Ragionamento Automatico
Model checking

Lezione 12

Sommario

♢ Capitolo 3 paragrafo 6 del libro di M. Huth e M. Ryan: *Logic in Computer Science: Modelling and reasoning about systems* (Second Edition) Cambridge University Press, 2004.

♢ Verifica/certificazione di sistemi dinamici

♢ Logiche Temporali

♢ Model Checking

Model checking

Vogliamo un algoritmo per verificare se $\mathcal{M}, s \models \phi$.

Nel caso in cui ϕ non è verificato l'algoritmo deve produrre **una traccia che non verifica** ϕ.

Quindi, per esempio, possiamo debuggare il sistema cercando di eliminare le cause per cui ϕ non è verificato.

Formulazione del problema

Consideriamo la logica CTL

♢ Possiamo provare a vedere se $\mathcal{M}, s_0 \models \phi$

♢ Possiamo partire da $\mathcal{M}$ e ϕ, calcolare tutti gli s che soddisfano ϕ. Poi vediamo se s_0 sta o no nell'insieme risultante.

Quest'ultimo problema è più facile da risolvere.

Algoritmo di etichettatura

INPUT: $\mathcal{M} = \langle \mathcal{S}, \rightarrow, L \rangle$ un modello per CTL,
ϕ una formula CTL.

OUTPUT: L'insieme di stati di $\mathcal{M}$ che soddisfano ϕ

Le formule vengono riscritte in termini di $\bot, \neg, \wedge, \mathbf{AF}, \mathbf{EX}$. Se ϕ contiene altri connettivi (proposizionali o modali) useremo al suo posto **TRANSLATE**(ϕ), risultato della trasformazione in termini dei suddetti connettivi.

Algoritmo di etichettatura (cont)

Supponiamo che ψ sia una sottoformula di ϕ e supponiamo che **tutti gli stati** in cui valgono tutte le **sottoformule immediate** di ψ siano stati etichettati.

Vediamo ora quali stati etichettare con ψ.

Se ψ è:

$\bot$:	nessuno stato è etichettato con $\bot$
p:	etichettiamo s con p sse $p \in L(s)$
$\psi_1 \wedge \psi_2$:	etichettiamo s con $\psi_1 \wedge \psi_2$ sse è già etichettato con ψ_1 e ψ_2
$\neg\psi_1$:	etichettiamo s con $\neg\psi_1$ sse non è già etichettato con ψ_1

continua

Algoritmo di etichettatura (cont)

$\mathbf{AF}\psi_1$:	– se uno stato è etichettato con ψ_1 lo etichettiamo con $\mathbf{AF}\psi_1$ – *Repeat* etichetta uno stato s con $\mathbf{AF}\psi_1$ sse tutti i suoi successori sono etichettati con $\mathbf{AF}\psi_1$ *until* non ci sono più cambiamenti
$\mathbf{E}[\psi_1 \mathbf{U} \psi_2]$:	– se uno stato è etichettato con ψ_2 lo etichettiamo con $\mathbf{E}[\psi_1 \mathbf{U} \psi_2]$ – *Repeat* etichetta uno stato s con $\mathbf{E}[\psi_1 \mathbf{U} \psi_2]$ sse è etichettato con ψ_1 e almeno uno dei suoi successori è etichettato con $\mathbf{E}[\psi_1 \mathbf{U} \psi_2]$ *until* non ci sono più cambiamenti
$\mathbf{EX}\psi_1$:	etichettiamo uno stato con $\mathbf{EX}\psi_1$ sse uno dei suoi successori è etichettato con ψ_1

Complessità

L'algoritmo di etichettatura si può dimostrare essere

$$O(f \times \mathcal{V} \times (\mathcal{V} + \mathcal{E}))$$

dove f è il numero dei connettivi nella formula, $\mathcal{V}$ è il numero di stati e $\mathcal{E}$ è il numero di transizioni di $\mathcal{M}$.

Quindi l'algoritmo è **lineare** nella dimensione della formula e **quadratico** nella dimensione del modello.

Algoritmo di etichettatura (cont)

Si può fare di meglio trattando **EG** direttamente.

