Politecnico di Milano
Scuola di Ingegneria dell'Informazione
Corso di laurea in Ingegneria Informatica

FabuPlan: un planner multimodale per la generazione di fabule

Tesi di Laurea di:
Canio Massimo TRISTANO Matr. 755382

Relatore: Prof. Marco COLOMBETTI

Anno Accademico 2011-2012

No, non mi interessa sviluppare un cervello elettronico potente.
Mi accontento di uno mediocre, un po' come quello del presidente dell'AT&T.

Alan Turing

Indice

Elenco delle figure

Questo documento presenta un generatore di fabule, in forma simbolica, basato su planning ad ordinamento parziale. Affinchè l'output generato sia una fabula di una storia e non un piano di azioni, è stato studiato un linguaggio multimodale sufficientemente espressivo per modellare la conoscenza e l'intenzionalità dei personaggi, in modo che la sequenza di eventi non soltanto sia logicamente coerente, ma anche credibile.

–

This document presents a symbolic fabula generator based on partial order planning. In order to generate a story fabula, and not a plan of actions, we have designed a multimodal language expressive enough to model characters knowledge and intentionality, so that the sequence of events feels not only logically coherent but believable too.

Introduzione

La narrativa è senza dubbio una delle abilità degli esseri umani più antiche ed importanti. Molto prima della nascita della scrittura, le storie venivano raccontate per condividere episodi di caccia, capacità ed esperienze acquisite, tramandandole alle generazioni successive per via verbale. L'uomo moderno continua a usare la composizione, comunicazione e comprensione di storie per una moltitudine di scopi: dall'educazione dei propri figli a quella di classi di studenti, per la condivisione di episodi di vita vissuta, per l'intrattenimento attraverso vari media (romanzi, film, videogiochi), per la socializzazione.

Non sorprende dunque che l'importanza di questa attitudine negli esseri umani abbia stimolato i ricercatori a studiare come le storie vengono costruite, se abbiano una struttura più o meno generale, quali sono le tecniche narrative più frequenti, come vengono decodificate, interiorizzate, memorizzate.

L'Intelligenza Artificiale nasce negli anni '60 come nuova area di ricerca interdisciplinare sullo studio delle capacità della mente umana e dei sistemi formali in grado di replicare tali funzionalità. Data l'importanza del linguaggio naturale negli esseri umani, ben presto i ricercatori di IA cominciarono a studiare i meccanismi inerenti la formulazione, la comunicazione e la comprensione del linguaggio naturale. Da questi studi, durante il decennio successivo, studiosi di IA e di narratologia gettarono le basi sullo studio metodico della narrazione e su ipotetici sistemi in grado di replicare un particolare aspetto del complesso sistema di facoltà che insieme compongono la narrativa negli esseri umani, di fatto originando una nuova area di ricerca chiamata *Intelligenza Narrativa*.

Oggi, possiamo dire che una parte consistente dell'industria dell'intrattenimento ha a che fare con la narrazione. L'industria dell'editoria, dello spettacolo, del cinema e, più recentemente, dei videogiochi sono accumunate dal fatto che essenzialmente tutte raccontano storie, sebbene il mezzo di comunicazione sia molto differente. Mentre i mezzi classici (il libro, il film, il teatro) svolgono una comunicazione unidirezionale (verso l'audience), i più recenti mezzi digitali consentono un flusso di informazioni bidirezionale, aggiungendo *interattività* allo storytelling. È il caso del videogioco, che ha registrato un'inarrestabile crescita di interesse da parte dell'industria e dei consumatori.

Una delle misure di qualità di un gioco è la *rigiocabilità*, ovvero il grado di interesse da parte di un giocatore a rigiocare il gioco una volta concluso, poiché questo presenterà situazioni di gioco distinte da quelle già vissute. Un esempio di gioco con alta rigiocabilità sono gli Scacchi, dove ogni partita è sensibilmente differente dalle altre per via dell'elevato numero di possibili situazioni di gioco. Ad oggi, le trame dei videogiochi vengono definite mediante la composizione di un insieme di *script* da parte dei progettisti, programmi che definiscono volta per volta cosa accade nel mondo di gioco. Sebbene la rigida definizione degli eventi di un gioco via scripting consenta agli sviluppatori di avere completo controllo sulla storia in modo da garantire un determinato livello di qualità, ogni esecuzione del gioco mostra poca o nessuna variabilità dalle altre, limitando il numero di situazioni in cui il giocatore può trovarsi e, dunque, la rigiocabilità. Inoltre un maggior coinvolgimento del giocatore si potrebbe ottenere adattando automaticamente la storia del gioco ai suoi gusti e preferenze. Questi fattori, insieme ad altri, giustificano la ricerca nei sistemi informatici di *generazione di narrativa*.

I narratologi dividono la storia in due livelli: *fabula* e *intreccio*. La prima è composta dagli eventi della storia in ordine cronologico, mentre il secondo esprime gli eventi secondo l'ordine in cui vengono narrati. Sono stati studiati diversi approcci alla generazione di fabule che si distinguono principalmente dal punto di

vista dell'agente che produce narrativa. Se l'agente è unico e distinto dai protagonisti e ha totale controllo sugli eventi e sulla struttura della fabula, l'approccio è di tipo *deliberativo* o *top-down*. Negli approcci *simulativi* o *bottom-up* la storia *emerge* dalla simulazione di un sistema multiagente dove ogni agente è indipendnte dall'altro ed è associato ad un singolo personaggio.

Il presente documento descrive FabuPlan, un sistema di generazione di fabula deliberativo. FabuPlan, come il nome suggerisce, fa uso di un planner per la costruzione di una fabule sotto forma di piano d'azione. Il planning rappresenta un idoneo approccio a questo compito data la sua capacità di costruire sequenze di azioni o eventi in modo che il risultato finale garantisca in modo coerente la loro causalità. Nel planning classico, il piano soluzione di un problema viene costruito supponendo che esista un solo agente che svolge le azioni oppure vi siano più agenti tutti cooperanti per il raggiungimento degli obiettivi. Queste assunzioni sono troppo limitanti quando il piano generato va inteso come fabula di una storia in quanto in una narrazione i protagonisti non sono né necessariamente cooperanti né necessariamente antagonisti, ma hanno intenzioni e desideri personali. Mentre un piano d'azione va eseguito da uno o più agenti, una storia andrebbe raccontata, e tra le due attività esistono misure di valutazioni differenti, come ad esempio la brevità che sebbene sia desiderata nel piano d'azione di una agente, non rappresenta un indice di bontà nella valutazione di una storia. Inoltre occorre poter modellare la conoscenza del mondo di ciascun protagonista in modo da generare situazioni interessanti. Per questi e altri motivi, FabuPlan contiene caratteristiche che lo differenziano da un planner classico.

Il prodotto di FabuPlan è una fabula sotto forma di grafo di eventi, il primo di tre passi necessari, a vista dell'autore, per la costruzione di una storia completa. Il secondo passo si occuperebbe di progettare un sistema per convertire la fabula in intreccio, che ordina la narrazione degli eventi in modo da suscitare specifiche emozioni nell'audience mentre il terzo di convertire il formato simbolico dell'out-

put dell'intreccio, adatto al calcolo automatico, in uno naturale più consono alla comunicazione agli esseri umani, come ad esempio un testo in linguaggio naturale. Sia il generatore di intreccio che il convertitore in formato natuarale sono al di là degli obiettivi del presente lavoro.

Struttura del documento

L'esposizione del lavoro svolto è suddivisa nei seguenti capitoli:

- ◇ **Capitolo 1.** Viene introdotta l'Intelligenza Narrativa, la sua storia e gli obiettivi, si argomentano le motivazioni che giustificano il presente lavoro, viene mostrata una panoramica dei lavori correlati più importanti e si definisce l'obiettivo a lungo termine di un sistema più complesso di cui FabuPlan fa parte.

- ◇ **Capitolo 2.** Si motivano le scelte tecniche e di design operate durante la progettazione di FabuPlan e vengono illustrate le caratteristiche del planner e del linguaggio di definizione di un problema.

- ◇ **Capitolo 3.** Viene descritto formalmente l'algoritmo di planning e vengono illustrati i problemi tecnici più rilevanti.

- ◇ **Capitolo 4.** Si esemplificano una serie di casi d'uso dei vari costrutti supportati e viene illustrata l'esecuzione dell'algoritmo durante la generazione di una fabula d'esempio.

- ◇ **Conclusioni.** Vengono riesaminati gli obiettivi in rapporto al lavoro svolto e si mostrano le possibili direzioni di sviluppo futuro.

Capitolo 1

Intelligenza Narrativa

Il presente capitolo introduce al lettore alcuni concetti fondamentali in merito all'Intelligenza Narrativa, alle motivazioni del presente lavoro e ad una panoramica sui precedenti lavori nel campo.

1.1 La narrativa negli esseri umani

La narrativa costituisce uno strumento di assoluta importanza in tutte le fasi della vita di un essere umano. La capacità di produrre narrazioni nasce, sebbene in modo molto rudimentale, insieme all'apprendimento delle prime capacità verbali [10].

Durante l'infanzia le storie rappresentano un primo e insostituibile mezzo di insegnamento, educazione (il famigerato "uomo nero"), intrattenimento e trasmissione della conoscenza del mondo. Le storie sono indispensabili per i bambini affinché riescano a dare un senso al mondo e ad apprendere metodi di approccio alla vita. La narrativa costituisce allo stesso modo un importante elemento della vita degli adolescenti e degli adulti: le storie vengono raccontate per condividere esperienze – nel caso di avvenimenti vissuti – per scopi educativi o, più banalmente, per fini ludici-intrattenitivi. Mentre un tempo le storie erano raccontate unicamente per via orale o, successivamente, attraverso la scrittura e lettura di libri, oggi conosciamo una moltitudine di complessi e moderni canali di comuni-

cazione, specialmente efficaci nell'intrattenimento, come il teatro, il cinema e più recentemente, i videogiochi.

Nonostante i media abbiano subito nel tempo una graduale evoluzione, la narrativa è rimasta una delle attività umane in grado di mantenere invariate le sue funzioni e i suoi obiettivi.

1.2 Intelligenza Narrativa

Vista l'importanza del linguaggio naturale negli esseri umani, molti ricercatori negli anni '70, spinti dall'entusiasmo della nuova e promettente Intelligenza Artificiale nata pochi anni prima, avviarono la ricerca sui processi e le strutture mentali degli esseri umani per comprendere il linguaggio naturale. Ben presto questi studi mostrarono come fosse complesso estrarre il significato delle frasi prese singolarmente e come, in realtà, fosse assolutamente indispensabile associare le frasi tra loro, combinandole con la conoscenza del dominio e il contesto del discorso, per comprenderne i significati. Questi risultati dirottarono la ricerca verso lo studio della comprensione e generazione della narrativa.

1.2.1 Generazione della narrativa

I successivi studi di Intelligenza Artificiale nel campo della narrativa, combinati con il crescente sviluppo e importanza dell'interazione uomo-macchina, portarono negli anni '80 alla definizione di una branca dell'IA chiamata *Intelligenza Narrativa* che, a sua volta, ricopriva una moltitudine di campi, come le *interfacce narrative*, il *design di agenti narrativi* e i sistemi di *story-understanding* [10]. Tra i campi studiati, quello che più da vicino riguarda l'oggetto del presente lavoro è quello della *generazione della narrativa*.

L'oggetto di studio della *narrativa computazionale* volta alla generazione di storie è, come il nome suggerisce, quello di studiare sistemi automatici in grado di comporre narrazioni in un ben definito dominio con il minor intervento umano

possibile. La definizione di cos'è una "'narrazione"' comporta una trattazione più dettagliata, discussa nel paragrafo 1.3.

1.2.2 Motivazioni

Le ragioni che spingono i ricercatori a procedere in questa direzione sono giustificate dal fatto che la narrazione rappresenta un elemento importante per gli esseri umani, in quanto consente loro di comunicare, intrattenere ed educare. I sistemi computazionali in grado di manipolare e generare storie, declinando la narrazione in tutte le sue forme, forniscono un miglioramento nell'interazione uomo-macchina, rendendo i computer migliori comunicatori, intrattenitori ed educatori.

Ma la chiave di volta che ha indotto a investire sforzi e denaro verso l'indagine di nuove tecniche narrative più interattive ed evolute, è stata l'applicazione di queste stesse tecniche alle così dette *applicazioni di intrattenimento interattivo-ludiche* meglio conosciute come *videogeochi*. La crescente utenza che è stata registrata negli ultimi anni traina la ricerca ed il mercato ad un'attento sviluppo di moderni sistemi computazionali per la generazione di storie, da applicare nel campo del gaming. Infatti un sondaggio del 2011 prova che il 72% delle famiglie americane gioca con i videogiochi [1], registrando una crescita del 3% rispetto allo stesso sondaggio eseguito nel 2006.

Ad oggi, l'approccio standard per definire una trama in un videogioco avviene mediante la definizione di una collezione di *script* da parte degli sviluppatori che, tuttavia, causano in ogni esecuzione poca o nessuna variazione alla trama. Questa caratteristica, sebbene fornisca agli sviluppatori maggior controllo sulla qualità della trama, mina fortemente la *rigiocabilità*[1], presentando sempre una struttura fissa per ogni giocatore e per ogni sessione di gioco. Le storie scriptate hanno inoltre la debolezza di poter risultare troppo complesse per un determinato tipo di giocatore o troppo semplici per un altro e non hanno nulla a che fare con

[1]La *rigiocabilità* rappresenta l'effettivo interesse del giocatore a rigiocare il gioco dopo averlo completato con successo.

i veri interessi del giocatore e i suoi gusti. Un sistema configurabile dal giocatore stesso ad avvio della sessione di gioco o, meglio, che si adatti automaticamente e dinamicamente al suo personale stile di gioco non soltanto aumenta considerevolmente l'intimità del rapporto tra giocatore e gioco, ma aumenta anche la sua rigiocabilità di quest'ultimo. Infine, un sistema di narrativa generativa consente ai giocatori di giocare una storia non prevista dagli sviluppatori stessi.

1.3 La storia

Se dunque l'obiettivo è lo *storytelling*, il prodotto finale generato dal sistema dovrebbe essere una storia completa. Nasce la questione di definire cos'è una storia, di quali parti è composta e, naturalmente, come costruire le parti necessarie. [15] definisce una storia come il "racconto di una sequenza di eventi che hanno un soggetto continuo e costituiscono un intero". Ai fini pratici, è più interessante la suddivisione comunemente accordata dai narratologi in due livelli: *fabula* e *intreccio* (*sjuzet* in inglese) [2].

1.3.1 Fabula

La fabula, in una storia, è la sequenza di eventi espressi in ordine (totale o parziale) cronologico. Nelle favole, o generalmente nelle storie per bambini, la fabula spesso coincide con la storia stessa, ovvero, la storia racconta gli eventi nell'ordine cronologico in cui avvengono.

Il tipo di rappresentazione schematica di una fabula dipende dal tipo di ordinamento temporale tra gli eventi della storia. Se l'ordinamento è totale, ovvero per ogni evento è noto quali eventi accadono prima e quali dopo, è sufficiente l'adozione di una lista che ordini secondo una sequenza cronologica gli eventi. Se la relazione di ordinamento temporale definito tra gli eventi è invece parziale, ovvero è noto solo quali eventi necessariamente accadono prima di un altro, una struttura a grafo è sicuramente più indicata, in quanto rende immediatamente vi-

sibili le sequenze di eventi che accadono in parallelo ad altre (ad esempio quando due personaggi svolgono una sequenza di azioni indipendentemente). Come verrà discusso successivamente, il sistema progettato appartiene al secondo caso, poiché definisce soltanto parzialmente l'ordinamento tra gli eventi.

1.3.2 Intreccio

Nelle favole, la fabula coincide con la storia narrata, ma le favole non rappresentano l'unico caso in cui non è necessario (e probabilmente neanche desiderato) avere un più complesso ordinamento degli eventi: nel caso di un videogioco, potrebbe essere sufficiente disporre della fabula, se la storia narrata è *locale*, ad esempio è stata vissuta ed è raccontata da un personaggio del gioco direttamente al giocatore. In altre circostanze la fabula non è sufficiente, ad esempio quando la storia è di più ampio respiro, come nel caso della trama di un libro, di un film o del videogioco stesso. In questo caso l'obiettivo della storia è quello di evocare sensazioni nell'"audience", ma la fabula da sola non è sufficiente per realizzare molti degli artifici narrativi in grado di trasmettere emozioni.

Per poter fare ciò, il secondo livello è occupato dall'intreccio. Questo è costituito dall'ordine in cui effettivamente gli eventi vengono raccontati nella storia. Libri e film fanno ampio uso di tecniche narrative per manipolare le emozioni dell'audience e per più efficacemente raccontare la storia. Due esempi sono l'*analessi*, o flashback, il racconto di fatti accaduti cronologicamente nel passato, e la *prolessi*, racconto di fatti accaduti nel futuro, anticipati.

1.4 Approcci

Gli approcci sino ad oggi considerati per la generazione di narrativa sono sostanzialmente di due tipi: *bottom-up* (via *simulazione*) e *top-down* (via sistema di *reasoning* centrale).

1.4.1 Approccio bottom-up

Questo approccio si basa sull'idea di ottenere una storia come prodotto di una simulazione del mondo nel quale la storia è ambientata e dei suoi personaggi. In altre parole, i personaggi sono *agenti* dotati di un sistema comportamentale e di decision-making, in grado di programmare la sequenza di azioni da svolgere per raggiungere un obiettivo personale nel mondo. Tale obiettivo a sua volta, evolve in seguito agli effetti delle azioni svolte dai personaggi. Questo approccio viene anche chiamato *emergente*, in quanto la storia emerge dall'interazione dei personaggi con il mondo.

Il primo lavoro che seguì questa strada fu Tale-Spin negli anni '70 [11], un sistema che genera fiabe sullo stile di quelle di Esopo, scegliendo momento per momento le azioni di ogni personaggio con un motore di inferenza basato su teorie sul *ragionamento di senso comune*. A causa della località della ricerca, in certe circostanze Tale-Spin genera fiabe mal strutturate e troppo brevi come, ad esempio, in caso di cattiva scelta dell'azione da svolgere. Il più recente Oz[9][3] del Carnegie Mellon University fa uso di un sistema di agenti reattivi e autonomi per rappresentare tutti i personaggi in un mondo dinamico. Gli agenti sono dotati di procedure di decision-making "shallow and broad" tali da coprire un vasto repertorio di attività credibili dei personaggi [4] [5]. Per correggere i problemi precedentemente citati di Tale-Spin, Oz fa uso di un agente speciale, chiamato *drama manager*, incaricato di osservare la storia in generazione e prevenire narrative mal strutturate o scarsamente interessanti, che emergono dall'interazione tra gli agenti. Infine il sistema I-Storytelling [6], come il sistema Oz, fa uso di agenti autonomi e reattivi, ma a differenza di questo fa uso di un planner Hierarchical Task Network (HTN) per raggiungere obiettivi pre-determinati.

Tutti questi sistemi in un approccio *divide-et-impera* decompongono il problema generale di costruire una storia interessante costruendo agenti autonomi in grado di prendere decisioni in virtù di un obiettivo esterno e di combinare il

risultato in modo da originare una storia interessante e ben strutturata. Per questo motivo, l'approccio bottom-up produce risultati meno validi quando l'analisi locale non è sufficiente, ma è necessario un punto di vista globale.

1.4.2 Approccio top-down

Duale al precedente, questo approccio, chiamato anche *deliberativo*, parte da una base di conoscenza più generale e globale rispetto al punto di vista dei personaggi, per generare direttamente la storia senza decomposizione. MINSTREL, basandosi su concetti già esistenti e predeterminati come storie pre-esistenti, utilizza un sistema di creatività computazionale per manipolare tale base di conoscenza, ottenendo nuove storie. In modo simile MÉXICA combina frammenti di storie già esistenti che corrispondono al pattern corrente di emozione e tensione, con un planner ad ordinamento parziale per mantenere l'integrità causale tra i frammenti inseriti.

Avvicinandoci al metodo adottato dal presente lavoro, Porteous e Cavazza [14] adottano una variante del planner *Fast-Forward* per trovare una sequenza di eventi, che costituiscono la storia, da una descrizione di mondo iniziale a quella finale. Invece di considerare gli obiettivi della storia distinti da quelli dei personaggi, il sistema permette all'autore di definire dei *landmark* del mondo attraverso i quali la storia generata deve passare, per convogliare la soluzione verso la struttura da lui desiderata.

Il sistema al quale il presente lavoro più si avvicina e con il quale più direttamente si confronta è FABULIST [16]. Riedl e Young usano una variante del planner UCPOP, un planner STRIPS ad ordinamento parziale, per generare la fabula di una storia via planning, a partire da un modello iniziale del mondo, in modo che gli obiettivi della storia definiti dall'autore siano soddisfatti quando la storia termina. La modifica apportata a UCPOP consente al planner di mantenere gli obiettivi dei personaggi distinti da quelli della storia, modellando efficacemente le intenzioni dei personaggi e, dunque, la loro credibilità.

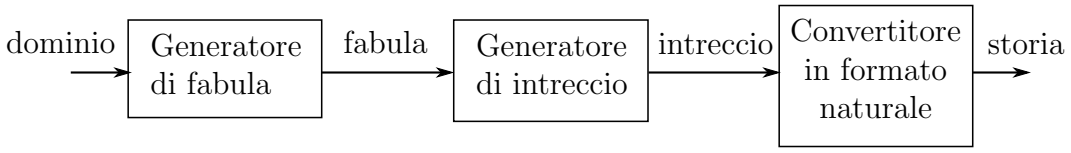


Figura 1.1: *Schema del sistema completo.*

1.5 FabuPlan e obiettivo a lungo termine

FabuPlan rappresenta il primo passo di un lavoro più ampio, che si pone l'obiettivo di generare un racconto in linguaggio naturale, partendo da una base di conoscenza definita dall'autore. Per poter raggiungere quest'obiettivo, nella visione dell'autore il sistema finale deve essere composto da tre sottosistemi: un generatore di fabula, un generatore di intreccio e un convertitore in un formato naturale.

1.5.1 Generatore di fabula

Questo sistema parte da una definizione di mondo iniziale e finale e costruisce il grafo degli eventi parzialmente ordinati. Questo è l'obiettivo di FabuPlan, l'oggetto del presente lavoro, il quale verrà illustrato dettagliatamente nei capitoli successivi.

1.5.2 Generatore di intreccio

Il passo successivo comprenderebbe lo sviluppo di un sistema di generazione di intreccio. Il sistema prende in input il grafo di eventi generato da FabuPlan e mette in sequenza gli eventi secondo l'ordine della narrazione (totalmente ordinato). Il sistema dunque si propone di *linearizzare* il grafo della fabula in modo opportuno costruendo un intreccio interessante che generi emozioni, tensione, suspense nell'audience.

1.5.3 Convertitore in formato naturale

L'output ottenuto dal *generatore di intreccio* è una sequenza di strutture dati che descrivono, simbolicamente, gli eventi della storia. Questo formato non è naturalmente idoneo alla *comunicazione* della storia stessa. La storia potrebbe essere convertita in un testo in inglese ad esempio, comunicando il racconto attraverso la lettura. Allo stesso modo, l'intreccio potrebbe venir convertito in un formato adatto al playback in un ambiente grafico virtuale, ad esempio animando i personaggi virtuali in un videogioco. Anche questa fase è al di là degli obiettivi del presente lavoro.

Capitolo 2

FabuPlan: un planner multimodale

In questo capitolo è illustrata l'architettura di FabuPlan, sono descritte in dettaglio le sue caratteristiche e funzionalità supportate e motivate le scelte tecniche e di design.

2.1 Scelte principali

Questa sezione descrive FabuPlan ad alto livello, soffermandosi sulle soluzioni adottate durante la sua progettazione.

2.1.1 Approccio

FabuPlan si inserisce nella schiera dei sistemi di generazione di narrativa top-down deliberativi. Avendo un unico sistema centrale incaricato di generare la fabula, si ottiene più controllo sulla scelta delle azioni svolte dai personaggi mantenendo una visione globale e, dunque, ottenendo più controllo sulla struttura finale della fabula. In un sistema simulativo, infatti, bisognerebbe avere un componente che a posteriori giudichi il racconto emerso, accettandolo o rifiutandolo in base a dei criteri prestabiliti, oppure che interagisca con i piani d'azione elaborati da ciascun agente manipolandoli affinché la storia finale abbia un aspetto coerente e globalmente sensato. I sistemi deliberativi, invece, non lavorando su informa-

zioni temporalmente localizzate ma globali, evitano i problemi della progressione logica e di struttura perché ragionano sulla struttura della narrazione nella sua interezza.

