CAP_3_4

February 11, 2025

1 Rappresentazione e Ragionamento Proposizionale

1.1 Introduzione

Per la trattazione di questo argomento, l'obiettivo fissato è stato quello di **costruire una base di conoscenza in logica proposizionale** che permetta di **modellare un sistema di ragionamento automatico** per determinare le possibilità di incidente in base a determinate condizioni ambientali e regole logiche. L'intento era quello di sviluppare un sistema che, utilizzando **meccanismi di inferenza automatica**, potesse analizzare e dedurre se un incidente potenzialmente grave possa verificarsi in un dato scenario.

Abbiamo cercato di costruire una base di conoscenza che prenda in considerazione vari fattori, come **buio**, **pioggia**, **ghiacciato**, **nebbia**, e altre condizioni ambientali, nonché le condizioni superficiali, per determinare se queste possono contribuire alla gravità di un incidente. L'applicazione delle regole logiche definisce quando una serie di condizioni potrebbe portare a un incidente grave.

In particolare, l'approccio utilizzato ha permesso di applicare **forward chaining**, **backward chaining**, sfruttando le caratteristiche del linguaggio Prolog per inferire automaticamente nuove informazioni a partire da fatti noti. Questo sistema non solo analizza le condizioni presenti, ma è anche in grado di fare ipotesi su situazioni potenzialmente rischiose, anche in assenza di un dataset esterno specifico.

L'obiettivo finale era dunque quello di comprendere come un sistema automatico possa utilizzare la logica proposizionale per valutare la possibilità di un incidente in tempo reale, partendo da condizioni ambientali e superficiali, senza dover fare affidamento su dati esterni complessi.

1.2 Teoria Della Logica Proposizionale

La logica proposizionale si basa su proposizioni atomiche (atomi) che rappresentano enunciati semplici, come "buio" o "pioggia". Questi atomi possono essere combinati usando connettivi logici come la negazione (-), la congiunzione ( ), la disgiunzione ( ), l'implicazione ($\rightarrow$), e l'equivalenza ( ).

Nel codice, le proposizioni atomiche sono rappresentate da predicati Prolog come buio, pioggia, ghiacciato, ecc. Le regole logiche sono definite usando la sintassi di Prolog, che permette di esprimere implicazioni e condizioni complesse.

2 Sintassi e Semantica nel Codice

2.1 Proposizioni Atomiche:

- Le proposizioni atomiche sono rappresentate da predicati come `buio`, `pioggia`, `ghiacciato`, ecc.
- Questi predicati possono essere veri o falsi, a seconda delle condizioni ambientali.

2.2 Connettivi Logici:

- **Negazione**: La negazione è rappresentata dal simbolo `\+` in Prolog. Ad esempio, `\+ buio` significa "non buio".
- **Congiunzione**: La congiunzione è rappresentata dalla virgola `,`. Ad esempio, `buio, pioggia` significa "buio e pioggia".
- **Disgiunzione**: La disgiunzione è rappresentata dal punto e virgola `;`. Ad esempio, `buio; pioggia` significa "buio o pioggia".
- **Implicazione**: L'implicazione è rappresentata dalla freccia `:-`. Ad esempio, `incidente_grave :- condizione_pericolosa(...)` significa "incidente_grave se condizione_pericolosa(…)".

2.3 Regole e Fatti:

- **Fatti**: Sono proposizioni atomiche che vengono asserite come vere. Ad esempio, `superficie(asciutto)` è un fatto che indica che la superficie è asciutta.
- **Regole**: Sono implicazioni che definiscono condizioni complesse. Ad esempio, la regola `incidente_grave :- condizione_pericolosa(...)` definisce che un incidente grave si verifica se una condizione pericolosa è soddisfatta.

2.4 Metodi di Inferenza

2.5 Forward Chaining

Il **forward chaining** (catena in avanti) è una strategia di inferenza **bottom-up** che parte dai **fatti noti** e applica iterativamente le **regole** per dedurre nuovi fatti. Questo metodo è utilizzato nei **sistemi di produzione** e nei **motori inferenziali** per l'**intelligenza artificiale**.

In termini pratici, si parte da una **base di fatti iniziali** e si utilizzano le **regole di inferenza** per aggiungere nuove conoscenze alla **base di dati** fino a raggiungere la **conclusione desiderata**.

