Metodi Quantitavi per l'Informatica

Formalizzazione di un problema in logica proposizionale

Autore: Alessia Borghini

Professore: Maurizio Lenzerini

Laurea Triennale in Ingegneria Informatica e Automatica

Dipartimento di Ingegneria Informatica, Automatica e Gestionale

Sapienza Università di Roma

Giugno 2021

Indice

1 Introduzione

L'obiettivo del progetto è quello di scegliere un problema, di formalizzarlo in logica proposizionale e di risolverlo. In particolare quindi, dopo aver definito il problema bisogna scriverlo in logica proposizionale, ottenendo una formula booleana che sarà soddisfacibile se e solo se è possibile trovare una combinazioni di valori booleani che rendano la formula vera, ovvero se il problema ha una soluzione. Rimandiamo alla Sezione 3 per ulteriori dettagli.

Il problema che abbiamo deciso di risolvere è un problema di planning, l'area dell'intelligenza artificiale che ha come scopo quello di pianificare una sequenza di azioni che permetta ad un agente di raggiungere uno stato finale dato. Nello specifico il problema scelto è un problema di classical planning [3], che impone i seguenti vincoli alla definizione del problema:

- l'ambiente è totalmente osservabile, deterministico e statico; solo l'agente può modificare lo stato del mondo in cui opera;
- il numero degli stati e quello delle azioni possibili del mondo sono finiti, anche se molto grandi;
- assumeremo che le azioni sono istantanee, quindi dobbiamo solamente occuparci della sequenza di svolgimento.

Il problema è quindi formato da i) uno stato iniziale, ii) delle azioni e dei vincoli e iii) uno stato finale. Lo stato iniziale descrive l'ambiente in cui si trova l'agente, quando l'agente non ha ancora effettuato mosse. Le azioni hanno precondizioni, che impongono vincoli e specificano in che configurazione del mondo possono essere eseguire tali azioni, e gli effetti che descrivono come cambia il mondo dopo l'esecuzione dell'azione stessa. Lo stato finale descrive l'obiettivo dell'agente, la configurazione dell'ambiente quando l'agente ha compiuto una sequenza di mosse vincenti. Una definizione più dettagliata del problema si trova nella Sezione 2.

Nella Sezione 3 verrà risolto il problema come SAT, mentre nella Sezione 4 verrà risolto utilizzando il linguaggio PDDL, e verrà fatto un confronto tra i due metodi nella Sezione 5.

PDDL [4], Planning Domain Definition Language, è uno dei linguaggi più diffusi per la risoluzione di problemi di planning.

2 Definizione del problema

In questa sezione introduciamo il problema scelto, descrivendo l'ambiente, l'agente e le azioni che quest'ultimo è in grado di compiere.

L'ambiente considerato, mostrato in Figura 1, è un grid-world di dimensioni 3×3 rappresentante un giardino. Nello specifico, ogni casella di tale giardino può essere: i) *vuota*, ii) *sana* (i.e., contenente piante sane), o iii) *infestata* (i.e., contenente piante infestanti).

L'agente, un robot giardiniere, può muoversi[1] tra le celle di tale giardino, e compiere le seguenti azioni:

1. quando si trova in una cella *sana* può *innaffiare* le piante presenti;
2. quando si trova in una cella *infestata* può *estirpare* le piante presenti.

L'obiettivo dell'agente è quello di trovare un percorso che gli permetta di i) estirpare tutte le piante infestanti e, ii) innaffiare tutte quelle sane.

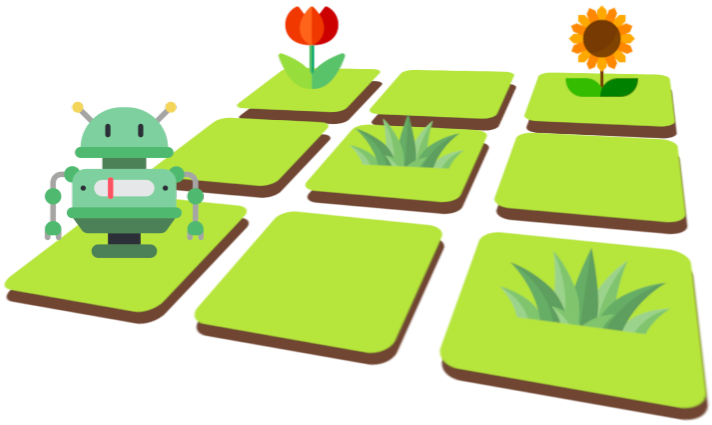

Figura 1: Rappresentazione grafica dell'ambiente considerato.

3 SAT

Il problema SAT, come definito nel libro "The Handbook of Satisfiability" [1], è il problema di determinare se una data formula booleana è soddisfacibile o meno, in particolare la definizione dice. Il problema SAT ha come input una CNF e come output un valore booleano, che indica se la formula è soddisfacibile o meno. Definiamo per prima cosa la CNF.

Definizione 1 (CNF) Una formula in forma normale congiuntiva (*conjunctive normal form*), è una congiunzione di clausole, ovvero solo legate dall'operatore logico and. Una clausola è una disgiunzione di letterali, ovvero sono legati dall'operatore logico or. Un letterale di una variabile booleana x è x se il letterale è positivo, o $\neg x$, se il letterale è negativo.

Definizione 2 (Soddisfacibilità) Una formula F è *soddisfacibile* se esiste un'assegnamento di verità ϕ che soddisfi F, cioè ϕ è *assegnamento soddisfacente* di F. Altrimenti la formula F è insoddisfacibile.

Definizione 3 (Assegnamento Soddisfacente) Un assegnamento di verità ϕ di una formula F assegna un valore vero alle sue variabili. L'assegnazione ϕ soddisfa:

- un letterale positivo se gli assegna un valore *True* alla sua variabile;

[1]per muoversi da una cella all'altra, la casella di partenza in cui si trova il robot deve essere adiacente a quella di arrivo. In particolare, consideriamo adiacenza-4.

- un letterale negativo se gli assegna un valore *False* alla sua variabile;
- una clausola se soddisfa almeno uno dei suoi letterali;
- una formula CNF se soddisfa ognuna delle sue clausole.

In questa sezione seguirà la formalizzazione del problema in logica proposizionale, e più nel dettaglio descriveremo il problema come formula booleana in CNF.

3.1 Formalizzazione generale del problema

In questa sezione mostriamo come ricondurre il problema definito nella Sezione 2 ad un problema di soddisfacibilità. Per ottenere la formula in CNF, ci occuperemo prima della formalizzazione di sottoformule che descrivono parzialmente il problema e poi le congiungeremo tramite and logici per ottenere la formula completa.

Forniamo una formalizzazione che si basa sulla *closed-world assumption*, ovvero tutto ciò che non è specificato è falso. L'ambiente è una griglia di dimensione $x_{max} \times y_{max}$ ed ogni casella è rappresentata dalla coppia di coordinate (x, y) con $x \in \{0, x_{max}\}$ e $y \in \{0, y_{max}\}$. In particolare, la casella in alto a sinistra è quella di coordinate $(0, 0)$, mentre quella in basso a destra è di coordinate $(2, 2)$. Definisco quindi l'insieme delle coordinate ammissibili:

$$C = \{\forall(x, y) | 0 < x < x_{max}, 0 < y < y_{max}\} \tag{1}$$

Il numero di mosse minimo n, che impiega l'agente per raggiungere l'obiettivo indica che l'agente nell'istante di tempo $n+1$ raggiunge lo stato finale, infatti un'azione svolta nell'istante i ha effetto sull'ambiente nell'istante successivo, quindi nell'istante $i + 1$.