2.1.2 Planning

In linea al lavoro di Riedl e Young, FabuPlan produce una fabula sotto forma di grafo di eventi simbolici attraverso il *planning*. Informalmente, in Intelligenza Artificiale un sistema di planning è un programma che costruisce un piano d'azione coerente, una sequenza ordinata o grafo aciclico di azioni, che una volta eseguito consente di raggiungere degli obiettivi finali preimpostati, a partire da una descrizione iniziale del mondo. Un esempio storicamente importante di planner è STRIPS (STanford Research Institute Problem Solver), strumento realizzato negli anni '70 all'Università di Stanford per la costruzione di piani di azione di un robot.

Il motivo principale per la scelta di implementare FabuPlan come planner risiede nel fatto che grazie ad esso è possibile disaccoppiare la descrizione del mondo con quella degli eventi verificabili ed esprimere ciascuno in modo molto modulare e facilmente estendibile, attraverso l'uso di una logica. La potenza espressiva del planner con la quale descrivere mondo ed eventi dipende fortemente dalla potenza espressiva dalla logica adoperata. STRIPS rappresenta la conoscenza mediante predicati del prim'ordine: ad esempio, il fatto che una particolare penna è su un particolare tavolo in STRIPS può essere espresso con

$$On(pen, table)$$

dove *pen* e *table* sono due termini (variabili o costanti individuali). Sia il mondo iniziale e che gli obiettivi finali vengono definiti mediante un insieme di predicati del prim'ordine.

Oltre a questi due elementi, un planner in senso stretto necessita anche di una libreria di schemi di azioni che l'agente può svolgere per raggiungere gli obiettivi.

Quest schemi di azione chiamati in STRIPS *operatori*, sono rappresentati con un insieme di *precondizioni*, fatti che devono essere verificati nel mondo un istante prima che l'azione sia eseguita, ed *effetti*, condizioni del mondo che cambiano una volta che l'azione è stata compiuta dall'agente.

È importante osservare che FabuPlan utilizza il planning seguendo un approccio deliberativo, a differenza di I-Storytelling che adotta un planner (di tipo Hierarchical Task Network) in un approccio simulativo. La differenza risiede nel fatto che mentre I-Storytelling usa il planner per la costruzione dei piani di azione di ciascun personaggio come singolo agente autonomo e indipendente per poi combinarli in una fabula, FabuPlan costruisce direttamente la fabula, pianificando contemporaneamente per *tutti* i personaggi. Per questa ragione, in FabuPlan il termine "azione" è improprio e viene invece usato il termine "evento", per sottolineare che può coinvolgere un agente, più agenti o nessuno.

Causalità

Un requisito essenziale affinché un piano o un racconto abbia senso è che naturalmente le precondizioni degli eventi siano vere prima che l'evento si verifichi (ad esempio per piovere, vi devono essere nuvole), detti anche vincoli di causa-effetto (causalità). Tali vincoli non sono rispettati se esiste una precondizione di un evento istanziato non vera nel mondo nell'istante precedente all'evento, e questo può verificarsi sia quando la precondizione non è mai resa vera a partire dal mondo iniziale, sia quando viene resa vera per effetto di un evento precedente, ma tra questo evento e il primo può accadere un evento che la rende falsa. Questo è naturalmente un requisito fondamentale affinché la storia non sia una collezione di eventi puramente casuale, ed è garantito dall'algoritmo di planning adottato se questo è *corretto*, ovvero se ogni soluzione trovata rispetta i vincoli di causa-effetto. FabuPlan è corretto, essendo basato sull'algoritmo UCPOP la cui correttezza è ben nota[13].

2.1.3 Tipologia di planner

La teoria del planning gode di notevole maturità dato il suo uso diffuso nella pianificazione aziendale. Ad oggi esistono numerosi planner molto differenti tra loro per complessità ed espressività. Un modo per classificare i sistemi di planning esistenti riguarda il modo in cui le azioni costituenti il piano sono ordinate tra loro nel piano in output.

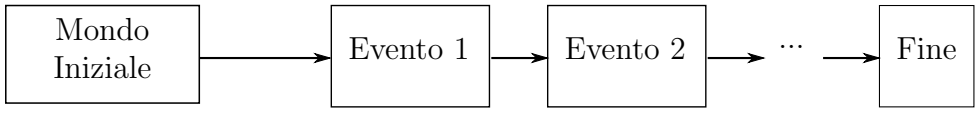
Planner ad ordinamento totale

Un primo approccio consiste nel mantenere le azioni del piano totalmente ordinate, esprimendo il piano soluzione come *sequenza* di eventi, dal primo eseguito all'ultimo, in ordine cronologico (esempi sono SHOP, FAST-FORWARD, GRAPH-PLAN, SAT-PLAN). Formalmente per ogni coppia (x, y) di eventi nel piano necessariamente $x \prec y$ o $y \prec x$ dove $\prec$ è la relazione di ordinamento temporale ("accade prima di"). Sebbene un piano lineare vada bene nel planning classico in cui l'obiettivo è un piano d'azione *eseguito* da uno o più agenti, il totale ordinamento rappresenta una restrizione eccessiva nel caso in cui il parallelismo dei sottopiani non sia indesiderato o sia addirittura desiderato. Si potrebbe tuttavia analizzare successivamente la sequenza di eventi individuando gli eventi non in conflitto e parallelizzandoli, al costo di implementare questo componente aggiuntivo.

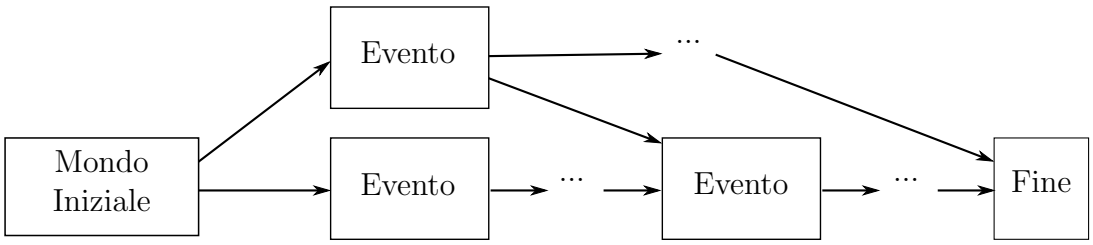
Planner ad ordinamento parziale

L'altra classe di planner produce uno piano dove le azioni sono *parzialmente ordinate* (UCPOP, REPOP). Questi planner chiamati Partial Order Causal Link (POCL), nativamente producono un piano con il minor numero possibile di vincoli di ordinamento tra gli eventi, pur mantenendo consistenti i vincoli di causalità. Se tra due eventi non è definito l'ordinamento relativo, i due eventi possono accadere in qualunque ordine o addirittura in parallelo. Il piano non ha dunque la forma di una lista, ma di un grafo.

Nel capitolo precedente si è mostrato che per costruire un intreccio a partire da una fabula non è sufficiente disporre degli eventi della storia in sequenza, ma occorre una struttura a grafo che mantenga l'ordinamento relativo tra gli eventi. Inoltre è noto che la mente di uno scrittore umano compone storie in ordinamento parziale per poi ricombinarle in ordini di narrazione opportuni. Per entrambi i motivi, è stato scelto di progettare FabuPlan come planner ad ordinamento parziale: l'output prodotto è un grafo di eventi dove ogni arco, oltre a possibilmente contenere altre informazioni, impone un ordinamento temporale tra i due eventi collegati.



(a) Ordinamento totale.



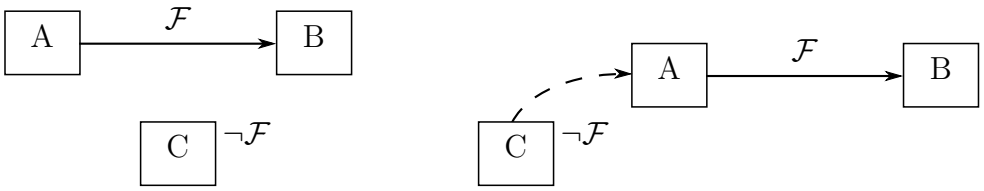
(b) Ordinamento parziale.

Figura 2.1: *Esempi grafici dei due tipi di ordinamento degli eventi.*

Per comprendere meglio i paragrafi successivi, viene illustrato informalmente come funziona l'algoritmo di planning alla base di FabuPlan, mentre una descrizione più formale è tratteggiata nel paragrafo 3.1.4. Il planner, come accennato in precedenza, prende in input una descrizione del mondo in un linguaggio logico, ad esempio nella forma di una congiunzione di formule atomiche, una descrizione parziale del mondo verificata quando la storia termina (gli obiettivi della storia) e una libreria di azioni (in FabuPlan eventi) che descrivono quello che può accadere nel mondo, quali sono le condizioni di applicabilità e quali sono i loro effetti sul mondo. Partendo dagli obiettivi specificati dall'autore, FabuPlan cerca di risolverli uno per volta unificando l'obiettivo corrente con un effetto già esistente

nel mondo iniziale, con un effetto di qualche già istanziato, oppure istanziando un nuovo evento opportuno. In quest'ultimo caso, le precondizioni del nuovo evento vanno ad aggiungersi alla lista degli obiettivi da risolvere. Quando questa lista è vuota, ovvero tutti gli obiettivi sono stati soddisfatti dagli eventi istanziati o dal mondo iniziale, una soluzione è stata trovata. Se un obiettivo è irrisolvibile, il planner non può trovare alcuna soluzione per il problema specificato.

La particolarità di un planner POCL risiede nel fatto che non è noto l'istante di tempo, misurato a partire dall'inizio della storia, in cui gli eventi accadono ma solo quale evento deve accadere prima di un altro (Partial Order). Quando una precondizione aperta (un obiettivo) di un evento e viene risolta dall'effetto di un evento f (già esistente o appena istanziato), viene creato un *collegamento causale* tra e e f. Questo link si dice che *protegge* l'effetto che soddisfa la precondizione. La protezione fa sì che nessun evento che ha come effetto la negazione dell'effetto protetto, o che in generale lo rende *falso*, può accadere tra e e f altrimenti la precondizione di f non sarebbe in realtà soddisfatta. Quando si verifica questo conflitto tra un evento g e un collegamento causale tra due eventi e e f, allora il planner deve risolvere il pericolo di inconsistenza in qualche modo. Un modo ad esempio è quello di vincolare l'evento g ad accadere prima di e o dopo di f in modo che non sia possibile che si verifichi proprio tra e e f. Se non c'è modo di risolvere il conflitto, il sottopiano corrente è inconsistente e viene scartato.



(a) Il collegamento causale che protegge $\mathcal{F}$ tra gli eventi A e B è minacciato da un evento C avente l'effetto $\neg\mathcal{F}$.

(b) Il planner risolve il conflito vincolando C ad accadere prima di A.

Figura 2.2: *Esempio di conflitto tra collegamento causale ed evento. L'arco solido indica il collegamento causale, quello trattegiato il vincolo di ordinamento.*

2.1.4 Requisiti di espressività del linguaggio

Conoscenza

Un planner classico a la STRIPS non fornisce alcun modo per specificare la conoscenza dei singoli agenti. È possibile asserire che il personaggio John sia ricco, ad esempio con $Rich(john)$ ma non che Mary *creda* che John sia ricco. Per aggirare questa limitazione, uno potrebbe definire ulteriori predicati di conoscenza caso per caso. Ad esempio, uno potrebbe definire il predicato $ThinksIsRich(x, y)$ che significa appunto "x crede che y sia ricco". Oltre ad essere estremamente tedioso, il problema si complica ulteriormente se si volesse modellare la negazione, come in "x pensa che y *non* sia ricco" o si volesse poter esprimere una catena di credenze, come in "Mary pensa che John creda che Mary sia ricca": il numero di predicati da definire per modellare questi casi crescerebbe esponenzialmente dunque questo sistema è poco pratico.

Riedl e Young non si pongono il problema di rappresentare la conoscenza di ciascun personaggio. FabuPlan fa uso di un linguaggio più espressivo di STRIPS, discusso in seguito, per poter esprimere la conoscenza privata dei personaggi sullo stato del mondo.

Modellare la conoscenza è indispensabile per una moltitudine di situazioni fondamentali presenti in tutte le storie. Con un sistema per definire e manipolare la conoscenza individuale dei personaggi si possono modellare atti comunicativi, miscredenze, equivoci, inganni, menzogne, e via dicendo. Ad esempio la menzogna è modellabile con la comunicazione da parte di un personaggio di un fatto che lui creda essere non vero, ma che desidera che l'altro pensi essere vero.

Intenzioni e desideri

Come accennato in precedenza, il rispetto dei vincoli di causa-effetto è essenziale per avere una storia di senso compiuto. Questa proprietà è sufficiente se il planner viene usato come sistema di decision-making negli agenti come in I-

STORYTELLING. Ma quando il planner non pianifica le azioni di un personaggio ma gli eventi di una storia, la causalità non è più da sola sufficiente per modellare l'intenzionalità dei personaggi, per i seguenti motivi:

- ◇ Nel planning classico i piani sono costruiti per un singolo agente o una collezione di agenti cooperanti; in una storia i personaggi non sono né necessariamente cooperanti, né necessariamente antagonisti.

- ◇ Nel planning classico, tutti gli agenti (personaggi) *intendono* svolgere azioni che supportino il raggiungimento degli obiettivi finali; nella pianificazione di una fabula, non necessariamente i personaggi *intendono* dar luogo ad eventi per il raggiungimento degli obiettivi della storia, ma addirittura potrebbero avere obiettivi personali in conflitto con quelli della storia.

Ad esempio una storia può terminare quando due personaggi si sposano anche se inizialmente non è specificato che essi *desiderino* essere sposati. In FABUPLAN sebbene gli obiettivi della storia sono sempre pre-determinati e costanti, gli obiettivi dei personaggi possono essere definiti a priori o generati dinamicamente.

Consideriamo il seguente semplice esempio: supponiamo l'autore definisca l'evento *kill* nel quale un personaggio può ucciderne un altro. Supponiamo le (semplificate) precondizioni prevedano che l'omicida sia fisicamente vicino alla vittima e abbia in mano un coltello. L'effetto è naturalmente la morte della vittima. In STRIPS, l'evento verrebbe modellato con

Evento $kill(x, y)$:
Precondizioni: $Near(x, y)$, $Holds(x, knife)$;
Effetti: $Dead(y)$;

dove x è la variabile che rappresenta l'omicida, y la vittima. Ora supponiamo che l'autore abbia specificato nella descrizione del mondo iniziale che John ami Mary. Se l'obiettivo di questa triste storia è la morte di Mary, il planner istanzia l'evento *kill* assegnando $y = mary$. Il planner deve quindi assegnare alla variabile

libera x uno tra John e Bill (chi compie l'omicidio). Casualmente viene deciso $x = John$, quindi il planner risolve il fatto che John abbia in mano il coltello prima di commettere l'omicidio e che sia vicino a Mary. Il piano è coerente in quanto rispetta i vincoli di causa-effetto e soddisfa tutti gli obiettivi finali. Ma un lettore di questo frammento di storia rimarrebbe alquanto deluso dal fatto che John, apparentemente, non aveva nessun motivo per uccidere Mary, quando in realtà l'amava. Il planner ha coinvolto ciecamente i personaggi per completare gli obiettivi della storia ($Dead(mary)$), senza curarsi di dare una motivazione al perché un personaggio voglia che l'evento che causa avvenga. In una storia generata mediante planning classico, tutti i personaggi intendono partecipare a qualunque evento senza alcuna motivazione necessaria. Manca la capacità da parte del planner di pianificare le *intenzioni* dei personaggi in modo che l'audience percepisca come *credibile* il *comportamento* degli stessi. Questa proprietà è fortemente minata quando l'audience non è in grado di comprendere gli obiettivi dalle azioni dei personaggi e viceversa di giustificare le azioni dei personaggi conoscendone gli obiettivi.

Come per la conoscenza, uno potrebbe apparentemente risolvere la questione modellando l'intenzione e i desideri dei personaggi con predicati ad-hoc, come $WantsRich(x, y)$ per dire che "x desidera che y sia ricco". Oltre ai problemi analoghi a quelli della conoscenza menzionati in precedenza, si presentano ulteriori complicazioni voledo modellare i desideri: spesso un desiderio giustifica un altro, ad esempio un personaggio può voler avere in mano un coltello perché vuole uccidere un altro personaggio. Senza comunque un arricchimento dell'algoritmo di planning classico, questo importante fenomeno non sarebbe modellabile.

Riedl e Young risolvono il problema dell'intenzionalità modificando internamente un planner POCL in modo da supportare la pianificazione delle intenzioni. Quando un evento viene istanziato, viene scelto a caso un effetto e marcato come desiderio del personaggio *attore* dell'evento, cioè quello che razionalmente intende

l'evento. Stabilendo il nuovo desiderio del personaggio, il planner crea un *frame of commitment*, un intervallo di tempo nel quale viene asserito il nuovo desiderio del personaggio, che intende l'evento producente il fatto desiderato. Quindi il planner continua la pianificazione giustificando *perché* il personaggio debba desiderare quell'effetto[16].

Multimodalità

Ispirato agli Atti Comunicativi di Cohen e Perrault [7], FabuPlan segue il tradizionale approccio *Belief-Desire-Intention* (BDI) dei sistemi multiagente, risolvendo entrambi i problemi della conoscenza e del desiderio/intenzione via arricchimento del linguaggio STRIPS alla *multimodalità*. Per la precisione, il linguaggio adottato da FabuPlan è più simile a una logica *dinamica* del prim'ordine, in quanto gli operatori modali sono parametrizzati con un termine (individuo o variabile). Sebbene sia possibile definire nuovi *operatori modali*, FabuPlan ne usa due di default: l'operatore *believes* **b** e l'operatore *wants* **w**. Il primo indicato con $\mathbf{b}_t$ ha il significato "t crede che ...", mentre il secondo indicato con $\mathbf{w}_t$ ha associato l'interpretazione "t desidera che sia vero ...".

L'estensione alla multimodalità consente di esprimere fatti complessi come la credenza e i desideri dei personaggi circa lo stato del mondo, e gli eventuali intrecci di credenze e desideri. Se per dire che $john$ è ricco si usa $Rich(john)$ allora si esprimerebbe "Mary crede che John sia ricco" con $\mathbf{b}_{mary}Rich(john)$, "John vuole diventar ricco" con $\mathbf{w}_{john}Rich(john)$, "John desidera che Mary creda che lui sia ricco" con $\mathbf{w}_{john}\mathbf{b}_{mary}Rich(john)$ e così via.

È opportuno infine soffermarsi sul significato di *intenzione* contro quello di *desiderio*. In [7], le intenzioni sono modellate attraverso l'operatore modale *intends*. Nella teoria classica BDI dei sistemi multiagente, gli agenti comunicano con *intends* l'intenzione di svolgere una specifica *azione* e ciò è molto diverso dall'operatore modale *wants* che invece si applica a un fatto e che corrisponde al desiderio da parte dell'agente che il fatto desiderato sia vero nel mondo. L'agente $john$ non

intende $Rich(john)$, ma lo desidera. Probabilmente in virtù di questo desiderio intenderà l'azione $WorksHard$ in modo che $Rich(john)$,essendo effetto dell'azione, soddisfi il desiderio. FabuPlan non supporta le intenzioni in senso stretto, ovvero, non è esprimibile direttamente la volontà di un personaggio di compiere una specifica azione. Il motivo risiede nel fatto che questo è comunque esprimibile indirettamente aggiungendo un nuovo predicato con lo stesso nome e argomenti dell'azione negli effetti dell'azione e indicando l'intenzione del personaggio con l'operatore modale *wants* applicato a tale predicato.

2.1.5 Perché un nuovo planner?

Dopo un accurato studio dei più famosi approcci al planning (ricerca totale basato su euristica in avanti, in regressione, Hirarchical Task Network, via SAT Solver, ecc.), si è optato per un planner di tipo Partial Order Causal Link. Si sono considerati i due più recenti lavori sul planning ad ordinamento parziale, UCPOP[13] e RePOP[12]. Il primo è un elegante planner ad ordinamento parziale scritto in Common Lisp con supporto a un sottinsieme di ADL: effetti condizionali e precondizioni, postcondizioni e obiettivi quantificabili universalmente. Il secondo estende UCPOP aggiungendovi un euristica che guidi la selezione del prossimo evento parziale da processare nella ricerca, e altri accorgimenti per una maggiore efficienza.

Sebbene RePOP sia stato un ottimo candidato come punto di partenza per realizzare FabuPlan (il sorgente è libero), in ultima analisi è stato scelto di sviluppare un planner ad ordinamento parziale ex novo basato sulle idee di UCPOP e RePOP, entrambi implementati in Common Lisp. La scelta è motivata principalmente dal fatto che FabuPlan estende ampiamente il linguaggio e l'algoritmo di UCPOP, dalla multimodalità all'inferenza delle intenzioni implicite. La scrittura ex-novo di FabuPlan è stata fondamentale per padroneggiare le meccaniche interne del planning POCL, acquisendo la conoscenza tecnica necessaria per realizzare le funzionalità aggiuntive, in modo che l'intero sistema risultasse organico

ed efficiente. Inoltre FabuPlan rappresenta la prima parte di un sistema più complesso di generazione di narrativa destinato a media differenti, dal testuale ai videogames. Nell'industria ludica digitale, il linguaggio standard de-facto è il C++, linguaggio di alto livello orientato agli oggetti molto efficiente poiché compilato in linguaggio macchina. Inoltre per la sua natura, il C++ è facilmente interoperabile con una moltitudine di altri linguaggi di programmazione, il che lo rende il miglior linguaggio candidato.

Ciò nonostante, FabuPlan prende direttamente spunto dagli algoritmi e le idee dei due lavori menzionati, con l'aggiunta di nuove per supportare i prerequisiti di espressività stabiliti e, per quanto possibile, efficienza.

2.2 Panoramica del linguaggio e delle caratteristiche di FabuPlan

In questa sezione verrà illustrato il ventaglio di caratteristiche supportate dal linguaggio di FabuPlan per la definizione delle informazioni di dominio passate in input al planner.

2.2.1 Tipi

La prima estensione di STRIPS riguarda il supporto ai tipi delle costanti individuali e delle variabili. La tipizzazione è parzialmente esprimibile in STRIPS usando predicati unari che indicano che l'argomento è di un certo tipo, ad esempio $Person(john)$, e aggiungendoli alle precondizioni degli eventi per specificare il tipo delle variabili e nella dichiarazione del mondo per specificare il tipo delle costanti individuali. Questo espediente non è più sufficiente però quando si vuole classificare i tipi in una gerarchia mediante la relazione *is-a*. Sarebbe estremamente tedioso inserire esplicitamente nelle precondizioni una disgiunzione comprendente ogni possibile sottotipo del tipo desiderato.

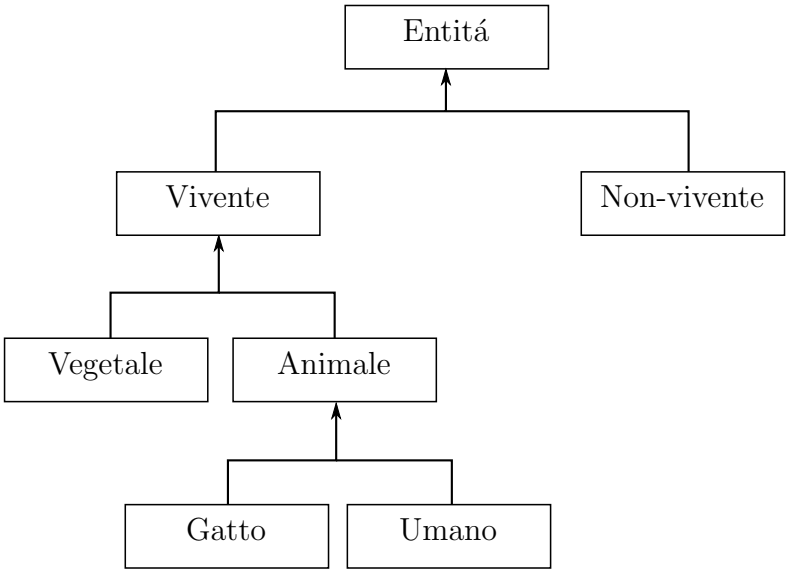


Figura 2.3: *Esempio di gerarchia di tipi. Le frecce indicano la relazione* is-a.

Per questo motivo, l'uso di tipi e sottotipi richiede che il planner li supporti nativamente. Nel linguaggio di definizione di FabuPlan è possibile dichiarare un tipo con l'espressione

```
type nome-tipo-1, nome-tipo-2, ...;
```

ad esempio

```
type Person;
```

Per indicare che un tipo è sottotipo di un altro, si utilizza la parola chiave `is`.

```
type Man, Woman is Person;
```

Quindi se all'interno della descrizione del mondo si dichiarano costanti individuali di tipo `Man` e `Woman`, questi saranno *disponibili* quando vi è una restrizione di tipo `Person`.