Il **forward chaining** è particolarmente utile quando vogliamo esplorare **tutte le possibili deduzioni** a partire da informazioni di base.

```
?- assertz(buio), assertz(nebbia), evaluta_condizioni([], Risultato).
Risultato = true.
```

Se i fatti **buio** e **nebbia** vengono asseriti, il sistema deduce che c'è **scarsa_visibilità**, che contribuisce a rendere vero **incidente_grave**.

2.6 Backward Chaining

Il **backward chaining** (catena all'indietro) è una strategia di inferenza **top-down** che parte da un **obiettivo da dimostrare** e cerca di verificare se i **fatti noti** supportano questa conclusione.

Questo metodo è molto usato nei **sistemi basati su regole** per la **risoluzione di problemi** e nei **sistemi esperti**.

Il processo inizia con un **obiettivo da dimostrare** e cerca di verificare se esistono **fatti** o **regole** che possono derivarlo. Se una **regola** porta alla **conclusione desiderata**, vengono verificate le **condizioni** della regola stessa.

Questo metodo è più **efficiente** quando vogliamo dimostrare una **specifica affermazione** senza esplorare l'intero spazio delle conoscenze.

```
?- incidente_grave.
```

Il sistema cercherà di provare **incidente_grave** controllando se esistono **fatti** e **regole** che lo rendono vero, attivando la valutazione delle **condizioni pericolose** definite nelle regole.

2.7 Esempio di Regole nel Codice

1. **Condizioni di illuminazione**:

```
luce :- \+ buio.
buio_luce_artificiale :- buio, luce_artificiale.
```

Queste regole definiscono quando il tempo è sereno (se non piove, non grandina, non nevica e non c'è vento forte) e quando c'è nebbia (se c'è vento forte o piove).

2.8 Condizioni Metereologiche

```
tempo_sereno :- \+ (pioggia; grandine; neve; vento_forte).
nebbia :- vento_forte; pioggia.
```

Queste regole definiscono quando il tempo è sereno (se non piove, non grandina, non nevica e non c'è vento forte) e quando c'è nebbia (se c'è vento forte o piove).

2.9 Condizioni Superficiali:

```
asciutto :- superficie(asciutto).
bagnato :- superficie(bagnato).
ghiacciato :- superficie(ghiacciato).
```

Queste regole definiscono le condizioni della superficie (asciutta, bagnata, ghiacciata).

2.10 Incidente Grave

```
incidente_grave :-
    condizione_pericolosa([scarsa_visibilita, meteo_estremo, superficie_pericolosa, combinazio
```

Questa regola definisce che un incidente grave si verifica se almeno una delle condizioni pericolose è soddisfatta.

2.11 Gestione Dinamica Delle Condizioni

Il codice utilizza predicati dinamici per gestire le condizioni ambientali. Ad esempio, buio, pioggia, ghiacciato sono predicati dinamici che possono essere modificati durante l'esecuzione del

programma. Questo permette di simulare diverse condizioni ambientali e verificare se portano a un incidente grave.

2.12 Inizializzazione:

```
:- initialization((
    retractall(buio),
    retractall(luce_artificiale),
    retractall(pioggia),
    retractall(grandine),
    retractall(neve),
    retractall(vento_forte),
    retractall(superficie(_)),
    retractall(nebbia),
    retractall(ghiacciato),
    assertz(superficie(asciutto))
)).
```

Questo blocco di codice inizializza le condizioni ambientali, rimuovendo tutte le asserzioni precedenti e impostando la superficie come asciutta.

2.13 Gestione delle Condizioni:

```
evaluta_condizioni(Condizioni, Risultato) :-
    setup_call_cleanup(
        apply_conditions(Condizioni, Aggiunte),
        (incidente_grave -> Risultato = true ; Risultato = false),
        maplist(retract_condition, Aggiunte)
    ).
```

Questo predicato permette di valutare diverse condizioni ambientali e determinare se portano a un incidente grave. Le condizioni vengono applicate temporaneamente e poi rimosse dopo la valutazione.

2.14 Esempi di Esecuzione

```
2 ?- evaluta_condizioni([buio, neve, superficie(bagnato), nebbia], Risultato).
Risultato = true.
```

La combinazione di **buio**, **neve**, **superficie(bagnato)** e **nebbia** è estremamente pericolosa, poiché ci sono molteplici fattori che riducono la visibilità e aumentano il rischio di slittamento o incidenti.

```
9 ?- evaluta_condizioni([superficie(bagnato), pioggia], Risultato).
Risultato = true.
```

Anche se la pioggia non è considerata una condizione di meteo pericolosa, il fatto che la **superficie** sia **bagnata** porta a far scattare l'allarme per un incidente grave.

```
10 ?- evaluta_condizioni([buio, superficie(asciutto), \+ pioggia], Risultato).
Risultato = true.
```