3.1.1 Configurazione iniziale

Attraverso tale sottoformula descriviamo la situazione dell'ambiente e dell'agente nell'istante $i = 0$, quando nessuna mossa è stata effettuata. Ponendo determinate variabile a *true* otteniamo delle precondizioni necessarie affinchè possano essere eseguite, negli istanti successivi a $i = 0$, delle azioni.

Indico con $r_{i,(x,y)}$ la posizione del robot nell'istante i-esimo, se $r_{i,(x,y)} = 1$ il robot all'istante i si trova in posizione (x, y), se (x, y) è una coppia di coordinate ammissibili. Per esprimere la posizione iniziale del robot, inizializzo le variabili $r_{i,(x,y)}$, per $i = 0$:

$$\begin{aligned} r_{0,(x,y)} &= 1 \quad \forall(x, y) = (x_{in}, y_{in}) \\ r_{0,(x,y)} &= 0 \quad \forall(x, y) \neq (x_{in}, y_{in}) \end{aligned} \tag{2}$$

In particolare, devo inizializzare sia la variabile relativa alla posizione effettiva del robot a *true*, sia le posizioni in cui non si trova a *false*, perchè il robot si può trovare in una e una sola posizione in ogni istante. In logica proposizionale 2 diventa:

$$r_{0,(x_{in},y_{in})} \wedge \neg \bigwedge_{(x,y)\neq(x_{in},y_{in})} r_{0,(x,y)} \tag{3}$$

Come specificato nella Sezione 2, ci sono due tipologie di piante, quelle *sane* e quelle *infestanti*. Per descrivere le posizioni delle piante, definisco l'insieme P il quale contiene tutte le coppie di coordinate (x, y) che corrispondono a celle in cui è presente una pianta

$$P = \{\forall(x, y)| \text{ in } (x, y) \text{ si trova una pianta}\}$$

Chiamo $p_{i,(x,y)}$ le variabili che rappresentano il fatto che nella casella di coordinate (x, y), all'istante i, ci sia una pianta, questo accade per tutte le coppie di coordinate contenute nell'insieme P. Fisso quindi a *true* le variabili relative alle posizioni delle piante presenti nel giardino nell'istante iniziale:

$$p_{0,(x,y)} = 1 \quad \forall(x, y) \in P, \tag{4}$$

questo si scrive in logica proposizionale:

$$\bigwedge_{(x,y)\in P} p_{0,(x,y)} \wedge \neg \bigwedge_{(x,y)\in C\setminus P} p_{0,(x,y)} \tag{5}$$

Come descritto nella Sezione 2, quando il robot si trova in una cella in cui è presente una pianta, deve *capire* se la pianta è infestante, e quindi va estirpata, o è sana e quindi va innaffiata. L'insieme delle posizioni relative alle piante infestanti I, è definito come segue:

$$I = \{\forall(x, y) \in P| \text{ in } (x, y) \text{ si trova una pianta infestante}\} \subseteq P$$

Per descrivere la condizione infestante della pianta definiamo le seguenti variabile (indipendenti dall'istante di tempo considerato):

$$\begin{aligned} infestante_{(x,y)} &= 1 \ \forall(x, y) \in I \\ infestante_{(x,y)} &= 0 \ \forall(x, y) \notin I \end{aligned} \tag{6}$$

In logica proposizionale otteniamo:

$$\bigwedge_{(x,y)\in I} infestante_{(x,y)} \wedge \neg \bigwedge_{(x,y)\notin I} infestante_{(x,y)} \tag{7}$$

Se la pianta non è *infestante*, l'agente dovrà innaffiarla, definiamo quindi delle variabili che indichino il fatto che nell'istante i la pianta è innaffiata. Inizialmente tutte queste variabili sono *false*, perchè nessuna pianta è stata ancora innaffiata:

$$innaffiata_{0,(x,y)} = 0 \ \forall(x, y) \in C \tag{8}$$

Ciò diventa in logica proposizionale:

$$\bigwedge_{(x,y)\in C} \neg innaffiata_{0,(x,y)} \tag{9}$$

3.1.2 Azioni

Per raggiungere l'obiettivo, l'agente, in ogni istante, deve svolgere l'azione più consona alla configurazione in cui si trova. In particolare l'agente valuta le precondizioni imposte dalle definizioni delle varie azioni e *sceglie* l'azione da svolgere tra le seguenti, come specificato nella Sezione 2:

1. muoversi da una casella ad un'altra;

2. estirpare una pianta infestante;

3. innaffiare una pianta sana.

L'agente può spostarsi su una casella, azione 1, se si trova in una delle caselle adiacenti a questa, indichiamo quindi con $A_{(x,y)}$ l'insieme delle adiacenze della cella (x, y). L'azione $move\_to_{i,(x,y)}$ permette al robot di arrivare nella casella (x, y) nell'istante $i + 1$ se all'istante i si trovava in una delle caselle adiacenti a (x, y)

$$\begin{gathered} move\_to_{i,(x,y)} \rightarrow (\bigvee_{(x_i,y_i)} r_{i,(x_i,y_i)} \wedge r_{i+1,(x,y)}) \\ \forall (x,y) \mid (x,y) \in C,\ \forall (x_i,y_i) \mid (x_i,y_i) \in A_{(x,y)},\ \forall i \in \{0,..,n\} \end{gathered} \tag{10}$$

Trasformiamo ora tale formula in CNF

$$\begin{gathered} \neg move\_to_{i,(x,y)} \vee (\bigvee_{(x_i,y_i)} r_{i,(x_i,y_i)} \wedge r_{i+1,(x,y)}) \\ (\neg move\_to_{i,(x,y)} \vee \bigvee_{(x_i,y_i)} r_{i,(x_i,y_i)}) \wedge (\neg move\_to_{i,(x,y)} \vee r_{i+1,(x,y)}) \\ \forall (x,y) \mid (x,y) \in C,\ \forall (x_i,y_i) \mid (x_i,y_i) \in A_{(x,y)},\ \forall i \in \{0,..,n\} \end{gathered} \tag{11}$$

L'azione 2 rigurda l'estirpazione delle piante *infestanti*, una pianta può essere estirpata nell'istante i solo se in tale istante la posizione dell'agente coincide con quella di una pianta e se la pianta è *infestante*.

$$\begin{gathered} estirpa_{i,(x,y)} \rightarrow (r_{i,(x,y)} \wedge p_{i,(x,y)} \wedge infestante_{(x,y)} \neg \wedge p_{i+1,(x,y)}) \\ \forall (x,y)\ |(x,y) \in P,\ \forall i \in \{0,..,n\} \end{gathered} \tag{12}$$

Trasformiamo la formula in CNF:

$$\begin{gathered} (\neg estirpa_{i,(x,y)} \vee r_{i,(x,y)}) \wedge (\neg estirpa_{i,(x,y)} \vee p_{i,(x,y)}) \wedge \\ \wedge (\neg estirpa_{i,(x,y)} \vee infestante_{(x,y)}) \wedge (\neg estirpa_{i,(x,y)} \vee \neg p_{i+1,(x,y)}) \\ \forall (x,y)\ |(x,y) \in P,\ \forall i \in \{0,..,n\} \end{gathered} \tag{13}$$