Per costruzione interna, FabuPlan richiede che la gerarchia dei tipi sia un grafo aciclico. Quindi non è consentito che un tipo sia, nella chiusura transitiva della relazione *is-a*, sottotipo di se stesso.

2.2.2 Fatti

L'elemento al cuore del sistema è senza dubbio il *fatto*, un'espressione simbolica di una parte dello stato del mondo espressa nel linguaggio predicativo del prim'ordine multimodale illustrato nella sezione precedente. Come abbiamo visto, esempi di fatti sono $Loves(john, mary)$, $\mathbf{b}_{john} Loves(john, mary)$, $\mathbf{w}_x At(x, home)$. Diamo anzitutto alcune definizioni preliminari.

Definizione 1. L'*alfabeto* del linguaggio include:

- ◇ simboli per denotare *variabili*: x, y, z, ...
- ◇ simboli per denotare *costanti individuali*: a_1, a_2, ...
- ◇ simboli per denotare *predicati* ciascuno con un fisso numero di argomenti: P_1, P_2, ...
- ◇ simboli per denotare gli operatori modali: $\mathbf{m}_1$, $\mathbf{m}_2$, ...

Nella sintassi di FabuPlan, i nomi delle variabili appaiono sempre precedute dal punto di domanda '`?`', ad esempio `?x`, `?person`, sono variabili (mentre `person` è considerata una costante individuale).

Diamo quindi la definizione di *termine*.

Definizione 2. :

- ◇ una costante individuale è un termine.
- ◇ una variabile è un termine.

Definizione 3. Si definisce $\mathbf{m}_t$ *operatore modale*, con $\mathbf{m}$ un simbolo di operatore modale e t un termine.

Schemi di fatti

Praticamente la teoria di tutte le logiche usano il concetto di *schema* per facilitare la scrittura di assiomi e tautologie. Ad esempio, nella logica proposizionale è una tautologia (*modus ponens*)

$$(\mathcal{A} \ \wedge \ (\mathcal{A} \rightarrow \mathcal{B})) \rightarrow \mathcal{B} \tag{2.1}$$

dove $\mathcal{A}$ e $\mathcal{B}$ sono *schemi di formule*, ovvero sostituendoli uniformemente in 2.1 con una qualunque sottoformula, il risultato è ancora una tautologia. La formula 2.1 indica in realtà un *insieme* (infinito nella logica proposizionale) di formule aventi questa struttura.

In FabuPlan non abbiamo schemi di formule ma di *fatti*. Si supponga di voler esprimere "se qualcuno desidera qualcosa allora sa di desiderarla", abbiamo bisogno di esprimere sinteticamente la struttura di questo enunciato. Come nell'esempio precedente in logica proposizionale, potremmo esprimere l'enunciato con

$$\mathbf{w}_x\mathcal{A} \rightarrow \mathbf{b}_x\mathbf{w}_x\mathcal{A}$$

FabuPlan supporta gli schemi di fatti con la restrizione che in una formula compaia al più un simbolo di schema, che indichiamo con $\bullet$. La formula precedente diventa

$$\mathbf{w}_x\bullet \rightarrow \mathbf{b}_x\mathbf{w}_x\bullet$$

dove $\bullet$ può essere uniformemente sostituito con un qualunque fatto.

Possiamo ora dare la definizione formale di *fatto*.

Definizione 4. Induttivamente

- $\diamond$ $\bullet$ è un fatto;
- $\diamond$ una sequenza di simboli della forma $P(t_1, \ldots, t_n)$, dove P è un simbolo di predicato n-ario e t_i è un termine, è un fatto;

- ◇ se $\mathcal{A}$ è un fatto, allora la sua negazione $\neg\mathcal{A}$ è un fatto;
- ◇ se $\mathcal{A}$ è un fatto, allora $\mathbf{m}_t\mathcal{A}$ è un fatto, dove m è un simbolo di operatore modale e t un termine.

La sequenza di operatori modali $\mathbf{m}^1_{t_1} \ldots \mathbf{m}^k_{t_k}$ che precede $P(t_1, \ldots, t_n)$ o $\bullet$ è detta *modalità*. Quando i termini di un fatto sono tutti costanti individuali (nessuno è una variabile), il fatto si dice *ground*.

Nella sintassi di FABUPLAN, gli operatori modali $\mathbf{m}_t$ vengono indicati usando le parentesi angolate come in `m<t>` e la negazione con il carattere '`!`'. Ad esempio $\neg\mathbf{b}_{john}Loves(john, mary)$ diventa `!b<john>Loves(john, mary)`. Il simbolo $\bullet$ invece è espresso con i tre puntini `...`, ad esempio `b<x>w<y>...`.

Unificazione

Cominciamo dal definire l'uguaglianza tra due fatti.

Definizione 5 (Uguaglianza di fatti)**.** Due fatti $\mathcal{F}$ e $\mathcal{G}$ sono uguali ($\mathcal{F} = \mathcal{G}$) se e solo se

- ◇ $\mathcal{F} = \bullet$ e $\mathcal{G} = \bullet$;
- ◇ oppure se $\mathcal{F} = P(t_1, \ldots, t_m)$ e $\mathcal{G} = Q(u_1, \ldots, u_n)$ allora $P = Q$ dunque $m = n$ e $\forall i = 1, \ldots, m, t_i = u_i$;
- ◇ oppure se $\mathcal{F} = \neg\mathcal{A}$ e $\mathcal{G} = \neg\mathcal{B}$ allora $\mathcal{A} = \mathcal{B}$;
- ◇ oppure se $\mathcal{F} = \mathbf{m}_t\mathcal{A}$ e $\mathcal{G} = \mathbf{n}_u\mathcal{B}$ allora $\mathbf{m} = \mathbf{n} \wedge t = u \wedge \mathcal{A} = \mathcal{B}$;

Quando un fatto è non-ground (contiene almeno una variabile) o almeno uno tra i due fatti confrontati è uno schema, l'uguaglianza stretta è un po' limitante in quanto potrebbe essere possibile applicare opportune sostituzioni alle variabili e a $\bullet$ in modo da renderli uguali anche se originariamente non lo sono. Se questo è possibile, diciamo che i due fatti sono *unificabili*.

Definizione 6. Una *sostituzione di variabile* θ è una coppia (v, t) dove v è una variabile e t un termine (valore). Il risultato dell'applicazione di θ a un fatto $\mathcal{F}$ indicato con $\alpha(\theta, \mathcal{F})$ è il fatto risultante dalla sostituzione di ogni occorrenza di v in $\mathcal{F}$ con t. Il risultato dell'applicazione di Θ a un fatto $\mathcal{F}$ indicato con $\alpha(\Theta, \mathcal{F})$ è il fatto risultante dall'applicazione di tutti i $\theta \in \Theta$ a $\mathcal{F}$.

Analogamente definiamo le sostituzioni operate sugli schemi.

Definizione 7. Una *sostituzione di schema* σ è un fatto. σ può essere *indefinito*, in tal caso $\sigma = \perp$. Il risultato dell'applicazione di σ a un fatto $\mathcal{F}$ indicato con $\alpha(\sigma, \mathcal{F})$ è $\mathcal{F}$ se $\sigma = \perp$ altrimenti il fatto risultante dalla sostituzione di $\bullet$ in $\mathcal{F}$ con σ.

Si noti che, a differenza delle sostituzioni di variabili, in una sostituzione di schema non è indicata una "variabile di schema" in quanto in FabuPlan sarebbe inutile, dato che non è possibile inserire più di uno schema in una formula.

Possiamo ora definire il significato di due fatti unificabili.

Definizione 8 (Unificatore)**.** Si chiama *unificatore* una coppia $U = \langle \Theta, \sigma \rangle$ con

- $\Theta = \{\theta_1, \ldots, \theta_n\}$ insieme possibilmente vuoto di sostituzioni di variabili $\theta_i = (v_i, t_i)$ tale che $\forall i, j \in \{1, \ldots, n\}$ se $v_i = v_j$ allora $i = j$ e σ sostituzione di schema;
- σ sostituzione di schema (possibilmente $\sigma = \perp$).

Definizione 9. Il risultato dell'applicazione di un unificatore $U = \langle \Theta, \sigma \rangle$ a un fatto $\mathcal{F}$ indicato con $\alpha(U, \mathcal{F})$ equivale a $\alpha(\Theta, \alpha(\sigma, \mathcal{F}))$ ovvero il risultato dell'applicazione di ogni sostituzione $\theta \in \Theta$ al risultato della sostituzione di σ a $\bullet$, se contenuto in $\mathcal{F}$.

Definizione 10. Due fatti $\mathcal{F}$ e $\mathcal{G}$ sono *unificabili* se esiste un unificatore $U = \langle \Theta, \sigma \rangle$ tale che se $\mathcal{H}_{max}$ è il fatto tra $\mathcal{F}$ e $\mathcal{G}$ con modalità più lunga e $\mathcal{H}_{min}$ più corta, allora $\alpha(U, \mathcal{H}_{min}) = \alpha(\Theta, \mathcal{H}_{max})$.

Si consideri l'esempio seguente: vogliamo verificare che $\mathcal{H}_{min} = \mathbf{b}_a\bullet$ unifichi con $\mathcal{H}_{max} = \mathbf{b}_x\mathbf{w}_bA(a,b)$ con A predicato binario, a,b costanti individuali e x variabile. I due fatti unificano perché esiste l'unificatore $U = \langle\Theta, \sigma\rangle$ con $\Theta = \{(x,b)\}$ e $\sigma = \mathbf{w}_bA(a,x)$ tale che $\alpha(U, \mathcal{H}_{min}) = \alpha(\Theta, \alpha(\sigma, \mathcal{H}_{min})) = \alpha(\Theta, \alpha(\sigma, \mathbf{b}_a\bullet)) = \alpha(\Theta, \mathbf{b}_a\mathbf{w}_bA(a,x)) = \mathbf{b}_a\mathbf{w}_bA(a,b)$ e $\alpha(\Theta, \mathcal{H}_{max}) = \alpha(\Theta, \mathbf{b}_x\mathbf{w}_bA(a,b)) = \mathbf{b}_a\mathbf{w}_bA(a,b)$ che risultano essere uguali.

Per la definizione data, Θ di un unificatore U può contenere più sostituzioni del necessario oppure σ può essere superfluo se nessuno di $\mathcal{F}$ e $\mathcal{G}$ contiene $\bullet$ o se sono uguali (non ci sarebbe nessun "eccesso" tra i fatti che andrebbe sostituito a $\bullet$ di quello con modalità più breve). Conviene dunque la seguente

Definizione 11 (Unificatore minimo)**.** Un unificatore $U = \langle\Theta, \sigma\rangle$ che unifica due fatti $\mathcal{F}$ e $\mathcal{G}$ si dice *minimo* se:

1. $\forall\theta = (v,t) \in \Theta$, almeno uno tra $\mathcal{F}$ o $\mathcal{G}$ contiene v;
2. se $\sigma \neq \bot$ allora almeno uno tra $\mathcal{F}$ e $\mathcal{G}$ contiene $\bullet$.

Osserviamo che se $\mathcal{F}$ e $\mathcal{G}$ hanno stessa modalità a meno dei termini di ciascun operatore modale ed entrambi contengono $\bullet$ allora σ di un unificatore minimo $U = \langle\Theta, \sigma\rangle$ è sicuramente $\bot$ altrimenti U non sarebbe un unificatore dei due fatti.

2.2.3 Basi

In FABUPLAN la descrizione dello stato del mondo iniziale e degli obiettivi finali della storia sono espressi mediante una *base*. Le basi, definite con la parola chiave `base` contengono una serie di *enunciati*. FABUPLAN supporta due tipi di enunciati, all'interno delle basi:

1. Dichiarazione di costanti individuali e loro tipo, nella sintassi

   ```
   nome-tipo individuo-1, ..., individuo-n;
   ```

   come ad esempio

```
Man john, arthur;
```

2. Asserzione di fatti *veri* nella base.

Un esempio completo di base è il seguente:

```
base world {
   Man john;
   Woman mary;

   b<john>Loves(john, mary), b<mary>Loves(mary, john);
   Rich(john), !b<john>Rich(john);
}
```

dove `Man` e `Woman` sono due tipi precedentemente definiti. Si osservi che in questo esempio manca il fatto `Loves(john,mary)` e viceversa poiché non è necessario che *John* e *Mary* si amino davvero, ma che l'uno *pensi* di amare l'altro. Inoltre si osservi che *John* può essere effettivamente ricco (`Rich(john)` e non sapere di esserlo `!b<john>Rich(john)`.

Mondo chiuso vs mondo aperto

FabuPlan funziona con assunzione di *mondo chiuso*. La differenza tra *mondo aperto* e *mondo chiuso* è che nel primo il valore di verità di tutto quello che non è dichiarato è sconosciuto al planner. Se nella dichiarazione del mondo il fatto $Loves(john, mary)$ non è dichiarato allora il planner non sa se john ami mary oppure no. Nel processo di pianificazione, il planner utilizza solo le informazioni esplicitamente dichiarate. Nell'assunzione di *mondo chiuso* invece, ciò che non è esplicitamente dichiarato è implicitamente falso. Il planner per valutare se è vero $\neg Loves(john, mary)$ in una base, può controllare che il fatto sia esplicitamente contenuto all'interno della base oppure che non sia dichiarato $Loves(john, mary)$. Quando il fatto contiene variabili, se esiste un fatto nella base che unifica con esso, per dire che è falso occorre garantire che in realtà una tra le sostituzioni di variabili Θ sia resa impossibile, con un vincolo di disuguaglianza tra la variabile e il valore. Questa situazione è illustrata meglio con un esempio: se la base contiene il

fatto $Loves(john, mary)$, il fatto $Loves(x, alice)$ è sicuramente falso in quanto non unifica con $Loves(john, mary)$, ma $Loves(x, mary)$ è falso se e solo se $x \neq john$ e $Loves(x, y)$ è falso se e solo se $x \neq john$ oppure $y \neq mary$. Con l'assunzione di mondo chiuso, l'autore quindi non è tenuto a dichiarare esplicitamente tutto ciò che è falso, ma gli basta dichiarare solo ciò che è vero per implicitamente asserire che è falso tutto il resto.

2.2.4 Formule

Abbiamo visto che i fatti rappresentano frammenti di conoscenza sullo stato del mondo. Come nella logica proposizionale o del prim'ordine, combinando logicamente fatti si possono costruire enunciati più complessi ed espressivi.

STRIPS non ha un vero supporto alle formule, così come UCPOP, RePOP e Fabulist (Riedl & Young): nelle precondizioni degli operatori e negli effetti sono permessi soltanto congiunzioni di fatti. Inoltre, nessuno supporta precondizioni negative.

FabuPlan ha pieno supporto a formule arbitrarie nelle precondizioni, inclusa la negazione. Diamo dunque la definizione di una *formula ben formata* o semplicemente *formula*.

Definizione 12 (Formula ben formata)**.** Induttivamente:

- ◇ un fatto $\mathcal{F}$ è una formula ben formata.
- ◇ se $\mathcal{A}$ è una formula ben formata allora $\neg\mathcal{A}$ è una formula ben formata.
- ◇ se $\mathcal{A}$ e $\mathcal{B}$ sono formule ben formate allora $\mathcal{A} \wedge \mathcal{B}$, $\mathcal{A} \vee \mathcal{B}$, $\mathcal{A} \rightarrow \mathcal{B}$ sono formule ben formate.
- ◇ se $\mathcal{A}$ è una formula ben formata allora $(\mathcal{A})$ è una formula ben formata.

I simboli $\wedge, \vee, \rightarrow$ hanno il significato comune di AND, OR e IMPLICA. In FabuPlan le parole chiave per questi operatori logici sono rispettivamente `and`,`or`,`implies` mentre per $\neg$ si utilizza la parola chiave `not`.

Ecco un esempio di formula in FabuPlan:

```
w<x>A(x,b) implies not (B(x) or b<x>...)
```

Poiché ci sarà utile in seguito, definiamo una funzione che restituisce le variabili contenute in una formula o fatto.

Definizione 13. La funzione *var* associa ad ogni formula l'insieme delle variabili che appaiono nella formula.

2.2.5 Definizioni di argomenti

In FabuPlan esistono diversi costrutti nei quali compaiono variabili, ad esempio i fatti. Poiché FabuPlan supporta i tipi associati a costanti individuali e variabili, prima che una variabile compaia in un fatto è necessario che sia precedentemente stata definita con il suo tipo.

Definizione 14. Una *definizione di argomenti* Γ è una funzione parziale che associa ad una variabile v un tipo T.

La sintassi di una definizione di argomenti appare in FabuPlan nel modo seguente

```
(tipo-1 nome-var-1, ..., tipo-m nome-var-m) [with (
   nome-var-1 != termine-1, ..., nome-var-n != termine-n)
```

dove il segmento `with (...)` specifica quali vincoli di disuguaglianza agiscono tra gli argomenti definiti, ed è facoltativo. *tipo-i* è il nome del tipo della variabile i precedentemente definito oppure la parola chiave `any` se la variabile può essere di qualunque tipo. Un esempio completo di definizione di argomenti è

```
(Person ?p1, Person ?p2, Weapon ?w) with (?p1 != ?p2)
```

che dichiara due argomenti distinti di tipo `Person` e uno di tipo `Weapon`.

Nelle regole, negli schemi di implicazione e negli schemi di eventi (definiti in seguito) possono comparire fatti contenenti simboli di variabili. Questi termini

in realtà non sono variabili perché le variabili sono create e gestite automaticamente dal planner, ma riferimenti a un argomento di una associata definizione di argomenti.

2.2.6 Regole

Quasi tutte le logiche sono basate su un insieme di tautologie, formule vere in ogni modello. Nella definizione di uno specifico dominio, il progettista definisce le verità del dominio mediante una sequenza di enunciati la cui verità in ogni modello è asserita dal progettista, chiamati *assiomi*. Un esempio di assioma in un dominio fisico afferma che se un oggetto è vicino a un altro allora anche l'altro è vicino al primo (simmetria della relazione di *vicinanza*). In logica del prim'ordine potremmo esprimere questo assioma con la formula

$$(\forall x)(\forall y)(Near(x, y) \rightarrow Near(y, x)) \tag{2.2}$$

Vediamo ora come estendere le funzionalità di un planner per supportare la scrittura di assiomi. Nel planning in avanti e ad ordinamento totale, il planner sceglie la prossima azione a partire da una descrizione totale del mondo corrente. In ogni stato, il planner cerca tutte gli schemi di azione applicabili (le cui precondizioni sono in qualche modo soddisfatte), ne istanzia uno indeterministicamente e costruisce lo stato successivo applicando gli effetti dell'azione allo stato corrente. In un planner di questo tipo, dove la definizione dello stato del mondo in ogni istante di tempo è *esplicita* e totale, potremmo applicare allo stato corrente tutti gli assiomi definiti ottenendo una nuova definizione di stato con la conoscenza implicita contenuta negli assiomi resa esplicita. Ad esempio se lo stato del mondo in un certo istante è $Near(a, b)$ allora applicando l'assioma 2.2 otteniamo un nuovo stato $Near(a, b) \wedge Near(b, a)$ sul quale poi verificare l'applicabilità di ciascuno schema d'aazione definito.

In un planner ad ordinamento parziale, non esiste alcuna definizione totale ed esplicita dello stato del mondo come congiunzione di fatti, a meno dell'istante

iniziale. In altre parole, in un qualunque istante tra lo stato iniziale e quello finale, non è noto esplicitamente lo stato del mondo, quindi l'approccio precedente non è utilizzabile.

La possibilità di definire assiomi non soltanto aumenta notevolmente l'espressività del linguaggio complessivo di un planner, e quindi allegerisce il lavoro del progettista nella definizione del dominio, ma può aiutare il planner a capire meglio il problema che sta risolvendo, con miglioramenti sulla qualità della soluzione trovata e sull'efficienza della sua ricerca.

FabuPlan supporta la definizione di assiomi sotto forma di *regole*.

Definizione 15 (Regola)**.** Una *regola* $\mathcal{R}$ è una formula della forma di una implicazione avente come antecedente una disgiunzione di fatti e come conseguente una congiunzione di fatti. In altri termini, una *regola* è una formula del tipo

$$\mathcal{F}_1 \vee \mathcal{F}_2 \vee \ldots \vee \mathcal{F}_m \rightarrow \mathcal{G}_1 \wedge \mathcal{G}_2 \wedge \ldots \wedge \mathcal{G}_n$$

dove $\mathcal{F}_i$ e $\mathcal{G}_j$ sono fatti. Ad ogni regola $\mathcal{R}$ è sempre associata una definizione di argomenti $\Gamma_{\mathcal{R}}$ tale che $var(\mathcal{R})$ siano definite in $\Gamma_{\mathcal{R}}$.

In FabuPlan, una regola viene definita con la seguente sintassi

```
rule definizione-argomenti: fatto-1 or ... or fatto-n
   implies fatto-1 and ... and fatto-k;
```

dove `dichiarazione-argomenti` è la dichiarazione di argomenti associata alla regola.

Ad esempio l'assioma 2.2 viene modellato in FabuPlan con

```
rule (any ?x, any ?y): Near(?x, ?y) implies Near(?y, ?x);
```

Si osservi che non è possibile definire una regola per modellare la transitività di una relazione. La transitività di una relazione R infatti è modellata in logica del prim'ordine con l'assioma

$$(\forall x)(\forall y)(\forall z)(R(x,y) \wedge R(y,z) \rightarrow R(x,z)) \tag{2.3}$$

ma nell'antecedente delle regole non possono comparire congiunzioni ma solo disgiunzioni. La regola

```
rule (any ?x, any ?y, any ?z): R(?x, ?y) or R(?y, ?z)
   implies R(?x, ?z);
```

naturalmente *non* modella la transitività di R.

Ci si potrebbe dunque chiedere il perché di questa limitazione alla struttura di una regola. Per spiegarne il motivo occorre prima illustrare come FabuPlan usa le regole. Iniziamo col definire le seguenti applicazioni:

Definizione 16. $\overrightarrow{\gamma}$ è una applicazione che associa ad ogni fatto $\mathcal{F}$ la sua chiusura transitiva delle regole in avanti via unificazione, ovvero

- $\{\mathcal{F}\} \subseteq \overrightarrow{\gamma}(\mathcal{F})$;
- per ogni regola $\mathcal{F}_1 \vee \ldots \vee \mathcal{F}_m \rightarrow \mathcal{G}_1 \wedge \ldots \wedge \mathcal{G}_n$ se
  - esistono un fatto $\mathcal{H} \in \overrightarrow{\gamma}(\mathcal{F})$ e un $i \in \{1, \ldots, m\}$ tali che $\mathcal{H}$ unifica con $\mathcal{F}_i$ per un unificatore U allora $\forall j \in \{1, \ldots, n\}, \alpha(U, \mathcal{G}_j) \in \overrightarrow{\gamma}(\mathcal{F})$;
  - esistono un fatto $\mathcal{H} \in \overrightarrow{\gamma}(\mathcal{F})$ e un $j \in \{1, \ldots, n\}$ tali che $\mathcal{H}$ unifica con $\neg\mathcal{G}_j$ per un unificatore U allora $\forall i \in \{1, \ldots, m\}, \alpha(U, \neg\mathcal{F}_i) \in \overrightarrow{\gamma}(\mathcal{F})$ (per contrapposizione);

Analogamente definiamo la chiusura all'indietro.

Definizione 17. $\overleftarrow{\gamma}$ è una applicazione che associa ad ogni fatto $\mathcal{F}$ la sua chiusura transitiva delle regole all'indietro via unificazione, ovvero

- $\{\mathcal{F}\} \subseteq \overleftarrow{\gamma}(\mathcal{F})$;
- per ogni regola $\mathcal{F}_1 \vee \ldots \vee \mathcal{F}_m \rightarrow \mathcal{G}_1 \wedge \ldots \wedge \mathcal{G}_n$ se
  - esistono un fatto $\mathcal{H} \in \overleftarrow{\gamma}(\mathcal{F})$ e un $j \in \{1, \ldots, n\}$ tali che $\mathcal{H}$ unifica con $\mathcal{G}_j$ per un unificatore U allora $\forall i \in \{1, \ldots, m\}, \alpha(U, \mathcal{F}_i) \in \overleftarrow{\gamma}(\mathcal{F})$;

– esistono un fatto $\mathcal{H} \in \overleftarrow{\gamma}(\mathcal{F})$ e un $i \in \{1, \ldots, m\}$ tali che $\mathcal{H}$ unifica con $\neg\mathcal{F}_i$ per un unificatore U allora $\forall j \in \{1, \ldots, n\}, \alpha(U, \neg\mathcal{G}_j) \in \overleftarrow{\gamma}(\mathcal{F})$ (per contrapposizione);

Si osservi che nella generazione di $\overrightarrow{\gamma}$ e $\overleftarrow{\gamma}$ vengono costruiti una serie di unificatori $U_k = \langle \Theta_k, \sigma_k \rangle$. Affinché i fatti in $\overleftrightarrow{\gamma}(\mathcal{F})$ siano validi è necessario rendere *persistenti* le sostituzioni in Θ_k imponendo vincoli di uguaglianza tra le variabili e i valori, necessari per l'unificazione.

