Analisi statica e verifica del software

Simone Colli, Manuel Di Agostino
simone.colli@studenti.unipr.it, manuel.diagostino@studenti.unipr.it

Appunti del corso tenuto dal **Prof. Vincenzo Arceri**

Università degli Studi di Parma
Anno Accademico 2025/2026

Indice

1 Introduzione

1.1 Cos'è l'affidabilità del software?

Definizione 1.1: Affidabilità del Software (IEEE 610.121990)

"The ability of a system or component to perform its required functions under stated conditions for a specified period of time"
Ovvero:
"La capacità di un sistema o componente di eseguire le funzioni richieste in condizioni specificate per un periodo di tempo specificato"

Definizione 1.2: Gestione dell'Affidabilità del Software (IEEE 982.11988)

"The process of optimizing the reliability of software through a program that emphasizes software error prevention, fault detection and removal, and the use of measurements to maximize reliability in light of project constraints such as resources, schedule and performance"
Ovvero:
"Il processo di ottimizzazione dell'affidabilità del software attraverso un programma che enfatizza la prevenzione degli errori software, il rilevamento e la rimozione dei guasti, e l'uso di misurazioni per massimizzare l'affidabilità alla luce dei vincoli del progetto come risorse, tempi e prestazioni"

Sfruttando le definizioni Definizione 1.1 e Definizione 1.2, è possibile riassumere che l'affidabilità del software consiste in 3 attività:

- Prevenzione degli errori.
- Rilevamento e rimozione dei guasti.
- Valutazione per massimizzare l'affidabilità, in particolare valutazioni a supporto delle prime due attività.

1.2 Perché l'affidabilità del software è importante?

L'importanza dell'affidabilità del software deriva dalle enormi conseguenze economiche e operative che i bug possono causare.

Alcune stime indicano che:

- I bug nei software costano circa 60 miliardi di dollari all'anno negli Stati Uniti.
- L'economia mondiale perde circa 250 miliardi di dollari all'anno a causa di qualsiasi tipo di attacco palese (overt attack).
- I difetti nel software rendono la programmazione molto dolorosa.

Alcuni esempi di fallimenti dovuti a bug software includono:

- Il disastro del volo 501 di Ariane 5 nel 1996, che è esploso dopo 37 secondi dal decollo a causa di un errore di overflow durante la conversione da un float 64 bit ad un intero 16 bit. Questo evento ha causato una perdita di circa 370.000.000 dollari.
- Il bug del Pentium FDIV nel 1994, che ha causato errori di calcolo nelle divisioni in virgola mobile. Questo bug ha portato a una perdita di circa 475.000.000 dollari.
- L'aggiornamento difettoso di CrowdStrike nel 2024, che ha causato più di 5000 voli cancellati. Questo evento ha impattato su banche, governi e infrastrutture critiche. Le perdite economiche sono state stimate a circa 5.400.000.000 dollari.

La lista di esempi potrebbe continuare, ma l'importante è comprendere che l'affidabilità del software è cruciale per evitare perdite economiche significative e garantire il corretto funzionamento dei sistemi.

1.3 Validazione e Verifica

La validazione e la verifica sono due processi distinti ma complementari nell'ambito dello sviluppo del software, entrambi mirati a garantire che il software soddisfi i requisiti e funzioni correttamente.

La **validazione** si concentra sulla domanda: "Stiamo costruendo il prodotto giusto?". Essa verifica che il software soddisfi le esigenze e le aspettative degli utenti finali. La **verifica**, d'altra parte, si concentra sulla domanda: "Stiamo costruendo il prodotto nel modo giusto?". Essa assicura che il software sia sviluppato correttamente secondo le specifiche tecniche e i requisiti di progetto.

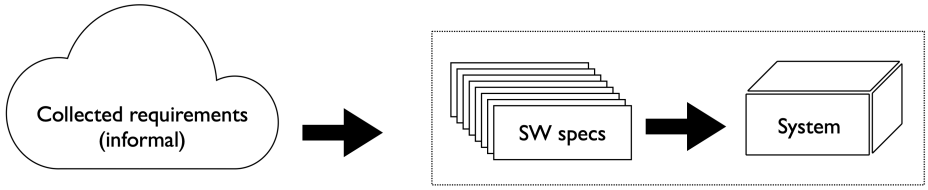


Figura 1: L'immagine illustra la differenza tra validazione e verifica.

Esempio 1.1: Risposta dell'ascensore

Un esempio pratico di validazione e verifica può essere trovato nel comportamento di un sistema di risposta dell'ascensore.
Una specifica validabile ma non verificabile potrebbe essere: "Se un utente preme il pulsante di richiesta a un piano i, un ascensore disponibile deve arrivare al piano i entro un tempo ragionevole".
Una specifica verificabile è: "Se un utente preme il pulsante di richiesta a un piano i, un ascensore disponibile deve arrivare al piano i entro 30 minuti".

1.4 Verifica del Software

Lo scopo della verifica del software è verificare alcune affermazioni di correttezza riguardante un programma:

- La **correttezza funzionale**, ovvero che il programma soddisfi i requisiti funzionali specificati.
- L'**assenza di errori non funzionali** (non cosa fa il programma, ma come lo fa), come ad esempio: soddisfacimento di vincoli spaziali e/o temporali, assenza di errori a runtime, soddisfacimento di vincoli di sicurezza, ecc.

