento opposto di a è **unico**.

  (Dimostrazione slide 8 del pacco di slide n. 3 di algebra)
- In un gruppo, $\forall x, y. (x \circ y)^{-1} = y^{-1} \circ x^{-1}$.

  (Dimostrazione slide 9 del pacco di slide n. 3 di algebra)
- In un gruppo, $\forall x. (x^{-1})^{-1} = x$.

  (Dimostrazione slide 10 del pacco di slide n. 3 di algebra)
- In un gruppo, $x = y \iff x \circ y^{-1} = e \iff x^{-1} \circ y = e$

  (Dimostrazione slide 11 del pacco di slide n. 3 di algebra)

Esempi di gruppi

- Il gruppo degli **interi con l'addizione** è $(\mathbb{Z}, +, 0, \cdot^{-1})$. Scriviamo $\cdot^{-1}$ come $-n$, infatti ad esempio $3 + (-3) = 0 = (-3) + 3$.
- Il gruppo degli **interi modulo 2 con l'addizione** corrisponde al gruppo $(\mathbb{Z}_2, +, 0, \cdot^{-1})$, nel quale $\mathbb{Z}_2 = \{0, 1\}$. Costituisce un gruppo in quanto l'addizione viene definita secondo l'aritmetica modulare, ad esempio $1 + 1 = 0$ e $0 + 1 = 1$, e per quanto riguarda gli elementi opposti abbiamo che $1^{-1} = 1$ e $0^{-1} = 0$, infatti $1 + 1^{-1} = 1 + 1 = 0$ e $0 + 0^{-1} = 0 + 0 = 0$.

  Più in generale, per ogni $n \in \mathbb{N}$, gli interi modulo n formano un gruppo $(\mathbb{Z}_n, +, 0, \cdot^{-1})$.
- Dato un poligono regolare P_n con n lati, le isometrie su di esso formano un gruppo chiamato il gruppo diedrale D_n in cui:
  - L'operazione (composizione) del gruppo è la composizione di isometrie.
  - L'elemento neutro è l'isometria identità.
  - L'inverso di un'isometria è l'isometria inversa.

I gruppi di permutazioni

Una famiglia importante di gruppi è formata dai gruppi di permutazioni.

> Sia $\mathbb{A}$ un insieme. Una **permutazione** di $\mathbb{A}$ è semplicemente una funzione biettiva $\pi : \mathbb{A} \to \mathbb{A}$ (vengono modificate le posizioni degli elementi all'interno dell'insieme).

Dato un insieme $\mathbb{A}$, abbiamo che:

- $Perm(\mathbb{A})$ è l'insieme di tutte le permutazioni dell'insieme $\mathbb{A}$, spesso chiamato anche **gruppo simmetrico** di $\mathbb{A}$.
- Siano π_1, π_2 due permutazioni di $\mathbb{A}$, anche $\pi_1 \circ \pi_2$ lo è.
- La funzione identità $id : \mathbb{A} \to \mathbb{A}$ è una permutazione.
- Sia π una permutazione di $\mathbb{A}$, anche π^{-1}, ovvero la funzione inversa di π, lo è.

> $(Perm(\mathbb{A}), \circ, id, \cdot^{-1})$ è un gruppo, chiamato **gruppo delle permutazioni** di $\mathbb{A}$.

Teorema di Cayley

> Il **teorema di Cayley** afferma che ogni gruppo che ha come sostegno un insieme $\mathbb{A}$ è isomorfo a un sottogruppo del gruppo $Perm(\mathbb{A})$ delle permutazioni di $\mathbb{A}$.

Supponiamo infatti di avere un gruppo che ha come sostegno un insieme $\mathbb{A}$ composto in questo modo $\{a_1, a_2, a_3, \dots, a_n\}$. Ora prendiamo uno alla volta tutti gli elementi dell'insieme ed effettuiamo l'operazione $\circ$ del gruppo per ogni elemento dell'insieme, ottenendo un insieme di permutazioni $\{a \circ a_1, a \circ a_2, a \circ a_3, \dots, a \circ a_n\}$, dove al posto di a vengono inseriti tutti gli elementi dell'insieme $\mathbb{A}$ uno ad uno. Otteniamo una **permutazione** in quanto dobbiamo vedere l'operazione $a \circ a_i$ ad una posizione differente dell'insieme.

Rappresentiamo l'operazione effettuata in questo modo $\pi_a : \{1, 2, 3, ..., n\} \to \{1, 2, 3, ..., n\}$ dove $\pi_a(i)$ è definito come $a_j = a \circ a_i$. È possibile inoltre dimostrare che il morfismo $\pi = a \mapsto \pi_a$ è un **isomorfismo** tra $(\mathbb{A}, \circ, e, \cdot^{-1})$ ed un sottogruppo del gruppo $Perm(\mathbb{A})$ delle permutazioni di $\mathbb{A}$ (dimostrazioni da slide 40 a 42 del pacco di slide della lezione n. 3 di algebra).

L'isomorfismo mappa ad un **sottogruppo** di $Perm(\mathbb{A})$ e non a tutto il gruppo in quanto se l'insieme $\mathbb{A}$ ha n elementi l'isomorfismo creerà n permutazioni diverse, ognuna data moltiplicando ogni elementi dell'insieme $\mathbb{A}$ per uno degli n elementi dell'insieme $\mathbb{A}$. Il gruppo di $Perm(\mathbb{A})$ però è formato invece da $n!$ permutazioni differenti, dunque se $n > 2$ l'isomorfismo mapperà ad un sottogruppo di $Perm(\mathbb{A})$.

Generalizzando, questa costruzione ci dice che qualsiasi gruppo può essere visto come un gruppo di permutazioni, dunque, in un certo senso, anche la teoria dei gruppi non è altro che la teoria delle permutazioni.

Esempio:

- Applichiamo il morfismo π al gruppo $(\mathbb{Z}_2, +, 0, \cdot^{-1})$, in cui $Z_2 = \{0, 1, 2\}$.

  Otteniamo le seguenti permutazioni:

  - $\pi_0 : 0 + 0 = 0, 1 + 0 = 1, 2 + 0 = 2$.
  - $\pi_1 : 0 + 1 = 1, 1 + 1 = 2, 2 + 1 = 0$

- $\pi_2 : 0 + 2 = 2, 1 + 2 = 0, 2 + 2 = 1$

Otteniamo dunque un sottogruppo del gruppo $(Perm(Z_2), \circ, id, \cdot^{-1})$, in quanto ad esempio la permutazione $f : 0 \mapsto 0, 1 \mapsto 2, 2 \mapsto 1$ non l'abbiamo ottenuta, dunque non è nell'immagine dell'isomorfismo.