Logică și structuri discrete

Relații. Funcții parțiale

Casandra Holotescu

casandra@cs.upt.ro

https://tinyurl.com/lecturesLSD

Relații – aspecte teoretice

Relații în lumea reală și informatică

O relație (matematică) modelează *legătura* dintre două entități (posibil de *tip* diferit)

relații subiect-obiect: un om *cine?* ←relația→ a citit *ce?* o carte

relații umane: copil , părinte , prieten

relații cantitative : egal, mai mic

Relații în lumea reală și informatică

O relație (matematică) modelează *legătura* dintre două entități (posibil de *tip* diferit)

relații subiect-obiect: un om *cine?* ←relația→ a citit *ce?* o carte

relații umane: copil , părinte , prieten

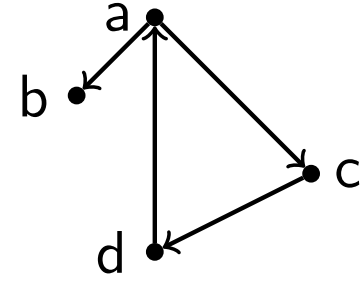
relații cantitative : egal, mai mic

Transpuse în informatică:

rețele sociale : “prieten”, “follow”, “în cercuri”, etc.

O relație între elementele *aceleiași* mulțimi definește un *graf* (elementele sunt noduri, relația e reprezentată prin muchii)

$\Rightarrow$ relațiile sunt o noțiune cheie în teoria grafurilor



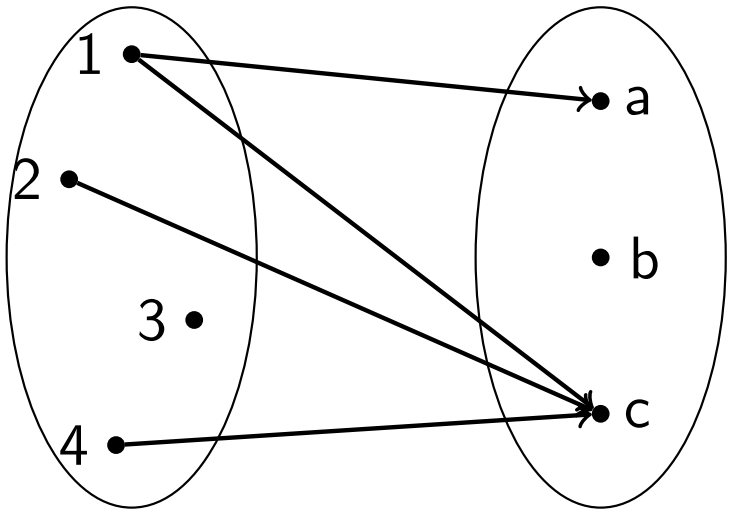
O relație e o mulțime de perechi

O *relație binară* R între două mulțimi A și B e o *mulțime de perechi*:
o *submulțime a produsului cartezian* $A \times B$: $R \subseteq A \times B$

O relație e o mulțime de perechi

O *relație binară* R între două mulțimi A și B e o *mulțime de perechi*:
o *submulțime a produsului cartezian* $A \times B$: $R \subseteq A \times B$

Notăm $(x, y) \in R$ sau $x\ R\ y$ sau $R(x, y)$ x e *în relație* cu y



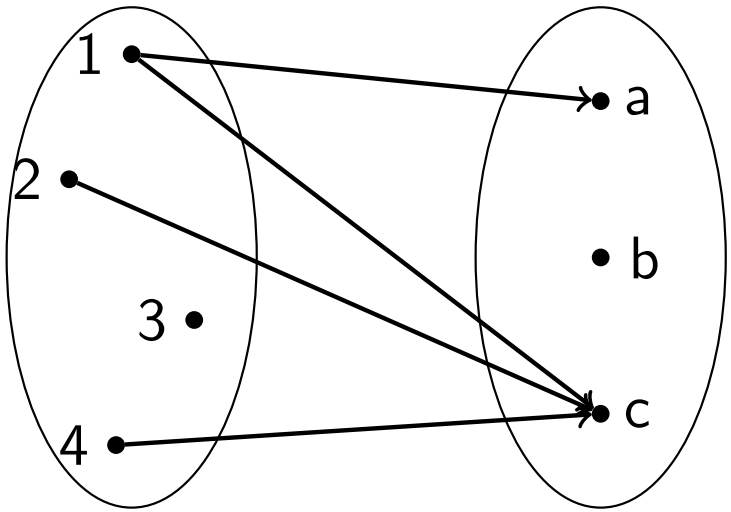
$A = \{1, 2, 3, 4\}, \quad B = \{a, b, c\}$

$R = \{(1, a), (1, c), (2, c), (4, c)\}$

O relație e o mulțime de perechi

O *relație binară* R între două mulțimi A și B e o *mulțime de perechi*:
o *submulțime a produsului cartezian* $A \times B$: $R \subseteq A \times B$

Notăm $(x, y) \in R$ sau $x\ R\ y$ sau $R(x, y)$ x e *în relație* cu y



$A = \{1, 2, 3, 4\}, \quad B = \{a, b, c\}$

$R = \{(1, a), (1, c), (2, c), (4, c)\}$

O relație e o noțiune *mai generală* decât o funcție: o funcție asociază *fiecărui* $x \in A$ *un singur* $y \in B$

Într-o relație putem avea (vezi figura):

1: are asociate *mai multe* elemente: a, c

2: are asociat *un singur* element: c

3: nu are asociat *niciun* element din B

Relații: generalități

În general, o relație *nu* e o noțiune simetrică:
produsul cartezian/perechea sunt noțiuni ordonate, $(x, y) \neq (y, x)$

Există, desigur, relații simetrice (în lumea reală și în matematică)

Relații: generalități

În general, o relație *nu* e o noțiune simetrică:
produsul cartezian/perechea sunt noțiuni ordonate, $(x, y) \neq (y, x)$

Există, desigur, relații simetrice (în lumea reală și în matematică)

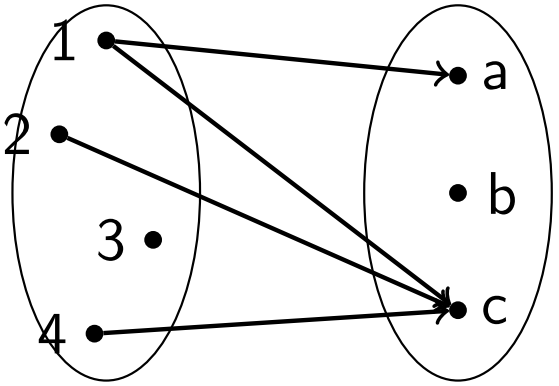
Generalizat, putem avea o *relație n-ară* care e o mulțime de n-tupluri (din produsul cartezian a n mulțimi).