Bad news: theres no silver bullet

Non esiste un metodo di verifica che sia contemporaneamente:

- **Automatico**: non richiede interazione umana.
- **Potente**: è in grado di dimostrare proprietà non banali (*non-trivial properties*).
- **Corretto (Sound)**: non dimostra mai una proprietà se questa non è vera.
- **Completo (Complete)**: dimostra sempre la proprietà se essa è vera e non fallisce mai nel dichiarare la correttezza di un programma corretto.

Il **teorema di Rice** afferma che tutte le proprietà non banali del comportamento di un programma scritto in un linguaggio di programmazione Turing-completo sono indecidibili. L'**indecidibilità** di una proprietà implica che non esiste alcun metodo di verifica automatico che sia allo stesso tempo **corretto** e **completo**.

2 Background matematico

Per poter discutere in modo rigoroso di tecniche di verifica del software, è necessario introdurre alcuni concetti matematici di base relativi alla teoria della computazione e alla logica formale.

2.1 Set notation

Definizione 2.1: Insieme

Un insieme (set) è una collezione di oggetti ben definiti e distinti. La collezione stessa è considerata un oggetto a sé stante.
Dato un insieme S è possibile dire che:

- $s \in S$ significa che l'elemento s appartiene all'insieme S.
- $S_1 \subseteq S_2 \triangleq \forall s \in S_1 \Rightarrow s \in S_2$ significa che l'insieme S_1 è un sottoinsieme di S_2 se ogni elemento di S_1 appartiene anche a S_2.
- $S_1 \cup S_2 = \{s | s \in S_1 \vee s \in S_2\}$ è l'unione di due insiemi, ovvero l'insieme di tutti gli elementi che appartengono ad almeno uno dei due insiemi.
- $S_1 \cap S_2 = \{s | s \in S_1 \wedge s \in S_2\}$ è l'intersezione di due insiemi, ovvero l'insieme di tutti gli elementi che appartengono ad entrambi gli insiemi.

2.2 Partial order

Definizione 2.2: Ordine parziale

Un ordine parziale (partial order) è una relazione binaria $\sqsubseteq$ su un insieme X che soddisfa le seguenti proprietà:

- Riflessività: $\forall x \in X \Rightarrow x \sqsubseteq x$, indica che ogni elemento è in relazione con se stesso.
- Anti-simmetria: $\forall x, y \in X, (x \sqsubseteq y \wedge y \sqsubseteq x) \Rightarrow x = y$, indica che se un elemento x è in relazione con un altro elemento y e viceversa, allora x e y sono lo stesso elemento.
- Transitività: $\forall x, y, z \in X, (x \sqsubseteq y \wedge y \sqsubseteq z) \Rightarrow x \sqsubseteq z$, indica che se un elemento x è in relazione con un elemento y, e y è in relazione con un elemento z, allora x è in relazione con z.

Esempio 2.1: Esempio informale di ordine parziale su $\mathbb{Z}$

Informalmente è possibile considerare l'insieme dei numeri interi $\mathbb{Z}$ con la relazione "minore o uguale di" ($\leq$) come un esempio di ordine parziale. In questo caso è possibile verificare che:

- Riflessività: Per ogni numero intero i, vale che $i \leq i$.
- Anti-simmetria: Se $i_1 \leq i_2$ e $i_2 \leq i_1$, allora $i_1 = i_2$.

- Transitività: Se $i_1 \leq i_2$ e $i_2 \leq i_3$, allora $i_1 \leq i_3$.

2.3 Powerset

Definizione 2.3: Insieme delle parti

Dato un insieme S, definito in Definizione 2.1, l'insieme delle parti (powerset) di S è l'insieme di tutti i sottoinsiemi di S, ed è indicato con $\mathcal{P}(S)$.
Dato un insieme S con $|S| = n$ elementi è possibile determinare che l'insieme delle parti di S ha $|\mathcal{P}(S)| = 2^n$ elementi.

Esempio 2.2: Alcuni esempi di powerset

Dato l'insieme vuoto $\emptyset$, il suo powerset è:

$$\mathcal{P}(\emptyset) = \{\emptyset\}$$

ed ha $2^0 = 1$ elemento.
Dato un insieme $X = \{a, b\}$, il suo powerset è:

$$\mathcal{P}(X) = \{\emptyset, \{a\}, \{b\}, \{a, b\}\}$$

ed ha $2^2 = 4$ elementi.
Dato l'insieme $\mathbb{Z}$ dei numeri interi, il suo powerset è:

$$\mathcal{P}(\mathbb{Z}) = \{\emptyset, \{\ldots, -2, -1, 0, 1, 2, \ldots\}, \{0\}, \{1, 2, 3\}, \ldots\}$$

ed ha un numero infinito di elementi.

Dato un powerset è possibile definire un ordine parziale $\subseteq$ su di esso e verificare che esso soddisfa le proprietà di un ordine parziale:

- **Riflessività:** $\forall X_1 \in \mathcal{P}(X).X_1 \subseteq X_1$
- **Anti-simmetria:** $\forall X_1, X_2 \in \mathcal{P}(X).X_1 \subseteq X_2 \wedge X_2 \subseteq X_1 \Rightarrow X_1 = X_2$
- **Transitività:** $\forall X_1, X_2, X_3 \in \mathcal{P}(X).X_1 \subseteq X_2 \wedge X_2 \subseteq X_3 \Rightarrow X_1 \subseteq X_3$

Esercizio 2.1: Inclusione inversa su powerset

L'inclusione inversa su un powerset $\mathcal{P}(X)$ è un ordine parziale? Per verificare se $\supseteq$ è un ordine parziale su $\mathcal{P}(X)$, è necessario verificare se soddisfa le tre proprietà di un ordine parziale.

