Relazione per l'approfondimento su Computational Tree Logic

Filippo Mameli, Federico Schipani

22 gennaio 2017

Indice

1 Introduzione a CTL

Computational Tree Logic (CTL) è una logica proposta da Clarke e Emerson per far fronte ad alcuni problemi noti di Linear Temporal Logic (LTL). In LTL il concetto di tempo è lineare, ciò vuol dire che in un determinato momento abbiamo un unico possibile futuro. Ciò comporta che una determinata formula ϕ è valida in uno stato s, se e solo se tutte le possibili computazioni che partono da quello stato soddisfano la formula. Più formalmente:

$$s \vDash \phi \iff \pi \vDash \phi \ \forall \ \mathtt{paths}\ \pi\ \mathtt{che}\ \mathtt{inizia}\ \mathtt{in}\ s \tag{1.0.1}$$

Come si può notare dalla Formula (1.0.1) non è possibile imporre facilmente condizioni di soddisfacibilità solo su alcuni di questi path. Dato uno stato s, verificare che solo alcune computazioni soddisfano una formula ϕ può essere fatto usando la dualità tra l'operatore universale ed esistenziale. Quindi verificare $s \vDash \exists\phi$ corrisponde a verificare $s \vDash \forall\neg\phi$. Se quest'ultima non è soddisfatta allora esisterà una computazione che soddisferà ϕ, altrimenti non esisterà.

Questo sotterfugio non è possibile usarlo per proprietà più complicate. Per esempio la proprietà

Proprietà 1. *Per ogni computazione è sempre possibile ritornare in uno stato iniziale*

non è possibile esprimerla in LTL. Un tentativo potrebbe essere $\Box\Diamond start$, dove $start$ indica uno stato iniziale. Tuttavia una formula di questo tipo è troppo forte, in quanto questa formula impone che una computazione ritorni sempre in uno stato iniziale, e non soltanto eventualmente.

CTL risolve questi problemi introducendo una nozione di tempo che si basa sulle diramazioni. Quindi non abbiamo più un infinita sequenza di stati, ma un infinito albero di stati. Questo comporta che in un determinato momento avremo diversi possibili futuri.

La semantica di questa logica è definita in termini di infiniti alberi, dove ogni diramazione rappresenta un singolo percorso. L'albero quindi è una fedele rappresentazione di tutti i possibili path, e si può facilmente ottenere schiudendo il Transition System (TS).

In CTL sono presenti quantificatori, definiti sui path, di tipo esistenziale ($\exists$) ed universale ($\forall$). La Proprietà $\exists\Diamond\psi$ dice che esiste una computazione che soddisfa $\Diamond\psi$, più intuitivamente vuol dire che esisterà almeno una possibile computazione nel quale uno stato s che soddisfa ψ verrà eventualmente raggiunto. Tuttavia questo non esclude la possibilità che ci possono essere computazioni per le quali questa proprietà non viene soddisfatta. La proprietà 1 citata in precedenza è possibile ottenerla annidando quantificatori esistenziali ed universali in questo modo:

$$\forall \Box \exists\Diamond start \tag{1.0.2}$$

La Formula (1.0.2) si legge come: in ogni stato ($\Box$) di ogni possibile computazione ($\forall$), è possibile ($\exists$) eventualmente ritornare in uno stato iniziale ($\Diamond start$).

1.1 Sintassi di CTL

CTL ha una sintassi a due livelli, dove le formule sono classificate in *formule sugli stati* e *formule sui path*.

$$\boldsymbol{\phi} ::= \text{true} \mid a \mid \boldsymbol{\phi}_1 \wedge \boldsymbol{\phi}_2 \mid \neg\boldsymbol{\phi} \mid \exists\varphi \mid \forall\varphi$$

Grammatica 1: Grammatica per le formule sugli stati

Le prime sono definite dalla Grammatica 1 dove $a \in AP$ e φ è una formula sui path.

$$\varphi ::= \bigcirc \boldsymbol{\phi} \mid \boldsymbol{\phi}_1 \,\mathbf{U}\, \boldsymbol{\phi}_2$$

Grammatica 2: Grammatica per le formule sui path

Le *formule sui path* sono invece definite dalla Grammatica 2. Intuitivamente si può dire che le formule sugli stati esprimono una proprietà su uno stato,

mentre le formule sui path esprimono proprietà sui infinite sequenze di stati. Per esempio la formula $\bigcirc \phi$ è vera per un path se lo stato successivo, in quel path, soddisfa ϕ. Una formula sugli stati può essere trasformata in una formula sui path aggiungendo all'inizio un quantificatore esistenziale ($\exists$) o universale ($\forall$). Per esempio la formula $\exists\varphi$ è valida in uno stato se esiste almeno un percorso che soddisfa φ.