Exemplu: $R \subseteq \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z}$
$R(x, y, m)$ dacă m e un multiplu comun pentru x și y:

$R(2, 9, 18)$, $R(6, 9, 18)$, $R(2, 9, 36)$, etc.

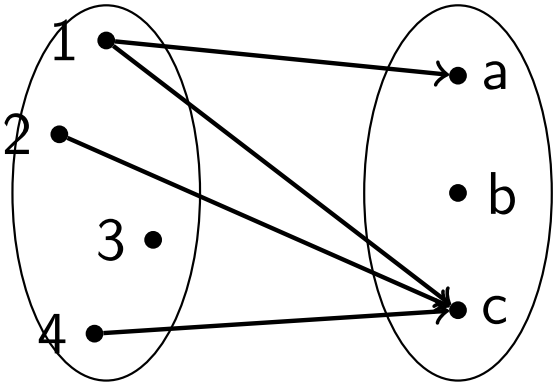
Reprezentarea unei relații

Explicit, prin *mulțimea perechilor*
(dacă e finită)
$R \subseteq \{1, 2, 3, 4\} \times \{a, b, c\}$
$R = \{(1, a), (1, c), (2, c), (4, c)\}$



Reprezentarea unei relații

Explicit, prin *mulțimea perechilor* (dacă e finită)
$R \subseteq \{1, 2, 3, 4\} \times \{a, b, c\}$
$R = \{(1, a), (1, c), (2, c), (4, c)\}$

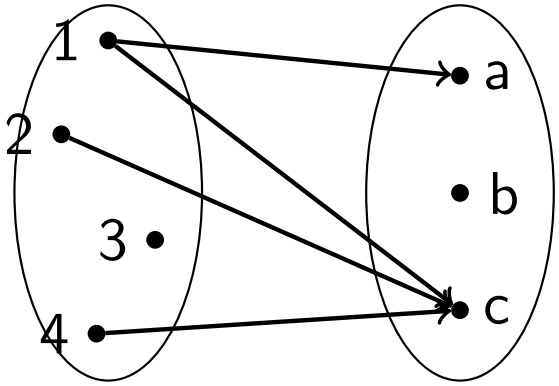


Printr-o *regulă* care leagă elementele:

$$R = \{(x, x^2 + 1) \mid x \in \mathbb{Z}\}$$

Reprezentarea unei relații

Explicit, prin *mulțimea perechilor*
(dacă e finită)
$R \subseteq \{1,2,3,4\} \times \{a,b,c\}$
$R = \{(1,a),(1,c),(2,c),(4,c)\}$



Printr-o *regulă* care leagă elementele:

$$R = \{(x, x^2+1) \mid x \in \mathbb{Z}\}$$

Ca *matrice* booleană/binară, dacă A, B finite,
linii indexate după A, și coloanele după B
$m_{xy} = 1$ dacă $(x,y) \in R$, $m_{xy} = 0$ dacă $(x,y) \notin R$
în practică: dacă A și B nu sunt foarte mari

	a	b	c
1	1	0	1
2	0	0	1
3	0	0	0
4	0	0	1

Relația văzută ca funcție

O *relație* $R \subseteq A \times B$ poate fi văzută ca o *funcție* $f_R : A \to \mathcal{P}(B)$ de la A la *mulțimea părților* lui B:

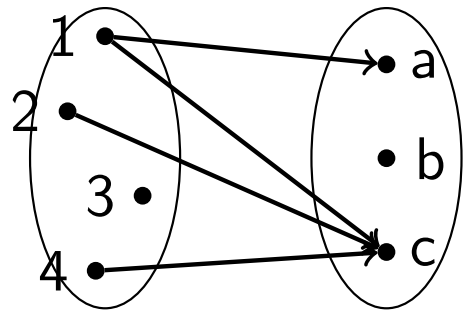
$$f_R(x) = \{y \in B \mid (x, y) \in R\}$$

Relația văzută ca funcție

O *relație* $R \subseteq A \times B$ poate fi văzută ca o *funcție* $f_R : A \to \mathcal{P}(B)$ de la A la *mulțimea părților* lui B:

$$f_R(x) = \{y \in B \mid (x, y) \in R\}$$

Asociază fiecărui x mulțimea elementelor lui B cu care x e în relație (posibil vidă):
$f_R(1) = \{a, c\}$, $f_R(3) = \emptyset$



Un vector de biți/booleni poate reprezenta o mulțime:

a	b	c
1	0	1

reprezintă $\{a, c\}$ (prin funcția caracteristică)

Numărul de relații între două mulțimi

Între A și B (finite) există $2^{|A|\cdot|B|}$ relații $R \subseteq A \times B$

Rezultă direct din definiție: o relație e o submultime $R \subseteq A \times B$.
Deci, $R \in \mathcal{P}(A \times B)$. Dar $|\mathcal{P}(A \times B)| = 2^{|A\times B|} = 2^{|A|\cdot|B|}$.

Sau, folosind reprezentarea ca matrice, care are $|A| \cdot |B|$ elemente.
fiecare: ales independent în 2 feluri: 0 sau 1, deci $2^{|A|\cdot|B|}$ variante.