- **Riflessività:** $\forall X_1 \in \mathcal{P}(X) \Rightarrow X_1 \supseteq X_1$, indica che ogni insieme è sovrainsieme di se stesso.
- **Anti-simmetria:** $\forall X_1, X_2 \in \mathcal{P}(X), (X_1 \supseteq X_2 \wedge X_2 \supseteq X_1) \Rightarrow X_1 = X_2$, indica che se ogni elemento di X_1 è anche in X_2 e che ogni elemento di X_2 è anche in X_1, allora X_1 e

X_2 sono lo stesso insieme.

- **Transitività:** $\forall X_1, X_2, X_3 \in \mathcal{P}(X), (X_1 \supseteq X_2 \land X_2 \supseteq X_3) \Rightarrow X_1 \supseteq X_3$, indica che se X_1 è un sovrainsieme di X_2, e X_2 è un sovrainsieme di X_3, allora X_1 è un sovrainsieme di X_3.

Poiché tutte e tre le proprietà sono soddisfatte, è possibile concludere che l'inclusione inversa $\supseteq$ è un ordine parziale su il powerset $\mathcal{P}(X)$. Quindi $\langle X, \supseteq \rangle$ è un poset.

2.4 Diagramma di Hasse

Il diagramma di Hasse è una rappresentazione grafica di un poset.

Sia $X = \{x, y, z\}$ un insieme (Definizione 2.1), ed un poset $\langle X, \sqsubseteq \rangle$ su di esso è possibile rappresentarlo tramite un diagramma di Hasse dove una linea che connette x e y indica che:

- $x \subseteq y$
- $\nexists z \in X$ tale che $x \subseteq z \subseteq y$

È possibile trovare la presenza di più livelli nel diagramma di Hasse, dove un elemento x è posizionato più in alto di un elemento y se x è "immediatamente maggiore" di y.

Nota 2.1: Poset inverso

Un poset inverso è un poset ottenuto invertendo la relazione di ordine di un poset originale. Graficamente, questo si traduce nel riflettere il diagramma di Hasse rispetto ad un asse orizzontale, ovvero ruotando di 180 gradi il diagramma.

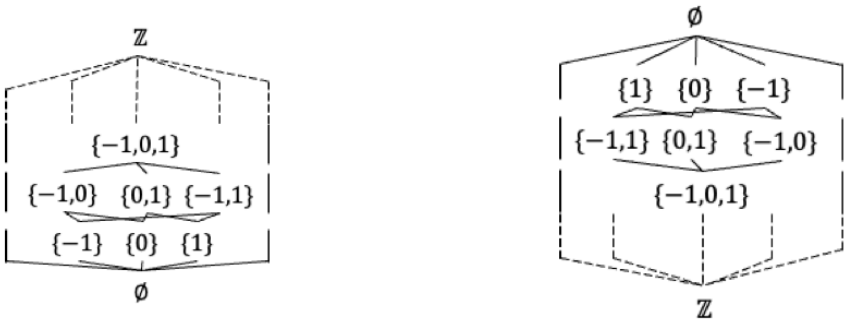


Figura 2: L'immagine illustra un esempio di diagramma di Hasse per un poset e il suo poset inverso.

2.5 Upper e lower bounds

Definizione 2.4: Upper bound and least upper bound

Dato un poset $\langle X, \sqsubseteq \rangle$ e un sottoinsieme $Y \subseteq X$ è possibile dire che:

1. $y \in X$ è un upperbound di Y se $\forall y' \in Y, y' \sqsubseteq y$, ovvero se ogni elemento dell'insieme Y è minore o uguale a y.
2. y è l'upperbound più piccolo tra tutti gli opperbound di Y, ovvero $\forall y' \in UB(Y).y \sqsubseteq y'$, dove $UB(Y)$ è l'insieme di tutti gli upperbound di Y.

Definizione 2.5: Lower bound e greatest lower bound

Dato un poset $\langle X, \sqsubseteq \rangle$ e un sottoinsieme $Y \subseteq X$ è possibile dire che:

1. $y \in X$ è un lowerbound di Y se $\forall y' \in Y, y \sqsubseteq y'$, ovvero se ogni elemento dell'insieme Y è maggiore o uguale a y.
2. y è il lowerbound più grande tra tutti i lowerbound di Y, ovvero $\forall y' \in LB(Y).y' \sqsubseteq y$, dove $LB(Y)$ è l'insieme di tutti i lowerbound di Y.

Esercizio 2.2: Lub e glb su powerset

Dato il Poset $\langle \mathcal{P}(X), \subseteq \rangle$, e siano $S_1, S_2 \in \mathcal{P}(X)$:

1. $S_1 \cup S_2$ è l'estremo superiore (lub) di $\{S_1, S_2\}$?
2. $S_1 \cap S_2$ è l'estremo inferiore (glb) di $\{S_1, S_2\}$?