La seguente query afferma che c'è **buio**, una **superficie asciutta** e non c'è la **pioggia**. Per la nostra base di conoscenza, la regola per determinare un incidente grave è già soddisfatta, poiché la presenza di **buio** è sufficiente a scatenare l'allarme.

```
3 ?- evaluta_condizioni([\+buio, superficie(asciutto), \+pioggia], Risultato).
Risultato = false.
```

In questa situazione, stiamo dicendo che **non c'è buio**, quindi c'è **luce**, abbiamo una **superficie asciutta** e **non c'è pioggia**, quindi **non c'è rischio di un incidente grave**.

3 RAPPRESENTAZIONE E RAGIONAMENTO RELAZIONE (LOGICA DI PRIMO ORDINE)

3.1 Obiettivo

Per trattare la classificazione della gravità degli incidenti stradali, abbiamo sviluppato una base di conoscenza utilizzando la logica del primo ordine. Il nostro sistema classifica gli incidenti come **"gravi"** o **"lievi"** in base a determinate condizioni ambientali e stradali, utilizzando inferenza logica e abduzione per determinare le cause e le condizioni mancanti.

L'obiettivo principale è fornire un modello di ragionamento basato su regole logiche, che consenta di interpretare le condizioni di un incidente in modo dinamico, permettendo un'inferenza flessibile e scalabile. Questo approccio consente di adattarsi a nuovi scenari senza la necessità di modificare direttamente la base di conoscenza, migliorando l'efficacia del sistema nel supporto decisionale.

3.2 Introduzione alla Logica del Primo Ordine

La **logica del primo ordine** (FOL, *First-Order Logic*) è un sistema formale che estende la logica proposizionale includendo variabili, quantificatori e relazioni tra oggetti.

3.3 Componenti fondamentali

- **Simboli di Costante**: rappresentano entità specifiche (es. `"pioggia"`, `"ghiacciata"`).
- **Simboli di Variabile**: rappresentano entità generiche (es. `ID` per identificare un incidente).
- **Simboli di Predicato**: esprimono proprietà o relazioni tra entità (es. `illuminazione(ID, buio)`).

3.4 Quantificatori

- **(per ogni)**: indica una generalizzazione.
- **(esiste)**: afferma che almeno un elemento soddisfa una condizione.

3.5 Esempi di utilizzo dei quantificatori nella base di conoscenza

3.6 Quantificatore Universale( )

```
% ID: incidente_grave(ID) se ci sono almeno due condizioni pericolose
incidente_grave(ID) :-
    almeno_due_condizioni_pericolose(ID, Conditions),
    length(Conditions, N),
```

```
    N >= 2.
```

Questo implica che per ogni ID incidente, la regola vale universalmente.

3.7 Quantificatore Esistenziale ( ):

```
% ID: almeno_due_condizioni_pericolose(ID, Conditions) se esiste un insieme di condizioni peri
almeno_due_condizioni_pericolose(ID, ConditionsWithDetails) :-
    findall(Condition-Descr, (
        (scarsa_visibilita(ID, Descr), Condition = 'Scarsa visibilità');
        (meteo_estremo(ID, Descr), Condition = 'Meteo estremo');
        (superficie_pericolosa(ID, Descr), Condition = 'Superficie pericolosa');
        (combinazione_critica(ID, Descr), Condition = 'Combinazione critica')
    ), ConditionsWithDetails).
```

Qui il predicato **findall** dimostra che esiste almeno un insieme di condizioni pericolose per un incidente.

3.8 Completamento di Clark

Il **Completamento di Clark è una tecnica per trasformare una base di conoscenza in una forma chiusa, in cui si assume che tutto ciò che non è noto come vero sia falso (chiusura del mondo). Questo principio è utile per limitare l'inferenza logica e garantire una maggiore efficienza computazionale.

In termini formali, per ogni predicato **P(X)** nella base di conoscenza, il completamento di Clark afferma:

```
P(X) :- Condizione1(X).
P(X) :- Condizione2(X).
```

è equivalente a

```
P(X) <-> (Condizione1(X)  Condizione2(X)).
```

3.9 Applicazione del Completamento di Clark nel nostro sistema.

Nella nostra base di conoscenza, il completamento di Clark viene applicato per definire relazioni chiuse tra le condizioni di pericolosità e la gravità dell'incidente.

```
% ID: scarsa_visibilita(ID, Descrizione)  (illuminazione(ID, buio)  ¬illuminazione(ID, artifi
scarsa_visibilita(ID, Descrizione) :-
    (illuminazione(ID, buio), \+ illuminazione(ID, artificiale)) ->
        Descrizione = 'Buio senza illuminazione artificiale'
    ; (condizione_meteo(ID, nebbia)) ->
        Descrizione = 'Presenza di nebbia'
    ; fail.
```

Questa regola rappresenta il **completamento di Clark**, assumendo che l'unico modo per avere scarsa visibilità sia attraverso buio senza **illuminazione** o **nebbia**.