La presenza delle regole fornisce al planner e al progettista due notevoli vantaggi.

Inferenza di effetti alternativi

Quando il planner cerca di soddisfare un fatto obiettivo $\mathcal{F}$ aperto, indeterministicamente sceglie di risolvere l'obiettivo unificandolo con un effetto di un evento già istanziato, con un fatto (se esiste) dichiarato nel mondo iniziale, oppure istanziando un nuovo evento che lo produce come effetto. Questi tentativi di unificazione avvengono sempre tra $\mathcal{F}$ e un fatto disponibile o istanziabile.

Poiché una regola è un assioma del dominio nella forma di un'implicazione tra una disgiunzione e una congiunzione di fatti, se è vero uno dei fatti dell'antecedente allora anche *tutti* i fatti del conseguente devono essere veri. Supponiamo esista una regola $\mathcal{A} \rightarrow \mathcal{B}$ e un effetto già istanziato o parte della descrizione iniziale del mondo che unifica con $\mathcal{A}$ per un unificatore U. Per l'esistenza della regola, dunque nello stesso istante in cui è vero $\mathcal{A}$ è anche vero $\alpha(U, \mathcal{B})$. Questo è un nuovo, implicito effetto che il planner può considerare durante la scelta dell'effetto da usare per risolvere un obiettivo aperto.

In altre parole, per risolvere un obiettivo $\mathcal{F}$ il planner può naturalmente utilizzare un effetto già istanziato o un fatto del mondo iniziale $\mathcal{G}$ che unifica con $\mathcal{F}$, ma anche tutti i fatti in $\mathcal{H} \in \overleftarrow{\gamma}(\mathcal{F})$ che unificano con $\mathcal{F}$ poiché, appunto, la verità del fatto $\mathcal{H} \in \overleftarrow{\gamma}(\mathcal{F})$ implica, per una qualche catena di regole, la verità di $\mathcal{F}$.

Si consideri il seguente esempio. Supponiamo di aver definito nel classico dominio del mondo a blocchi movibili, la regola

$$On(x, y) \rightarrow \neg Free(y)$$

dove x e y sono variabili di blocchi impilabili. La regola dice, in italiano, che "se un blocco x è su un oggetto y allora y non è libero (da altri blocchi)". Ora supponiamo vi sia un obiettivo aperto $\neg Free(a)$ che significa "il blocco a non è libero". Intuitivamente se ci fosse un oggetto su a, allora a non sarebbe libero. Questo ragionamento viene effettuato dal reasoner cercando la chiusura transitiva all'indietro delle regole $\overleftarrow{\gamma}(\neg Free(a))$, che nell'esempio risulta essere banalmente $\overleftarrow{\gamma}(\neg Free(a)) = \neg Free(a), On(x_1, a)$ dove x_1 è una nuova variabile libera. Quindi supponendo nel mondo iniziale sia solo definito $On(b, a)$, il planner può comunque soddisfare l'obiettivo $\neg Free(a)$ con $On(b, a)$ mediante l'unificatore $U = \langle\{(x_1, b)\}, \bot\rangle$.

Questo sistema riduce notevolmente il lavoro del progettista nella definizione del dominio poiché non è più necessario esplicitare le alternative di caso in caso. Si pensi al predicato $Near$. Una precondizione aperta come $Near(a, b)$ è sia soddisfacibile dall'effetto $Near(a, b)$ ma anche dall'effetto $Near(b, a)$. Dover aggiungere manualmente la disgiunzione $Near(x, y) \vee Near(y, x)$ ogni qual volta è necessario il predicato $Near$ in una precondizione diventerebbe ovviamente abbastanza tedioso.

Vediamo ora perché le regole devono presentare la forma di implicazione tra disgiunzione di fatti e congiunzione di fatti. Se le regole non avessero questa restrizione, non potremmo calcolare efficacemente la lista dei fatti alternativi a una precondizione. Intuitivamente, partendo dalla precondizione, cerchiamo una regola che ha tra gli effetti uno che unifica con la precondizione. Se fosse permesso inserire indeterminismo nel conseguente di una regola attraverso una disgiunzione, il planner non sarebbe *sicuro* che quell'effetto sussiste. Ad esempio, se la seguente

regola fosse una permessa

$$\mathcal{A} \to \mathcal{B} \vee \mathcal{C}$$

e $\mathcal{C}$ unificasse con la precondizione per un unificatore U, non potremmo deterministicamente affermare che $\alpha(U, \mathcal{C})$ è vero quando lo è $\alpha(U, \mathcal{A})$ perché potrebbe invece essere vero soltanto $\alpha(U, \mathcal{B})$. Avendo invece solo una congiunzione di fatti nel conseguente, siamo sicuri che ciascun fatto della congiunzione è vero quando è vero l'antecedente.

Ora ragioniamo sulla restrizione sulla precondizione. Supponiamo sia possibile in questa inserire una congiunzione di fatti, ad esempio nella seguente regola

$$\mathcal{A} \wedge \mathcal{B} \to \mathcal{C}$$

e al solito sia $\mathcal{C}$ un fatto che unifichi con la precondizione per un unificatore U. Sappiamo dunque che è vero $\alpha(U, \mathcal{C})$ quando sono veri sia $\alpha(U, \mathcal{A})$ che $\alpha(U, \mathcal{B})$. Poiché FabuPlan e tutti i planner ad ordinamento parziale risolvono le precondizioni aperte una per volta collegandole via singolo collegamento causale a un effetto istanziato, per risolvere la precondizione usando questa regola dovremmo cercare *due* effetti istanziati, l'uno che unifichi con $\alpha(U, \mathcal{A})$ l'altro con $\alpha(U, \mathcal{B})$ e creare due collegamenti causali per una singola precondizione. Questa aggiunta è realizzabile, ma è esattamente il modo in cui funzionano le regole complesse (descritte in seguito), dunque sarebbe una non necessaria complicazione dell'algoritmo di planning.

Inferenza di fatti mutuamente esclusivi

Come menzionato nel paragrafo 2.1.3, FabuPlan è un planner di tipo POCL, Partial Order Causal Link. STRIPS, UCPOP[13] e Fabulist[16] (basato su UCPOP), per determinare i fatti in conflitto tra un evento e un collegamento causale, fanno unicamente uso della *lista di rimozione degli effetti* (*delete list*) di ogni azione, una lista di effetti che vengono *rimossi* dallo stato del mondo quando l'azione è compiuta (e quindi resi falsi per assunzione di mondo chiuso). Se un

collegamento causale protegge l'effetto $\mathcal{F}$ allora un conflitto nasce quando vi è un effetto unificabile con $\mathcal{F}$ nella *delete list* di un evento istanziato nel piano, in quanto quella condizione andrebbe rimossa. Ne consegue che in questi planner solo le condizioni aventi stesso predicato possono essere in conflitto: la mutua esclusione del fatto $On(x, y)$ con $Free(y)$ verrebbe "derivata" dal planner al costo di molte iterazioni.

Consideriamo ora la seguente regola

$$\mathcal{F}_1 \vee \ldots \vee \mathcal{F}_m \rightarrow \mathcal{G}_1 \wedge \ldots \wedge \mathcal{G}_n$$

Se $\mathcal{F}$, il fatto protetto da un collegamento causale, unifica per qualche i con $\mathcal{F}_i$, per un unificatore $U = \langle \Theta, \sigma \rangle$, allora ogni fatto $\alpha(U, \mathcal{G}_j)$ è implicato da $\mathcal{F}$: quindi se un evento ha un effetto $\mathcal{H}$ che unifica con $\neg\alpha(U, \mathcal{G}_j)$ per qualche j, allora vi è un conflitto tra l'evento e il collegamento causale, perché se l'evento accadesse tra i due eventi del collegamento causale, si avrebbe una contraddizione in quanto sono veri sia $\alpha(U, \mathcal{G}_j)$ che $\neg\alpha(U, \mathcal{G}_j)$, a patto che $\Theta = \emptyset$ oppure che venga reso persistente nel piano ogni $\theta \in \Theta$. Quindi non soltanto $\neg\mathcal{F}$ è mutex con $\mathcal{F}$ ma anche tutti gli $\neg\alpha(U, \mathcal{G}_j)$ $\forall j$.

Per *contrapposizione* dell'implicazione della regola precedente abbiamo che

$$\neg(\mathcal{G}_1 \wedge \ldots \wedge \mathcal{G}_n) \rightarrow \neg(\mathcal{F}_1 \vee \ldots \vee \mathcal{F}_m)$$

ovvero (De Morgan)

$$\neg\mathcal{G}_1 \vee \ldots \vee \neg\mathcal{G}_n \rightarrow \neg\mathcal{F}_1 \wedge \ldots \wedge \neg\mathcal{F}_m$$

Anche per questa regola è possibile ripetere un procedimento analogo. Se $\mathcal{F}$, sempre il fatto protetto dal collegamento causale, unifica per un unificatore $U = \langle \Theta, \sigma \rangle$ con $\neg\mathcal{G}_j$ per qualche j, allora ogni fatto $\alpha(U, \neg\mathcal{F}_i)$ è implicato da $\mathcal{F}$: quindi se un evento ha un effetto $\mathcal{H}$ che unifica con $\neg\alpha(U, \neg\mathcal{F}_i) = \alpha(U, \mathcal{F}_i)$ per qualche i, allora vi è nuovamente un conflitto tra l'evento e il collegamento causale (sempre se $\Theta = \emptyset$ o viene reso persistente nel piano ogni $\theta \in \Theta$). Quindi anche tutti gli $\alpha(U, \mathcal{F}_i)$ $\forall i$ sono mutex con $\mathcal{F}$.

Concludiamo dicendo che

Teorema 1. $\mu(\mathcal{F}) = \{\neg\mathcal{G} \mid \mathcal{G} \in \vec{\gamma}(\mathcal{F})\}$ *è l'insieme di fatti mutuamente esclusivi con un fatto* $\mathcal{F}$*.*

Dimostrazione. La dimostrazione segue dal ragionamento precedente. □

Come mostrato da [12], disporre per ogni collegamento causale la lista di fatti mutex con il fatto protetto permette più precocemente di individuare piani parziali inconsistenti, arrestando quel ramo di ricerca e ottenendo, generalmente, migliori prestazioni.

Vediamo anche in relazione a questo vantaggio i motivi della restrizione delle regole a implicazione tra una disgiunzione e congiunzione di fatti, come abbiamo fatto nel paragrafo precedente. L'obiettivo è quello di trovare i fatti che potrebbero essere in mutex con il fatto protetto da un collegamento causale. Se fosse possibile definire una congiunzione nell'antecedente di una regola come in

$$\mathcal{A} \wedge \mathcal{B} \rightarrow \mathcal{C}$$

allora sarebbe impossibile determinare i fatti in mutex con quello protetto, perché supponendo questo sia unificabile con $\mathcal{A}$ per un unificatore U allora non possiamo dir niente sulla verità di $\alpha(U, \mathcal{C})$ perché avremmo bisogno di dimostrare anche la verità di $\alpha(U, \mathcal{B})$ e questo non è possibile, in quanto conosciamo solo la verità di un fatto, quello protetto, nell'arco di tempo che intercorre tra i due eventi collegati dal collegamento causale. Dunque non potremmo essere certi che $\alpha(U, \mathcal{C})$ è mutex con il fatto protetto.

Infine nelle regole non è consentito indeterminismo nel conseguente come in

$$\mathcal{A} \rightarrow \mathcal{B} \vee \mathcal{C}$$

altrimenti, supponendo il fatto protetto unifichi con $\mathcal{A}$ per un unificatore U, non potremmo deterministicamente dire quale tra $\alpha(U, \mathcal{B})$ e $\alpha(U, \mathcal{C})$ sia mutex con il fatto protetto.

2.2.7 Schemi di implicazione e regole complesse

Gli *schemi di implicazioni* sono postcondizioni che si verificano soltanto quando certe precondizioni sono verificate. In FabuPlan gli schemi di implicazione sono usati per implementare gli effetti condizionali (come in ADL), se definiti all'interno di schemi di eventi, oppure per definire assiomi non esprimibili con una regola (le *regole complesse*).

Definizione 18. Uno schema di implicazione $\mathcal{I}$ è una formula della forma

$$\mathcal{I} := \mathcal{P} \rightarrow \mathcal{E}$$

dove $\mathcal{P}$ è una formula qualunque e $\mathcal{E}$ è una formula del tipo $\mathcal{E} = \mathcal{F}_1, \ldots, \mathcal{F}_n$ dove ogni $\mathcal{F}_i$ è un fatto. Ad ogni schema di implicazione $\mathcal{I}$ è associata una definizione di argomenti $\Gamma_{\mathcal{I}}$ tale che $var(\mathcal{I})$ siano definite in $\Gamma_{\mathcal{I}}$.

Uno schema di implicazione o "blocco *when*" presenta nel linguaggio di FabuPlanla seguente struttura:

```
when dichiarazione-argomenti {
   in: precondizioni;
   out: effetti;
}
```

`dichiarazione-argomenti` è la dichiarazione di argomenti associata allo schema di implicazione. `precondizioni` è una formula qualunque (può quindi comprendere gli operatori logici `and`, `or`, `implies` e `not`) di fatti. Se in un qualunque momento questa formula è soddisfacibile mediante opportune sostituzioni alle variabili, gli effetti devono essere veri nello stesso momento. Questi vengono indicati in `effetti` che a differenza di `precondizioni` ammette soltanto l'operatore `and`: `effetti` è una formula vincolata ad essere sempre una congiunzione di fatti. Questa limitazione risiede nel fatto che sebbene FabuPlan supporti l'indeterminismo nelle precondizioni, non lo supporta negli effetti e dunque sia $\vee$ che $\neg$ non sono permessi ($\neg(\mathcal{A} \wedge \mathcal{B}) = \neg\mathcal{A} \vee \neg\mathcal{B}$). Il motivo è puramente

tecnico, in quanto aggiungerebbe notevole addizionale complessità all'algoritmo di planning in realtà non necessaria, dato che è sempre possibile creare due versioni di un evento inserendo in ciascuna versione una clausola congiuntiva di effetti.

Abbiamo visto che le regole non supportano formule arbitrarie nelle precondizioni, ma solo una disgiunzione di fatti. Quando si vuole definire un assioma complesso non esprimibile con una regola, possiamo utilizzare in alternativa una regola complessa, come nel caso della transitività. Una regola complessa è un assioma sotto forma di blocco `when` che compare all'esterno di uno schema di evento (è indipendente). Ad esempio per dire che il predicato $Near$ identifica una relazione transitiva, possiamo dichiarare la seguente regola complessa:

```
when (any ?x, any ?y, any ?z) {
   in: Near(?x, ?y) and Near(?y, ?z);
   out: Near(?x, ?z);
}
```

Modellare un'assioma mediante una regola complessa piuttosto che con una regola non fornisce, tuttavia, i vantaggi delle regole (inferenza di alternative di una precondizione e dei fatti mutex a un fatto protetto), per i motivi spiegati nel paragrafo 2.2.6.

2.2.8 Schemi di eventi

FabuPlan istanzia opportunamente una collezione di eventi in modo da costruire una fabula da una descrizione di mondo iniziale avente certi obiettivi finali. Gli eventi che costituiscono la fabula sono istanze di *schemi di eventi* (chiamati *operatori* in STRIPS), definiti dall'autore. Uno schema d'evento, definito in FabuPlan con la parola chiave `event`, è composto di:

- ◇ un nome, ad esempio *WalksTo*;
- ◇ una definizione, associata allo schema, di argomenti validi all'interno dello schema;

- ◇ una formula di precondizioni qualunque (come negli schemi di implicazione);
- ◇ una formula di effetti limitata a congiunzione di fatti (come negli schemi di implicazione);
- ◇ nessuno o più effetti condizionali.

Quindi diamo la seguente

Definizione 19. Uno *schema di eventi* $\mathcal{E}$ è una tupla $\langle \mathcal{I}, E \rangle$ dove $\mathcal{I}$ è uno schema di implicazione e E è un insieme di schemi di implicazioni chiamato insieme degli *effetti condizionali*.

Le precondizioni e gli effetti dello schema d'evento sono rispettivamente l'antecedente e il conseguente di $\mathcal{I}$ mentre la definizione di argomenti associata allo schema d'evento è dato da quello di $\mathcal{I}$ cioè $\Gamma_{\mathcal{I}}$.

Consideriamo, a titolo d'esempio l'evento *WalksTo*, ovvero l'evento coinvolgente un personaggio che si dirige da un luogo a un altro. Un modo in cui *WalksTo* è modellabile in FABUPLAN è il seguente:

```
event WalksTo(Person ?p, Place ?from-place, Place ?to-
   place) with (?from-place != ?to-place) {
   in: At(?p, ?from-place) and b<?p>At(?from-place) and w
      <?p>At(?to-place) and CanWalk(?p) and b<?p>CanWalk
      (?p);
   out: !At(?p, ?from-place) and At(?p, to-place) and b<?
      p>!At(?from-place) and b<?p>At(?to-place);
}
```

Una descrizione informale dell'evento è la seguente: affinché si verifichi l'evento che una persona vada da un luogo ad un altro, è necessario che:

- ◇ la persona sia fisicamente in un qualche luogo iniziale;
- ◇ la persona creda di essere in tale luogo (potrebbe ad esempio essere stata rapita e portata in un luogo sconosciuto);
- ◇ la persona desidera trovarsi in un altro luogo.

◇ la persona può camminare;

◇ la persona sa di poter camminare.

allora, conclutosi l'evento, nel mondo è vero che:

◇ la persona non è più nel luogo iniziale, essendosi spostata;

◇ la persona è nel luogo destinazione;

◇ la persona sa di esserlo;

Quest'esempio di schema di evento può essere istanziato in una moltitudine di eventi al variare della persona e dei due luoghi.

Lo schema *WalksTo* non contiene effetti condizionali: tutti gli eventi *WalksTo* hanno le stesse precondizioni e gli stessi effetti, coincidenti con rispettivamente le formule `in` e `out`. Ci sono casi però in cui alcuni effetti di un evento si verificano solo a fronte di specifiche precondizioni aggiuntive. Queste situazioni sono risolte aggiungendo all'interno di uno schema uno o più schemi di implicazione (blocchi `when`) in questo caso *effetti condizionali*. Ad esempio, vogliamo modellare il fatto che quando una persona rientra in casa diventa felice. Per farlo, modifichiamo lo schema *WalksTo* aggiungendo un blocco `when`:

```
event WalksTo(Person ?p, Place ?from-place, Place ?to-
   place) with (?from-place != ?to-place) {
   in: At(?p, ?from-place) and b<?p>At(?from-place) and w
      <?p>At(?to-place) and CanWalk(?p) and b<?p>CanWalk
      (?p);
   out: !At(?p, ?from-place) and At(?p, ?to-place) and b
      <?p>!At(?from-place) and b<?p>At(?to-place);
   when () {
      in: HasHouse(?p, ?to-place);
      out: Happy(?p);
   }
}
```

Si noti che, in questo caso, il blocco `when` non dichiara ulteriori argomenti anche se ciò è permesso. Un planner in avanti dispone della descrizione totale del mondo dopo ogni azione: grazie ad essa, una volta che ha scoperto che una azione è applicabile, può controllare che ogni effetto condizionale dell'azione sia anch'esso applicabile e, in tal caso, aggiungere gli effetti al prossimo stato del mondo. FabuPlan, essendo un planner POCL, non dispone dell'esplicito stato del mondo prima di ogni evento: quello che può fare è aggiungere le precondizioni di un effetto condizionale agli obiettivi e cercare di risolverli in futuro. Supponiamo che il planner stia risolvendo l'obiettivo `At(john, green-house)`, ovvero il fatto che `john` sia nella `green-house`. Per farlo, decide di istanziare un nuovo evento `WalksTo` che ha come effetto `At(?p, ?to-place)` che unifica con `At(john, green-house)`. Quindi sviluppa ed aggiunge le precondizioni unificate di `WalksTo` alla lista degli obiettivi da risolvere. Dopo qualche iterazione, si presenta la necessità di risolvere l'obiettivo `Happy(john)`. Qui il planner può decidere di istanziare un nuovo evento `WalksTo` dato che `Happy(?p)` è un effetto *possibile* oppure *riutilizzare* l'evento `WalksTo` precedentemente istanziato, aggiungendovi però la precondizione, da risolvere, che `green-house` sia la casa di `john`, ovvero `HasHouse(john, green-house)`.

2.2.9 Scope degli schemi di fatti

Quando FabuPlan soddisfa un obiettivo aperto unificandolo per un unificatore minimo U con l'effetto contenente $\bullet$ di un nuovo evento istanziato, per unificazione $\bullet$ di tutti i fatti nei blocchi `in` e `out` vengono sostituiti con σ. In generale diciamo che lo *scope* di uno schema di fatto si estende all'interno della stessa formula. Osserviamo che le formule `in` e `out` sono in realtà rispettivamente l'antecedente e il conseguente di una stessa implicazione, quindi sono considerati appartenere alla stessa formula.

Dunque se $\sigma \neq \bot$ nell'unificazione con un fatto di una regola, $\bullet$ di tutti i fatti della stessa regola verranno sostituiti da σ. Analogamente ciò accade per i

fatti contenuti nelle formule `in` e `out` di uno schema di evento o di implicazione. Dunque se viene istanziato uno schema d'evento per via di un effetto condizionale contenente •, allora solo • dei fatti contenuti all'interno dello stesso effetto condizionale saranno sostituiti con σ mentre un eventuale fatto contenente • nelle precondizioni o effetti dello schema d'eventi non subirà la sostituzione.

2.2.10 Giustificazione delle intenzioni implicite

Abbiamo visto nel paragrafo 2.1.4 che il planning classico pianifica per un unico agente o per una moltitudine di agenti ciecamene cooperanti alla riuscita degli obiettivi e che quindi, per modellare le intenzionalità dei personaggi in modo da renderli credibili dall'audience, è stato necessario aggiungere espressività al linguaggio STRIPS, inserendo operatori modali che modellassero la conoscenza e le intenzioni sotto forma di desideri sullo stato del mondo di ciascun personaggio. La sola estensione del linguaggio alla multimodalità non è tuttavia sufficiente affinché il planner si accorga che più desideri interconnessi di uno stesso personaggio facciano parte della medesima intenzionalità.

Consideriamo lo scenario seguente: nella definizione di una storia, l'autore dichiara uno schema d'evento *Marry* nel modo seguente

```
event Marry(Man ?groom, Woman ?bride) {
   in: w<?groom>Married(?groom, ?bride) and w<?bride>
      Married(?groom, ?bride) and At(?groom, church) and
      At(?bride, church) ad /* ... */;
   out: Married(?groom, ?bride) and /* ... */;
}
```

con eventuali ulteriori precondizioni ed effetti omessi. Supponiamo la storia termini con `jafar` e `jasmine` sposati, dunque `Married(jafar, jasmine)`. FABUPLAN comincia risolvendo questo (unico) obiettivo: istanzia lo schema d'evento `Marry` con un opportuno unificatore e aggiunge agli obiettivi il risultato dell'applicazione dell'unificatore alle precondizioni. Supponiamo `w<jafar>Married(jafar, jasmine)` e `w<jasmine>Married(jafar, jasmine)` siano fatti veri nel

mondo iniziale e supponiamo inizialmente sia *Jafar* che *Jasmine* siano al *castello* (dunque non in *chiesa*). Ora il planner deve risolvere la precondizione aperta `At(jafar, church)`. Per farlo, istanzia il seguente schema d'evento

```
event GoesTo(Person ?person, Place ?from, Place ?to) with
    (?from != ?to) {
   in: w<?person>At(?person, ?to) and At(?person, ?from)
      and /* ... */;
   out: At(?person, ?to) and /* ... */;
}
```

Consideriamo ora la precondizione `w<jafar>At(jafar, church)` di questo nuovo evento istanziato. Nel mondo iniziale non è definito esplicitamente che *Jafar* voglia andare in *chiesa*: dunque questa precondizione sarebbe irrisolvibile e la generazione della fabula fallirebbe. In realtà osserviamo che il desiderio di essere in chiesa può essere *giustificato* dal fatto che uno voglia sposarsi, che per farlo, deve essere in *chiesa*. FABUPLAN dunque dovrebbe poter inderministicamene scegliere di risolvere la precondizione `w<jafar>At(jafar, church)` mediante l'effetto (del mondo) `w<jafar>Married(jafar, jasmine)`. Generalizziamo questa *giustificazione di intenzione implicita* con la seguente proposizione:

Proposizione 1 (Giustificazione di intenzione implicita)**.** Ogni precondizione $\mathcal{P} = \mathbf{w}_t\mathcal{A}$ può essere soddisfatta da un effetto $\mathcal{E} = \mathbf{w}_u\mathcal{B}$ tale che esiste un unificatore U che unifica $\mathbf{w}_t\bullet$ con $\mathbf{w}_u\bullet$ e esiste un evento istanziato che abbia $\alpha(U, \mathcal{A})$ tra le precondizioni e $\alpha(U, \mathcal{B})$ negli effeti.

