Progetto di Sistemi Intelligenti

Cucina Nintendo

Arangio Paolo/Belloni Lorenzo

Cosa sono AP e PDDL: Abbiamo deciso di svolgere un compito sull'automated planning(AP), un processo di razionalizzazione di tutte quelle azioni, ragionamenti e calcoli necessari a svolgere un compito dinamico.
Attraverso un "macro linguaggio" di programmazione, Il PDDL(Planning Domain Definition Language), abbiamo scritto il corpo principale del programma detto Domain. All'interno di questo sono indicati:

- I tipi. Tutte le tipologie di variabili che è possibile incontrare e lavorare.
- I predicati. Degli stati che possono essere controllati e modificati che sia come precondizione di funzionamento di una azione o come effetto risultante dalla stessa.
- Le azioni. Una serie di comportamenti del planner con cui decide come interagire con il problema. Ogni azione ha delle precondizioni che devono essere vere per permettere lo svolgimento dell'azione e degli effetti. Questi modificano lo stato dei predicati per gestire l'andamento e le scelte del programma.

Non elencheremo ogni struttura presente nel pddl; le precedenti erano solo le più importanti per la comprensione.

Il nostro progetto: La nostra idea è quella di un piccolo ristorante interamente gestito da un solo chef di nome Mario. Egli lavora da solo quindi possiede un unico coltello e una sola padella. Possiede 5 zone lavorative:

- Il bancone, dove taglia gli ingredienti.
- Il fornello, dove li cuoce.
- Il lavandino, dove pulisce accuratamente i suoi strumenti tra la lavorazione di un ingrediente e un altro .
- Il magazzino, dove sono conservati gli ingredienti.
- Il tavolo, dove porta ogni ingrediente quando è pronto.

Visto lo spazio ridotto possiede un solo tavolo per la clientela, quindi può ricevere un solo ordine alla volta.

Il Codice PDDL: Di seguito indichiamo qualche esempio di azione e ne spieghiamo il funzionamento.

Move:

```
(:action move
    :parameters (?x - chef      ?y1 - location       ?y2 - location)
    :precondition (and
    (chef_location ?x ?y1)
    )
    :effect (and
    (not(chef_location ?x ?y1))
    (free ?y1)
    (chef_location ?x ?y2)
    (not(free ?y2))
    (increase (total-cost) 0.5)
    )
)
```

-Questa azione permette a Mario di muoversi per le varie zone della cucina. Controlla che quella location(per usare i termini del programma) non sia occupata da qualcun'altro* e ci si sposta aggiornando la propria posizione.

Chop:

```
(:action chop
    :parameters (?x -chef      ?y -ingredient  )
    :precondition (and
    (is_holding ?x knife)
    (ing_holding ?x ?y)
    (chef_location ?x counter)
    (not(cooked ?y))
    (not(chopped ?y))
    (not(dirty knife))
    )
    :effect (and
    (chopped ?y)
    (dirty knife)
    (not(needs_chopping ?y))
    (increase (total-cost) 2)
    )

)
```

Questa azione controlla che Mario abbia in mano coltello e ingrediente da tagliare, che non sia in mano a nessun'altro* ,che lo strumento non sia sporco e che si trovi al bancone.

Come risultato l'ingrediente è tagliato e viene marchiato come "non più da tagliare", il coltello si sporca e si incrementa di 2 il total cost(ne parleremo dopo)
Un'azione quasi identica è Cook che fa la stessa cosa ma piuttosto che tagliare con il coltello, cuoce con la padella.

Take_tool:

```
(:action take_tool
    :parameters (?x - chef      ?z - tool       ?y - location)
    :precondition (and
    (chef_location ?x ?y)
    (tool_location ?z ?y)
    (not(held ?z))
    (not(is_holding_tool ?x))
    )
    :effect (and
    (held ?z)
    (is_holding ?x ?z)
    (is_holding_tool ?x)
    (not(tool_location ?z ?y))
    (increase (total-cost) 2)
    )
)
```

Questa azione controlla che lo chef si trovi nella stessa posizione dell'oggetto che desidera prendere,che non abbia altre cose in mano e che quello strumento non sia in mano ad altri*. Come risultato l'oggetto viene preso in mano dallo chef e, per semplicità di gestione, non risulta più presente in alcuna location fino a che non viene appoggiato nuovamente da qualche parte. Anche qui vediamo l'aumento al costo.
Esiste l'azione complementare per gli ingredienti e, per entrambe, esiste l'opposta che li mette giù.