3.10 ABDUZIONE

L'abduzione è un processo inferenziale che consente di ipotizzare nuove informazioni per spiegare un fatto osservato. Nel nostro sistema, l'abduzione viene utilizzata per dedurre condizioni mancanti che potrebbero giustificare la gravità di un incidente.

Esempio di abduzione nel nostro sistema:

```
abduzione_da_meteo(ID) :-
    (condizione_meteo(ID, pioggia), \+ superficie_stradale(ID, _) ->
        assertz(superficie_stradale(ID, bagnata)),
        format('Abduzione: Dedotta superficie bagnata per pioggia~n')
    ; true).
```

Se piove e non è specificata la condizione della strada, il sistema deduce che la superficie è bagnata.

L'abduzione aiuta a colmare lacune nella conoscenza senza richiedere informazioni complete, migliorando l'accuratezza del sistema.

3.11 Forward Chaining

Il **Forward Chaining** è un approccio in cui il sistema parte dai dati iniziali (fatti noti) e applica le regole per dedurre nuove informazioni. L'idea è di partire dalle informazioni già disponibili e applicare le regole fino a raggiungere una conclusione.

Nel codice, il Forward Chaining è applicato nel contesto dell'abduzione e della deduzione delle informazioni mancanti. Ad esempio, se la condizione meteorologica è pioggia ma la superficie stradale non è definita, il sistema deduce che la superficie è bagnata.

```
abduzione_da_meteo(ID) :-
    (condizione_meteo(ID, pioggia),
     \+ superficie_stradale(ID, _) ->
        assertz(superficie_stradale(ID, bagnata)),
        format('Abduzione: Dedotta superficie bagnata per presenza di pioggia~n')
    ; true).
```

In questo caso, partendo dalla condizione meteorologica (pioggia), il sistema deduce che la superficie deve essere bagnata. Questo è un esempio di Forward Chaining, poiché si parte dai fatti (pioggia) per dedurre nuove informazioni (superficie bagnata).

3.12 Backward Chaining

Il **Backward Chaining** è un approccio che parte dalla conclusione desiderata e cerca di risalire, applicando le regole a ritroso, per verificare quali fatti devono essere veri affinché quella conclusione sia valida.

Nel codice, il Backward Chaining viene utilizzato in particolare nella valutazione dell'incidente. Il sistema parte dall'obiettivo di determinare la gravità dell'incidente (grave o lieve) e cerca di risalire per verificare se le condizioni pericolose (come scarsa visibilità, meteo estremo, superficie pericolosa) sono soddisfatte.

Esempio nel codice:

```
incidente_grave(ID) :-
    almeno_due_condizioni_pericolose(ID, Conditions),
    length(Conditions, N),
    N >= 2.
```

In questo caso, il sistema sta tentando di risalire alla conclusione (incidente grave) e applica le regole per verificare se esistono almeno due condizioni pericolose. Se le condizioni pericolose sono soddisfatte, l'incidente viene classificato come grave. Qui, il Backward Chaining è evidente perché il sistema parte dalla conclusione (grave) e cerca di verificare se le precondizioni (due condizioni pericolose) sono soddisfatte.

3.13 Integrazione di Forward e Backward Chaining

Il sistema integra sia il **Forward Chaining** che il **Backward Chaining** per ottenere una logica di deduzione potente e flessibile. Le regole di abduzione (Forward Chaining) vengono utilizzate per dedurre automaticamente le informazioni mancanti, mentre le regole di valutazione dell'incidente (Backward Chaining) vengono utilizzate per determinare la gravità dell'incidente sulla base delle condizioni pericolose identificate.