Si osservi che l'unificatore U così costruito è sempre della forma $U = \langle \emptyset, \bot \rangle$ oppure $U = \langle \{\theta\}, \bot \rangle$, ovvero contiene al più una sostituzione di variabile e σ indefinito. Ha dunque l'unico scopo di unificare t con u e applicare l'eventuale sostituzione a $\mathcal{A}$ e $\mathcal{B}$.

Nell'esempio, $\mathcal{P} = \mathbf{w}_{jafar}At(jafar, church)$ e $\mathcal{E} = \mathbf{w}_{jafar}Married(jafar, jasmine)$. L'unificatore è $U = \langle \emptyset, \bot \rangle$ in quanto nessuna sostituzione di variabili è necessaria. $Married(jafar, jasmine)$ è un effetto dell'evento *Marry* istanziato e dun-

que $\mathbf{w}_{jafar} Married(jafar, jasmine)$ implica che $\mathbf{w}_{jafar} At(jafar, church)$, il quale può essere risolto dal primo.

Capitolo 3

Implementazione

Questo capitolo illustra l'algoritmo di planning di FabuPlan in modo più formale, per poi trattare i più interessanti dettagli tecnici.

3.1 Algoritmo di planning

Vediamo dunque come internamente FabuPlan costruisce il grafo di fabula a partire degli elementi definiti nel capitolo precedente.

3.1.1 Eventi

FabuPlan istanzia una serie di schemi di evento che costituiscono i nodi del grafo della fabula. Gli schemi di evento sono appunto schemi, in quanto per essere inseriti in una fabula, gli argomenti devono essere uniformement sostituiti con costanti individuali, variabili in uso o variabili appena create. Quando FabuPlan risolve una precondizione aperta può indeterministicamente istanziare uno schema d'evento che produce un effetto unificabile ad essa. L'istanza dello schema d'evento prende il nome di *evento istanziato* o semplicemente *evento*.

Nel risolvere un effetto istanziando uno schema d'evento, possono accadere due situazioni:

1. L'effetto desiderato è negli effetti non condizionali dello schema (il blocco `out` interno al blocco `event`);

2. L'effetto desiderato è un effetto condizionale (interno al blocco `out` di un blocco `when` nello schema).

In entrambi i casi, viene costuito un unificatore U che unifica la precondizione aperta con l'effetto di un nuovo evento e appena creato. Nel primo caso, U è applicato a tutti gli effetti dello schema che vengono aggiunti agli *effetti istanziati* di e e a tutte le precondizioni. Nel secondo caso, l'unificatore U è applicato sia agli effetti del blocco `when` contenente l'effetto utile a risolvere la precondizione, sia agli effetti non condizionali dello schema d'evento e alle precondizioni condizionali e non condizionali. Sia in un caso che nell'altro, tutte le precondizioni istanziate diventano nuove precondizioni aperte (dunque obiettivi del planner) e tutti gli effetti istanziati sono nuovi effetti a disposizione del planner per risolvere le prossime precondizioni aperte. Il metodo di istanziazione delle regole complesse è del tutto analogo alla prima situazione.

Entrambi le situazioni prevedono che un nuovo evento venga istanziato. Può capitare però che l'effetto condizionale desiderato appartenga a uno schema d'eventi già precedentemente istanziato in un'iterazione precedente. Se il planner si accorge che esiste un evento istanza dello stessa schema e compatibile con il vecchio unificatore, potrebbe indeterministicamente decidere di non istanziare completamente un nuovo evento ma istanziare solo l'effetto condizionale desiderato all'interno dell'istanza di evento già esistente.

Consideriamo l'esempio seguente. Modelliamo con il seguene schema il fatto che una persona che compra un'edifico non solo poi possiede l'edificio ma anche il mobilio ivi contenuto.

```
event BuysBuilding(Person ?p, Building ?b) {
   in: w<?p>Owns(?p, ?b) and HasMoney(?p) and /* ... */;
   out: Owns(?p, ?b) and b<?p>Owns(?p, ?b) and /* ... */;
   when (Object ?o) {
      in: In(?o, ?b);
      out: Owns(?p, ?o) and b<?p>Owns(?p, ?o);
   }
}
```

Ora supponiamo che FabuPlan risolva l'obiettivo `Owns(john,green-house)` istanziando questo schema d'evento. Viene creato un nuovo evento con le precondizioni

```
w<john>Owns(john, green-house), HasMoney(john), ...
```

e gli effetti

```
Owns(john, green-house), b<john>Owns(john,green-house)
   , ...
```

Supponiamo successivamente il planner debba risolvere l'obiettivo `Owns(?x1, wooden-table)` dove `?x1` è una variabile precedentemente creata. FabuPlan potrebbe istanziare un nuovo evento `BuysBuilding` oppure riutilizare quello appena creato. Infatti `Owns(john,?o)`, con `?o` una variabile argomento, unifica con l'obietivo `Owns(?x1, wooden-table)` sostituendo `john` a `?x1` e `wooden-table` a `?o`. Supponiamo FabuPlan decida di riutilizzare l'evento, quindi aggiunge all'evento la precondizione aperta

```
In(john, wooden-table)
```

e gli effetti

```
Owns(john, wooden-table),  b<john>Owns(john, wooden-
   table)
```

La situazione sarebbe del tutto analoga alla precedente se in seguito si dovesse presentare un nuovo obiettivo simile, come ad esempio `Owns(john, chair)`.

Abbiamo visto che gli eventi possono avere una qualunque formula di precondizioni: questo significa che le disgiunzioni di formule, e quindi di fatti, sono consentite. Nel planning ad ordinamento parziale, le disgiunzioni di fatti sono realizzate ponendo la formula in forma normale a clausole congiuntive e scegliendo indeterministicamente una clausola come precondizioni dell'evento.

Ad esempio la precondizione $A \wedge \neg(B \wedge C) = A \wedge (\neg B \vee \neg C) = A \wedge \neg B \vee A \wedge \neg C$ ovvero equivale alle due clausole congiuntive $\{A, \neg B\}$ e $\{A, \neg C\}$. Il planner quindi sceglie indeterministicamente una delle due, ad esempio le precondizioni di questo evento sarebbero A e $\neg C$.

3.1.2 Vincoli di ordinamento

La fabula generata da FabuPlan è una rappresentazione simbolica degli eventi che la costituiscono, collegati tra loro da relazioni di causa-effetto e vincoli di ordinamento sotto forma di una relazione di ordinamento parziale.

Definizione 20. Sia E l'insieme degli eventi istanziati in una fabula. Si definisce $\prec \subseteq E \times E$ una relazione di ordinamento parziale, irriflessiva, antisimmetrica e transitiva.

FabuPlan delega la gestione dei vincoli di ordinamento a un sottomodulo che ha il compito di mantenere $\prec$ una relazione di ordinamento e di individuare le inconsistenze. Ad esempio l'ordinamento (e, e) con $e \in E$ è inconsistente in quanto e non può accadere prima di se stesso. In generale questo modulo si accorge della creazione di un ciclo nel grafo degli ordinamenti, marcando la fabula parziale corrente come inconsistente. Il funzionamento di questo modulo è illustrato più in dettaglio nel paragrafo 3.2.6

3.1.3 Collegamenti causali

Abbiamo visto che quando FabuPlan risolve una precondizione aperta di un evento f con un effetto istanziato da un evento e, crea un collegamento causale tra e e f, per indicare che l'effetto di e motiva la precondizione di f ed è protetto da possibili effetti mutex ad esso.

Formalmente

Definizione 21. Un collegamento causale è un'espressione del tipo $e \xrightarrow{\mathcal{F}} f$ dove e è l'evento (istanziato) che produce un effetto $\mathcal{E}$, f è l'effetto (istanziato) avente precondizione aperta $\mathcal{P}$ risolvibile da $\mathcal{E}$ e $\mathcal{F}$ è il risultato dell'unificazione di $\mathcal{E}$ con $\mathcal{P}$.

Un collegamento causale implica il vincolo d'ordinamento $e \prec f$ in quanto e, per poter soddisfare la precondizione di f, deve necessariamente accadere prima.

Quando viene aggiunto un nuovo effetto appartenente ad un evento, FABU-PLAN verifica che l'effetto unifichi con qualche fatto mutex a quello protetto da qualche collegamento causale. Quando questo avviene, si verifica una *minaccia* alla consistenza del piano, che il planner deve in qualche modo risolvere. Una minaccia si verifica anche quando viene aggiunto un nuovo collegamento causale ed esiste un effetto istanziato che unifica con un fatto mutex a quello protetto. Anche in questo caso, la minaccia va in qualche modo risolta.

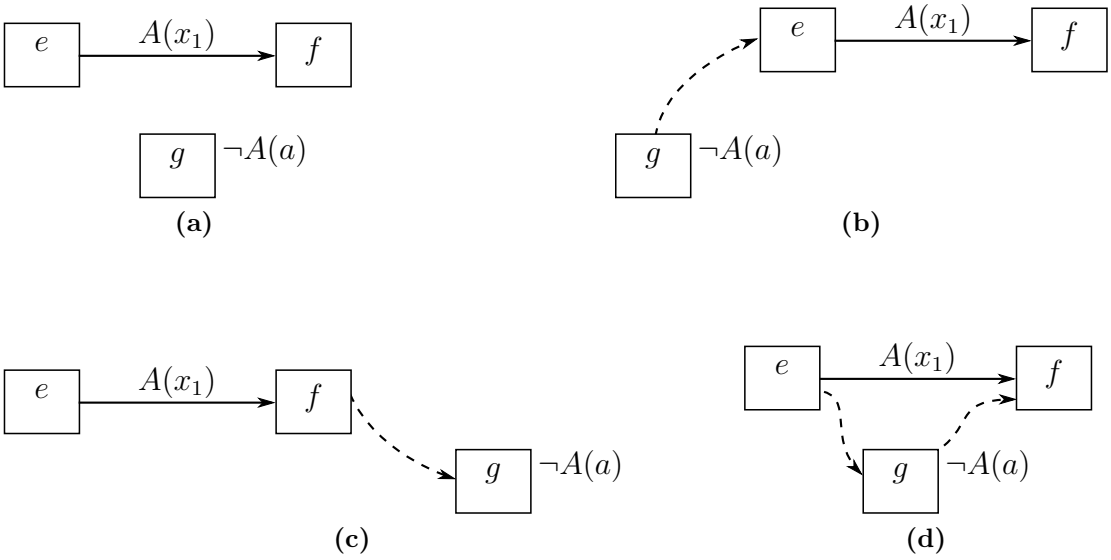


Figura 3.1: *Modi di risoluzione di una minaccia a un collegamento causale: (a) situazione iniziale: una minaccia è stata individuata tra l'evento g e il collegamento causale da e a f; (b) risoluzione per retrocessione; (c) risoluzione per promozione; (d) risoluzione per separazione (aggiunto il vincolo $x_1 \neq a$).*

Quando un collegamento causale $e \xrightarrow{\mathcal{F}} f$ è minacciato dall'effetto $\mathcal{E}$ di un evento g, è possibile mantenere la coerenza del piano in uno dei modi seguenti[13]:

1. *Retrocessione.* Se possibilmente $g \prec e$, aggiungi ai vincoli di ordinamento il vincolo $g \prec e$, ovvero forza l'evento g ad accadere prima di e. In questo modo la minaccia è risolta, poiché $\mathcal{F}$ è reso vero nel mondo *dopo* $\mathcal{E}$ e dunque f può sfruttarlo per risolvere la precondizione aperta.

2. *Promozione.* Se possibilmente $f \prec g$, aggiungi ai vincoli di ordinamento il vincolo $f \prec g$, ovvero forza l'evento g dopo l'evento f (che necessita di $\mathcal{F}$). La minaccia è risolta poiché la cancellazione di $\mathcal{F}$ avviene dopo il suo utilizzo.

3. *Separazione.* Se l'unificatore $U = \langle \Theta, \sigma \rangle$ tra l'effetto mutex di $\mathcal{F}$ e $\mathcal{E}$ è tale che $|\Theta| > 0$, allora scegli indeterministicamente un $\theta = (v, t) \in \Theta$ e imposta $v \neq t$. In questo modo la minaccia è risolta perché almeno una coppia di termini che dovrebbero essere uguali per avere unificazione sono diversi, e dunque $\mathcal{E}$ è sicuramente diverso dal fatto mutex a $\mathcal{F}$. Aggiungi i vincoli di ordinamento $e \prec g \prec f$, ovvero fa che g accada tra e e f.

FabuPlan sceglie indeterministicamente uno di questi tre modi per risolvere una minaccia. Osserviamo, però, che non sempre un modo è applicabile. Ad esempio nel caso 3, Θ dev'essere non vuoto, mentre nei casi 1 e 2, l'aggiunta del vincolo di ordinamento deve mantenere la coerenza del piano (no cicli nell'ordinamento). Se nessuno dei tre modi applicabile, la minaccia non può essere risolta e la fabula parziale corrente è dimostrata essere incoerente e può essere scartata. Se $|\Theta| > 1$ allora ci sono più modi per separare i fatti in conflitto, in quanto basta soltanto una disuguaglianza tra variabile e valore di una sostituzione dell'unificatore per renderli diversi.

Si osserva che i tre casi generano situazioni disgiunte tra loro e che tutti i modi in cui è logicamente possibile risolvere una minaccia sono coperti da questi tre metodi[13].

3.1.4 Pseudocodice

Abbiamo definito singolarmente tutti gli elementi che compongono un problema di planning di FabuPlan. Riassumendo

Definizione 22. Un *problema* è una tupla $\langle T, W, G, \Lambda, R \rangle$ dove

- ◇ T è un insieme di tipi;
- ◇ W è una base che definisce il mondo iniziale, avente il tipo delle costanti individuali contenute in T;
- ◇ G è una base che definisce gli obiettivi della storia, avente il tipo delle costanti individuali contenute in T;
- ◇ Λ è un insieme degli schemi di eventi e regole complesse basati su T;
- ◇ R è un insieme di schemi di regole basate su T.

FabuPlan considera numerosi piani parziali prima di trovare quello soluzione.

Definizione 23. Unq *fabula* è una tupla $\Phi = \langle E, V, O, L \rangle$ dove

- ◇ E è l'insieme degli eventi/regole complesse istanziate;
- ◇ V è un contesto di variabili;
- ◇ O è l'insieme dei vincoli di ordinamento tra gli elementi di E;
- ◇ L è l'insieme dei collegamenti causali tra gli elementi di E.

L'algoritmo di planning comincia costruendo, in un primo piano parziale, due eventi *fittizi*: il primo riflette il mondo iniziale e contiene come effetti istanziati tutti i fatti definiti nella base del mondo iniziale, mentre il secondo riflette gli obiettivi della storia, non ha nessun effetto e ha come precondizioni tutti e soli gli obiettivi della storia. Questi due eventi fittizi rappresentano l'inizio e la fine della fabula.

In Figura 3.2 è mostrato lo pseudocodice dell'algoritmo di planning di FabuPlan. $\langle E, V, O, L \rangle$ è una fabula; E contiene due eventi fittizi: e_w contenente tutti e soli i fatti della base del mondo iniziale, e e_g contenente gli obiettivi della storia come precondizioni. O inizialmente contiene il vincolo $e_w \prec e_g$ (il mondo iniziale viene prima della fine della fabula). F è l'insieme degli obiettivi aperti (*flaw*),

FabuPlan $(\langle E, V, O, L\rangle, F, \Lambda, R)$

1. **Terminazione.** Se V o O sono inconsistenti, fallimento. Altrimenti se F è vuoto restituisci $\langle E, V, O, L\rangle$.

2. Altrimenti:

   (a) **Selezione di un obiettivo.** Scegli una condizione aperta (flaw) all'interno di $f = \langle e_{pre}, \mathcal{P}\rangle \in F$, quindi sia $F' \leftarrow F \ \{f\}$.

   (b) **Selezione dell'operatore.** Scegli indeterministicamente:

      i. un evento (o regola complessa) s_{eff} già istanziato;
      ii. istanzia un nuovo evento (o regola complessa) s_{eff};
      iii. istanzia un effetto condizionale su un evento s_{eff} già istanziato;
      iv. istanzia un effetto condizionale su un nuovo evento s_{eff};

      e sia $\mathcal{E}$ un effetto istanziato di s_{eff} tale che unifichi con un fatto $\mathcal{A} \in \overleftarrow{\gamma}(\mathcal{P})$ per un unificatore $U = \langle\Theta, \sigma\rangle$. Se non esiste tale s_{eff} fai backtracking. Altrimenti, sia $S' = S \cup \{e_{eff}\}$, $O' = O \cup \{e_{eff} \prec e_{pre}\}$, $\forall (v,t) \in \Theta$ aggiungi in V il vincolo $v = t$, $L' = L \cup \{e_{eff} \xrightarrow{\alpha(\mathcal{U},\mathcal{E})} e_{pre}\}$, scegli indeterministicamente una clausola congiuntiva delle eventuali nuove precondizioni di e_{eff} e per ogni fatto $\mathcal{Q}$ nella clausola, aggiungi in F' il nuovo flaw $\langle e_{eff}, \mathcal{Q}\rangle$. Con e_{world} l'evento fittizio del mondo iniziale e e_{goal} l'evento finale degli obiettivi, aggiungi i vincoli di ordinamento $e_{world} \prec e_{eff} \prec e_{goal}$. Se durante l'aggiunta di un qualunque vincolo hai individuato una inconsistenza, fai backtracking.

   (c) **Risoluzione delle minacce.** Per ogni evento istanziato e se esistono un effetto dell'evento $\mathcal{E}$ e un collegamento causale $e_j \xrightarrow{\mathcal{F}} e_k$ tali che $\mathcal{E}$ unifica con qualche fatto $\mathcal{G} \in \mu(\mathcal{F})$ per un unificatore $U = \langle\Theta, \sigma\rangle$ allora scegli non deterministicamente:

      ◇ **Retrocessione.** Se possibilmente $e \prec e_j$ (non è incoerente), allora imposta $O' \leftarrow O' \cup \{e \prec e_j\}$;

      ◇ **Promozione.** Se possibilmente $e_k \prec e$ (non è incoerente), allora imposta $O' \leftarrow O' \cup \{e_k \prec e\}$;

      ◇ **Separazione.** Se $|\Theta| > 0$ scegli indeterministicamente un $(v,t) \in \Theta$ tale che non è dimostrabile $v = t$ in V' e imposta $V' \leftarrow V' \cup \{v \neq t\}$ e $O' \leftarrow O' \cup \{e_j \prec e, e \prec e_k\}$.

   (d) **Chiamata ricorsiva.** Chiama $FabuPlan(\langle E', V', O', L'\rangle, F', \Lambda, R)$.

Figura 3.2: *Pseudocodice dell'algoritmo di planning di* FabuPlan. $\langle E, V, O, L\rangle$ *è una fabula parziale,* F *è l'insieme dei* flaw, Λ *l'insieme degli schemi di eventi e regole complesse,* R *l'insieme delle regole.*

inizialmente per ogni obiettivo della storia $\mathcal{G}$, $\langle \mathcal{G}, e_g \rangle \in F$. Λ è l'insieme degli schemi di evento e regole complesse definiti. R è l'insieme delle regole definite.

L'algoritmo comincia con il controllare che la fabula parziale sia ancora consistente. Se non lo è, viene effettuato backtracking all'ultima fabula parziale coerente. Quando una fabula è coerente e $F = \emptyset$, una soluzione è stata trovata e l'algoritmo termina: completa la fabula assegnando valori casuali e coerenti alle variabili libere. Il secondo passo è quello di risolvere un flaw rimosso da F. Indeterministicamente viene scelto un evento (o regola complessa) nuovo o precedentemente istanziato oppure viene istanziato un nuovo effetto condizionale su un nuovo o precedentemente istanziato evento tale che un suo effetto unifica con un fatto nella chiusura delle regole all'indietro via unificazione del fatto obiettivo. In altre parole se l'obiettivo è un fatto $\mathcal{P}$ allora $\overleftarrow{\gamma}(P)$ sono tutti i fatti che sicuramente implicano $\mathcal{P}$ incluso $\mathcal{P}$ stesso. Quindi se uno di questi è un effetto di qualche evento, tale effetto implica $\overleftarrow{\gamma}P$ è può essere usato per risolvere $\mathcal{P}$. Viene dunque creato un collegamento causale che protegge l'effetto tra l'evento che lo fornisce e l'evento che lo usa, viene aggiunto un vincolo di ordinamento tra i due eventi e ogni fatto nella clausola congiuntiva delle eventuali nuove precondizioni scelta indeterministicamente viene aggiunto a F.

Quindi l'algoritmo passa alla risoluzione delle minacce. Ogni minacca a un collegamento causale da parte di un evento, viene risolta in uno dei tre modi illustrati precedentemente (retrocessione, promozione o separazione). Quindi l'algoritmo reinvoca se stesso ricorsivamente sulla nuova fabula parziale costruita. Un esempio d'esecuzione dell'algoritmo discusso è mostrato nel paragrafo 4.4.

3.2 Dettagli tecnici

In questa sezione vengono discusse le soluzioni di FabuPlan ai più importanti problemi tecnici, essenziali per una corretta e, per quanto possibile, efficiente implementazione dell'algoritmo di planning.

3.2.1 Panoramica del sistema

Dalla descrizione in pseudocodice dell'algoritmo di planning di FabuPlan non emergono i numerosi dettagli che rendono l'implementazione di una algoritmo di planning un compito arduo. FabuPlan riduce la complessità del problema seguendo un approccio *divide-et-impera*, scomponendo il problema generale in un numero di sotto-problemi, e delegando la gestione di ciascuno di questi a specifici moduli indipendenti.

Alla base di qualunque modulo vi è il concetto di *simbolo*. I simboli sono usati per identificare simboli di predicato, di variabile, costanti individuali e operatori modali. Sia per motivi di debug che per mantenere un collegamento con il dominio, una classe di nome `SymbolDictionary` si occupa di mantenere il mapping tra i nomi dei simboli (sotto forma di stringhe di caratteri) e i simboli stessi. Ogni simbolo è rappresentato da un intero senza segno, riservando lo zero come simbolo speciale (*nil*).

I fatti sono rappresentati da una classe `Fact` che contiene essenzalmente una sequenza di simboli. Per motivi di efficienza e di memoria, sia la lunghezza delle modalità, che il numero di argomenti di un predicato sono limitati superiormente da costanti globali. Per gli stessi motivi, la negazione di un fatto o sotto-fatto è rappresentata con un singolo bit. `Fact` espone un metodo per la creazione di un unificatore, se possibile, con un altro fatto.

I tipi, come i simboli, sono gestiti da un *dictionary* che li cataloga e calcola, al momento della definizione di un nuovo tipo, il suo insieme degli antenati, in modo da poter determinare efficientemente se un tipo è sottotipo di un altro. Quindi vi sono classi per rappresentare regole, regole complesse e schemi di eventi. Tutte sono riunite in una libreria centrale che fornisce efficienti metodi per il recupero di tutte le regole aventi un fatto nell'antecedente (o nel conseguente) che unifica con uno dato, e per il recupero degli schemi di eventi che hanno un effetto che unifica con uno dato.

Anche le formule sono gestite da una classe, che si occupa di convertirle in clausole congiuntive.

Fondamentali sono i contenitori di fatti, strutture dati in grado di contenere e catalogare i fatti in modo da poter individuare tutti quelli che unificano con uno dato senza confrontarli uno per uno. La ricerca di fatti in un insieme che unificano con uno esterno è cruciale in ogni parte della pianificazione quindi è stata mostrata particolare cura nell'implementazione di questi contenitori, come meglio illustrato nel paragrafo 3.2.4.

Le basi sono modellate usando i contenitori di fatti. In aggiunta, definiscono e gestiscono le costanti individuali, associandole al loro tipo e offrendo un sistema di ricerca di tutte le costanti individuali di un certo tipo (considerando anche i sotto-tipi).

Non necessario per il planning ma comunque fondamentale è il parser di FabuPlan che legge i file di definizione del problema e genera gli opportuni simboli, popola la libreria di regole e eventi e costruisce la base del mondo iniziale e finale. Il parser è di tipo LL-2 e, come l'analizzatore lessicale, è stato costruito interamente a mano.

La classe `Fabula` offre la funzione di accesso alla generazione di una fabula. Questa, tuttavia, non contiene l'algoritmo di pianificazione, ma implementa la strategia di ricerca A* o hill-climbing per la selezione della prossima fabula parziale da esplorare.