Sau, considerând funcția corespunzătoare, $f : A \to \mathcal{P}(B)$.
Numărul de funcții e $|\mathcal{P}(B)|^{|A|} = (2^{|B|})^{|A|} = 2^{|B|\cdot|A|}$

Funcții parțiale

O funcție parțială $f : A \nrightarrow B$ e un *caz particular de relație*:
asociază câte *un singur* element din B (ca funcția)
dar *nu neapărat fiecărui* element din A (cum e obligată funcția)

Funcții parțiale

O funcție parțială $f : A \nrightarrow B$ e un *caz particular de relație*:
asociază câte *un singur* element din B (ca funcția)
dar *nu neapărat fiecărui* element din A (cum e obligată funcția)

Funcții parțiale sunt utile
– când domeniul *exact* al funcției *nu* e cunoscut
(funcții care nu sunt neapărat calculabile în orice punct).
în conjectura Collatz $(3 \cdot n + 1)$, pentru anumiți n
numărul de pași până la 1 ar putea să nu existe (infinit)

– când domeniul de definiție al funcției e foarte mare sau nelimitat,
dar reprezentăm funcția explicit *doar* pentru valorile de interes
Exemplu: populația unei localități
posibil să nu știm populația pentru toate localitățile
dacă argumentul e un șir, nu orice șir e nume de localitate

Relații binare. Proprietăți

Relații binare pe o mulțime

Următoarele proprietăți sunt definite pentru relații binare pe *o* (aceeași) mulțime X: $R \subseteq X \times X$

Relații binare pe o mulțime

Următoarele proprietăți sunt definite pentru relații binare pe *o* (aceeași) mulțime X: $R \subseteq X \times X$

reflexivă: pentru orice $x \in X$ avem $(x, x) \in R$

ireflexivă: pentru orice $x \in X$ avem $(x, x) \notin R$

Relații binare pe o mulțime

Următoarele proprietăți sunt definite pentru relații binare pe *o* (aceeași) mulțime X: $R \subseteq X \times X$

reflexivă: pentru orice $x \in X$ avem $(x, x) \in R$

ireflexivă: pentru orice $x \in X$ avem $(x, x) \notin R$

simetrică: pentru orice $x, y \in X$,
dacă $(x, y) \in R$ atunci și $(y, x) \in R$

antisimetrică: pentru orice $x, y \in X$,
dacă $(x, y) \in R$ și $(y, x) \in R$, atunci $x = y$

Relații binare pe o mulțime

Următoarele proprietăți sunt definite pentru relații binare pe *o* (aceeași) mulțime X: $R \subseteq X \times X$

reflexivă: pentru orice $x \in X$ avem $(x, x) \in R$

ireflexivă: pentru orice $x \in X$ avem $(x, x) \notin R$

simetrică: pentru orice $x, y \in X$,
dacă $(x, y) \in R$ atunci și $(y, x) \in R$

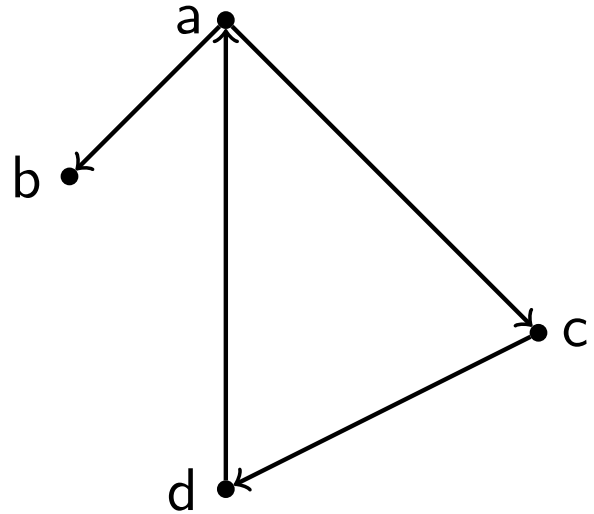
antisimetrică: pentru orice $x, y \in X$,
dacă $(x, y) \in R$ și $(y, x) \in R$, atunci $x = y$

tranzitivă: pentru orice $x, y, z \in X$,
dacă $(x, y) \in R$ și $(y, z) \in R$, atunci $(x, z) \in R$

Relații binare și grafuri

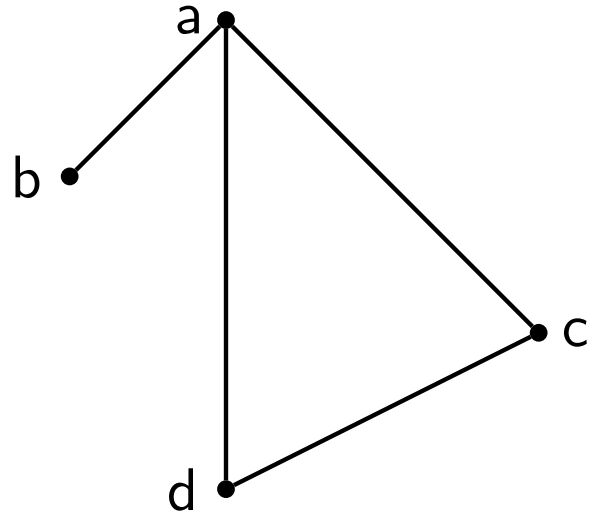
O relație binară pe o mulțime X poate fi reprezentată ca un *graf* cu X ca mulțime de noduri:

graf *orientat*: relație *oarecare*



$R = \{(a,b),(a,c),(c,d),(d,a)\}$

graf *neorientat*: relație *simetrică*



$R = \{(a,b),(a,c),(a,d),(b,a),$
$(c,a),(c,d),(d,a),(d,c)\}$

Relații de echivalență

O relație e *de echivalență* dacă e *reflexivă*, *simetrică* și *tranzitivă*

Relația de egalitate e (evident) o relație de echivalență.

Relația de congruență modulo un număr:
$a \equiv b \pmod{n}$ dacă $n \mid a - b$ (divide diferența)

Relații de echivalență

O relație e *de echivalență* dacă e *reflexivă*, *simetrică* și *tranzitivă*

Relația de egalitate e (evident) o relație de echivalență.