L'azione 3 rigurda l'innaffiamento delle piante sane, può essere effettuata nell'istante i solo se in tale istante la posizione del robot coincide con quella di una pianta e se la pianta è *sana*, ovvero non è *infestante*.

$$\begin{gathered} innaffia_{i,(x,y)} \rightarrow (r_{i,(x,y)} \wedge p_{i,(x,y)} \wedge \neg infestante_{(x,y)} \wedge \neg innaffiata_{i,(x,y)} \wedge innaffiata_{i+1,(x,y)}) \\ \forall (x,y)\ |(x,y) \in P,\ \forall i \in \{0,..,n\} \end{gathered} \tag{14}$$

Analogamente ai casi precedenti, portiamo la formula in CNF:

$$\begin{gathered} (\neg innaffia_{i,(x,y)} \vee r_{i,(x,y)}) \wedge (\neg innaffia_{i,(x,y)} \vee p_{i,(x,y)}) \wedge (\neg innaffia_{i,(x,y)} \vee \neg infestante_{(x,y)}) \\ \wedge (\neg innaffia_{i,(x,y)} \vee \neg innaffiata_{i,(x,y)}) \wedge (\neg innaffia_{i,(x,y)} \vee innaffiata_{i+1,(x,y)}) \\ \forall (x,y)\ |(x,y) \in P,\ \forall i \in \{0,..,n\} \end{gathered} \tag{15}$$

3.1.3 Vincoli

L'agente in ogni istante si trova in una ed una sola posizione, devo definire un limite inferiore e uno superiore al numero di posizioni in cui si trova. Per far sì che l'agente in ogni istante si trovi in almeno una posizione devo assicurarmi che per ogni istante, almeno una variabile relativa alla posizione sia *true*

$$\begin{gathered}\bigvee_{(x,y)} r_{i,(x,y)} \\ \forall (x,y) \mid (x,y) \in C,\ \forall i \in \{0,..,n\}\end{gathered} \tag{16}$$

Per avere il limite superiore devo controllare il fatto che per ogni coppia di azioni possibile in un dato istante di tempo al massimo un'azione sia *true*

$$\begin{gathered}\neg r_{i,(x_1,y_1)} \vee \neg r_{i,(x_2,y_2)} \\ \forall (x_1,y_1),\ \forall (x_2,y_2) |\ (x_1,y_1) \in C,\ (x_2,y_2) \in C,\ (x_1,y_1) \neq (x_2,y_2),\ \forall i \in \{0,...,n\}\end{gathered} \tag{17}$$

Il numero di azioni che l'agente può svolgere è esattamente uno per ogni istante di tempo, quindi devo rappresentare tramite una CNF anche questa condizione. Indico con M l'insieme delle mosse che possono essere effettutate, $M = \{move\_to, innaffia, estirpa\}$. In modo analogo al caso precedente scrivo il limite inferiore e quello superiore per ottenere 1 sola azione *true* per istante:

$$\begin{gathered}\bigvee a_{i,(x,y)} \\ \forall (x,y) \mid (x,y) \in C,\ \forall a \in M,\ \forall i \in \{0,..,n-1\}\end{gathered}$$

$$\begin{gathered}\neg a^1_{i,(x,y)} \vee \neg a^2_{i,(x,y)} \\ \forall (x,y) \mid (x,y) \in C,\ \forall a^1, a^2 \mid a^1, a^2 \in M,\ a^1 \neq a^2,\ \forall i \in \{0,..,n\}\end{gathered} \tag{18}$$

L'agente può essere in una data configurazione, ovvero delle variabili hanno un certo valore, in un istante solo se nell'istante precedente erano verificate determinate condizioni. In particolare, dobbiamo ottenere le implicazioni inverse delle formule che descrivono le azioni, definite nella Sezione 3.1.2.
Per quanto riguarda la posizione dell'agente, sappiamo che è raggiungiungibile nell'istante $i+1$ solo se abbiamo effettuato una *move_to* nell'istante precedente o se eravamo in questa posizione nell'istante i e abbiamo effettuato una tra le azioni *estirpa* e $innaffia$ rimanendo quindi nella stessa posizione. In CNF questo diventa:

$$\begin{gathered}\neg r_{i+1,(x,y)} \vee \left(move\_to_{i,(x,y)} \vee estirpa_{i,(x,y)} \vee innaffia_{i,(x,y)}\right) \\ \forall (x,y) \mid (x,y) \in P\ \forall i \in \{0,..,n\}\end{gathered} \tag{19}$$

$$\begin{gathered}\neg r_{i+1,(x,y)} \vee move\_to_{i,(x,y)} \\ \forall (x,y) \mid (x,y) \in C \setminus P\ \forall i \in \{0,..,n\}\end{gathered} \tag{20}$$

Analogamente, per le variabili $p_{i+1,(x,y)}$ che è *false* solo se è stata estirpata nell'istante i la pianta che si trovava in (x,y), oppure se la pianta non era già presente nell'stante i, in CNF otteniamo:

$$\begin{gathered}p_{i+1,(x,y)} \vee \left(estirpa_{i,(x,y)} \vee \neg p_{i,(x,y)}\right) \\ \forall (x,y) \mid (x,y) \in P\ \forall i \in \{0,..,n\}\end{gathered} \tag{21}$$

Infine, le variabili $innaffiata_{i+1,(x,y)}$ sono *true* solo se è stata innaffiata la pianta in posizione (x, y) nell'istante precendente o se era già stata innaffiata la pianta nell'istante i, ovvero $innaffiata_{i,(x,y)}$ è *true*, in CNF diventa:

$$\begin{gathered}\neg innaffiata_{i+1,(x,y)} \vee \left(innaffia_{i,(x,y)} \vee \neg innaffiata_{i,(x,y)}\right) \\ \forall (x, y) \mid (x, y) \in P \; \forall i \in \{0, .., n\}\end{gathered} \tag{22}$$

3.1.4 Configurazione finale

La configurazione finale rappresenta il compimento dell'obiettivo, quindi le variabili che rappresentano le piante innaffiati devono essere *true* e le variabili che indicano la presenza di piante infestanti devono essere *false*

$$\left(\bigwedge_{(x,y)\in P\setminus I} innaffiata_{n,(x,y)} \wedge \neg \bigwedge_{(x,y)\in I} p_{n,(x,y)} \right) \tag{23}$$

3.2 Formalizzazione del problema 3×3

In questa sezione mostreremo come formalizzare il problema scelto, nel caso di griglia di dimensione 3×3. Per raggiungere questo scopo sono stati creati diversi script Python che analizzeremo nel dettaglio nella sezione. Come spiegato all'inizio della Sezione 3 per risolvere il problema tramite SAT bisogna ottenere una formula in CNF. La formula che formalizza il problema sarà scritta all'interno di un file di testo nel seguente formato:

- ogni riga rappresenta una clausola, quindi tutte le righe sono in "and" tra loro
- ogni elemento della riga rappresenta un letterale, quindi tutti gli elementi di una riga sono in "or" tra di loro. Se un elemento inizia con il simbolo "-" significa che il letterale è negato.