Parliamo ora del "total-cost". Nel PDDL è importante dare un "peso" agli step eseguiti ; per ogni azione che viene compiuta si assegna un costo che è dipendente dalla tipologia dell'azione. In particolar modo nel nostro progetto era intelligente indicare ogni azione con lo stesso costo, così che non avesse preferenze su quale azione svolgere.

Fase di scrittura e Testing: Svolgere questo progetto non è stato per nulla facile. In particolar modo è stato frustrante renderci conto di aver commesso alcuni errori durante la scrittura del codice, che ci hanno portato a soluzioni non valide dal

planner e ci hanno obbligato a riscrivere tutto almeno un paio di volte. In molte occasioni PDDL ha fornito piani errati e svolto azioni inutili.
Abbiamo faticato non poco a comprendere quali fossero le parti da modificare. Non realizzare appieno cosa stai sbagliando porta a questo processo laborioso e frustrante in cui tenti di modificare tutto un pezzo alla volta, finché qualcosa non cambia.
Ma pddl non è stato l'unico fattore di difficoltà. Per quanto internet e la IA offrano un aiuto incredibile con la programmazione più "standard", anche formare l'interfaccia grafica in python è stata una sfida che ci ha forzato ad alcuni compromessi.

Risultati tecnici: Il programma di pddl gestisce fino a 8 ingredienti richiesti nel goal. Questo numero non è scelto a caso, siccome il processing di domain e problem con la richiesta di un piano *ottimale* funzionava solo per goal estremamente piccoli e poco realistici. Il risultato del nostro planner è restituito con un processing s*ub-ottimale* che, quantomeno ad occhio umano, restituisce un plan pulito e senza azioni inutili. 8 è un numero per cui il planner restituisce un plan in tempi brevi(<10 sec). Vista la natura esponenziale di pddl, 9 ingredienti risultano tediosi e restituiscono un plan in circa 1 minuto e mezzo con processing sub-ottimale. Senza processing si ottiene un plan in poco meno di una dozzina di secondi anche per i goal più importanti.

Miglioramenti futuri: Come indicato alla fine di questo documento, l'idea iniziale di questo progetto era differente; prevedeva l'esistenza di un secondo chef di nome Luigi che lavorasse in contemporanea a Mario. Il primo miglioramento desiderato è sicuramente l'implementazione di questo nuovo lavoratore.
Si potrebbero aggiungere nuove tipologie di lavorazione del cibo oltre al taglio e alla cottura, come ad esempio la tostatura o il mescolamento di più ingredienti per formarne uno nuovo.
Si potrebbero aggiungere nuovi strumenti per aumentare la profondità del lavoro, così da richiedere più azioni per ogni ingrediente.
Un'implementazione divertente sarebbe un grado di cottura del cibo scandito nel tempo, così che uno chef sbadato possa effettivamente bruciare gli ingredienti da lui lavorati se spende troppo tempo a fare altro.
Per ultimo, e come indicato nel precedente punto, una razionalizzazione nel tempo. Questo potrebbe portare a più azioni svolte in contemporanea che necessitano di un tempo minimo per il proprio svolgimento.

***Un piccolo appunto finale:** Questo progetto è nato con un'idea leggermente differente, poi successivamente scartata per via di problemi durante lo sviluppo, motivo per il quale all'interno del programma in codice sono presenti delle caratteristiche che potrebbero sembrare inutili o errate. Alcune di queste incertezze sono state modificate e adeguate all'attuale progetto. Altre, si voglia per tempo o per complicatezza sono state mantenute perché sono, al più, ridondanti e non generano alcuna problematica.