Relația de congruență modulo un număr:
$a \equiv b \pmod{n}$ dacă $n \mid a - b$ (divide diferența)

Clasa de echivalență a lui x
e mulțimea elementelor aflate în relație cu x
$\{y \mid (y, x) \in R\}$ notată $\hat{x}$ sau $[x]$

O relație de echivalență pe X definește o *partiție* a lui X
(două clase de echivalență sunt fie identice, fie disjuncte)

Relații de ordine. Latice. Punct fix

Relații de ordine stricte și totale

O relație $\prec$ e o *ordine strictă* dacă e *ireflexivă* și *tranzitivă*

nu există x cu $x \prec x$

dacă $x \prec y$ și $y \prec z$ atunci $x \prec z$

Exemple: relațiile $<$ și $>$ între numere (întregi, reale, etc.)

Relația "descendent" între persoane

Relații de ordine stricte și totale

O relație $\prec$ e o *ordine strictă* dacă e *ireflexivă* și *tranzitivă*
nu există x cu $x \prec x$
dacă $x \prec y$ și $y \prec z$ atunci $x \prec z$

Exemple: relațiile $<$ și $>$ între numere (întregi, reale, etc.)
Relația "descendent" între persoane

O relație $\preceq$ e o *ordine totală* dacă e
reflexivă,
antisimetrică (dacă $x \preceq y$ și $y \preceq x$ atunci $x = y$),
tranzitivă, și în plus
oricare două elemente sunt *comparabile*,
adică pentru orice x, y avem $x \preceq y$ sau $y \preceq x$

Exemple: relațiile $\leq$ și $\geq$ între numere (întregi, reale, etc.)

Relații de ordine parțială

În practică apar adesea relații de ordine care nu sunt totale:
clasament pe grupe, dar nu și între grupe diferite
știm ordinea sosirii mesajelor, dar nu și ordinea trimiterii lor
în expresia $f(x) + g(x)$, f și g se apelează *înainte* de adunare,
dar nu știm dacă se evaluează întâi f sau g

O relație e o *ordine parțială* (non-strictă), dacă e
reflexivă, *antisimetrică* și *tranzitivă*

relația de divizibilitate între întregi
relația de incluziune $\subseteq$ pe mulțimea părților

Orice ordine totală e și o ordine parțială (dar nu și reciproc).

Orice ordine parțială induce o ordine strictă, și reciproc:

Definim: $a \prec b$ dacă $a \preceq b$ și $a \neq b$
Invers, definim $a \preceq b$ dacă $a \prec b$ sau $a = b$

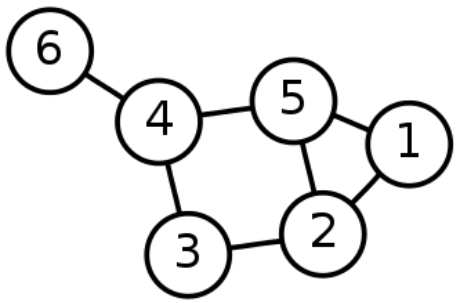
Noțiunea de punct fix

$x \in X$ e *punct fix* pentru funcția $f : X \to X$ dacă $f(x) = x$.
(privind f ca o transformare, ea nu îl modifică pe x)

Noțiunea de punct fix

$x \in X$ e *punct fix* pentru funcția $f : X \to X$ dacă $f(x) = x$.
(privind f ca o transformare, ea nu îl modifică pe x)

Exemplu: fie un graf $G = (V, E)$, și pentru $X \subseteq V$ funcția
$f(X) = X \cup \bigcup_{v \in X} vecini(v)$ (adăugăm la X toți vecinii)
$f(U) = U \Rightarrow$ din nodurile U, urmărind vecinii nu găsim noduri noi

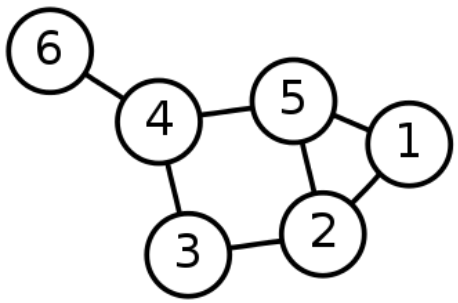


Pornind de la $S_0 = \{6\}$ calculăm $S_1 = f(S_0) = \{6, 4\}$, $S_2 = \{6, 4, 3, 5\}$, $S_3 = \{6, 4, 3, 5, 1, 2\}$, $S_4 = S_3$. Am atins un punct fix: avem toate nodurile care pot fi atinse din 6.

Noțiunea de punct fix

$x \in X$ e *punct fix* pentru funcția $f : X \to X$ dacă $f(x) = x$.
(privind f ca o transformare, ea nu îl modifică pe x)

Exemplu: fie un graf $G = (V, E)$, și pentru $X \subseteq V$ funcția
$f(X) = X \cup \bigcup_{v \in X} vecini(v)$ (adăugăm la X toți vecinii)
$f(U) = U \Rightarrow$ din nodurile U, urmărind vecinii nu găsim noduri noi



Pornind de la $S_0 = \{6\}$ calculăm $S_1 = f(S_0) = \{6, 4\}$, $S_2 = \{6, 4, 3, 5\}$, $S_3 = \{6, 4, 3, 5, 1, 2\}$, $S_4 = S_3$. Am atins un punct fix: avem toate nodurile care pot fi atinse din 6.

Multe prelucrări repetitive pot fi definite ca transformări care se opresc când atingem un *punct fix*

care sunt toate configurațiile posibile într-un joc?

care sunt toate variabilele de care depinde o variabilă dată? etc.

Existența unui punct fix e legată de *ordini parțiale* și *latice*.