Tramite lo script Python `problem_generator.py` viene creato il file di testo `problem.txt` che contiene tutto il problema, formato cioè dalla configurazione iniziale, dalle azioni, dai vincoli e dallo stato finale.

```
def problem_gen():
    i_s = ""
    with open("initial_state.txt", "r") as f:
        f.readline()
        for line in f:
            i_s += line

    a = ""
    with open("moves.txt", "r") as f:
        for line in f:
            a += line

    f_s = ""
```

```
with open("final_state.txt", "r") as f:
    for line in f:
        f_s += line

with open("problem.txt", "w") as f:
    f.write(i_s+"\n")
    f.write(a+"\n")
    f.write(f_s)
```

Listing 1: problem_generator.py

Mostriamo ora nel dettaglio i contenuti dei file `initial_state.txt`, in Sezione 3.2.1, `moves.txt`, in Sezione 3.2.2 e `final_state.txt`, in Sezione 3.2.3.

3.2.1 Configurazione iniziale

Per quanto riguarda la configurazione iniziale abbiamo creato un file di testo, `initial_state.txt` tramite il quale viene definito lo stato iniziale.

La prima riga del file indica il numero di mosse massimo che l'agente avrà a disposizione per raggiungere l'obiettivo, in particolare il numero di mosse `n` implica il fatto che tramite l'ultima mossa, che viene effettuata nell'istante `n`, si raggiunge lo stato finale all'istante `n+1`. Successivamente viene inizializzata la posizione del robot, le variabili sono della forma `r_i,x,y`, dove, `i` indica l'istante di tempo, che ovviamente è `0` e la coppia `x,y` indica la coppia di coordinate in cui può trovarsi l'agente. Una sola tra le variabili `r` deve essere positiva, senza il segno - davanti, tutte le altre devono essere negate, con il segno - davanti. Inizializziamo poi le posizioni delle piante, le variabili `p_i,x,y` sono della stessa forma di quelle `r`, e sono positive tutte quelle relative alle piante presenti nel giardino. Per specificare quali tra le piante presenti sono infestanti utilizziamo le variabili `infestante_x,y` positive. Mentre per specificare che le piante sono sane vengono negate le variabili `infestante_x,y` corrispondenti alle piante sane che si trovano in posizione `x,y`. Infine, ci sono le variabili che inizializzano tutte le piante come non innaffiate.

```
11

r_0,0,2
-r_0,0,0
-r_0,0,1
-r_0,1,0
-r_0,1,1
-r_0,1,2
-r_0,2,0
-r_0,2,1
-r_0,2,2

p_0,0,0
p_0,1,1
p_0,2,0
p_0,2,2
```

```
-p_0,0,1
-p_0,0,2
-p_0,1,0
-p_0,1,2
-p_0,2,1

-infestante_0,0
-infestante_0,1
-infestante_0,2
-infestante_1,0
infestante_1,1
-infestante_1,2
-infestante_2,0
-infestante_2,1
infestante_2,2

-innaffiata_0,0,0
-innaffiata_0,1,1
-innaffiata_0,2,0
-innaffiata_0,2,2
```

3.2.2 Azioni e vincoli

Per generare le azioni e i vincoli per tutti gli istanti di tempo, una volta scelto il numero di mosse, viene utilizzato lo script Python `moves_generator.py`. Tale script prende in input il file di testo `actions_and_constraints.txt` – che specifica tutte le azioni possibili e i vincoli da soddisfare in un singolo istante di tempo – per generare le azioni e i vincoli per tutti i restanti istanti di tempo. In particolare, il file `actions_and_constraints.txt` esprime ciò che è stato illustrato nelle Sezioni 3.1.2 e 3.1.3 per il primo istante di tempo ed è riportato di seguito:

```
-move_to_0,0,0 r_0,0,1 r_0,1,0
-move_to_0,0,0 r_1,0,0
-move_to_0,1,0 r_0,0,0 r_0,1,1 r_0,2,0
-move_to_0,1,0 r_1,1,0
-move_to_0,2,0 r_0,1,0 r_0,2,1
-move_to_0,2,0 r_1,2,0
-move_to_0,0,1 r_0,0,0 r_0,1,1 r_0,0,2
-move_to_0,0,1 r_1,0,1
-move_to_0,1,1 r_0,0,1 r_0,1,0 r_0,1,2 r_0,2,1
-move_to_0,1,1 r_1,1,1
-move_to_0,2,1 r_0,2,0 r_0,1,1 r_0,2,2
-move_to_0,2,1 r_1,2,1
-move_to_0,0,2 r_0,0,1 r_0,1,2
-move_to_0,0,2 r_1,0,2
-move_to_0,1,2 r_0,0,2 r_0,1,1 r_0,2,2
-move_to_0,1,2 r_1,1,2
-move_to_0,2,2 r_0,1,2 r_0,2,1
-move_to_0,2,2 r_1,2,2
```

```
-estirpa_0,0,0 r_0,0,0
-estirpa_0,0,0 p_0,0,0
-estirpa_0,0,0 infestante_0,0
-estirpa_0,0,0 -p_1,0,0
-estirpa_0,0,0 r_1,0,0

-estirpa_0,1,1 r_0,1,1
-estirpa_0,1,1 p_0,1,1
-estirpa_0,1,1 infestante_1,1
-estirpa_0,1,1 -p_1,1,1
-estirpa_0,1,1 r_1,1,1

-estirpa_0,2,0 r_0,2,0
-estirpa_0,2,0 p_0,2,0
-estirpa_0,2,0 infestante_2,0
-estirpa_0,2,0 -p_1,2,0
-estirpa_0,2,0 r_1,2,0

-estirpa_0,2,2 r_0,2,2
-estirpa_0,2,2 p_0,2,2
-estirpa_0,2,2 infestante_2,2
-estirpa_0,2,2 -p_1,2,2
-estirpa_0,2,2 r_1,2,2

-innaffia_0,0,0 r_0,0,0
-innaffia_0,0,0 p_0,0,0
-innaffia_0,0,0 -infestante_0,0
-innaffia_0,0,0 -innaffiata_0,0,0
-innaffia_0,0,0 innaffiata_1,0,0
-innaffia_0,0,0 r_1,0,0

-innaffia_0,1,1 r_0,1,1
-innaffia_0,1,1 p_0,1,1
-innaffia_0,1,1 -infestante_1,1
-innaffia_0,1,1 -innaffiata_0,1,1
-innaffia_0,1,1 innaffiata_1,1,1
-innaffia_0,1,1 r_1,1,1

-innaffia_0,2,0 r_0,2,0
-innaffia_0,2,0 p_0,2,0
-innaffia_0,2,0 -infestante_2,0
-innaffia_0,2,0 -innaffiata_0,2,0
-innaffia_0,2,0 innaffiata_1,2,0
-innaffia_0,2,0 r_1,2,0

-innaffia_0,2,2 r_0,2,2
-innaffia_0,2,2 p_0,2,2
-innaffia_0,2,2 -infestante_2,2
-innaffia_0,2,2 -innaffiata_0,2,2
-innaffia_0,2,2 innaffiata_1,2,2
-innaffia_0,2,2 r_1,2,2

-move_to_0,0,0 -move_to_0,1,0
-move_to_0,0,0 -move_to_0,2,0
```

```
-move_to_0,0,0 -move_to_0,0,1
-move_to_0,0,0 -move_to_0,1,1
-move_to_0,0,0 -move_to_0,2,1
-move_to_0,0,0 -move_to_0,0,2
-move_to_0,0,0 -move_to_0,1,2
-move_to_0,0,0 -move_to_0,2,2
-move_to_0,0,0 -estirpa_0,0,0
-move_to_0,0,0 -estirpa_0,1,1
-move_to_0,0,0 -estirpa_0,2,0
-move_to_0,0,0 -estirpa_0,2,2
-move_to_0,0,0 -innaffia_0,0,0
-move_to_0,0,0 -innaffia_0,1,1
-move_to_0,0,0 -innaffia_0,2,0
-move_to_0,0,0 -innaffia_0,2,2
-move_to_0,1,0 -move_to_0,2,0
-move_to_0,1,0 -move_to_0,0,1
-move_to_0,1,0 -move_to_0,1,1
-move_to_0,1,0 -move_to_0,2,1
-move_to_0,1,0 -move_to_0,0,2
-move_to_0,1,0 -move_to_0,1,2
-move_to_0,1,0 -move_to_0,2,2
-move_to_0,1,0 -estirpa_0,0,0
-move_to_0,1,0 -estirpa_0,1,1
-move_to_0,1,0 -estirpa_0,2,0
-move_to_0,1,0 -estirpa_0,2,2
-move_to_0,1,0 -innaffia_0,0,0
-move_to_0,1,0 -innaffia_0,1,1
-move_to_0,1,0 -innaffia_0,2,0
-move_to_0,1,0 -innaffia_0,2,2
-move_to_0,2,0 -move_to_0,0,1
-move_to_0,2,0 -move_to_0,1,1
-move_to_0,2,0 -move_to_0,2,1
-move_to_0,2,0 -move_to_0,0,2
-move_to_0,2,0 -move_to_0,1,2
-move_to_0,2,0 -move_to_0,2,2
-move_to_0,2,0 -estirpa_0,0,0
-move_to_0,2,0 -estirpa_0,1,1
-move_to_0,2,0 -estirpa_0,2,0
-move_to_0,2,0 -estirpa_0,2,2
-move_to_0,2,0 -innaffia_0,0,0
-move_to_0,2,0 -innaffia_0,1,1
-move_to_0,2,0 -innaffia_0,2,0
-move_to_0,2,0 -innaffia_0,2,2
-move_to_0,0,1 -move_to_0,1,1
-move_to_0,0,1 -move_to_0,2,1
-move_to_0,0,1 -move_to_0,0,2
-move_to_0,0,1 -move_to_0,1,2
-move_to_0,0,1 -move_to_0,2,2
-move_to_0,0,1 -estirpa_0,0,0
-move_to_0,0,1 -estirpa_0,1,1
-move_to_0,0,1 -estirpa_0,2,0
-move_to_0,0,1 -estirpa_0,2,2
-move_to_0,0,1 -innaffia_0,0,0
-move_to_0,0,1 -innaffia_0,1,1
-move_to_0,0,1 -innaffia_0,2,0
```