Soluzione esercizio 1
$S_1 \cup S_2$ è l'estremo superiore (lub) di $\{S_1, S_2\}$?
Sia $L = S_1 \cup S_2$, che per essere lub deve soddisfare le condizioni elencate in Definizione 2.4 punto 1 e 2.
Punto 1:
$S_1 \subseteq L$ e $S_2 \subseteq L$.

Punto 2:
Sia U un qualsiasi insieme tale che $S_1 \subseteq U$ e $S_2 \subseteq U$. Allora U deve per contenere gli elementi di S_1 e S_2 deve essere proprio $(S_1 \cup S_2) \subseteq U$.

Soluzione esercizio 2
$S_1 \cap S_2$ è l'estremo inferiore (glb) di $\{S_1, S_2\}$?

2.6 Supremum e Infimum

Definizione 2.6: Supremum

Dato un poset $\langle X, \sqsubseteq \rangle$ e un sottoinsieme $S \subseteq X$ il supremum (o least upper bound) di S è l'elemento più piccolo tra tutti gli upper bound di S.

Definizione 2.7: Infimum

Dato un poset $\langle X, \sqsubseteq \rangle$ e un sottoinsieme $S \subseteq X$ l'infimum (o greatest lower bound) di S è l'elemento più grande tra tutti i lower bound di S.

2.7 Proprietà di lub e glb

Proposizione 2.1: Proprietà di lub e glb

Dato un poset $\langle X, \sqsubseteq \rangle$ è possibile definire:

- $\sqcup$ come il lub su X.
- $\sqcap$ come il glb su X.
- se $\sqcup$ e $\sqcap$ esistono sono unici.
- $\sqcup X$ esiste se e solo se X ha un elemento $\top$ ($\sqcup X = \top$).
- $\sqcap X$ esiste se e solo se X ha un elemento $\bot$ ($\sqcap X = \bot$).

2.8 Lattice

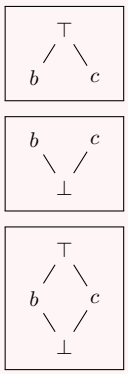
Definizione 2.8: Lattice

Dato un poset $\langle X, \sqsubseteq \rangle$, esso è un reticolo (lattice) se soddisfa entrambe le seguenti proprietà:

- $\forall x, y \in X, \exists x \sqcup y$, ovvero per ogni coppia di elementi qualsiasi di X deve esistere il loro lub. Un poset che soddisfa questa proprietà è detto "join semi lattice".
- $\forall x, y \in X, \exists x \sqcap y$, ovvero per ogni coppia di elementi qualsiasi di X deve esistere il loro glb. Un poset che soddisfa questa proprietà è detto "meet semi lattice".

In un reticolo è presente la relazione d'ordine parziale $\sqsubseteq$ su due elementi $x, y \in X$, ovvero $x \sqsubseteq y$ se e solo se:

- $x \sqcup y = y$
- $x \sqcap y = x$



Join semi-lattice

Meet semi-lattice

Lattice

Figura 3: Rappresentazione grafica di Join semi-lattice, Meet semi-lattice e Lattice.

2.9 Set lattice

Definizione 2.9: Set lattice

Le operazioni insiemistiche definiscono una struttura di reticolo. Dato il poset $\langle \mathcal{P}(X), \subseteq \rangle$, esso forma un reticolo rispetto alle operazioni di unione ($\cup$) e intersezione ($\cap$), denotato come:

$$\langle \mathcal{P}(X), \subseteq, \cup, \cap \rangle$$

Se l'insieme X è finito, allora il reticolo ha altezza finita.

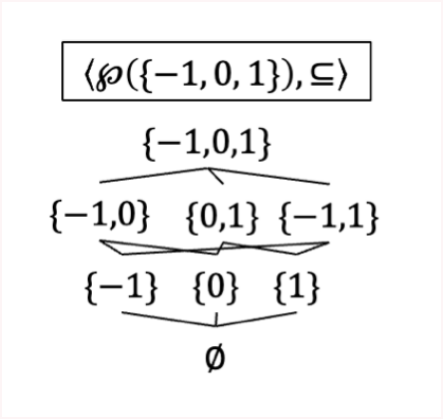


Figura 4: L'immagine illustra un esempio di reticolo su un poset finito.

Nota 2.2: Top e Bottom nei reticoli

Un reticolo non possiede necessariamente un elemento Top ($\top$) o un elemento Bottom ($\bot$). Ad esempio, il reticolo $\langle \mathbb{Z}, \leq, \max, \min \rangle$ non ha né un elemento massimo né un elemento minimo, poiché l'insieme dei numeri interi è illimitato in entrambe le direzioni.