Imagine: `https://upload.wikimedia.org/wikipedia/commons/5/5b/6n-graf.svg`

Latice

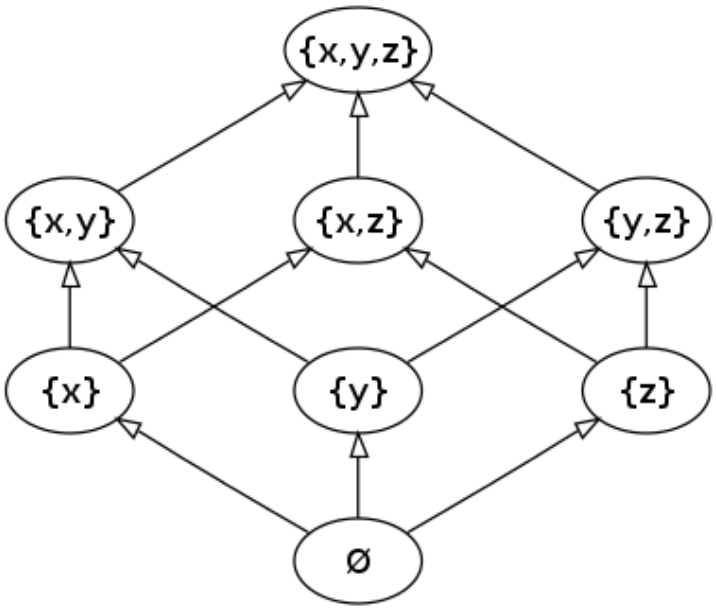
O *latice* e o mulțime *parțial ordonată*, în care orice două elemente au un *minorant* și un *majorant*.
(elemente mai mici, respectiv mai mari în ordine decât cele două).

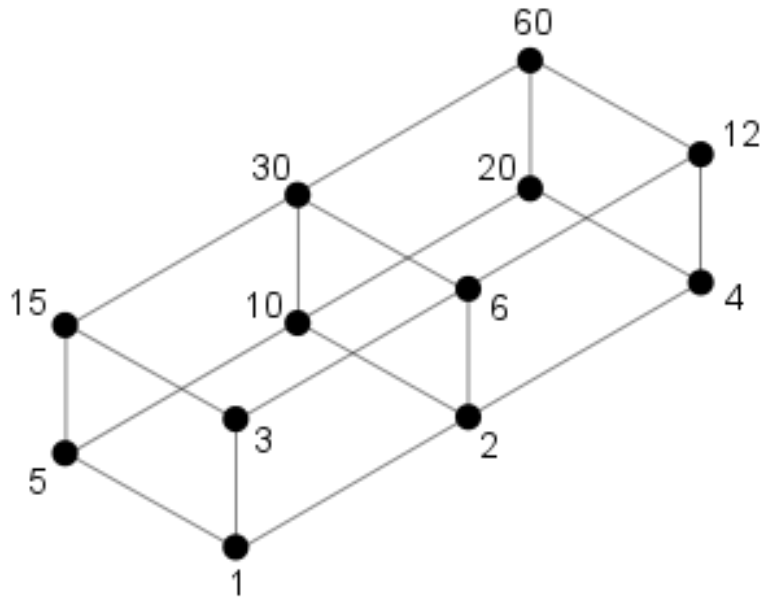
Latice

O *latice* e o mulțime *parțial ordonată*, în care orice două elemente au un *minorant* și un *majorant*.
(elemente mai mici, respectiv mai mari în ordine decât cele două).

Ex: mulțimea părților unei mulțimi (ordine: $\subseteq$; minor./maj.: $\cap$, $\cup$)

Ex: mulț. diviz. unui nr (ordine: $\vdots$, minor./maj.: *cmmdc*, *cmmmc*)





Aceste exemple sunt chiar *latice complete*.

Imagine: `http://en.wikipedia.org/wiki/File:Hasse_diagram_of_powerset_of_3.svg`

`http://en.wikipedia.org/wiki/File:Lattice_of_the_divisibility_of_60.svg`

Latice complete și puncte fixe

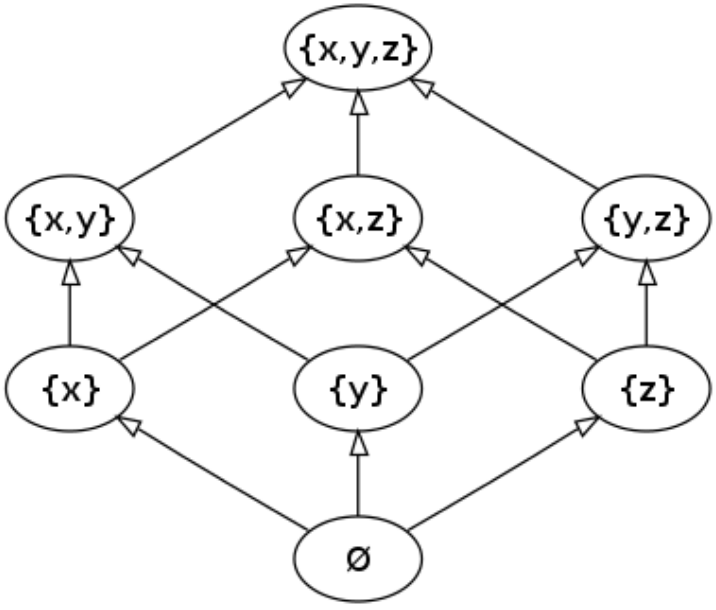
O latice L e *completă* dacă *orice* mulțime $S \subseteq L$ are un cel mai mic majorant (*supremum*) și un cel mai mare minorant (*infimum*).