-move_to_0,0,1 -innaffia_0,2,2
-move_to_0,1,1 -move_to_0,2,1
-move_to_0,1,1 -move_to_0,0,2
-move_to_0,1,1 -move_to_0,1,2
-move_to_0,1,1 -move_to_0,2,2
-move_to_0,1,1 -estirpa_0,0,0
-move_to_0,1,1 -estirpa_0,1,1
-move_to_0,1,1 -estirpa_0,2,0
-move_to_0,1,1 -estirpa_0,2,2
-move_to_0,1,1 -innaffia_0,0,0
-move_to_0,1,1 -innaffia_0,1,1
-move_to_0,1,1 -innaffia_0,2,0
-move_to_0,1,1 -innaffia_0,2,2
-move_to_0,2,1 -move_to_0,0,2
-move_to_0,2,1 -move_to_0,1,2
-move_to_0,2,1 -move_to_0,2,2
-move_to_0,2,1 -estirpa_0,0,0
-move_to_0,2,1 -estirpa_0,1,1
-move_to_0,2,1 -estirpa_0,2,0
-move_to_0,2,1 -estirpa_0,2,2
-move_to_0,2,1 -innaffia_0,0,0
-move_to_0,2,1 -innaffia_0,1,1
-move_to_0,2,1 -innaffia_0,2,0
-move_to_0,2,1 -innaffia_0,2,2
-move_to_0,0,2 -move_to_0,1,2
-move_to_0,0,2 -move_to_0,2,2
-move_to_0,0,2 -estirpa_0,0,0
-move_to_0,0,2 -estirpa_0,1,1
-move_to_0,0,2 -estirpa_0,2,0
-move_to_0,0,2 -estirpa_0,2,2
-move_to_0,0,2 -innaffia_0,0,0
-move_to_0,0,2 -innaffia_0,1,1
-move_to_0,0,2 -innaffia_0,2,0
-move_to_0,0,2 -innaffia_0,2,2
-move_to_0,1,2 -move_to_0,2,2
-move_to_0,1,2 -estirpa_0,0,0
-move_to_0,1,2 -estirpa_0,1,1
-move_to_0,1,2 -estirpa_0,2,0
-move_to_0,1,2 -estirpa_0,2,2
-move_to_0,1,2 -innaffia_0,0,0
-move_to_0,1,2 -innaffia_0,1,1
-move_to_0,1,2 -innaffia_0,2,0
-move_to_0,1,2 -innaffia_0,2,2
-move_to_0,2,2 -estirpa_0,0,0
-move_to_0,2,2 -estirpa_0,1,1
-move_to_0,2,2 -estirpa_0,2,0
-move_to_0,2,2 -estirpa_0,2,2
-move_to_0,2,2 -innaffia_0,0,0
-move_to_0,2,2 -innaffia_0,1,1
-move_to_0,2,2 -innaffia_0,2,0
-move_to_0,2,2 -innaffia_0,2,2
-estirpa_0,0,0 -estirpa_0,1,1
-estirpa_0,0,0 -estirpa_0,2,0
-estirpa_0,0,0 -estirpa_0,2,2
-estirpa_0,0,0 -innaffia_0,0,0

```
-estirpa_0,0,0 -innaffia_0,1,1
-estirpa_0,0,0 -innaffia_0,2,0
-estirpa_0,0,0 -innaffia_0,2,2
-estirpa_0,1,1 -estirpa_0,2,0
-estirpa_0,1,1 -estirpa_0,2,2
-estirpa_0,1,1 -innaffia_0,0,0
-estirpa_0,1,1 -innaffia_0,1,1
-estirpa_0,1,1 -innaffia_0,2,0
-estirpa_0,1,1 -innaffia_0,2,2
-estirpa_0,2,0 -estirpa_0,2,2
-estirpa_0,2,0 -innaffia_0,0,0
-estirpa_0,2,0 -innaffia_0,1,1
-estirpa_0,2,0 -innaffia_0,2,0
-estirpa_0,2,0 -innaffia_0,2,2
-estirpa_0,2,2 -innaffia_0,0,0
-estirpa_0,2,2 -innaffia_0,1,1
-estirpa_0,2,2 -innaffia_0,2,0
-estirpa_0,2,2 -innaffia_0,2,2
-innaffia_0,0,0 -innaffia_0,1,1
-innaffia_0,0,0 -innaffia_0,2,0
-innaffia_0,0,0 -innaffia_0,2,2
-innaffia_0,1,1 -innaffia_0,2,0
-innaffia_0,1,1 -innaffia_0,2,2
-innaffia_0,2,0 -innaffia_0,2,2

move_to_0,0,0 move_to_0,1,0 move_to_0,2,0 move_to_0,0,1 move_to_0,1,1 move_to_0,2,1 move_to_0,0,2 move

r_1,2,0 r_1,0,0 r_1,0,1 r_1,0,2 r_1,1,0 r_1,1,1 r_1,1,2 r_1,2,1 r_1,2,2

-r_1,0,0 -r_1,0,1
-r_1,0,0 -r_1,0,2
-r_1,0,0 -r_1,1,0
-r_1,0,0 -r_1,1,1
-r_1,0,0 -r_1,1,2
-r_1,0,0 -r_1,2,0
-r_1,0,0 -r_1,2,1
-r_1,0,0 -r_1,2,2
-r_1,0,1 -r_1,0,2
-r_1,0,1 -r_1,1,0
-r_1,0,1 -r_1,1,1
-r_1,0,1 -r_1,1,2
-r_1,0,1 -r_1,2,0
-r_1,0,1 -r_1,2,1
-r_1,0,1 -r_1,2,2
-r_1,0,2 -r_1,1,0
-r_1,0,2 -r_1,1,1
-r_1,0,2 -r_1,1,2
-r_1,0,2 -r_1,2,0
-r_1,0,2 -r_1,2,1
-r_1,0,2 -r_1,2,2
-r_1,1,0 -r_1,1,1
-r_1,1,0 -r_1,1,2
-r_1,1,0 -r_1,2,0
-r_1,1,0 -r_1,2,1
-r_1,1,0 -r_1,2,2
```

```
-r_1,1,1 -r_1,1,2
-r_1,1,1 -r_1,2,0
-r_1,1,1 -r_1,2,1
-r_1,1,1 -r_1,2,2
-r_1,1,2 -r_1,2,0
-r_1,1,2 -r_1,2,1
-r_1,1,2 -r_1,2,2
-r_1,2,0 -r_1,2,1
-r_1,2,0 -r_1,2,2
-r_1,2,1 -r_1,2,2

-r_1,0,0 move_to_0,0,0 innaffia_0,0,0 estirpa_0,0,0
-r_1,1,1 move_to_0,1,1 innaffia_0,1,1 estirpa_0,1,1
-r_1,2,0 move_to_0,2,0 innaffia_0,2,0 estirpa_0,2,0
-r_1,2,2 move_to_0,2,2 innaffia_0,2,2 estirpa_0,2,2
-r_1,0,1 move_to_0,0,1
-r_1,0,2 move_to_0,0,2
-r_1,1,0 move_to_0,1,0
-r_1,1,2 move_to_0,1,2
-r_1,2,1 move_to_0,2,1
p_1,0,0 -p_0,0,0 estirpa_0,0,0
p_1,1,1 -p_0,1,1 estirpa_0,1,1
p_1,2,0 -p_0,2,0 estirpa_0,2,0
p_1,2,2 -p_0,2,2 estirpa_0,2,2
p_0,0,0 -p_1,0,0
p_0,0,1 -p_1,0,1
p_0,0,2 -p_1,0,2
p_0,1,0 -p_1,1,0
p_0,1,1 -p_1,1,1
p_0,1,2 -p_1,1,2
p_0,2,0 -p_1,2,0
p_0,2,1 -p_1,2,1
p_0,2,2 -p_1,2,2
-innaffiata_1,0,0 innaffia_0,0,0 innaffiata_0,0,0
-innaffiata_1,1,1 innaffia_0,1,1 innaffiata_0,1,1
-innaffiata_1,2,0 innaffia_0,2,0 innaffiata_0,2,0
-innaffiata_1,2,2 innaffia_0,2,2 innaffiata_0,2,2
```