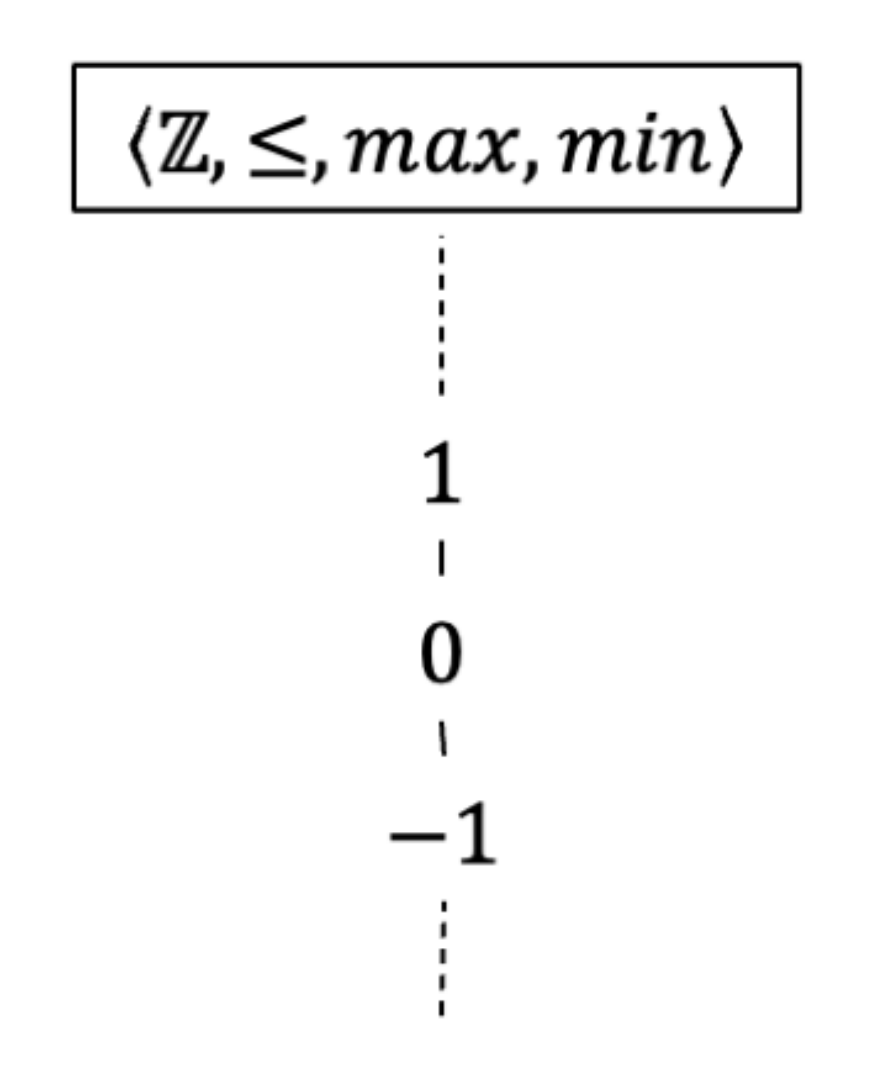


Figura 5: L'immagine illustra il reticolo su $\mathbb{Z}$, privo di elementi top e bottom.

2.10 Complete lattice

Definizione 2.10: Complete lattice

Un reticolo $\langle X, \sqsubseteq, \sqcup, \sqcap \rangle$ è detto completo se:

- Ogni sottoinsieme $Y \subseteq X$ ha un lub in X.
- È presente un elemento bottom $\bot$ in X.

Esempio 2.3: Esempi di lattice completo e non completo

Considerando l'insieme dei numeri interi $\mathbb{Z}$ con la relazione "minore o uguale di" ($\leq$), si può osservare che

$$\langle \mathbb{Z}, \leq, \max, \min \rangle$$

è un reticolo ma non è completo. Infatti, non esiste un elemento top ($\top$) in $\mathbb{Z}$, poiché non esiste un numero intero massimo.
Se all'insieme dei numeri interi si aggiungono gli elementi $+\infty$ e $-\infty$, si ottiene l'insieme esteso $\mathbb{Z} \cup \{+\infty, -\infty\}$. In questo caso, il reticolo

$$\langle \mathbb{Z} \cup \{+\infty, -\infty\}, \leq, \max, \min \rangle$$

diventa un reticolo completo, poiché ora esistono sia un elemento top ($+\infty$) che un elemento bottom ($-\infty$).