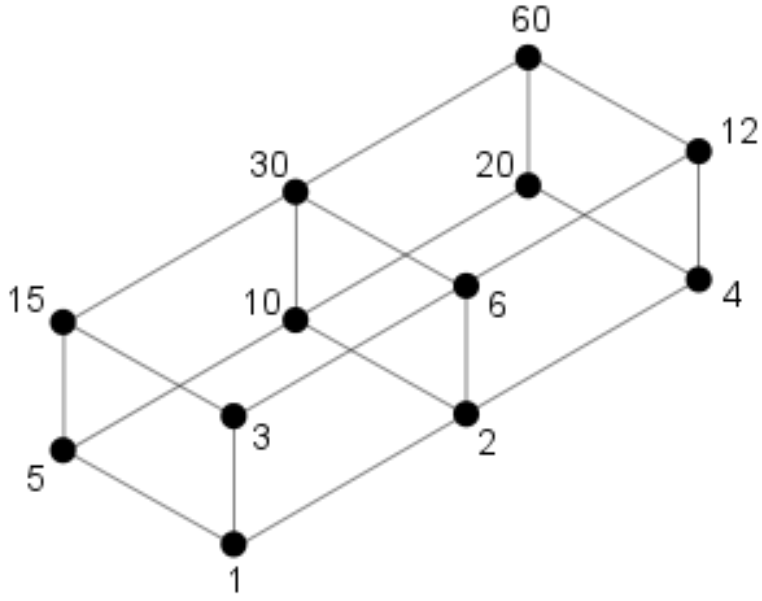
Latice complete și puncte fixe

O latice L e *completă* dacă *orice* mulțime $S \subseteq L$ are un cel mai mic majorant (*supremum*) și un cel mai mare minorant (*infimum*).

Sunt condiții mai puternice: pentru orice *submulțime* (și infinită); avem o *ordine* între majoranți/minoranți, și un *cel mai mic/mare*.

$\Rightarrow$ Luând $S = L$, rezultă că L are un element *minim* și unul *maxim*





Latice complete și puncte fixe

Teorema de punct fix (Knaster-Tarski):

Fie f o funcție *monotonă* pe o *latice completă*.

Atunci *mulțimea punctelor fixe* a lui f e tot o *latice completă*.

Latice complete și puncte fixe

Teorema de punct fix (Knaster-Tarski):
Fie f o funcție *monotonă* pe o *latice completă*.
Atunci *mulțimea punctelor fixe* a lui f e tot o *latice completă*.

Corolar:

O funcție monotonă pe o latice completă are
un *punct fix minimal* și un *punct fix maximal*.

se obțin pornind de la 0, f(0), f(f(0)), ... resp. M, f(M), f(f(M)), ...
unde 0 și M sunt elementul cel mai mic respectiv cel mai mare

Pentru orice x, șirul $x, f(x), f(f(x))), \ldots$ ajunge la un punct fix.

Compunerea de relatii. Închiderea tranzitivă

Inversa unei relații

Inversa unei relații $R \subseteq A \times B$ e relația

$R^{-1} \subseteq B \times A$,

cu $(y, x) \in R^{-1}$ dacă și numai dacă $(x, y) \in R$

$$R^{-1} = \{(y, x) \mid (x, y) \in R\}$$

Compunerea de relații

Fie două relații $R_1 \subseteq A \times B$ și $R_2 \subseteq B \times C$.

Compunerea $R_2 \circ R_1 \subseteq A \times C$ e relația

$$R_2 \circ R_1 = \{(x, z) \mid \text{ există } y \in B \mid (x, y) \in R_1 \text{ și } (y, z) \in R_2\}$$

Compunerea de relații

Fie două relații $R_1 \subseteq A \times B$ și $R_2 \subseteq B \times C$.

Compunerea $R_2 \circ R_1 \subseteq A \times C$ e relația

$$R_2 \circ R_1 = \{(x, z) \mid \text{ există } y \in B \mid (x, y) \in R_1 \text{ și } (y, z) \in R_2\}$$

La fel ca la funcții, scriem $R_2 \circ R_1$ și vedem că
pentru $x \in A$ găsim întâi $y \in B$ și apoi $z \in C$.

Se poate vedea simplu că $(R \circ S)^{-1} = S^{-1} \circ R^{-1}$

Compunerea de relații

Fie două relații $R_1 \subseteq A \times B$ și $R_2 \subseteq B \times C$.

Compunerea $R_2 \circ R_1 \subseteq A \times C$ e relația

$$R_2 \circ R_1 = \{(x, z) \mid \text{ există } y \in B \mid (x, y) \in R_1 \text{ și } (y, z) \in R_2\}$$

La fel ca la funcții, scriem $R_2 \circ R_1$ și vedem că
pentru $x \in A$ găsim întâi $y \in B$ și apoi $z \in C$.

Se poate vedea simplu că $(R \circ S)^{-1} = S^{-1} \circ R^{-1}$

Pentru o relație de echivalență R, $R = R^{-1}$

R e tranzitivă dacă și numai dacă $R \circ R \subseteq R$

Pentru o relație binară $R \subseteq A \times A$, se notează $R^2 = R \circ R$, etc.

Închiderea tranzitivă a unei relații

Dintr-o relație R, putem defini o nouă relație, prin "intermediari", ca în condiția de tranzitivitate.

Ex.: într-un graf, avem *muchii* și *drumuri* (relații între noduri):

Un drum e format din una sau mai multe muchii:

$drum(X, Y)$ dacă $muchie(X, Y)$
sau dacă $muchie(X, Z)$ și $muchie(Z, Y)$
sau dacă $muchie(X, Z)$ și $muchie(Z, U)$ și $muchie(U, Y)$ ...

Relația *drum* include relația *muchie* și e *tranzitivă*.

Închiderea tranzitivă a unei relații

Dintr-o relație R, putem defini o nouă relație, prin "intermediari", ca în condiția de tranzitivitate.

Ex.: într-un graf, avem *muchii* și *drumuri* (relații între noduri):

Un drum e format din una sau mai multe muchii:

drum(X, Y) dacă *muchie*(X, Y)
sau dacă *muchie*(X, Z) și *muchie*(Z, Y)
sau dacă *muchie*(X, Z) și *muchie*(Z, U) și *muchie*(U, Y) ...

Relația *drum* include relația *muchie* și e *tranzitivă*.

Sau, fie relația *copil(X, Y)* (X e copilul lui Y):

Definim relația *descendent*:

desc(X, Y) dacă *copil(X, Y)* (1)
desc(X, Z) dacă *desc(X, Y)* și *desc(Y, Z)* (2)

Relația *desc* include relația *copil* (1) și e tranzitivă (2).