A partire da tale file, tramite lo script Python `moves_generator.py` viene generato il file `moves.txt`, che sarà ~ 12 volte[2], più grande del file "actions_and_constraints.txt" riportato, che contiene le azioni possibili con annessi vincoli per ogni istante di tempo.

```python
def gen_moves(n):
    move = {}

    with open("actions_and_constraints.txt", "r") as f:
        move[0] = ""
        for line in f:
            if line!="\n":
```

[2] 12 è il massimo numero di mosse che l'agente ha a disposizione per risolvere il problema, specificato nella prima riga del file "actions_and_constraints.txt".

```
            move[0] += line

    for i in range(n):
        move[i+1] = move[0].replace("move_to_0", "move_to_"+str(i+1))
        move[i+1] = move[i+1].replace("estirpa_0", "estirpa_"+str(i+1))
        move[i+1] = move[i+1].replace("innaffiata_1", "innaffiata_"+str(i+2))
        move[i+1] = move[i+1].replace("innaffiata_0", "innaffiata_"+str(i+1))
        move[i+1] = move[i+1].replace("innaffia_0", "innaffia_"+str(i+1))
        move[i+1] = move[i+1].replace("p_1", "p_"+str(i+2))
        move[i+1] = move[i+1].replace("p_0", "p_"+str(i+1))
        move[i+1] = move[i+1].replace("r_1", "r_"+str(i+2))
        move[i+1] = move[i+1].replace("r_0", "r_"+str(i+1))

    with open("moves.txt", "w") as f:
        for k in move:
            f.write(move[k]+"\n")
```

Listing 2: moves_generator.py

3.2.3 Configurazione finale

La configurazione finale viene generata tramite lo script Python `final_state_generator.py` che viene chiamato al momento della risoluzione del problema.

```
def generate_final_state():
    with open("initial_state.txt", "r") as f:
        p = []
        inf = []
        num_mosse = int(f.readline().strip())
        for line in f:
            line = line.strip()
            if len(line)>1:
                if line[0]=="p":
                    p.append(line)
                elif "infestante" in line:
                    inf.append(line)

    final_state = ""
    coord = ""
    for pos in p:
        coord = pos[-5:]
        for i in inf:
            if i[-5:]==coord:
                if i[0]=="-":
                    final_state += "innaffiata_"+str(num_mosse)+","+coord+"\n"
                else:
                    final_state += "-p_"+str(num_mosse)+","+coord+"\n"
```

```
    with open("final_state.txt", "w") as f:
        f.write(final_state)
    return num_mosse
```

Listing 3: final_state_generator.py

Il risultato di tale script, applicato allo stato iniziale `initial_state.txt` mostrato in Sezione 3.2.1, è il seguente file, `final_state.txt`.

```
innaffiata_12,0,0
-p_12,1,1
innaffiata_12,2,0
-p_12,2,2
```