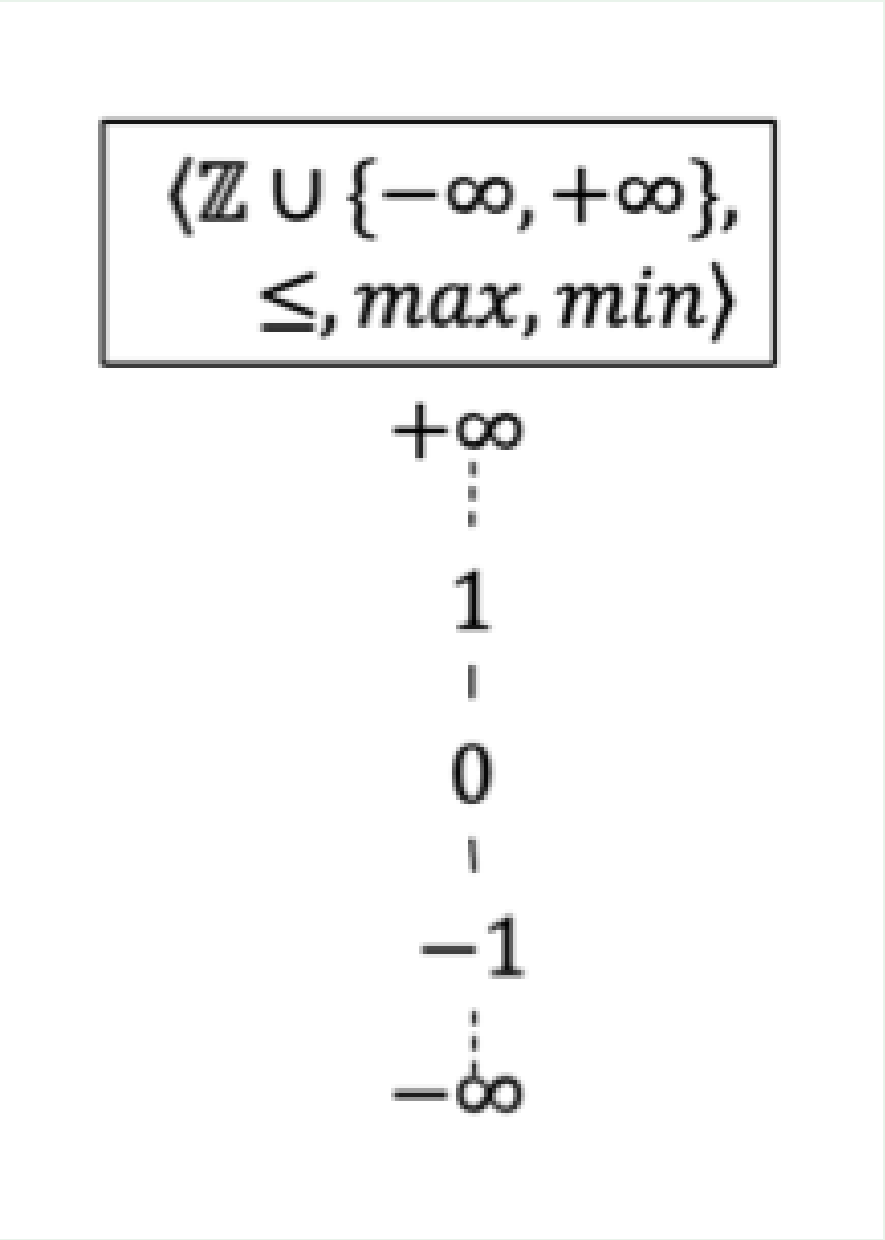


Figura 6: L'immagine illustra il reticolo su $\mathbb{Z} \cup \{+\infty, -\infty\}$.

Proposizione 2.2: Proprietà di un reticolo completo

Un reticolo completo possiede le seguenti proprietà:

- reticolo completo $\langle X, \sqsubseteq, \sqcup, \sqcap \rangle$ ha un elemento top $\top$ e un elemento bottom $\bot$ e si denota come
$$\langle X, \sqsubseteq, \top, \bot, \sqcup, \sqcap \rangle$$
- I reticoli finiti sono completi.
- $\sqcup$ induce $\sqcap : \sqcap S = \sqcup \{y | \forall x \in S . y \sqsubseteq x\}$
- $\sqcap$ induce $\sqcup : \sqcup S = \sqcap \{y | \forall x \in S . x \sqsubseteq y\}$

2.11 Relations

Definizione 2.11: Prodotto cartesiano

Dati due insiemi(Definizione 2.1) X e Y, il loro prodotto cartesiano $X \times Y$ è l'insieme di tutte le possibili coppie ordinate (x, y) dove il primo elemento appartiene a X e il secondo a Y .

$$X \times Y = \{(x, y) \mid x \in X \wedge y \in Y\}$$

Definizione 2.12: Relazione

Una relazione R tra due insiemi X e Y è definita come un sottoinsieme del loro prodotto cartesiano (Definizione 2.11):
$$R \subseteq X \times Y$$

Nota 2.3: Notazione per le relazioni

Dato una relazione $R \subseteq X \times Y$, è possibile indicare che un elemento $x \in X$ è in relazione con un elemento $y \in Y$ tramite due notazioni:

- $(x, y) \in R$, ovvero tramite relazione insiemistica.
- $x \; R \; y$, ovvero tramite notazione infissa.

Nota 2.4: Ordine parziale come relazione

Un ordine parziale $\sqsubseteq$ (Definizione 2.2) è un caso particolare di relazione definita sullo stesso insieme X. Formalmente, $\sqsubseteq$ è un sottoinsieme del prodotto cartesiano $X \times X$ ($\sqsubseteq \subseteq X \times X$). A differenza di una relazione generica, l'ordine parziale deve soddisfare le proprietà di riflessività, anti-simmetria e transitività (Definizione 2.2).

2.12 Functions (or maps)

Le funzioni (functions), sono un tipo particolare di relazione, e sono fondamentali per rioscire a modellare gli stati di un programma.

Definizione 2.13: Funzione

Una funzione f tra due insiemi X e Y è un particolare tipo di relazione $R \subseteq X \times Y$ che soddisfa la proprietà di unicità dell'immagine.
Formalmente, per ogni elemento x nel dominio, esiste al massimo un elemento y nel codominio tale che $(x, y) \in f$:

$$\forall (x, y) \in f, \nexists (x', y') \in f.(x = x' \land y \neq y')$$

Una funzione è definita al massimo una volta su un dato input.

Esempio 2.4: Cerchio su un piano cartesiano

Un cerchio su un piano cartesiano non rappresenta una funzione, poiché un singolo valore di x può essere associato a due valori distinti di y (uno sopra l'asse x e uno sotto). Questo viola la proprietà di unicità dell'immagine definita per le funzioni.

Nota 2.5: Non tutte le relazioni sono funzioni

Un ordine parziale (Definizione 2.2) non è generalmente una funzione, poiché un elemento può essere minore o uguale a molti altri elementi (uno-a-molti).