Închiderea tranzitivă a unei relații

Închiderea tranzitivă a unei relații $R \subseteq A \times A$

e relația *tranzitivă minimală* R^+ astfel ca $R \subseteq R^+$

Putem calcula $R^+ = \bigcup_{k=1}^{\infty} R^k = R \cup R^2 \cup \ldots$

$R^2 = R \circ R$

$R^3 = R \circ R \circ R$

Închiderea tranzitivă (cont.)

Un exemplu (fără cicluri):
copil(ana, ion), *copil(lia, ion)*, *copil(ion, mara)*, *copil(mara, eva)*.

$copil^2$: *desc(ana, mara)*, *desc(lia, mara)*, *desc(ion, eva)*
$copil^3$: *desc(ana, eva)*, *desc(lia, eva)*.
Nu sunt descendenți de ordin > 3.
Deci, relația $desc = copil \cup copil^2 \cup copil^3$

Închiderea tranzitivă (cont.)

Un exemplu (fără cicluri):
copil(ana, ion), *copil(lia, ion)*, *copil(ion, mara)*, *copil(mara, eva)*.

$copil^2$: *desc(ana, mara)*, *desc(lia, mara)*, *desc(ion, eva)*
$copil^3$: *desc(ana, eva)*, *desc(lia, eva)*.
Nu sunt descendenți de ordin > 3.
Deci, relația $desc = copil \cup copil^2 \cup copil^3$

Putem defini $f(X) = R \cup (X \circ R)$.

Atunci $f(R) = R \cup R^2$ și prin inducție, $f^n(R) = \bigcup_{k=1}^{n+1} R^k$.

f e monotonă: dacă $X \subseteq Y$, $X \circ R \subseteq Y \circ R$ și $f(X) \subseteq f(Y)$.
Deci f are un punct fix minimal, tocmai închiderea tranzitivă R^+,
iar pentru o mulțime finită calculul are un număr finit de pași.

Relații și dicționare în ML

Dicționare: funcții parțiale în ML

Un *dicționar* (*asociere*) memorează *perechi* care asociază o *cheie* (primul element) cu o *valoare* (al doilea element)

Dicționare: funcții parțiale în ML

Un *dicționar* (*asociere*) memorează *perechi* care asociază
o *cheie* (primul element) cu o *valoare* (al doilea element)

ML: modulul `Map` e *parametrizat* (ca la mulțimi) cu un modul care
definește tipul (ordonat) al *cheilor*. *Nu* precizează tipul *valorilor*.

`module M = Map.Make(String)`
creează un modul pentru asocieri între
chei șiruri (`string`)
și un *tip valoare neprecizat* încă

Not.: `key`: tipul cheilor
`'a t`: tipul dicționar cu valori de tip `'a`

Funcții predefinite in modulul `Map`

`M.empty` dicționarul vid (nicio asociere)

```
M.add: key -> 'a-> 'a t -> 'a t
```

```
let m1 = M.add "x" 5 M.empty
```

ia o cheie, valoare și dicționar
creează un dicționar care are în plus noua asociere
(suprascrie eventuala veche asociere pentru cheie)

```
M.remove : key -> 'a t -> 'a t
```

```
let m2 = M.remove "y" m1
```

creează un nou dicționar fără cheia dată (fie că există sau nu)

Funcții predefinite in modulul `Map`

```
M.fold: (key -> 'a -> 'b -> 'b) -> 'a t -> 'b -> 'b
```

Ca la mulțimi, dar *elementul curent* parcurs e dat de *doi* parametri:

- *cheia* (`key`)
- *valoarea* (`'a`).

Produce o valoare arbitrară (tipul `'b`).

Funcții predefinite in modulul Map

```
M.fold: (key -> 'a -> 'b -> 'b) -> 'a t -> 'b -> 'b
```

Ca la mulțimi, dar *elementul curent* parcurs e dat de *doi* parametri:

- *cheia* (`key`)
- *valoarea* (`'a`).

Produce o valoare arbitrară (tipul `'b`).

Exemplu:

```
let m3 = M.singleton "x" 5 |> M.add "y" 3

M.fold (fun k v acc -> (k, v)::acc) m3 []
```

dă lista de perechi din dicționar: `[("x",5);("y",3)]`

Funcții predefinite in modulul Map

```
M.fold: (key -> 'a -> 'b -> 'b) -> 'a t -> 'b -> 'b
```

Ca la mulțimi, dar *elementul curent* parcurs e dat de *doi* parametri:

- *cheia* (`key`)
- *valoarea* (`'a`).

Produce o valoare arbitrară (tipul `'b`).

Exemplu:

```
let m3 = M.singleton "x" 5 |> M.add "y" 3

M.fold (fun k v acc -> (k, v)::acc) m3 []
```

dă lista de perechi din dicționar: `[("x",5);("y",3)]`

Există predefinită: `M.bindings m3 (* [("x",5);("y",3)] *)`

`M.bindings: 'a t -> (key * 'a) list`

Lucrul cu excepții

Căutăm valoarea asociată unei chei în dicționar:

`M.find "x" m` returnează întregul 5

`M.find "y" m` generează *excepția* `Not_found`

O *excepție* este o condiție specială care întrerupe calculul normal

Lucrul cu excepții

Căutăm valoarea asociată unei chei în dicționar:

```
M.find "x"  m      returnează întregul 5
M.find "y"  m      generează excepția Not_found
```

O *excepție* este o condiție specială care întrerupe calculul normal

Funcția semnalează că nu poate returna un rezultat

dacă nu e *tratată*, se abandonează execuția programului

altfel, codul de *tratare a excepției* stabilește ce se face

Lucrul cu excepții

Unele funcții standard *generează* excepții:

`List.hd []` produce `Exception: Failure "hd"`
`char_of_int 300` dă `Invalid_argument "char_of_int"`

Operații matematice pot genera excepții:

`1 / 0` produce `Exception: Division_by_zero`

Aceste excepții (primele 2 cu parametru șir, ultima fără parametru) și altele sunt *predefinite* în modulul `Pervasives` *deschis implicit*

Generarea de excepții

Putem *genera* excepții cu funcția `raise` (parametru: excepție):
`raise Not_found` sau `raise (Failure "gresit")` etc.