1.2 Semantica di CTL

Le formule CTL sono interpretate sia sugli stati che sui path di un TS. Formalmente, dato un TS, la semantica di una formula è definita da due relazioni di soddisfazione: una per le formule di stato ed una per le formule di path. Per le formule di stato è un tipo di relazione tra gli stati del TS e la formula di stato. Si scrive che $s \vDash \boldsymbol{\phi}$ se e solo se la formula di stato $\boldsymbol{\phi}$ è vera nello stato s.

Per le formule di path la relazione $\vDash$ è una relazione definita tra un frammento di path massimale nel TS e una formula di path. Si scrive che $\pi \vDash \varphi$ se e solo se il path π soddisfa la formula φ.

Definizione 1. *Sia* $a \in AP$ *una proposizione atomica,* $TS = (S, act, \rightarrow, I, AP, L)$ *un Transition System (TS) senza stati terminali, stati* $s \in S$, $\boldsymbol{\phi}$, $\boldsymbol{\Psi}$ *formule CTL di stato e* φ *una formula CTL di path. La relazione di soddisfazione* $\vDash$ *per le formule di stato è definita come:*

$$s \vDash a \iff a \in L(s)$$

$$s \vDash \neg\boldsymbol{\phi} \iff \text{not } s \vDash \boldsymbol{\phi}$$

$$s \vDash \boldsymbol{\phi} \wedge \boldsymbol{\Psi} \iff (s \vDash \boldsymbol{\phi}) \text{ e } (s \vDash \boldsymbol{\Psi})$$

$$s \vDash \exists\varphi \iff \pi \vDash \varphi \text{ per alcuni } \pi \in Paths(s)$$

$$s \vDash \forall\varphi \iff \pi \vDash \text{ per tutti i } \pi \in Paths(s)$$

Per un path π*, la relazione di soddisfazione* $\vDash$ *per le formule di path è definita da:*

$$\pi \vDash \bigcirc \boldsymbol{\phi} \iff \pi[1] \vDash \boldsymbol{\phi}$$

$$\pi \vDash \boldsymbol{\phi}\ \mathbf{U}\ \boldsymbol{\Psi} \iff \exists j \geqslant 0.(\pi[j] \vDash \boldsymbol{\Psi} \wedge (\forall 0 \leqslant k < j.\pi[k] \vDash \boldsymbol{\phi}))$$

2 Model Checking di CTL

Data una formula CTL ϕ ed un TS l'obiettivo dell'algoritmo di Model Checking è quello di dire se il TS soddisfa o meno la formula. Gli algoritmi proposti lavorano su formule in Existential Normal Form (ENF), definite dalla Grammatica 3

$$\phi ::= \text{true} \mid a \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \exists \bigcirc \phi \mid \exists \Box \phi \mid \exists(\phi_1 \mathbf{U} \phi_2)$$

Grammatica 3: Grammatica delle formule in ENF

il che non è limitate in quanto il Teorema 2.1 dimostra che per ogni formula CTL esiste la corrispondente formula in ENF.

Teorema 2.1. *Per ogni formula CTL esiste un equivalente formula CTL in ENF*

Dimostrazione. Grazie alle leggi di dualità si ottengono delle regole di traduzione:

$$\forall \bigcirc \phi \equiv \neg \exists \bigcirc \neg \phi$$

$$\forall (\phi \mathbf{U} \Psi) \equiv \neg \exists (\Psi \mathbf{U} (\neg \phi \wedge \Psi)) \wedge \neg \exists \neq \Psi$$

□

L'algoritmo base, mostrato in Algoritmo **??**, risolve ricorsivamente il problema di verificare se un determinato TS soddisfa una formula ϕ. Fondamentalmente il calcolo consiste in un attraversamento dalle foglie alla radice dell'albero di parsing della formula sugli stati ϕ. In questo albero i nodi rappresentano le sottoformule Ψ di ϕ, mentre le foglie rappresentano le proposizioni atomiche $a \in AP$ e la costante $true$. Durante la computazione vengono calcolati ricorsivamente gli insiemi $Sat\Psi$ per ogni sottoformula Ψ di ϕ. In un generico passo intermedio il risultato della valutazione di un nodo figlio è usato e combinato in una maniera appropriata per stabilire il set di stati che soddisfa il nodo padre, chiamato v. Il tipo di computazione quando si raggiunge il nodo v dipende dal tipo di operatore che contiene, che può essere $\wedge$, $\exists \bigcirc$ oppure $\exists \mathbf{U}$.