Definizione 2.14: Notazione delle funzioni

Dato $f : X \to Y$, dove X è il dominio e Y è il codominio:

- Notazione estensionale: Una funzione definita su punti specifici si scrive come:

  $$f = [x_0 \mapsto y_0, x_1 \mapsto y_1, \ldots, x_n \mapsto y_n]$$

  che è equivalente all'insieme di coppie $\{(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)\}$.

- Lambda notazione: Si utilizza $\lambda x.espressione$ per definire una funzione anonima. Ad esempio $\lambda x.f(x) \equiv f$.

2.12.1 Monotonia, Embedding e Isomorfismo

Dati due posets $\langle X, \sqsubseteq_X \rangle$ e $\langle Y, \sqsubseteq_Y \rangle$, e una funzione $f : X \to Y$, è possibile dire che:

- La funzione f è monotona se $x_1 \sqsubseteq_X x_2 \Rightarrow f(x_1) \sqsubseteq_Y f(x_2)$.
- La funzione f è un embedding ordinato se $x_1 \sqsubseteq_X x_2 \iff f(x_1) \sqsubseteq_Y f(x_2)$.
- La funzione f è un isomorfa se è un embedding ordinato ed è suriettiva, ovvero: $\forall y \in Y, \exists x \in X.f(x) = y$.

Esempio 2.5: Esempio di funzione monotona, embedding ordinato e suriettiva

Data $f : \langle \mathbb{Z}, \leq \rangle \to \langle \mathbb{Z}, \leq \rangle$, definita come $f(x) = x + 1$, è possibile osservare che:

- La funzione è monotona.
- La funzione è un embedding ordinato.
- La funzione è suriettiva.

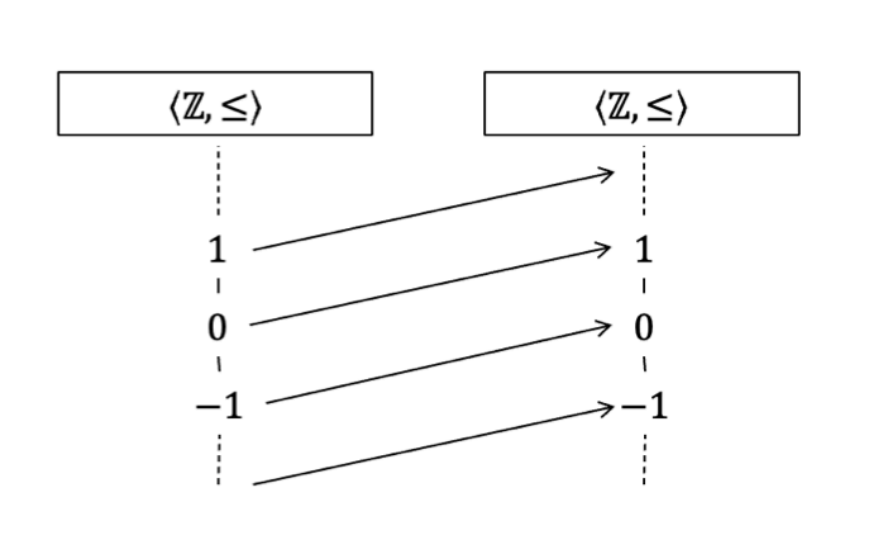


Figura 7: L'immagine illustra la funzione $f(x) = x + 1$ come monotona, embedding ordinato e suriettiva.

Esempio 2.6: Esempio di funzione monotona ma che non è un embedding

Data $f : \langle \mathcal{P}(\mathbb{Z}), \subseteq \rangle \to \langle \mathbb{Z}, \leq \rangle$, definita come $f(X) = \max(X)$, è possibile osservare che:

- La funzione è monotona.
- La funzione non è un embedding.

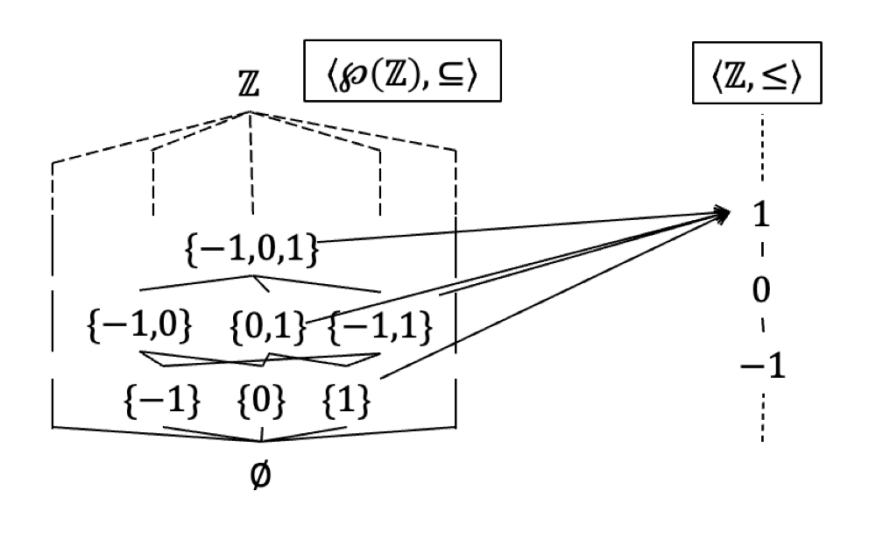


Figura 8: L'immagine illustra la funzione $f(X) = \max(X)$ come monotona ma non un embedding.