`failwith "mesaj"` e echivalent cu `raise (Failure "mesaj")`

`invalid_arg "msg"` e la fel cu `raise (Invalid_argument "msg")`

Tratarea excepțiilor

Trebuie să știm *ce excepții* pot genera funcțiile pe care le folosim și să le *tratăm* corespunzător

Sintaxa: `try` *expresie* e tot o formă de expresie
`with` *tipar*

unde *tipar* tratează una sau mai multe excepții și are forma

| *excepție-1* -> *expresie-1* (valoarea în acest caz)
| *excepție-2* -> *expresie-2* (valoarea în cazul 2)
...

Dacă *expresie* se evaluează normal, ea dă rezultatul;
altfel, dacă apare *excepție-k* se evaluează *expresie-k*

Pe *toate* ramurile, expresiile au *același tip* cu cea din `try` *expresie*

Tratarea excepțiilor

Dacă *expresie* se evaluează normal, ea dă rezultatul;
altfel, dacă apare *excepție-k* se evaluează *expresie-k*

Pe *toate* ramurile, expresiile au *același tip* cu cea din `try` *expresie*

```
fun x m -> try M.find x m
           with Not_found -> 0 (* pt. dict. cu val. int *)
```

Excepțiile se *propagă*, *terminând* fiecare funcție,
până când sunt "*prinse*" de un bloc de tratare
– altfel, programul e *abandonat*.

De la liste de asocieri la dicționare

E natural să construim un dicționar de la o listă de perechi.
Ea ar putea conține duplicate pentru primul element:

```
let lst = [("x", 5); ("y", 3); ("x", 2); ("a", 2)]
```

De la liste de asocieri la dicționare

E natural să construim un dicționar de la o listă de perechi.
Ea ar putea conține duplicate pentru primul element:

```
let lst = [("x", 5); ("y", 3); ("x", 2); ("a", 2)]
```

Putem să considerăm:

- *ultima* valoare (adăugăm necondiționat)

```
List.fold_left
  (fun dct (k,v) -> M.add k v dct) M.empty lst
```

De la liste de asocieri la dicționare

E natural să construim un dicționar de la o listă de perechi.
Ea ar putea conține duplicate pentru primul element:

```
let lst = [("x", 5); ("y", 3); ("x", 2); ("a", 2)]
```

Putem să considerăm:

- *ultima* valoare (adăugăm necondiționat)

```
List.fold_left
  (fun dct (k,v) -> M.add k v dct) M.empty lst
```

- *prima* valoare (adăugăm doar dacă nu există)

```
List.fold_left (fun dct (k,v) ->
  if M.mem k dct then dct
     else M.add k v dct) M.empty lst
```

De la liste de asocieri la dicționare

Dicționar de la o listă de perechi care poate conține chei duplicate:

```
let lst = [("x", 5); ("y", 3); ("x", 2); ("a", 2)]
```

- *toate* valorile: o *listă* sau *mulțime*

```
let addnew dict (k,v) =
  let lst = try M.find k dict
            with Not_found -> []
  in M.add k (v :: lst) dict;;

List.fold_left addnew M.empty lst
```

Relații cu ajutorul dicționarelor

Am văzut că o *relație* $R \subseteq A \times B$ poate fi privită ca o *funcție* $f_R : A \to \mathcal{P}(B)$ de la A la *mulțimea părților* lui B:

$$f_R(x) = \{y \in B \mid (x, y) \in R\}$$

Dicționarul va fi atunci de la A la *mulțimi* de elemente din B

Relații cu ajutorul dicționarelor

Am văzut că o *relație* $R \subseteq A \times B$ poate fi privită ca o *funcție* $f_R : A \to \mathcal{P}(B)$ de la A la *mulțimea părților* lui B:

$$f_R(x) = \{y \in B \mid (x, y) \in R\}$$

Dicționarul va fi atunci de la A la *mulțimi* de elemente din B

```
module M = Map.Make(String) (* dictionar pe siruri *)
module S = Set.Make(String) (* multimea de valori *)

let addpair m (x, y) =
  let oldset = try M.find x m (* multimea asociata cu x *)
  with Not_found -> S.empty (* nu e, deci multimea vida *)
  in M.add x (S.add y oldset) m
(* creeaza dictionar din lista *)
let setmap_of_pairs = List.fold_left addpair M.empty

setmap_of_pairs [("tms", "arad");("tms", "lugoj")];;
```

asociază "tms" cu mulțimea {"arad", "lugoj"}

Punctul fix în ML

Dacă șirul x, f(x), f(f(x)), ... atinge un punct fix, putem scrie:

```
let rec fix f x =
  let nxt = f x in
  if nxt = x then x else fix f nxt
```

`fix f x` compară $f(x)$ cu x. Dacă sunt egale, x e punct fix,
și e returnat. Dacă nu, reapelăm recursiv cu valoarea $f(x)$.

Apelul al n-lea va avea argumentul $f^{n-1}(x)$ și îl compară cu $f^n(x)$.
Dacă există n cu $f^{n-1}(x) = f^n(x)$, va fi găsit, $f^{n-1}(x)$ e punct fix.

Punctul fix în ML

Dacă șirul x, f(x), f(f(x)), ... atinge un punct fix, putem scrie:

```
let rec fix f x =
  let nxt = f x in
  if nxt = x then x else fix f nxt
```

`fix f x` compară $f(x)$ cu x. Dacă sunt egale, x e punct fix,
și e returnat. Dacă nu, reapelăm recursiv cu valoarea $f(x)$.

Apelul al n-lea va avea argumentul $f^{n-1}(x)$ și îl compară cu $f^n(x)$.
Dacă există n cu $f^{n-1}(x) = f^n(x)$, va fi găsit, $f^{n-1}(x)$ e punct fix.

Putem rescrie, folosind o funcție ajutătoare cu doar un parametru
(nu mai trebuie repetat la apel parametrul `f`):

```
let fix f =
  let rec fix1 x =
    let nxt = f x in
    if nxt = x then x else fix1 nxt
  in fix1
```