Il seguente teorema definisce come vengono generati gli insiemi di sottoformule.

Teorema 2.2. *Sia* $TS = (S, Act, \rightarrow, I, AP, L)$ *un TS senza stati terminali. Per tutte le formule CTL* ϕ, Ψ *su AP è vero che:*

$$Sat(true) = S \tag{2.2.1}$$

$$Sat(a) = \{s \in S | a \in L(s)\} \tag{2.2.2}$$

$$Sat(\phi \wedge \Psi) = Sat(\phi) \cap Sat(\Psi) \tag{2.2.3}$$

$$Sat(\neg \phi) = S \backslash Sat(\phi) \tag{2.2.4}$$

$$Sat(\exists \bigcirc \phi) = \{s \in S | Post(s) \cap Sat(\phi) \neq \varnothing\} \tag{2.2.5}$$

$$Sat(\exists(\phi \mathbf{U} \Psi)) \textit{ è il più piccolo sottoinsieme T di S tale per cui } (Sat(\Psi) \subseteq T \wedge (s \in Sat(\phi) \wedge Post(s) \cap T \neq \varnothing)) \implies s \in T \tag{2.2.6}$$

$$Sat(\exists(\Box \phi)) \textit{è il più grande sottoinsieme T di S tale che } T \subseteq Sat(\phi) \wedge s \in T \implies Post(s) \cap T \neq \varnothing \tag{2.2.7}$$

Dimostrazione. Da scrivere per (2.2.6) e (2.2.7). □

Algoritmo 1 Algoritmo di model checking base per CTL

```
1: procedure CTLModelChecking(TS, φ)
2:     for all i ≤ |φ| do
3:         for all Ψ ∈ Sub(φ) con |Ψ| = i do
4:             calcola Sat(Ψ) da Sat(Ψ′)
5:         end for
6:     end for
7:     return I ⊆ Sat(φ)
8: end procedure
```

Algoritmo 2 Algoritmo per $Sat(\exists(\phi \mathbf{U} \Psi))$

```
1: procedure ComputeExistsUntil(TS, φUΨ)
2:     E := Sat(Ψ)
3:     T :=E
4:     while E ≠ ∅ do
5:         let s¹ ∈ E
6:         E := E\{s¹}
7:         for all  s ∈ Pre(s¹) do
8:             if s ∈ Sat(φ)\T then
9:                 E := E ∪ {s}
10:                T := T ∪ {s}
11:            end if
12:        end for
13:    end while
14:    return T
15: end procedure
```

Algoritmo 3 Algoritmo per $Sat(\exists \Box \phi)$

```
1:  procedure ComputeExistsAlways(TS, (∃ □ φ))
2:      S := S\Sat(φ)
3:      T := Sat(φ)
4:      for all s ∈ Sat(φ) do
5:          count[s] := |Post(s)|
6:      end for
7:      while E ≠ ∅ do
8:          let s¹ ∈ E
9:          E := E\{s¹}
10:         for all  s ∈ Pre(s¹) do
11:             if s ∈ T then
12:                 count[s] := count[s] − 1
13:                 if count[s] = 0 then
14:                     T := T\{s}
15:                     E := E ∪ {s}
16:                 end if
17:             end if
18:         end for
19:     end while
20:     return T
21: end procedure
```

Algoritmo 4 Algoritmo alternativo per $Sat(\exists \Box \phi)$

```
1:  procedure AlternativeComputeExistsAlways(TS, (∃ □ φ))
2:      S¹ := Satφ
3:      →¹:=→ ∩(S¹ × Act × S¹)
4:      I¹ := I ∩ S¹
5:      L¹(s) := L(s)|s ∈ S¹
6:      TS[φ] := (S¹, Act, →¹, I¹, AP, L¹)
7:      T := ComputeSSC(TS[φ])
8:      return Reach(I, T)
9:  end procedure
```

2.1 Complessità dell'algoritmo

3 Model Checker in Python per CTL

4 Esempi di utilizzo del Model Checker di CTL in Python

CTL Computational Tree Logic

LTL Linear Temporal Logic

TS Transition System

ENF Existential Normal Form