2.12.2 Funzioni che preservano join e meet

Dati due reticoli $\langle X, \sqsubseteq_X, \sqcup_X, \sqcap_X \rangle$ e $\langle Y, \sqsubseteq_Y, \sqcup_Y, \sqcap_Y \rangle$, e una funzione $f : X \to Y$, è possibile dire che:

- La funzione f preserva i join se $f(x_1 \sqcup_X x_2) = f(x_1) \sqcup_Y f(x_2)$.
- La funzione f preserva i meet se $f(x_1 \sqcap_X x_2) = f(x_1) \sqcap_Y f(x_2)$.

Nota 2.6: Mantenimento della Monotonia

Una funzione che preserva i join o i meet è monotona, ma una funzione che è monotona non necessariamente preserva i join o i meet.

2.13 Chains

2.14 Fixpoint

3 Dominio degli intervalli

Il dominio degli intervalli è possibile definirlo come:

$$Intv = \{[a,b] \mid a \in \mathbb{Z} \cup \{-\infty\}, b \in \mathbb{Z} \cup \{\infty\}, a \leq b\} \cup \{\bot\}$$

L'elemento top del dominio è definito come:

$$\top_{Intv} = [-\infty, \infty]$$

L'elemento bottom del dominio viene introdotto e definito come elemento a se stante:

$$\bot_{Intv}$$

È possibile visualizzare il dominio degli intervalli come un reticolo:
Il reticolo su questo dominio può essere definito come segue:

$$\langle Intv, \sqsubseteq, \sqcup, \sqcap, \bot_{Intv}, \top_{Intv} \rangle$$

dove le operazioni di ordine parziale, unione e intersezione sono definite come segue:

- $[a,b] \sqsubseteq [c,d] \iff a \geq c \wedge b \leq d$
- $[a,b] \sqcup [c,d] = [\min(a,c), \max(b,d)]$. Questo non è sempre esatto.
- $[a,b] \sqcap [c,d] = \begin{cases} [\max(a,c), \min(b,d)] & \text{se } \max(a,c) \leq \min(b,d) \\ \bot & \text{altrimenti} \end{cases}$

Nota 3.1: Reticolo completo

Questo reticolo è completo, in quanto ogni sottoinsieme di intervalli ha sia un estremo superiore che un estremo inferiore.

Un reticolo sul dominio degli intervalli può avere le funzioni di concretizzazione:

- $\gamma(\bot) = \emptyset$
- $\gamma([a,b]) = \{n \in \mathbb{Z} \mid n \in [a,b]\}$

La funzione di astrazione può essere definita come:

$$\alpha(X) = \begin{cases} \bot & \text{se } X = \emptyset \\ [\min(X), \max(X)] & \text{altrimenti} \end{cases}$$

Dove $X \subseteq \mathbb{Z}$.

Nota 3.2: Su quali insieme lavora l'astrazione?

La funzione di astrazione lavora su $\mathbb{Z}, \mathbb{R}$ ma non su $\mathbb{Q}$ perché non garantisce la best abstraction.

3.1 Semantica astratta

La semantica astratta per le espressioni aritmetiche prende una memoria e ritorna un intervallo è definita come: $[[\dot{]}]^\# : \mathbb{M}^\# \to Intv$ Quindi:

- $[[c]]^\# m^\# = [n, n]$
- $[[e_1]]^\# m^\# = [a, b]$
- $[[e_2]]^\# m^\# = [c, d]$
- $-[[e_1]]^\# m^\# = [-b, -a]$
- $[[e_1 + e_2]]^\# m^\# = [a + c, b + d]$
- $[[e_1 - e_2]]^\# m^\# = [a - d, b - c]$
- $[[e_1 * e_2]]^\# m^\# = [\min(ac, ad, bc, bd), \max(ac, ad, bc, bd)]$
- $$[[e_1/e_2]]^\# m^\# = \begin{cases} [\min(a/c, a/b), \max(b/c, b/d)] & c \geq 1 \\ [\min(b/c, b/d), \max(a/c, a/b)] & d \leq -1 \\ ([a, b]/([c/d] \cup [1, +\infty])) \cup ([a, b]/([c, d] \cup [-\infty, -1])) & altrimenti \end{cases}$$

Nota 3.3: Differenza come somma di intervalli

La differenza tra intervalli può essere vista come una somma di intervalli, ovvero:

$$[[e_1 - e_2]]^\# m^\# = [[e_1 + (-e_2)]]^\# m^\# = [a + (-d), b + (-c)] = [a - d, b - c]$$

dove $-[[e_2]]^\# m^\# = -[c, d] = [-d, -c]$.

3.2 Operatori booleani

In questo caso devo scrivere le assunzioni sugli intervalli, ovvero:

$$assumeB_{expr} \times \mathbb{M}^\# \to \mathbb{M}^\#$$

quindi c'é da definire gli operatori booleani del tipo

$$[[e_1 \odot_{rel} e_2]]^\# m^\# = assume_{e_1 \odot_{rel} e_2}(m^\#)$$

Quindi $assume_{e_1 \leq e_2}(m^\#)$ è definito come:

$$assume_{e_1 \leq e_2}(m^\#) =$$

3.3 Esempio di analisi

```
x = 0;
while (x < 40) {
    x = x + 1;
}
```

Da qui è possibile estrarre il cfg: