Concorrenza e parallelismo

Vogliamo studiare linguaggi (e relativa semantica) per programmare l'esecuzione parallela di processi (agenti/demoni…).

Lo scenario comprende quindi una comunità di attori (unità di esecuzione) che manipolano delle risorse comuni agendo simultaneamente in maniera per quanto possibile indipendente.

Questo porta dei vantaggi in termini di rapidità di esecuzione e semplicità di programmazione del singolo attore, che può essere programmato per svolgere una singola attività specifica fra le tante che porteranno al risultato finale (corrispondente in un paradigma sequenziale ad un modulo).

D'altra parte progettare un attore in modo da funzionare indipendentemente dalle alterazioni che gli altri attori possono introdurre nell'ambiente presenta notevoli complicazioni.

I linguaggi che studieremo sono minimali nel senso che presentano il minimo indispensabile di costrutti per capire quali sono i problemi (e le soluzioni) dovuti al parallelismo; non sono certo linguaggi realistici per la programmazione di agenti/processi.

Esempio non informatico

Consideriamo un caso di parallelismo dal “mondo reale”: una coppia di sposi conduce le sue attività giornaliere.

Lei pianifica di riordinare la casa, andare a fare shopping oppure in palestra, passare a prendere due pizze e cenare a casa col marito.

Lui pianifica di lavorare mezza giornata, incontrare la sua amante, portarla a casa (approfittando dell’assenza della moglie), uscire a comprare delle bevande e tornare a cenare con la moglie.

In questo esempio si vedono in concreto alcuni concetti tipici della concorrenza:

sincronizzazione: marito e moglie devono essere a cena assieme

azioni asincrone: marito e moglie possono svolgere gran parte delle attività che hanno pianificato in maniera del tutto indipendente

risorse condivise: la casa viene usata da entrambi gli attori in competizione fra loro

cooperazione: la cena sarà pronta per la cooperazione fra moglie (che acquista le pizze) e marito (che acquista da bere)

non determinismo: la moglie non ha ancora deciso se fare shopping o palestra e a seconda della sua scelta alla fine della giornata il bilancio familiare sarà diverso.

Inoltre, a seconda dei tempi effettivi di esecuzione delle attività rispettive, l’andamento della giornata può essere ben diverso, perché se il riordino della casa si protrae …

Quest’ultimo punto fa capire che la semantica del sistema (coppia) non può essere definita in termini della semantica delle parti se quest’ultima si limita (come nel caso sequenziale) ad osservare i cambiamenti fra stato iniziale e finale

Parallelismo in informatica

Tutti i sistemi attuali hanno un certo grado di parallelismo (multitasking), quanto meno fra i vari pezzi del sistema operativo.
Il parallelismo può essere "virtuale" in caso di sistemi monoprocessore (interleaving) o effettivo per i sistemi multiprocessore o distribuiti.
A livello di utente, però, il parallelismo è spesso nascosto: ciscun task (applicazione, finestra…) lavora su una sua macchina virtuale e le interazioni fra queste macchine sono gestite dal sistema operativo.
Noi vogliamo studiare proprio questi meccanismi di gestione di cooperazione e competizione.
Ci sono due famiglie di modelli per la concorrenza distinte dal metodo usato per permettere interazione fra le componenti:
Memoria condivisa (shared memory): i vari processi usano le stesse "locazioni", per cui le modifiche fatte da uno possono essere usate da altri (il cui comportamento, quindi risulta influenzato…)
Scambio di messaggi (message passing): ciascun processo vive in splendido isolamento e comunica con (influenza) gli altri mandando e ricevendo messaggi, che possono contenere valori, sottoprocessi…
A seconda di quale sia il modello che si ha in mente, servono differenti primitive per la concorrenza (in un caso per modificare la memoria, nell'altro per scambiare messaggi).
Quindi vedremo due linguaggi uno adatto al modello a memoria condivisa (ParWhile) e l'altro per il modello con scambio di messaggi (CCS).