3.3 Risoluzione del problema

Per risolvere il problema per prima cosa bisogna convertire il file `problem.txt` in formato DIMACS CNF. Il formato DIMACS CNF rappresenta una formula in *conjunctive normal form*, dove gli interi positivi indicano letterali positivi e le loro negazioni corrispondono a letterali negati. Per effettuare la conversione utilizziamo lo script Python `convert2dimacs.py`.

```
def convert_to_dimacs():
    file = ""
    with open("problem.txt", "r") as f:
        for line in f:
            if line!="\n":
                file += line

    file = file.replace("(", "")
    file = file.replace(")", "")
    file = file.replace(",", "")
    file = file.replace("_", "")
    file = file.replace("infestante", "1")
    file = file.replace("innaffiata", "2")
    file = file.replace("moveto", "3")
    file = file.replace("estirpa", "4")
    file = file.replace("innaffia", "5")
    file = file.replace("p", "6")
    file = file.replace("r", "7")

    with open("problem_dimacs.txt", "w") as f:
        f.write(file)
```

Listing 4: convert2dimacs.py

Una volta ottenuto il file in formato dimacs, questo può essere utilizzato dallo script Python `solver.py` per risolvere il problema. Per risolvere il problema, utilizziamo uno dei solver di PySAT, MiniSat 2.2, la cui documentazione si può consultare al seguente link `https://github.com/pysathq/pysat`.

```
from pysat.solvers import *
import numpy as np

def solve():
    with open("report.txt", "w") as report:
        with open("problem_dimacs.txt", "r") as f:
            l = []
            flattened_list = []
            for line in f:
                line= [int(x) for x in line.strip().split(" ")]
                l.append(line)
                flattened_list.extend(line)

        s = Solver(use_timer=True)
        for elem in l:
            s.add_clause(elem)

        ris = s.solve()
        t = s.time()

        sat = "SAT" if ris else "UNSAT"
        report.write("RESULT: "+sat+"\n")
        report.write("EXEC-TIME: "+str(t)+"\n")
        print(sat)

        if ris:
            model = s.get_model()
            modello_txt = ""

            replace_words = {"1":"infestante", "2": "innaffiata", "3": "moveto",
                "4": "estirpa", "5": "innaffia", "6": "p", "7": "r"}
            for elem in model:
                if elem in flattened_list:
                    lit = str(elem)
                    literal = ""
                    if lit[0]!= "-":
                        literal = replace_words[lit[0]]
                        literal += "_" + lit[1:-2] + ",(" + lit[-2:-1] + "," +
                            lit[-1:] + ")"
                    else:
                        literal = "-"+replace_words[lit[1]]
                        literal += "_" + lit[2:-2] + ",(" + lit[-2:-1] + "," +
                            lit[-1:] + ")"
```

```
                modello_txt += literal+"\n"

            report.write("MODEL:\n"+modello_txt)

        else:
            report.write("MODEL: -\n")
        s.delete()
        print("Generated report file")
```

Listing 5: solver.py

Per risolvere il problema, abbiamo creato per comodità, lo script Python `solve.py` che svolge i passi sopra descritti. Una volta eseguito questo file, otteniamo sul terminale il risultato dell'esecuzione, quindi se il problema è SAT o UNSAT, e un messaggio che indica il fatto che è stato scritto il report dell'esecuzione nel file `report.txt`. Questo file, che è disponibile nella repository GitHub linkata nella Sezione 7, conterrà delle informazioni sull'esecuzione, come il tempo di esecuzione e il modello che rende soddisfacibile la formula.

```
from final_state_generator import generate_final_state
from solver import solve
from moves_generator import gen_moves
from problem_generator import problem_gen
from convert2dimacs import convert_to_dimacs

num_mosse = generate_final_state()
gen_moves(num_mosse)
problem_gen()
convert_to_dimacs()
solve()
```

Listing 6: solve.py

4 PDDL

PDDL (Planning Domain Definition Language) è un linguaggio di planning introdotto nel 1998 da Ghallab et al. [2]. Tale linguaggio prende ispirazione dalla formulazione STRIPS dei problemi di planning. Un problema di planning in PDDL è definito da due componenti: una descrizione del dominio, ed una descrizione del problema. La prima fornisce una definizione parametrizzata delle azioni che caratterizzano il comportamento dell'agente, così come la definizione di predicati, funzioni e tipi usati nel problema. La seconda, invece, fornisce la definizione dello stato iniziale, dello stato finale, e degli oggetti specifici che caratterizzano il problema.

Entrambe le componenti, dominio e problema, sono espresse tramite due file ".pddl". Nello specifico la struttura di base del dominio è la seguente:

dove:

```
(define
   (domain domain_name)

   (:requirements :strips ... )

   (:predicates p1 p2 ... )

   (:action a1
      :parameters (?param1 ?param2 ...)
      :precondition (and p1 ...)
      :effect (and (not(p1)) ...)
   )
)
```

- **:requirements** specifica le caratteristiche del linguaggio che servono;
- **:predicates** dichiara tutti i predicati (con o senza parametri) che intendiamo usare;
- **:action** definisce un azione che prende gli argomenti di **:parameters** come input e, se ogni espressione logica in **:precondition** è soddisfatta, ciò che è spicificato in **:effect** diventa vero.

La struttura del problema è invece rappresentata come segue:

```
(define
   (problem problem_name)

   (:domain domain_name)

   (:init p1 p2 (not(p3)) ...)

   (:goal (p3))
)
```

dove:

- **domain_name** deve essere lo stesso usato nel file in cui è definito il dominio;
- **:init** specifica tutti i letterali che inizialmente sono *true*;
- **:goal** definisce lo stato finale che vogliamo raggiungere.

Sebbene PDDL segua la formulazione STRIPS, esso offre molteplici estensioni che forniscono una maggiore espressività. In particolare, PDDL consente l'utilizzo di pre-condizioni negative, effetti condizionali e quantificatori, sia nelle pre- che nelle post- condizioni. Durante il processo di planning, ed in particolare durante la fase di grounding, le azioni parametrizzate sono rimpiazzate da azioni grounded dove i parametri formali sono sostituiti dagli oggetti

definiti nel problema.

Nel resto della sezione vedremo come formalizzare il problema definito nella Sezione 2.

4.1 Domain

Nel file del dominio, dobbiamo descrivere l'ambiente in cui agisce il robot giardiniere.

Dopo aver nominato il dominio, elenchiamo i requisiti necessari del file i) `:strips`, che permette di avere tutte le funzionalità di base di PDDL, ii) `:negative-preconditions`, il quale permette di utilizzare il `not` delle precondizioni e degli effetti, iii) `:equality`, necessario per usare l'uguaglianza tra variabili e iv) `:typing`, che permette di definire dei tipi. Definiamo poi i tipi, per l'agente, le piante e le celle, che verranno utilizzati per le azioni. I predicati che verranno utilizzati nella definizione del problema per definire delle caratteristiche degli oggetti. Infine, definiamo le azioni che possono essere effettuate nell'ambiente: una per far muovere l'agente, una per innaffiare una pianta e una per estirpare una pianta infestante.

```
(define (domain robot_giardiniere)
    (:requirements :strips :negative-preconditions :equality :typing)

    (:types agente pianta cella)

    (:predicates
        (adj ?from ?to) (innaffiata ?p) (infestata ?p)
        (pianta-in ?p ?c) (at ?robot ?c)
    )
        (:action move
        :parameters (?robot - agente ?from - cella ?to - cella)
        :precondition (and (at ?robot ?from) (adj ?from ?to) (not (= ?from ?to)))
        :effect (and (at ?robot ?to) (not (at ?robot ?from)))
    )
    (:action innaffia
        :parameters (?robot - agente ?c - cella ?p - pianta)
        :precondition (and (at ?robot ?c) (pianta-in ?p ?c) (not(infestante ?p))
            (not (innaffiata ?p)))
        :effect (and (innaffiata ?p))
    )

    (:action estirpa
        :parameters (?robot - agente ?c - cella ?p - pianta)
        :precondition (and (at ?robot ?c) (pianta-in ?p ?c) (infestante ?p))
        :effect (and (not (pianta-in ?p ?c)))
    )
)
```

4.2 Problem

Nel file del problema, dobbiamo descrivere il problema che dobbiamo risolvere.