3.14 Esecuzione del sistema ed esempi di test

Per eseguire il sistema, è necessario avere un interprete Prolog installato sul proprio computer. Seguire i passaggi seguenti:

1. **Scaricare e installare Prolog**
   - Se si utilizza SWI-Prolog, è possibile scaricarlo dal sito ufficiale: **https://www.swi-prolog.org**
2. **Aprire il terminale e caricare il file della base di conoscenza**
   - Posizionarsi nella directory dove si trova il file **cap5.pl**
   - Eseguire il comando: **swipl**
   - **All'interno dell'interprete,caricare il file con: [cap5].
3. **Eseguire delle query per eseguire il sistema** assertz(condizione_meteo(id1, pioggia)). assertz(condizione_meteo(id2, pioggia)). così aggiungiamo i fatti alla kb per testare usiamo: **analizza_incidente(1).**

Carichiamo il file.pl nel terminale

```
1 ?- [cap5].
true.
```

Inseriamo delle condizioni e verifichiamo

```
assertz(illuminazione(1, buio)),
assertz(condizione_meteo(1, neve)),
assertz(superficie_stradale(1, ghiacciata)),


analizza_incidente(1).
```

RISULTATO ATTESO

```
=== Analisi dell'incidente 1 ===
```

```
Condizioni dell'incidente 1:
- Meteo: neve
- Illuminazione: buio
- Superficie: ghiacciata
- Gravità : non ancora valutato

Condizioni dopo abduzione:
Condizioni dell'incidente 1:
- Meteo: neve
- Illuminazione: buio
- Superficie: ghiacciata
- Gravità : non ancora valutato
Rilevate 4 condizioni pericolose:
- Scarsa visibilitÃ : Buio senza illuminazione artificiale
- Meteo estremo: Presenza di neve
- Superficie pericolosa: Strada ghiacciata
- Combinazione critica: Superficie ghiacciata con precipitazioni

CONCLUSIONE: Incidente classificato come GRAVE
true.
```

CASO 2:
- **Condizione di illuminazione**: Buio
- **Condizioni di meteo**: Nebbia
- **Superficie**: Umida

Esito atteso: Incidente grave

```
assertz(illuminazione(2, buio)),
assertz(condizione_meteo(2, nebbia)),
assertz(superficie_stradale(2, umida)),


analizza_incidente(2).
```

RISULTATO ATTESO

```
=== Analisi dell'incidente 2 ===
Condizioni dell'incidente 2:
- Meteo: nebbia
- Illuminazione: buio
- Superficie: umida
- GravitÃ : non ancora valutato

Condizioni dopo abduzione:
Condizioni dell'incidente 2:
- Meteo: nebbia
- Illuminazione: buio
- Superficie: umida
- GravitÃ : non ancora valutato
```

```
Rilevate 4 condizioni pericolose:
- Scarsa visibilitÃ : Buio senza illuminazione artificiale
- Meteo estremo: Presenza di nebbia densa
- Superficie pericolosa: Strada umida con nebbia
- Combinazione critica: Buio con nebbia

CONCLUSIONE: Incidente classificato come GRAVE
true.
```

Test 3:
- **Condizione meteo**: Pioggia

Esito atteso: Il sistema di abduzione riesce a capire dalla condizione del meteo che la superficie potrebbe essere bagnata.

Esito atteso:
- La superficie bagnata, combinata con la pioggia, è una condizione **pericolosa**.
- L'incidente è classificato come **lieve**, in quanto c'è solo una condizione pericolosa.

OUTPUT

```
=== Analisi dell'incidente 3 ===
Condizioni dell'incidente 3:
- Meteo: pioggia
- Illuminazione: non specificato
- Superficie: non specificato
- GravitÃ : non ancora valutato
Abduzione: Dedotta superficie bagnata per presenza di pioggia

Condizioni dopo abduzione:
Condizioni dell'incidente 3:
- Meteo: pioggia
- Illuminazione: non specificato
- Superficie: bagnata
- GravitÃ : non ancora valutato
Rilevate 1 condizioni pericolose:
- Superficie pericolosa: Strada bagnata con pioggia

CONCLUSIONE: Incidente classificato come LIEVE
true.
```

3.15 Conclusioni e Differenze con la Logica Proposizionale

La **logica del primo ordine** offre vantaggi significativi rispetto alla **logica proposizionale**:

- **Espressività Maggiore**: consente di definire relazioni tra oggetti e proprietà dinamiche.
- **Generalizzazione**: le regole possono essere applicate a un numero infinito di situazioni.
- **Inferenza più potente**: grazie all'uso di quantificatori e variabili.

4 Possibili Estensioni

- **Integrazione con Ontologie** (SSN/SOSA, SUMO, TRANS-Model) per arricchire la conoscenza del sistema.
- **Applicazione del Machine Learning** per migliorare la classificazione degli incidenti.