Il cuore dell'algoritmo è implementato nella classe `PartialFabula`. Questa contiene l'interezza dell'algoritmo presentato, ma delega la gestione delle variabili e dei vincoli di ordinamento a sotto-moduli esterni. Questi due sotto-sistemi sono essenziali perché svolgono due funzioni critiche durante la pianificazione e meritano una trattazione più dettagliata, contenuta nei paragrafi successivi.

3.2.2 Contesto di variabili

Abbiamo visto che i termini sono sia costanti individuali che variabili. I simboli di variabile che compaiono in una definizione di argomenti sono in realtà variabili *ausiliarie*, in quanto fungono da segnaposto per gli effettivi argomenti noti solo durante il planning. Gli argomenti di una regola, schema di evento, ecc., sono assegnati a specifici valori in base alle sostituzioni di variabili contenute in un unificatore. Quando durante l'istanziazione di una definizione di argomenti qualche argomento rimane *libero* - non assegnato a nessun altro termine - questo viene assegnato a una nuova variabile creata all'interno della fabula in generazione.

FabuPlan come UCPOP, RePOP, Fabulist, e la maggior parte dei planner segue l'approccio del *least commitment* sull'assegnamento delle variabili che informalmente si traduce nel "cercare di assegnare una variabile il più tardi possibile". Dunque quando una variabile viene creata, per motivi di efficienza non viene indeterministicamente assegnata a qualche possibile costante individuale, ma viene lasciata libera e assegnata solo quando v'è davvero bisogno, ovvero quando è parte di una sostituzione di variabile in una unificazione che va resa *persistente*.

FabuPlan delega la gestione delle variabili a un sottosistema chiamato *contesto di variabili*. Un contesto si occupa di gestire la creazione, l'assegnamento e l'imposizione di vincoli di uguaglianza/disuguaglianza alle variabili. Dunque il contesto di variabili si occupa anche di memorizzare le sostituzioni di un unificatore che vanno rese persistenti.

La funzione cruciale di un contesto di variabili è la gestione automatica delle relazioni di uguaglianza e disuguaglianza tra le variabili. FabuPlan, infatti, continuamente imposta nuovi vincoli sulle variabili di un contesto e questo, a fronte di un nuovo vincolo inserito, verifica che il modello è ancora consistente, e in caso negativo lo comunica al planner che distrugge quel piano parziale.

Internamente, un contesto di variabili è un grafo dove i nodi sono le variabili contenute e le costanti individuali ad esse assegnate e gli archi i vincoli di

uguaglianza/disuguaglianza.

Definizione 24. Un *nodo* è un un termine, ovvero un simbolo di variabile o di costante individuale.

Ogni termine in FabuPlan è associato a un tipo costante. Per indicare il tipo associato a un termine t usiamo la notazione $Tipo[t]$.

Definizione 25. Un *contesto di variabili* Ω è una tupla $\Omega = \langle N, E, I \rangle$ dove N è un insieme di nodi, E la relazione di uguaglianza tra le variabili e I la relazione di disuguaglianza. E è una relazione d'equivalenza poiché riflessiva, simmetrica e transitiva. I una relazione simmetrica e irriflessiva.

Un contesto di variabili è un sistema di inferenza sull'uguaglianza/disuguaglianza delle variabili. In base ai vincoli mano mano accumulati, un contesto fa automaticamente inferenza dei vincoli impliciti. Un contesto offre e fa uso di due servizi di reasoning:

- ◇ alla domanda $x \stackrel{?}{=} y$ dove x e y sono due nodi, risponde affermativamente se e solo se è in grado di dimostrare che i valori dei due nodi sono uguali.
- ◇ alla domanda $x \stackrel{?}{\neq} y$ dove x e y sono due nodi, risponde affermativamente se e solo se è in grado di dimostrare che i valori dei due nodi sono diversi.

Quando non ha elementi sufficienti per dimostrare un'enunciato, la query restituisce il valore falso. Anche se la maggior parte dei nodi saranno variabili, il sistema traccia organicamente i vincoli di uguaglianza/disuguaglianza tra le variabili e tra variabili e costanti individuali (assegnamento). Questi servizi sono usati per determinare lo stato del modello a fronte di un nuovo vincolo. Si possono presentare due situazioni:

- ◇ viene aggiunto il nuovo vincolo $x = y$, ma il contesto può dimostrare che $x \neq y$.

- ◇ viene aggiunto il nuovo vincolo $x \neq y$, ma il contesto può dimostrare che $x = y$.

In entrambi i casi il modello è inconsistente in quanto x e y non possono essere contemporaneamente uguali e diversi, e l'inconsistenza viene segnalata al planner che scarta la fabula parziale corrente ed effettua backtracking.

Vediamo quindi come un contesto dimostra l'uguaglianza/disuguaglianza tra due nodi.

Teorema 2. *Sia $\Omega = \langle V, E, I \rangle$ un contesto di variabili. Siano $x, y \in N$ due nodi. Si può dimostrare che $x = y$ se $(x, y) \in E$.*

Dimostrazione. Se esiste un percorso da x a y tramite E e cioè x è uguale per transitività a y allora $(x, y) \in E$ in quanto E contiene la sia chiusura riflessiva, simmetrica e transitiva. □

Teorema 3. *Sia $\Omega = \langle V, E, I \rangle$ un contesto di variabili. Siano $x, y \in N$ due nodi. Si può dimostrare che $x \neq y$ se:*

1. *t_x e t_y sono due simboli di costante indviduale e $t_x \neq t_y$;*
2. *oppure né $Tipo[x]$ è sottotipo di $Tipo[y]$, né $Tipo[y]$ è sottotipo di $Tipo[x]$;*
3. *oppure $\exists$ due diverse costanti individuali $c_1, c_2 \in N$ tali che si può dimostrare che $x = c_1$ e $y = c_2$.*
4. *oppure $\exists z, w \in N | (x, z) \in E \wedge (z, w) \in I \wedge (w, y) \in E$.*

Dimostrazione. I casi 1,2 e 3 sono ovvi. Nel caso 4 abbiamo che essendo E una relazione di equivalenza è possibile partizionare N per E ottenendo l'insieme quoziente delle classi di equivalenza. Poiché i nodi all'interno di uno stessa classe sono uguali tra loro quindi hanno lo stesso valore, se tra gli elementi della classe di x c'è un elemento che è in relazione I (quindi diverso) da un elemento nella classe di y, allora sicuramente x è diverso da y. □

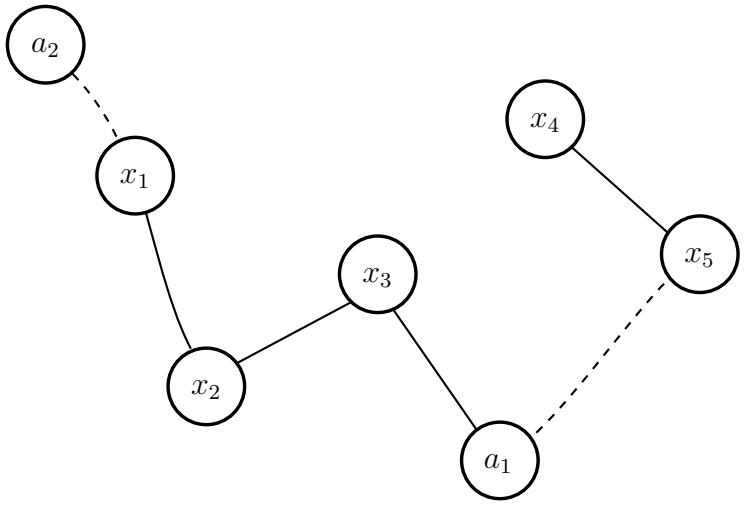


Figura 3.3: *Esempio di situazione iniziale iniziale. Gli archi solidi indicano i vincoli di uguaglianza quelli tratteggiati quelli di disuguaglianza.*

Vediamo un esempio. Supponiamo di avere un contesto di variabili nella situazione iniziale mostrata in figura 3.3 dove gli archi solidi indicano la relazione E, gli archi tratteggiati la relazione I, i nodi x_i le variabili mentre a_i le costanti individuali. Supponiamo di voler aggiungere il vincolo $x_1 = x_4$.

Anzitutto, la relazione E include la sua chiusura transitiva e riflessiva dunque in realtà il grafo completo è mostrato in Figura 3.4.

Il contesto di variabili si chiede quindi se può dimostrare che $x_1 \neq x_4$. Ci accorgiamo che, partendo da x_1 e passando per a_1 e x_5, arriviamo a x_4 (quarto caso dal teorema 3), e dunque la classe di x_1 è diversa da quella di x_4 (Figura 3.5).

3.2.3 Contesto dei vincoli di ordinamento

Come nel caso delle variabili, FabuPlan affida la gestione dei vincoli di ordinamento ad un sottomodulo specifico. Anche questo modulo mantiene una rappresentazione interna dell'ordinamento parziale tra gli eventi sotto forma di grafo orientato. Il suo compito essenziale è quello di individuare cicli all'interno di tale grafo.

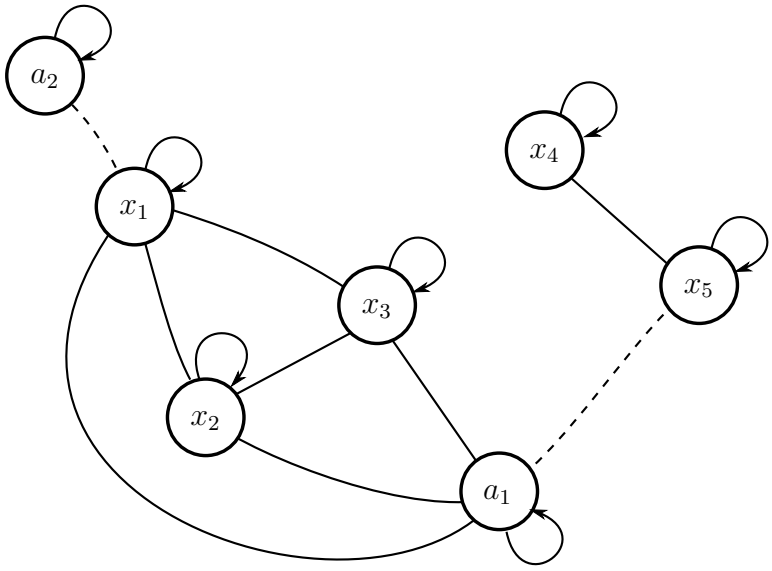


Figura 3.4: *Il contesto di variabile fa in modo di mantere la relazione E riflessiva, simmetrica e transitivia e I simmetrica e irriflessiva.*

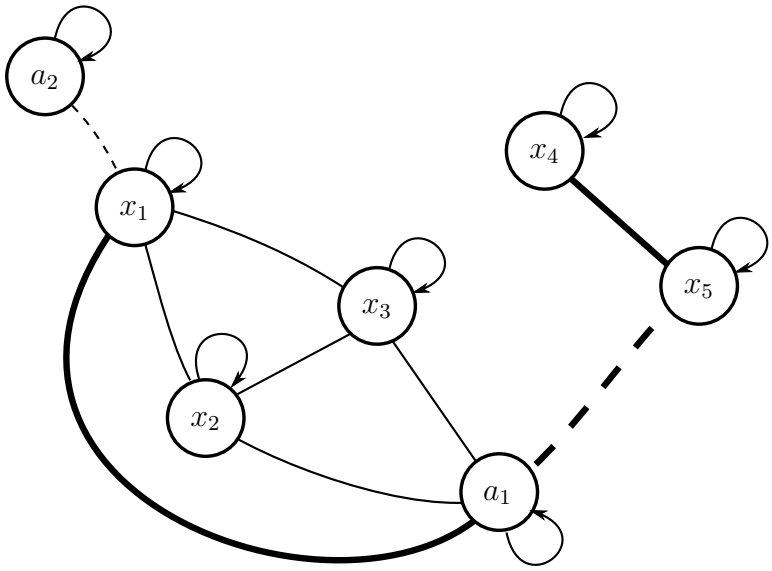


Figura 3.5: *Il percorso in evidenza mostra che $a_1 \neq x_5$ ma $x_1 = a_1$ e $x_4 = x_5$ e dunque $x_1 \neq x_4$*

Come suggerito da REPOP[12], un notevole miglioramento delle prestazioni si ottiene aggiungendo il supporto alle disgiunzioni di vincoli di ordinamento. Osserviamo che la retrocessione e la promozione di un evento rispetto a un collegamento causale sono due soluzioni alternative di risolvere una minaccia, ovvero se e è l'evento che minaccia il collegamento causale $e_j \xrightarrow{\mathcal{F}} e_k$ allora possiamo

modellare retrocessione e promozione in un unico vincolo

$$e \prec e_j \vee e_k \vee e \tag{3.1}$$

poiché per risolvere la minaccia è sufficiente che uno dei due sia vero.

Aggiungendo il supporto alla disgiunzione di vincoli di ordinamento riduciamo l'indeterminismo nella scelta del modo di risoluzione di una minaccia. In altre parole promozione e retrocessione vengono combinati in un unico caso, quello della risoluzione per *ordinamento*. Riassumendo, FabuPlan ha due modi alternativi di risolvere una minaccia: via ordinamento, aggiungendo la disgunzione di vincoli dell'equazione 3.1, e via separazione.

Il modulo delegato alla gestione dei vincoli di ordinamento è il *contesto dei vincoli di ordinamento*.

Definizione 26. Un contesto di vincoli di ordinamento è un insieme O di enunciati della forma

$$e_i \prec e_j \vee e_k \prec e_i$$

o semplicemente

$$e_i \prec e_j$$

dove e_i, e_j, e_k sono eventi istanziati.

Il contesto di vincoli di ordinamento, a fronte di un nuovo vincolo, svolge un processo di deduzione per *risoluzione* (ogni vincolo può essere considerato una clausola disgiuntiva): quando esistono due vincoli c_1: $e_i \prec e_j \vee e_k \prec e_i$ e c_2: $e_j \prec e_i$ allora c_1 viene rimosso e viene aggiunto il vincolo $e_k \prec e_i$. Vediamo un esempio: supponiamo che in una certa iterazione, un contesto di vincoli di ordinamento si presenti nel modo seguente:

$$e_1 \prec e_2 \vee e_3 \prec e_1$$

$$e_3 \prec e_4 \vee e_5 \prec e_3$$

$$e_2 \prec e_3$$

$$e_4 \prec e_5$$

e venga aggiunto un nuovo vincolo $e_3 \prec e_5$. Siccome questo vincolo è in contraddizione con $e_5 \prec e_3$ di $e_3 \prec e_4 \vee e_5 \prec e_3$, quest'ultimo viene rimosso e viene aggiunto il vincolo $e_3 \prec e_4$ (insieme a $e_3 \prec e_5$). Ora il contesto si presenta nel modo seguente:

$$e_1 \prec e_2 \vee e_3 \prec e_1$$

$$e_3 \prec e_4$$

$$e_2 \prec e_3$$

$$e_4 \prec e_5$$

$$e_3 \prec e_5$$

Quindi viene aggiunto il vincolo $e_5 \prec e_2$. Nessuna clausola è riducibile ma ci accorgiamo, con l'aggiunta di questo vincolo, che esiste un ciclo $e_2 \prec e_3 \prec e_5 \prec e_2$ che rende inconsistente il contesto e dunque la fabula di cui esso fa parte.

3.2.4 Contenitore efficiente di fatti

Gli insiemi di fatti sono usati pervasivamente in tutta l'implementazione di FabuPlan. In ogni iterazione di planning vengono effettuate numerose ricerche di fatti in un insieme che unificano con uno query. Vediamo in questo capitolo qualche dettaglio sull'implementazione degli insiemi di fatti in FabuPlan.

Un primo modo naïve di implementare la ricerca dei fatti unificanti in un insieme consiste nel ricercare linearmente, dal primo all'ultimo, i fatti che unificano

con quello dato. Sebbene per piccoli insiemi questo metodo sia sufficiente (e addirittura preferibile a sistemi più complessi), per grandi insiemi comporterebbe una sensibile inefficienza alla già grande complessità dell'algoritmo di planning. In FabuPlan ogni stringa di simbolo è associata a un intero positivo, quindi i fatti sono in ultima analisi sequenze di interi. Gli insiemi di fatti possono essere dunque ordinati per un'arbitraria relazione di ordinamento.

Il principio alla base della ricerca degli effetti unificanti in un insieme consiste nell'operare una serie di ricerche di estremo superiore e inferiore all'interno di intervalli di ricerca via via più piccoli. Supponiamo che i fatti abbiano tutti modalità nulla (nessun operatore modale). Possiamo rappresentare i fatti come una sequenza di interi $p, a_1, a_2, \ldots, a_n$ dove p è il simbolo di predicato e a_i il simbolo dell'argomento i-esimo. Supponiamo di ordinare i fatti in ordine lessicografico, ovvero $\{p, a_1, a_2, \ldots, a_n\} = \mathcal{F} < \mathcal{G} = \{p', a'_1, a'_2, \ldots a'_m\}$ se e solo se $p < q$ oppure $p = q$ e $n < m$ oppure con $n = m$ esiste un $i = 1, \ldots, n$ tale che $\forall j < i-1, a_j = a'_j$ e $a_i < a'_i$. Inoltre conveniamo che per ogni variabile v e costante individuale c, $v < c$.

Saremmo tentati di cercare un algoritmo con complessità nel caso pessimo $O(\log N)$ con N numero di fatti nell'insieme ad esempio mediante una ricerca binaria, ma questo non è possibile per la presenza delle variabili. Quello che cerchiamo infatti non è la presenza o meno di un fatto nell'insieme, ma tutti i fatti che unificano con uno query, che naturalmente possono essere più d'uno se sono presenti variabili.

La tecnica di ricerca utilizzata funziona nel seguente modo: sia A un insieme ordinato di fatti e $\mathcal{F} = \{p, a_1, a_2, \ldots, a_k\}$ il fatto che si vuole unificare con i fatti dell'insieme A. Sia $[l, u)$ la coppia di estremi inferiori e superiori dell'intervallo di ricerca attuale. Cominciamo con $[l, u)$ pari al limite inferiore e superiore di tutti i fatti aventi predicato p. Per cercare i due limiti è possibile utilizzare una ricerca binaria con complessità nel caso pessimo $O(\log N)$.

Sia $i = 1$:

1. Se $l = u - 1$ allora segnala il fatto con indice l come fatto unificante con $\mathcal{F}$;
2. Altrimenti:
   (a) Se a_i è una variabile, allora l'argomento i-esimo di $\mathcal{F}$ unifica con quello di tutti i fatti in $[l, u)$. Richiama la procedura ricorsivamente da (1) con $i' \leftarrow i + 1$ per ogni coppia $[l', u')$ con $l' \geq l \wedge u' \leq u$ tale che ogni fatto in $[l', u')$ ha stesso argomento i-esimo.
   (b) Se a_i è una costante individuale, allora sia $i' \leftarrow i + 1$. Sia $u_v \in [l, u]$ l'indice del primo fatto in $[l, u)$ che ha argomento i-esimo costante individuale. In altre parole u_v è il limite superiore dei fatti in $[l, u)$ aventi come argomento i-esimo una variabile (ogni variabile è minore di ogni costante individuale). Per quanto riguarda l'argomento i-esimo, tutti i fatti in $[l, u_v)$ sono unificabili con $\mathcal{F}$ perché l'argomento i-esimo di questi fatti è una variabile. Richiama la procedura ricorsivamente da (1) con $i' \leftarrow i + 1$ per ogni coppia $[l', u')$ con $l' \geq l \wedge u' \leq u_v$ tale che ogni fatto in $[l', u')$ ha stesso argomento i-esimo. Quindi chiama la procedura ricorsivamente da (1) con $i' \leftarrow i + 1$ per la coppia $[l', u')$, se esiste, con $l' \geq u_v \wedge u' \leq u$ tale che ogni fatto in $[l', u')$ ha l'argomento i-esimo pari a a_i.

Il calcolo formale della complessità dell'algoritmo mostrato è molto complesso, in quanto dipende anche da altri fattori come il numero di simboli di variabili e di costanti individuali oltre al numero di fatti nell'insieme. Intuitivamente l'algoritmo funziona bene se l'insieme è *sparso*, ovvero quando non contiene molti fatti "quasi uguali". L'algoritmo interrompe il ramo di ricerca attuale quando il prossimo sotto-intervallo calcolato ha i due estremi coincidenti (lunghezza nulla). Ad esempio, se l'argomento i-esimo del fatto query è una costante individuale, l'algoritmo è eseguito ricorsivamente su tutti gli intervalli con stessa variabile nella

colonna i-esima e su quello con l'argomento i-esimo pari alla costante individuale. Se ad esempio nessun sotto-intervallo valido è disponibile, l'algoritmo interrompe la ricerca per l'intervallo attuale. La ricerca di ciascun intervallo è logaritmica in quanto i fatti sono ordinati.

L'algoritmo presentato in realtà è diverso da quello implementato poiché è vincolato al caso molto più semplice in cui i fatti sono senza modalità e senza schema di fatti • che invece sono, come sappiamo, presenti in FabuPlan.

3.2.5 Euristiche

Nella definizione dell'algoritmo nel paragrafo 3.1.4, viene usata la parola "indeterministicamente". Nell'implementazione dell'algoritmo, per motivi di efficienza, una scelta indeterministica dev'essere guidata da una qualche strategia. Non è possibile determinare con sicurezza quale sarà la scelta giusta, altrimenti sarebbe equivalente a risolvere il problema stesso. Le scelte possono essere tuttavia guidate da funzioni euristiche che valutino le possibili opzioni e permettano di scegliere quella che *sembra* più promettente.

Un modo naïve consiste nel considerare ogni opzione di equa importanza. In questa situazione la scelta è indifferente e non si ottiene nessun guadagno in efficienza anche se ci sono delle scelte migliori di altre. Nell'implementazione di FabuPlan ci sono sostanzialmente due situazioni nelle quali operare una scelta: selezione della prossima fabula parziale e selezione del flaw da risolvere. Quando è possibile risolvere un flaw in più modi, FabuPlan crea altrettante copie della fabula corrente e in ciascuna risolve il flaw in modo differente. Questo insieme di fabule parziali è mantenuto in una *coda* ordinata in modo che la più promettente appaia in cima, ovvero in ordine crescente di f, valore calcolato nel seguente modo

$$f(\Phi) = c(\Phi) + h(\Phi) + n(\Phi)$$

dove

- Φ è una fabula;

- $c(\Phi)$ è la somma del costo degli eventi istanziati in Φ;
- $h(\Phi)$ è una stima euristica del costo degli eventi necessari a completare Φ;
- $n(\Phi)$ è un disturbo che aggiunge casualità alla fabula generata.

Il costo default di ogni evento è 1, ma il progettista può impostare un costo arbitrario nella definizione dell'evento con la parola chiave `cost`. La stima del costo al completamento è semplicemente il numero delle precondizioni aperte in Φ, supponendo, in modo molto ottimistico, che sia necessario un solo evento per soddisfare ogni precondizione aperta. Quest'euristica non è molto informata e migliori ma più complesse euristiche sono costruibili, come mostrato in [12]. $n(\Phi)$ è un rumore bianco con una certa ampiezza che aggiunge disturbo alla scelta della fabula più promettente in modo che non venga sempre generata la stessa fabula. Maggiore è l'ampiezza, più variabilità si ottiene nei piani generati, ma più fatica farà il planner a trovare una soluzione poiché diventa sempre meno indicativo il valore f.

L'altra scelta indeterministica importante riguarda quella del flaw da risolvere. FabuPlan segue il seguente schema:

1. viene scelto il flaw con *priorità* maggiore, dove la priorità è il numero di eventi che producono l'effetto per qualche unificazione;
2. altrimenti viene scelto il flaw inserito nell'iterazione più lontana da quella attuale;
3. altrimenti viene scelto il flaw con numero di argomenti maggiore;
4. altrimenti viene scelto il flaw con modalità più lunga.

La giustificazione del vantaggio dall'usare quest'euristica rispetto a selezionare un flaw casuale è motivato dai risultati empirici che mostrano un miglioramento nell'efficienza (rispetto a una scelta casuale) in virtù del minor numero di piani parziali esplorati per trovare una soluzione.

Per la selezione della fabula più promettente, FabuPlan può usare un approccio best-first (A*) o hill-climbing. Nel primo caso viene mantenuta una lista ordinata secondo il valore f e viene ogni volta selezionata la fabula con valore f minore. In questo modo non vengono "dimenticate" le soluzioni parziali che all'inizio sembravano poco promettenti, così se il problema è particolarmente difficile e le scelte che apparivano promettenti all'inizio erano sbagliate, il planner può fare backtracking alle fabule parziali inizialmente sembrate "controintuitive", che in realtà guidano verso una soluzione. A* garantisce completezza della ricerca, a scapito di un notevole uso di memoria. In alternativa, possiamo sacrificare la completezza per una maggiore efficienza: l'approccio *hill-climbing* usato da FabuPlan è una versione modificata dell'approccio classico. Nell'hill-climbing classico non viene mantenuta alcuna alternativa in memoria ma ad ogni scelta vengono generate le alternative e viene considerata solo quella più promettente (f minore) scartando tutte le altre. Com'è noto, questo approccio è soggetto ai minimi/massimi locali, e l'algoritmo potrebbe non trovare una soluzione sebbene questa esista (incompletezza). Un miglioramento consiste nel mantenere in memoria una variabile min e nell'utilizzare A* fin quando la migliore alternativa della lista aperta ha un valore di $h(\Phi)+n(\Phi) < min$. Quando questo avviene $min = h(\Phi)+n(\Phi)$ e la lista aperta viene svuotata, mantenendo solo la fabula parziale più promettente. Questo sistema forza la ricerca nel mantenere in memoria tutte le alternative finché una, concretamente, comporta un avvicinamento alla soluzione. Questo approccio si basato sul modo in cui il planner Fast-Forward[8] conduce la ricerca nello spazio degli stati.

3.2.6 Risoluzione di minacce: completezza contro efficienza

Abbiamo visto nel paragrafo 3.1.3 e che vi sono essenzialmente due modi di risolvere un collegamento causale: per ordinamento e per separazione. La scelta indeterministica di porre l'evento che causa il conflitto prima o dopo l'intervallo

protetto dal collegamento causale è risolta dal supporto alle disgiunzioni di vincoli di ordinamento. Se però Θ dell'unificazione contiene più di una sostituzione, la scelta della coppia variabile-valore in Θ da separare rimane indeterministica.

Nella risoluzione dei conflitti, ogni scelta indeterministica comporta la duplicazione delle alternative che si stanno generando. La situazione è meglio illustrata con un esempio: supponiamo venga risolto un flaw mediante l'istanziazione di un nuovo evento e supponiamo che vengano individuate due minacce. La prima è risolvibile via ordinamento e separazione per un unica sostituzione. La fabula, sulla quale il planner sta attualmente lavorando, viene duplicata: nella prima copia la minaccia viene risolta via ordinamento, nella seconda via separazione. Anche la seconda minaccia è risolvibile per ordinamento e per una separazione, quindi le due fabule precedenti vengono nuovamente duplicate ottenendo quattro alternative: nelle prime due la seconda minaccia è risolta per ordinamento, nelle altre due per separazione.

Questa continua duplicazione delle alternative per i modi in cui possiamo risolvere le minacce causa inefficenze congenite. Se però possiamo sacrificare completezza, ovvero la capacità di trovare una soluzione da parte di un algoritmo quando questa esiste, e quindi accettiamo che alcune volte il planner fallisca nella ricerca della soluzione anche se questa esiste, possiamo guadagnare in efficienza.

FabuPlan supporta tre modalità per regolare il trade-off tra completezza ed efficienza nella risoluzione delle minacce:

1. La più completa. Nessuna semplificazione, la completezza è mantenuta e ogni modo in cui è possibile risolvere una minaccia è applicato a una copia della fabula attuale;

2. Le minacce vengono risolte per ordinamento e per separazione, ma non viene creato un duplicato per ogni separazione possibile (ogni coppia variabile-valore), ma ne viene scelta una casualmente;

3. La più efficiente. Inizialmente scegli casualmente se risolvere la minaccia per ordinamento o per separazione. Se risolvi per separazione, scegli casualmente la separazione tra quelle possibili.

Naturalmente i metodi (2) e (3), non essendo completi, potrebbero portare il planner a non trovare soluzioni per un certo numero di esecuzioni per poi magari trovarne una all'esecuzione successiva (avendo cura di impostare il seme del generatore dei numeri casuali ad un valore differente per ogni esecuzione).

Capitolo 4

Esempi

In questo capitolo verranno illustrati una serie di esempi di definizioni, piccole esecuzioni e infine un esempio di esecuzione completo.

4.1 Esempio di gerarchia di tipi

I tipi consentono di modellare le entità del dominio di una storia. Sebbene FabuPlan non richieda di vincolare il tipo degli argomenti di un predicato, la consistenza dei tipi degli argomenti è fortemente consigliata.

Segue un semplice esempio di definizione di tipi.

```
type Entity;
type LivingEntity is Entity;
type Human is LivingEntity;
type Man,Woman is Human;
type Prince,King is Man;
type Princess,Queen is Woman;
type Prince,King,Princess,Queen is Monarch;
```

4.2 Esempi di regole

Le regole sono un ottimo strumento per la definizione degli assiomi dipendenti e indipendenti dal dominio.

4.2.1 Per assiomi del mondo

Le regole permettono di definire efficientemente e concisamente assiomi che modellano le meccaniche del mondo in un dominio. Possiamo modellare ad esempio la simmetria di una relazione:

```
rule (Person ?x, Person ?y) with (x != y): Friends(?x, ?y
   ) implies (Friends(?y, ?x);
```

I sotto concetti:

```
rule (Nation ?x, Nation ?y) with (x != y): InWar(?x, ?y)
   implies AreEnemy(?x,?y);
```

I fatti opposti:

```
rule (Person ?x): Alive(?x) implies !Dead(?x);
```

E inserendo un fatto dummy *Top* nel mondo iniziale, la riflessività di una relazione:

```
rule (Person ?x): Top implies Knows(?x, ?x); // ognuno
   conosce se stesso.
```

4.2.2 Per assiomi della logica

Ogni logica può avere uno schema di assiomi validi all'interno di essa. Ad esempio in una logica modale normale vale l'assioma K

$$\Box(\mathcal{A} \to \mathcal{B}) \to (\Box\mathcal{A} \to \Box\mathcal{B})$$

Possiamo usare le regole e gli schemi di fatti per introdurre assiomi fondamentali della logica di FabuPLANche modellino le implicazioni tra modalità. Ad esempio vorremmo modellare che chi crede qualcosa crede di crederla e viceversa.

```
rule (Person ?x): b<?x>... implies b<?x>b<?x>...;
```

e

```
rule (Person ?x): b<?x>b<?x>... implies b<?x>...;
```

Se qualcuno vuole volere qualcosa allora la vuole.

```
rule (Person ?x): w<?x>w<?x>... implies w<?x>...;
```

Se qualcuno sa di volere qualcosa allora la vuole.

```
rule (Person ?x): b<?x>w<?x>... implies w<?x>...;
```

e così via.

4.2.3 Atti comunicativi

L'uso di effetti condizionali contenenti schemi di fatti all'interno degli schemi di eventi gioca un ruolo fondamentale nella definizione dei possibili *atti comunicativi*[7] tra i protagonisti della storia. Supponiamo di voler definire l'evento $TalksTo$, la comunicazione verbale di un fatto tra due protagonisti. Un modo molto semplificato di farlo è il seguente:

```
event TalksTo(Person ?p1, Person ?p2) with (?p1 != ?p2) {
   in: b<?p1>... and w<?p1>b<?p2>... and CanSpeak(?p1)
      and CanHear(?p2);
   out: b<?p1>b<?p2>... and b<?p2>b<?p1>... and b<?p2
      >...;
}
```

In italiano sarebbe: "se una persona p_1 crede sia vero qualcosa, vuole che anche un'altra persona p_2 lo creda, p_1 può parlare e p_2 può sentire, allora p_1 gliela può dire e l'effetto sarebbe che p_1 crederà che p_2 creda che sia vero quel qualcosa (e viceversa) e che p_2 effettivamente crederà che quel qualcosa sia vero". Lo schema definito in questo modo presenta tre problemi:

1. Non è detto che p_2 creda ciecamente a quello che gli viene detto da chiunque;
2. Se p_1 volesse dire a p_2 più di un fatto, il planner avrebbe bisogno di più eventi $TalksTo$ piuttosto che, se fosse possibile, riutilizzare lo stesso evento per tutti i fatti comunicati (p_1 direbbe *più cose* a p_2 nello stesso evento di dialogo).

3. la comunicazione avverrebbe sempre e solo tra una persona e un altra. Potremmo desiderare di modellare il caso in cui il parlante è vicino a molti ascoltatori che ascoltano quello che dice.

Il primo problema può essere risolto aggiungendo il fatto $Trusts(x, y)$ per modellare il fatto che x si fida di y e che quindi crede a quel che y gli dice. Aggiungendo però la precondizione direttamente a $TalksTo$, si avrebbe che non può mai accadere l'evento $TalksTo$ se p_2 non si fida di p_1 e questo non è vero: p_1 potrebbe voler dire qualcosa a p_2 anche se questo non si fida del primo. Per risolvere correttamente il problema usiamo due effetti condizionali: nel primo modelliamo il caso in cui p_2 non si fida di p_1 e nell'altro invece sì. Un miglior modo di modellare $TalksTo$ è dunque il seguente:

```
event TalksTo(Person ?p1, Person ?p2) with (?p1 != ?p2) {
   in: Near(?p1, ?p2) and b<?p1>Near(?p1,?p2) CanSpeak(?
      p1) and CanHear(?p2);
   when () {
      in: !Trusts(?p2, ?p1) and b<?p1>... and w<?p1>b<?p2
         >...;
      out: b<?p1>b<?p2>... and b<?p2>b<?p1>...;
   }
   when () {
      in: Trusts(?p2, ?p1) and b<?p1>... and w<?p1>b<?p2
         >...;
      out: b<?p2>... and b<?p1>b<?p2>... and b<?p2>b<?p1
         >...;
   }
}
```

Questa soluzione risolve anche il secondo problema, dando la possibilità al planner di istanziare più volte gli effetti condizionali con diverso σ: in questo modo, più fatti possono essere comunicati con uno stesso evento $TalksTo$. Infine per risolvere il terzo problema, basta spostare la dichiarazione dell'argomento `?p2` all'interno degli effetti condizionali.

```
event TalksTo(Person ?p1) {
   in: CanSpeak(?p1);
   when (Person ?p2) with (?p1 != ?p2)  {
```

```
      in: !Trusts(?p2, ?p1) and b<?p1>... and w<?p1>b<?p2
         >... and Near(?p1, ?p2) and b<?p1>Near(?p1,?p2)
         and CanHear(?p2);
      out: b<?p1>b<?p2>... and b<?p2>b<?p1>...;
   }
   when (Person ?p2) with (?p1 != ?p2)  {
      in: Trusts(?p2, ?p1) and b<?p1>... and w<?p1>b<?p2
         >... and Near(?p1, ?p2) and b<?p1>Near(?p1,?p2)
         and CanHear(?p2);
      out: b<?p2>... and b<?p1>b<?p2>... and b<?p2>b<?p1
         >...;
   }
}
```

in questo modo più persone possono ascoltare più fatti diversi.

4.3 Esempio di problema di planning classico

Testiamo la correttezza del planning con il classico problema del mondo a blocchi. Un buono scenario di testing è quello che riproduce l'anomalia di Sussman, un problema complesso data la presenza di obiettivi non indipendenti tra loro. La definizione del problema è la seguente:

```
type Surface;
type Box is Surface;

base world {
   Surface table;
   Box a,b,c;
   On(c,a);
   // Zucchero sintattico per On(a,table), On(b,table)
   On([a,b],table);
   Free([b,c]);
}

base goal {
   On(a,b);
   On(b,c);
}
```

```
rule (Box ?b1, Surface ?b2): On(?b1,?b2) implies !On(?b2
   ,?b1);
rule (Box ?b1, Box ?b2): On(?b1,?b2) implies !Free(?b2);


event Move(Box ?box, Surface ?from, Box ?to) {
   in: On(?box,?from) and Free(?box) and Free(?to);
   out: On(?box,?to) and !On(?box,?from) and !Free(?to)
      and Free(?from);
}

event PutOnTable(Box ?box, Box ?from) {
   in: On(?box, ?from) and Free(?box);
   out: On(?box, table) and !On(?box,?from) and Free(?
      from);
}
```

La difficoltà del problema sta nel fatto che `On(a,b)` e `On(b,c)` sono obiettivi l'uno dipendente dall'altro, ovvero per poter risolvere il problema, il planner non può risolvere indipendentemente il primo obiettivo e poi il secondo. Fortunatamente i planner POCL (quindi FabuPlan) non soffrono di questa anomalia, ma il problema rimane non banale. La soluzione minima consiste nello spostare c sul tavolo, quindi spostare b su c e a su b. Attivando la modalità 1 di risoluzione dei conflitti (che mantiene la completezza), il piano viene trovato dopo 175 ms[1] esplorando 31 stati, con 495 piani parziali nella lista aperta e 19 in quella chiusa. Lo stesso problema risolto in modalità di risoluzione dei conflitti numero 3 (più efficiente ma non pià completo), la soluzione è stata trovata in in 146 ms, con un numero di stati esplorati che si riduce a 12, 8 piani parziali nella lista aperta e 5 in quella chiusa. Su dieci esecuzioni in questa modalità, tuttavia, cinque si sono concluse con esito positivo (hanno trovato una soluzione). In tutte le esecuzioni che hanno trovato una soluzione, questa era minima.

[1]Test eseguiti su una macchina con processore Intel i7-2620M, 2.70 GHz, 4,00 Gb RAM DDR3 665.3 Mhz.

4.4 Esempio di generazione di fabula

Proviamo a generare la fabula di una storia combinando gli strumenti offerti da FabuPlan. Si supponga di voler definire il dominio di una classica favola che narra di un cavaliere che, per trovar consorte, uccide un drago che difende una torre nella quale è confinata una principessa. Tuttavia il cavaliere all'inizio della storia non sa dove sia nascosto il drago, mentre un saggio che conosce la sua posizione.

Si definisce il dominio della fabula nel seguente modo.

```
type Person;
type Man, Woman is Person;
type Knight, WiseMan is Man;
type Princess is Woman;

type Dungeon;
type Tower is Dungeon;

type Monster;
type Dragon is Monster;

base world {
   Knight william;
   Princess elizabeth;
   WiseMan george;

   Tower dark-tower;
   Dragon gurag;

   Confined(elizabeth, dark-tower);
   Defends(gurag, dark-tower);
   Alive([william,elizabeth,george,gurag]);

   At(gurag, dark-tower);

   b<george>At(gurag, dark-tower);
   Powerful(william);

   w<william>IsMarried(william);
}
```

```
base goal {
   IsMarried(william);
}
```

La storia comincia con tre personaggi, il cavaliere *William*, la principessa *Elizabeth* e il saggio *George*. La principessa è confinata in una torre oscura difesa dal temibile drago *Gurag*. Naturalmente, all'inizio sia i personaggi che il drago sono vivi. Gurag è alla torre ma solo George lo sa. Il cavaliere William desidera aver consorte (essere sposato) con qualche fanciulla e la storia termina con William sposato.

Nel nostro dominio, se qualcuno ama una persona allora desidera sposarla. Definiamo quindi le seguenti regole.

```
rule (Man ?m, Woman ?w): Loves(?m, ?w) implies w<?m>
   Married(?m,?w);
rule (Man ?m, Woman ?w): Loves(?w, ?m) implies w<?w>
   Married(?m,?w);
```

Le regole seguenti descrivono l'ovvia implicazion che se qualcuno desidera essere sposato allora vorrà essere sposato con qualcuno e viceversa.

```
rule (Man ?m, Woman ?w): w<?m>Married(?m,?w) implies w<?m
   >IsMarried(?m);
rule (Man ?m, Woman ?w): w<?w>Married(?m,?w) implies w<?w
   >IsMarried(?w);

rule (Man ?m, Woman ?w): w<?m>IsMarried(?m) implies w<?m>
   Married(?m,?w);
rule (Man ?m, Woman ?w): w<?w>IsMarried(?w) implies w<?w>
   Married(?m,?w);
```

Modelliamo un evento per "trasmettere" desideri da un personaggio a un altro. Nell'evento *coerce* (in italiano *costringi*) se una persona è potente, allora può costringere un'altra persona a desiderare un suo stesso desiderio. Potremmo modellare altri eventi come ad esempio "chiede favore" o "assolda".

```
event Coerce(Person ?p1) cost 2{
   in: Powerful(?p1);
```

```
    when (Person ?p2) with (?p1 != ?p2) {
        in: w<?p1>...;
        out: w<?p2>...;
    }
}
```

Quindi modelliamo un evento per il trasferimento di conoscenza da un personaggio a un altro, ad esempio via comunicazione orale.

```
event Tell(Person ?p1) cost 2{
    when (Person ?p2) with (?p1 != ?p2) {
        in: b<?p1>... and w<?p1>b<?p2>...;
        out: b<?p2>... and b<?p1>b<?p2>... and b<?p2>b<?p1
            >...;
    }
}
```

Ora modelliamo l'evento in cui un cavaliere uccide un mostro in un luogo. Quest'evento ha l'effetto condizionale `Freed` per modellare che se il mostro ucciso difendeva un luogo e una persona era confinata nel luogo allora il cavaliere ha liberato questa persona.

```
event Slay(Knight ?k, Monster ?m, Dungeon ?d) {
    in: At(?m, ?d) and b<?k>At(?m,?d) and Alive(?m);
    out: !Alive(?m);
    when(Person ?p) with (?k != ?p) {
        in: Defends(?m,?d) and Confined(?p, ?d) and w<?k>
            Freed(?k,?p);
        out: Freed(?k, ?p);
    }
}
```

In questo semplice dominio cavalleresco, una principessa liberata da un uomo si innamora dell'uomo. Modelliamo quest'evento nel modo seguente:

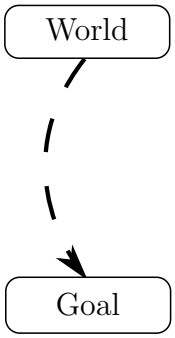
```
event FallInLove(Princess ?p, Man ?k) {
    in: Freed(?k, ?p);
    out: Loves(?p, ?k);
}
```

Infine se due persone vogliono essere sposate, allora si prendono in matrimonio.

```
event Marry(Man ?m, Woman ?w) {
```

```
    in: w<?w>Married(?m,?w) and w<?m>Married(?m,?w);
    out: IsMarried(?m) and IsMarried(?w) and Married(?m,?w
       );
}
```

Vediamo ora la sequenza di iterazioni che FabuPlan svolge per generare una fabula per questo dominio, supponendo faccia sempre la scelta giusta. Inizialmente sono presenti due eventi fittizi, l'uno (e_{world}) avente come effetti i fatti del mondo iniziale e l'altro (e_{goal}) avente come precondizioni gli obiettivi della storia. La situazione è la seguente.



La lista di flaw iniziale contiene soltanto una precondizione

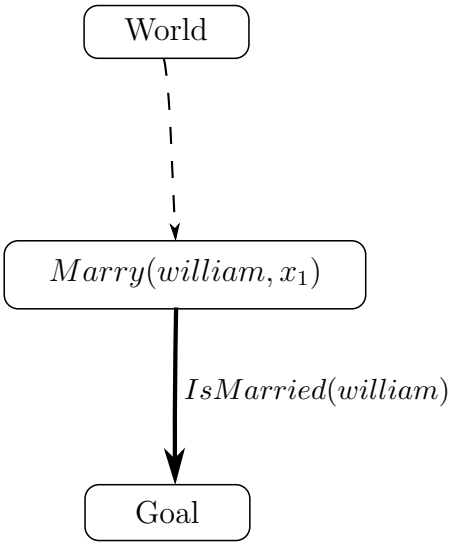
$$F = \{\langle e_{goal}, IsMarried(william)\rangle\}.$$

Viene scelto quindi questo (unico) flaw. L'unico modo è risolverlo mediante l'istanziazione di un nuovo evento $Married$. L'evento viene creato con l'opportuno unificatore minimo e viene creato un collegamento causale tra il nuovo evento e e_{goal} per proteggere

$$IsMarried(william)$$

quindi vengono aggiunti i vincoli di ordinamento con l'evento iniziale e finale.

Per mantenere la figura pulita, per convenzione nelle figure ogni arco solido (collegamento causale) implica un arco trattegiato (vincolo di ordinamento), e i vincoli di ordinamento ottenibili per transitività sono stati nascosti. L'etichetta dell'arco solido rappresenta il fatto protetto dal collegamento causale.



Successivamente vengono aggiunte le due precondizioni

$$w_{william} Married(william, x_1)$$

e

$$w_{x_1} Married(william, x_1).$$

Scegliamo di risolvere prima

$$\mathcal{P} = w_{william} Married(william, x_1).$$

Questo evento non è direttamente risolvibile da nessun effetto, né dal mondo iniziale. Ci accorgiamo però che $\overleftarrow{\gamma}(\mathcal{P})$ contiene

$$w_{william} IsMarried(william)$$

per la regola

```
rule (Man ?m, Woman ?w): w<?m>IsMarried(?m) implies w
   <?m>Married(?m,?w)
```

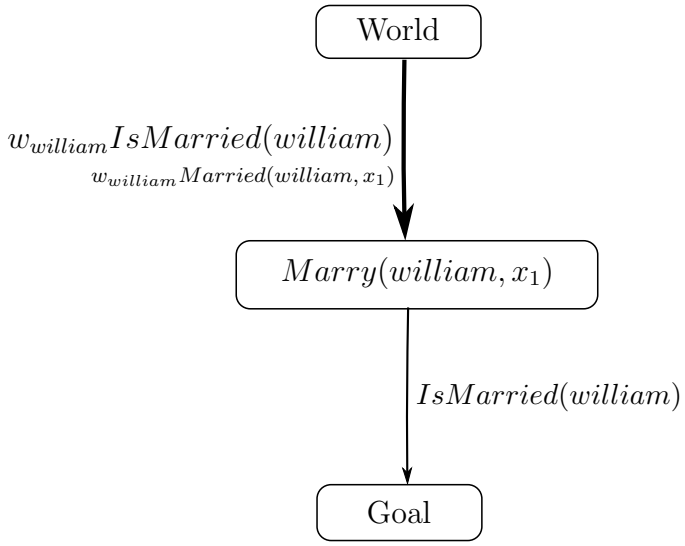
Dunque risolviamo

$$w_{william} Married(william, x_1)$$

per il fatto nel mondo

$$w_{william} IsMarried(william)$$

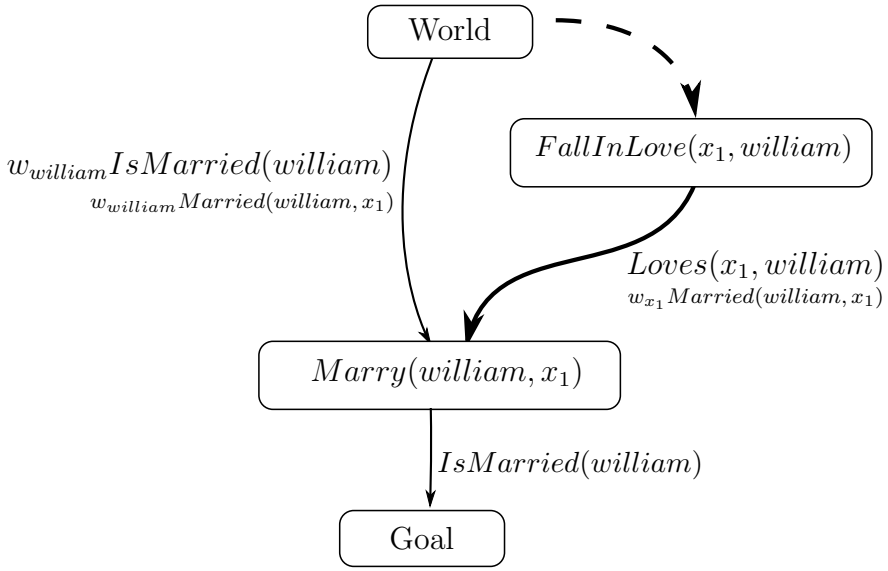
creando un collegamento causale tra l'evento del mondo iniziale e l'evento $Marry$. Poiché l'effetto protetto dal collegamento causale risolve una precondizione diversa da se stesso, indichiamo in piccolo la precondizione risolta.