Definiamo gli oggetti del problema il robot, le piante e le celle ai quali assegniamo i tipi definiti nel dominio. Nello stato iniziale, `:init`, vengono applicati i predicati definiti nel dominio agli oggetti del problema per definire la configurazione iniziale. Nello stato finale, `:goal`, vengono applicati di nuovo i predicati per esprimere la configurazione finale dell'ambiente.

```
(define (problem 3x3)
    (:domain robot_giardiniere)

    (:objects
        robot_giardiniere - agente
        p1 p2 p3 p4 - pianta
        c00 c01 c02 c10 c11 c12 c20 c21 c22 - cella
    )
    (:init
        (at robot_giardiniere c02)
        (pianta-in p1 c00)
        (pianta-in p2 c11) (infestante p2)
        (pianta-in p3 c22) (infestante p3)
        (pianta-in p4 c20)
        (adj c00 c01) (adj c00 c10) (adj c01 c00) (adj c01 c11) (adj c01 c02)
        (adj c02 c01) (adj c02 c12) (adj c10 c00) (adj c10 c11) (adj c10 c20)
        (adj c11 c01) (adj c11 c10) (adj c11 c12) (adj c11 c21) (adj c12 c02)
        (adj c12 c11) (adj c12 c22) (adj c20 c10) (adj c20 c21) (adj c21 c11)
        (adj c21 c20) (adj c21 c22) (adj c22 c12) (adj c22 c21)
    )

    (:goal
        (and
            (not (pianta-in p2 c11))
            (not (pianta-in p3 c22))
            (innaffiata p1)
            (innaffiata p4)
        )
    )
)
```

4.3 Plan

Tramite un PDDL Solver, come questo al link `https://web-planner.herokuapp.com/`, possiamo risolvere il problema. Il report dell'esecuzione è riportato di seguito, questo con-

tiene il risultato dell'esecuzione, quindi se è raggiungibile l'obiettivo e in caso affermativo, il piano risolutivo trovato.

```
Result: SUCCESS
Domain: robot_giardiniere
Problem: 3x3
Plan:
  (move robot_giardiniere c02 c01)
  (move robot_giardiniere c01 c00)
  (innaffia robot_giardiniere c00 p1)
  (move robot_giardiniere c00 c01)
  (move robot_giardiniere c01 c11)
  (estirpa robot_giardiniere c11 p2)
  (move robot_giardiniere c11 c10)
  (move robot_giardiniere c10 c20)
  (innaffia robot_giardiniere c20 p4)
  (move robot_giardiniere c20 c21)
  (move robot_giardiniere c21 c22)
  (estirpa robot_giardiniere c22 p3)
Execution time: 0.0019s
```

5 Confronto tra i due metodi

Dopo aver formalizzato il problema in logica proposizionale, in Sezione 3, e in PDDL, come mostrato in Sezione 4, in questa sezione possiamo fare un contronto tra i due procedimenti.

Nella formulazione SAT abbiamo ricondotto il problema in una CNF e l'abbiamo fornita in input ad un SAT Solver che ha valutato la sua soddisfacibilità, restituendo una sequenza di mosse che permetta al robot di raggiungere l'obiettivo. Sebbene tale sequenza di mosse restituita dal solver sia soddisfacente, il processo di modellazione di un problema di planning con SAT ha lo svantaggio di dover introdurre in maniera esplicita molteplici vincoli ausiliari al fine di permettere all'agente di scegliere la mossa corretta da effettuare. Inoltre, siccome il planning consiste in una serie di azioni nel tempo, tali vincoli devono essere propagati per ogni istante, rendendo difficile l'applicazione del SAT per problemi di planning complessi.

Per formulare invece il problema con PDDL dobbiamo definire: nel file di domain l'ambiente del problema e le azioni che può svolgere l'agente, e nel file problem gli oggetti del problema. Questo rende molto più agevole la risoluzione di problemi di planning complessi e, permette di cambiare le specifiche (i.e., lo stato iniziale) del file problem per risolvere una versione differente del problema riferita allo stesso dominio. Il lavoro che svolge il solver di PDDL per modificare la definizione del problema può essere riprodotto tramite script generativi, in particolare bisognerebbe generare il file `moves.txt`, mostrato in Sezione 3.2.2.

I tempi medi di esecuzione dei due metodi, mostrati in Tabella 5, mostrano che la risoluzione tramite PDDL è più veloce rispetto a quella ottenuta con il SAT Solver.

Metodo	Tempo
SAT Solver	0.0030s
PDDL	0.0022s

Con i due metodi raggiungiamo lo stato finale con lo stesso numero di mosse, quello ottimo, ma attraverso due percorsi differenti.

6 Conclusioni

In questo progetto abbiamo visto come risolvere un problema di planning con due metodologie differenti: SAT e PDDL.

L'obiettivo del problema, come illustrato in Sezione 2, era quello di fornire ad un robot giardiniere tutte le capacità (i.e., azioni consentite) e le conoscenze necessarie (i.e., struttura dell'environment, posizione degli oggetti) per permettergli di curare un giardino. In particolare, il robot doveva estirpare tutte le piante infestanti e innaffiare tutte le piante sane presenti nel giardino.

Quando abbiamo formalizzato il problema con SAT, rappresentandolo con una CNF, nonostante abbiamo trovato un piano soddisfacente, abbiamo notato lo svantaggio di dover definire tutti i vincoli relativi alle azioni e alle configurazioni ammissibili per ogni istante di tempo. Questo problema viene superato dalla risoluzione tramite PDDL, rendendo quest'ultimo approccio più conveniente per la risoluzione di problemi di planning.

7 Fruibilità del progetto

La repository GitHub al link `https://github.com/ABorghini/Planning-as-SAT-and-PDDL` contiene gli script Python del progetto e i file .pddl. Scaricando la repository ed eseguendo gli script è possibile provare a risolvere il problema tramite SAT e tramite PDDL.

Referenze

[1] A. Biere et al. *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications*. NLD: IOS Press, 2009. ISBN: 1586039296.

[2] M. Ghallab et al. "PDDL—The Planning Domain Definition Language". In: 1998. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.37.212.

[3] Malik Ghallab, Dana Nau e Paolo Traverso. *Automated Planning: Theory and Practice*. The Morgan Kaufmann Series in Artificial Intelligence. Amsterdam: Morgan Kaufmann, 2004. ISBN: 978-1-55860-856-6. URL: http://www.sciencedirect.com/science/book/9781558608566.

[4] Malik Ghallab et al. "PDDL - The Planning Domain Definition Language". In: (ago. 1998). URL: https://homepages.inf.ed.ac.uk/mfourman/tools/propplan/pddl.pdf.