Ora soddisfiamo la precondizione che qualcuno voglia sposare *William*, ovvero il fatto

$$\mathcal{P} = w_{x_1} Married(william, x_1).$$

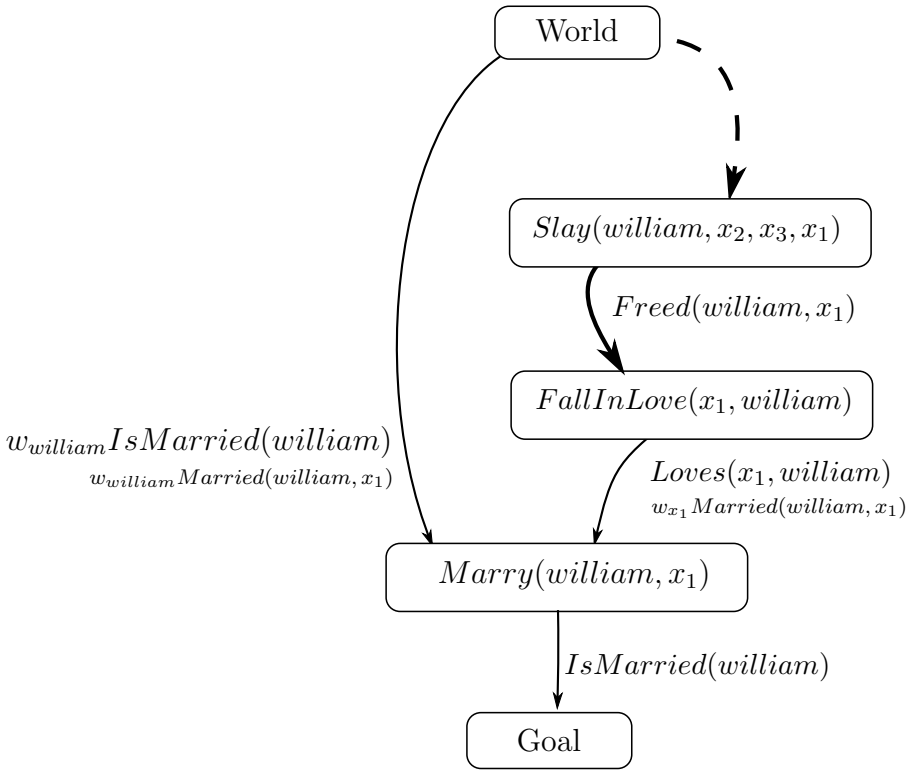
Anche questo fatto non è direttamente risolvibile, ma anche stavolta ci vengono in aiuto le regole, in quanto abbiamo definito che se qualcuno ama qualcun'altro allora vuole sposarlo. Il fatto $Loves(x_1, william)$ è infatti in $\overleftarrow{\gamma}(\mathcal{P})$ ed è effetto dell'unico evento $FallInLove$, dunque lo istanziamo.



La precondizione

$$Freed(william, x_1)$$

è data dall'evento *Slay* che viene istanziato. Vengono create le variabili x_2, il mostro ucciso, e x_3, il luogo nel quale il mostro viene ucciso. Siccome l'evento è istanziato da un effetto condizionale, le precondizioni di questo sono aggiunte a quelle dell'evento.



Sostituendo $x_1 = elizabeth$, $x_2 = gurag$ e $x_3 = darktower$ otteniamo che le precondizioni

$$Alive(x_2), Confined(x_{1,x}\,3), Defends(x_2, x_3) \text{ e } At(x_2, x_3)$$

sono direttamente risolvibili dal mondo iniziale. L'ultima precondizione dell'evento

$$w_{william} Freed(william, x_1)$$

richiede più attenzione: questa precondizione è risolvibile mediante giustificazione dell'intenzione implicita, in particolare, il fatto che william voglia liberare x_1 si

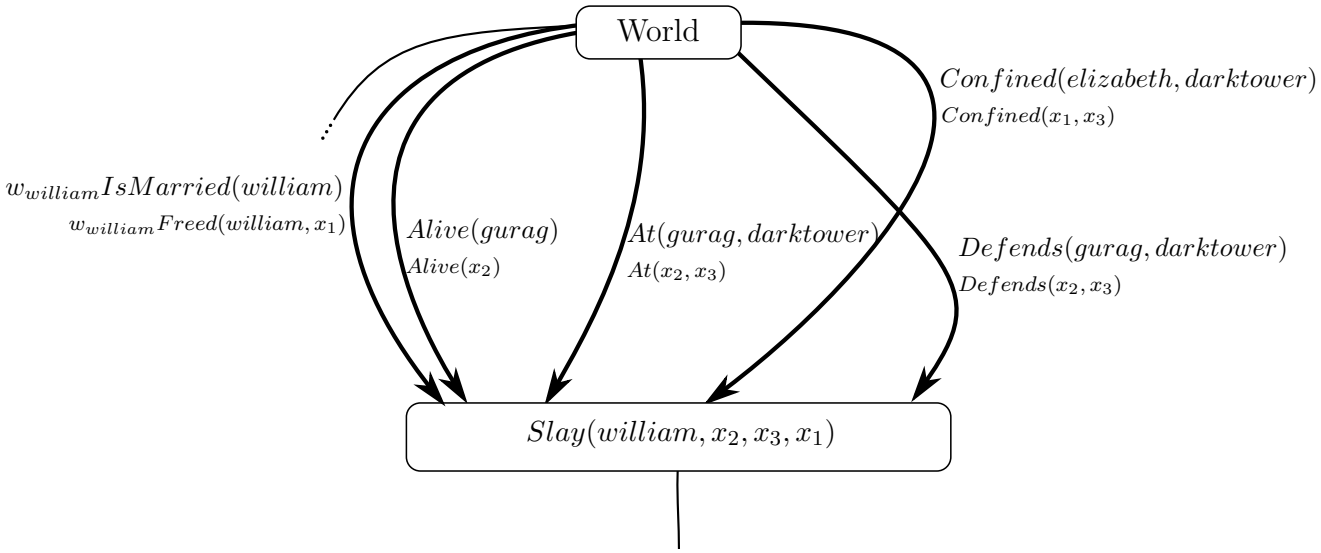
giustifica dal fatto che liberandola ella si innamorerà di lui, e l'innamoramento è una giustificazione al voler essere sposati. Ma *William* desidera essere sposato (mondo iniziale), e dunque vuole liberarla affinché si innamori di lui e lo voglia sposare. In altre parole

$$Freed(william, x_1)$$

è una precondizione di una catena di eventi che ha come prodotto finale qualcosa che lui desidera, e cioè

$$w_{william} IsMarried(william).$$

Il risultato è il seguente (una parte di grafo è stata omessa per leggibilità).

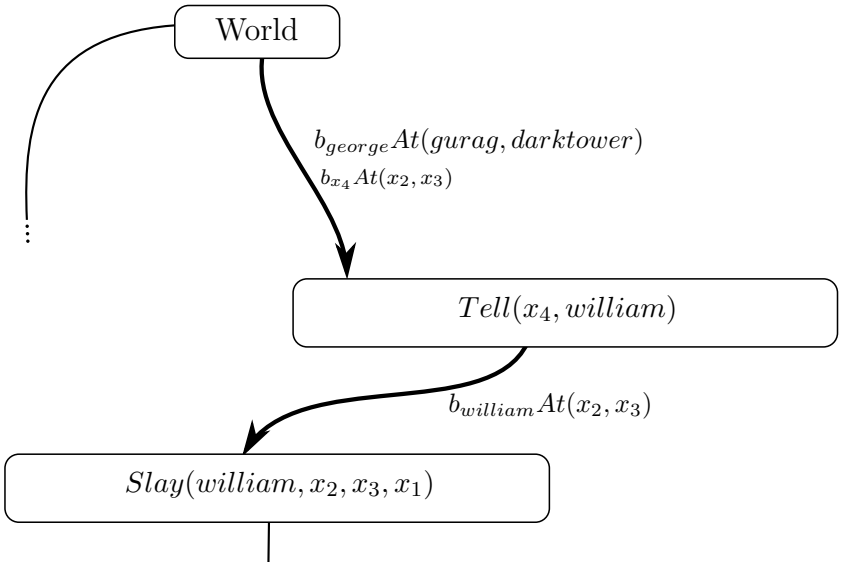


Rimane la precondizione di *Slay* che il cavaliere sappia dove sia il mostro, e cioè

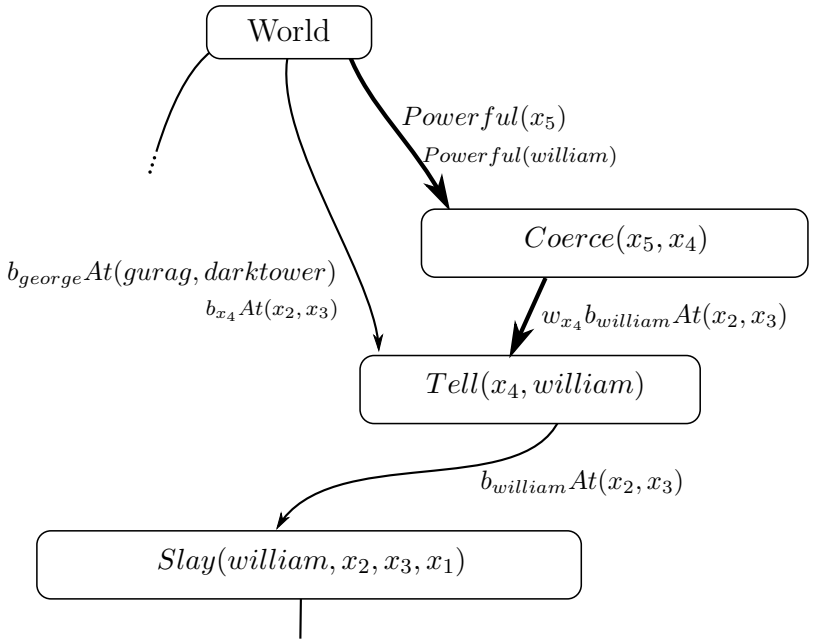
$$b_{william} At(x_2, x_3).$$

Questa precondizione è risolta dall'evento $Tell$ tra $william$ e $x_4 = george$, che è l'unico a sapere dove sia il mostro

$$b_{george} At(gurag, darktower).$$



Infine il fatto che *George* desideri che *William* sappia dove sia il drago, viene risolto dall'evento *Coerce*, che vede *William* costringere *George* a desiderare che il primo sappia la posizione del drago. Risolvendo anche la precondizione banale $Powerful(william)$ la fabula diventa la seguente.



Infine l'ultima precondizione di *Coerce*

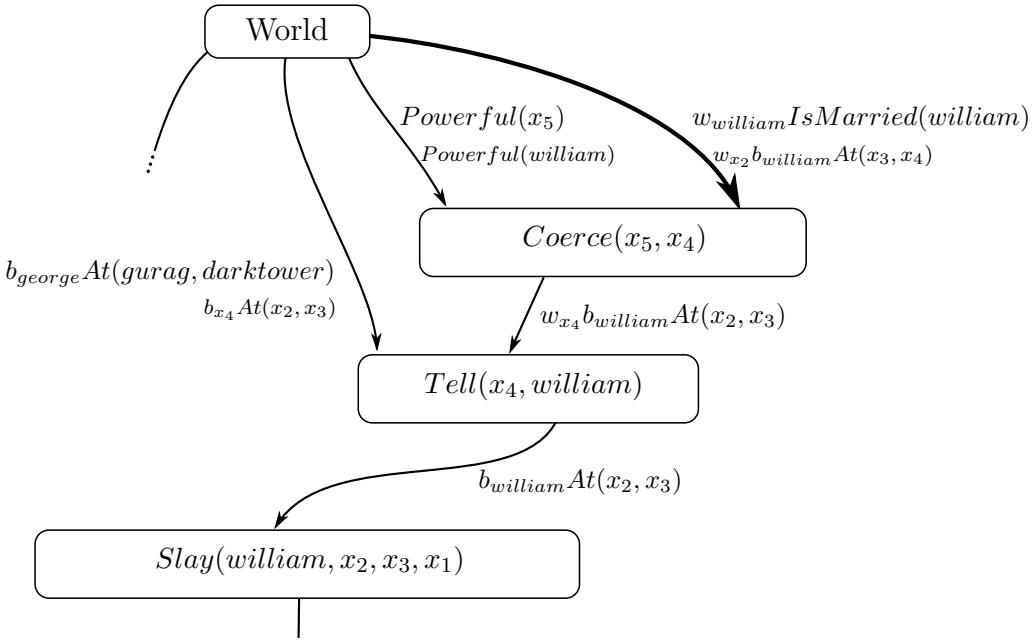
$$w_{x_5=william}b_{william}At(gurag, darktower)$$

è risolta nuovamente per giustificazione delle intenzioni, poiché $b_{william}At(gurag, darktower)$ è la precondizione dell'evento *Slay* che ha come effetto

$$Freed(william, x_2)$$

che porta x_2 a innamorarsi di *William* per poi sposarsi. Quindi il fatto che *William* voglia sposarsi è sufficiente per giustificare

$$w_{x_5=william} b_{william} At(gurag, darktower).$$



Il grafo completo della fabula finale, dopo le sostituzioni di variabili

$$x_2 = elizabeth, x_3 = gurag, x_4 = darktower, x_5 = william$$

è mostrato in figura 4.1.

Eseguendo la risoluzione del problema in modalità di risoluzione dei conflitti numero 2 (tradeoff tra efficienza e completezza), e abilitando la giustificazione delle intenzioni implicite, la fabula viene trovata dopo 585 ms, esplorando 77 piani parziali, con 96 piani parziali nella lista aperta, 74 in quella chiusa.

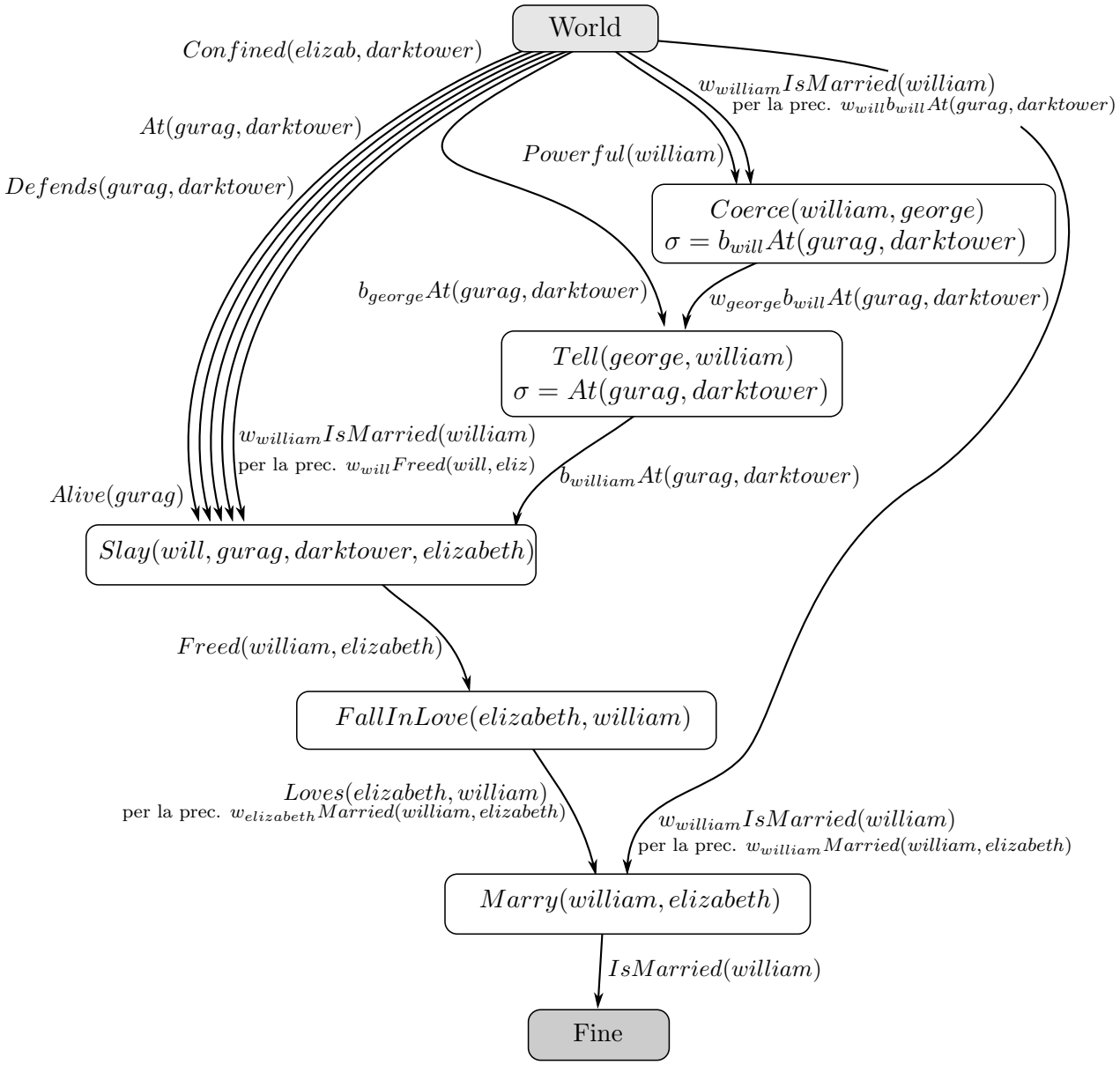


Figura 4.1: *Esempio completo di fabula generata con* FABUPLAN.

Conclusioni

L'obiettivo del lavoro presentato in questo documento è quello di studiare un approccio alla generazione di fabule, che, combinate con un intreccio, definiranno una storia. La generazione della fabula, ovvero la sequenza degli eventi di una storia espressa in ordine cronologico, è stata oggetto di svariati sistemi, classificabili in quelli basati su una *simulazione* e quelli *deliberativi*. I primi tendono a produrre narrative con una buona credibilità dei personaggi ma difficilmente producono narrative con una trama globale interessante. Il motivo risiede nel fatto che gli approcci simulativi modellano direttamente i personaggi con agenti autonomi, i quali determinano le azioni del personaggio associato di situazione in situazione in base alle informazioni locali possedute. Per la località della ricerca, gli approcci basati su simulazione sono soggetti a massimi locali. Al contrario, gli approcci deliberativi non soffrono di questa limitazione, in quanto ragionano direttamente sull'interezza della narrativa e generano quindi trame con una struttura globale più coerente. In generale, tuttavia, questi sistemi non considerano l'intenzionalità dei personaggi, poiché generano sequenze di eventi che, anche presentando una struttura globale coerente, possono risultare non credibili agli occhi dell'audience quando questa non è in grado di comprendere le intenzioni a partire dalle azioni dei personaggi o viceversa prevedere le prossime azioni conoscendone le intenzioni.

Un sistema deliberativo che risolve questo problema è FABULIST, un planner Partial Order Causal Link che modella e pianifica l'intenzionalità dei personaggi arricchendo direttamente l'algoritmo di planning. L'esigenza della modifica nasce dal fatto che il planning è convenzionalmente adottato per generare il piano

d'azione di un singolo agente o di più agenti cooperanti. Nelle storie, gli agenti – i personaggi – non sono né necessariamente cooperanti e né necessariamente antagonisti. Ciascuno è animato da desideri e obiettivi potenzialmente distinti da quelli finali della storia.

FabuPlan, oggetto del presente documento, è la soluzione proposta alla generazione deliberativa di fabule anch'esso basato su planning POCL. Per garantire la credibilità delle intenzioni dei personaggi da parte di un'ipotetica audience, FabuPlan risolve il problema primariamente estendendo il linguaggio logico di definizione della storia. Mentre Fabulist mantiene invariato il linguaggio STRIPS, un linguaggio basato sui predicati del prim'ordine, FabuPlan aggiunge a STRIPS la multimodalità, in modo da supportare il modello *Belief-Desire-Intention*. Con quest'aggiunta, non soltanto è possibile modellare i desideri dei personaggi, ma anche la *credenza* di ciascuno di essi circa lo stato del mondo, in un arbitrario livello di profondità. Ad esempio sono immediatamente modellabili intrecci di credenze e desideri come "un personaggio crede che un altro creda che..." oppure "un personaggio desidera che un altro personaggio creda che...", e così via. È stato comunque necessario modificare l'algoritmo di planning POCL classico, come in Fabulist, per poter individuare desideri che da soli giustificano o implicano altri desideri, in un sistema chiamato *giustificazione delle intenzioni implicite*: senza tale meccanismo, il planner non potrebbe da solo costruire l'intenzionalità che muove un personaggio attraverso vari eventi, a partire dai suoi desideri.

Per la complessità del nuovo linguaggio, si è scelto di non partire dal sorgente di un planner POCL completo come UCPOP scritto in Common Lisp, ma di costruirne uno ex-novo in C++. La scrittura di FabuPlan dal principio ha consentito di acquisire padronanza con i non ovvi dettagli implementativi di un planner, fondamentale per comprendere come realizzare le profonde modifiche necessarie per supportare le funzionalità aggiuntive richieste. Questa scelta ha, inoltre, permesso di sperimentare il supporto agli assiomi definibili dall'utente

da parte di un planner ad ordinamento parziale. Questo task è non banale in questo tipo di planning, poiché non è mai esplicito lo stato totale del mondo in un determinato istante di tempo, come invece accade per i pianificatori in avanti ad ordinamento totale. La soluzione adottata consiste nell'utilizzare gli assiomi per trovare gli effetti che implicitamente risolvono una precondizione aperta e per identificare i fatti mutualmente esclusivi ai fatti protetti dai collegamenti causali. Questi due notevoli vantaggi sono realizzabili solo se gli assomi, chiamati *regole*, presentano la forma di una implicazione tra una disgiunzione e una congiunzione di fatti. Abbiamo visto che l'espressività del linguaggio di definizione del dominio aumenta sensibilmente, riducendo il lavoro del progettista e del planner.

Il piano generato da FabuPlan è corretto, coerente e gestisce l'intenzionalità e la conoscenza dei personaggi. Il vero limite all'uso di FabuPlan risiede nella sua complessità computazionale. Questo è particolarmente vero se si vuole costruire un sistema di generazione di narrativa interattivo, che dinamicamente adatti la narrazione in base all'input dell'utente. Un'implementazione più attenta, che faccia uso di euristiche più informate[12], che individui ulteriori compromessi tra completezza ed efficienza è sicuramente fondamentale e può essere oggetto di lavori futuri. Inoltre si potrebbe raffinare la ricerca della fabula dotando il planner di funzioni online di valutazione di una fabula parziale da un punto di vista narrativo, in modo che si possa potare l'albero di ricerca quando una questa non soddisfa i requisiti stabiliti dall'autore. Lo studio e l'implementazione di queste funzioni di valutazioni rappresentano un altra possibile direzione di sviluppo.

Un sistema in grado di generare storie può essere usato per adattare la narrativa alle preferenze dell'utente e migliorare sensibilmente immersività e rigiocabilità nei videogiochi, dando luogo a situazioni di gioco non previste inizialmente dai progettisti. Più in generale, la generazione di narrativa muove verso una più naturale interazione uomo-macchina, con il risultato di generare sistemi computazionali con migliori capacità di comunicazione, intrattenimento ed educazione.

Bibliografia

[1] Entertainment Software Association. *Essential Facts About the Computer and Video Game Industry*. 2011. `http://www.theesa.com/facts/pdfs/ESA_EF_2011.pdf`.

[2] M. Bal. *Narratology: Introduction to the Theory of Narrative*. G - Reference, Information and Interdisciplinary Subjects Series. University of Toronto Press, 1997.

[3] J. Bates. The nature of character in interactive worlds and the oz project. Technical report, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1992.

[4] J. Bates, A. B. Loyall, and W. S. Reilly. Broad agents. *Proceedings AAAI Spring Symposium on Integrated Intelligent Architectures*, 1991.

[5] J. Bates, A. B. Loyall, and W. S. Reilly. An architecture for action, emotion, and social behavior. *Proceedings of the Fourth European Workshop on Modeling Autonomous Agents in a Multi-Agent World*, 1992.

[6] M. Cavazza, F. Charles, and S. Mead. Planning characters' behaviour in interactive storytelling. *Journal of Visualization and Computer Animation, 13*, pages 121–131, 2002.

[7] P. R. Cohen and C. R. Perrault. Elements of a plan-based theory of speech acts. *Cognitive Science 3*, pages 177–212, 1979.

[8] J. Hoffmann. Ff: The fast-forward planning system. *AI magazine, Volume 22*, pages 57–62, 2001.

[9] M. Mateas. An oz-centric review of interactive drama and believable agents. Technical report, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1997.

[10] M. Mateas and P. Sengers. *Narrative Intelligence*. Advances in Consciousness Research. J. Benjamins Pub., 2003.

[11] J. R. Meehan. Tale-spin: An interactive program that writes stories. *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, pages 91–98, 1977.

[12] X. Nguyen and S. Kambhampati. Reviving partial order planning. pages 459–464, 2001.

[13] S. J. Penberthy and D. S. Weld. Ucpop: A sound, complete, partial order planner for ADL. In Bernhard Nebel, Charles Rich, and William Swartout, editors, *KR'92. Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference*, pages 103–114. Morgan Kaufmann, San Mateo, California, 1992.

[14] J. Porteous and M. Cavazza. Controlling narrative generation with planning trajectories: the role of constraints. *Proceedings of the 2nd International Conference on Interactive Digital Storytelling*, 2009.

[15] G. Prince. *A Dictionary of Narratology*. University of Nebraska Press, 2003.

[16] M. O. Riedl and R. M. Young. Narrative planning: Balancing plot and character. *Journal of Artificial Intelligence Reserach 39*, pages 217–267, 2010.