EGψ_1: – etichettiamo TUTTI gli stati s con **EG**ψ_1
– Se uno stato non è etichettato con ψ_1 cancelliamo **EG**ψ_1
– *Repeat* cancelliamo **EG**ψ_1 da uno stato s
sse nessuno dei suoi successori è etichettato con **EG**ψ_1
until non ci sono più cambiamenti

Algoritmo di etichettatura più efficiente

Invece di usare **EX**, **EU** e **AF** usiamo **EX**, **EU** e **EG**
Per **EX**, **EU** si procede come prima
(backwards, breadth first per non passare sugli stati più volte)

Per **EG**ψ si fa come segue:
• Cancellare tutti gli stati che non soddisfano ψ e le transizioni che li riguardano
• Individuare tutte le **Componenti Fortemente Connesse*** Massimali.
• Usare backwards, breadth first per trovare tutti gli stati che possono raggiungere una di queste CFCM.

Questo algoritmo è $O(f \times (\mathcal{V} + \mathcal{E}))$,
lineare nella formula e nel modello

* C'è un cammino (finito) da tutti i nodi a tutti i nodi

ALGORITMO SAT

function $SAT(\phi)$
/* determina l'insieme di stati che soddisfano ϕ */
begin
case
ϕ è $\top$ **return** $\mathcal{S}$
ϕ è $\bot$ **return** $\{\}$
ϕ è atomica **return** $\{s \in \mathcal{S} \mid \phi \in L(s)\}$
ϕ è $\neg\phi_1$ **return** $\mathcal{S} - SAT(\phi_1)$
ϕ è $\phi_1 \wedge \phi_2$ **return** $SAT(\phi_1) \cap SAT(\phi_2)$
ϕ è $\phi_1 \vee \phi_2$ **return** $SAT(\phi_1) \cup SAT(\phi_2)$
ϕ è $\phi_1 \rightarrow \phi_2$ **return** $SAT(\neg\phi_1 \vee \phi_2)$

ALGORITMO SAT (cont)

ϕ è **AX**ϕ_1 **return** $SAT(\neg\mathbf{EX}\neg\phi_1)$
ϕ è **EX**ϕ_1 **return** $SAT_{EX}(\phi_1)$
ϕ è **A**$[\phi_1\mathbf{U}\phi_2]$ **return**
$SAT(\neg(\mathbf{E}[\neg\phi_2\mathbf{U}(\neg\phi_1 \wedge \neg\phi_2)] \vee \mathbf{EG}\neg\phi_2))$
ϕ è **E**$[\phi_1\mathbf{U}\phi_2]$ **return** $SAT_{EU}(\phi_1, \phi_2)$
ϕ è **EF**ϕ_1 **return** $SAT(\mathbf{E}[\top\mathbf{U}\phi_1])$
ϕ è **EG**ϕ_1 **return** $SAT(\neg\mathbf{AF}\neg\phi_1)$
ϕ è **AF**ϕ_1 **return** $SAT_{AF}(\phi_1)$
ϕ è **AG**ϕ_1 **return** $SAT(\neg\mathbf{EF}\neg\phi_1)$
end case
end function

Funzioni ausiliarie

$\text{pre}_\exists(Y) = \{s \in \mathcal{S} \mid \text{esiste } s', (s \to s' \text{ e } s' \in Y\}$
$\text{pre}_\forall(Y) = \{s \in \mathcal{S} \mid \text{per ogni } s', (s \to s' \text{ implica } s' \in Y)\}$

$\text{pre}_\exists(Y)$ prende un sottinsieme Y di stati e ritorna l'insieme di stati che **possono** fare una transizione in Y.

$\text{pre}_\forall(Y)$ prende un sottinsieme Y di stati e ritorna l'insieme di stati che fanno transizioni **solo** in Y.

SAT_{EX}

function $SAT_{EX}(\phi)$
/* determina l'insieme di stati che soddisfano **EX**ϕ */
local var X, Y
begin
    $X := SAT(\phi)$;
    $Y := \text{pre}_\exists(X)$;
    return Y
end

SAT_{AF}

function $SAT_{AF}(\phi)$
/* determina l'insieme di stati che soddisfano **AF**ϕ */
local var X, Y
begin
    $X := \mathcal{S}$;
    $Y := SAT(\phi)$;
    repeat until $X = Y$
    begin
        $X := Y$;
        $Y := Y \cup \text{pre}_\forall(Y)$
    end
    return Y
end

SAT_{EU}

function $SAT_{EU}(\phi, \psi)$
/* determina l'insieme di stati che soddisfano **E**[ϕ**U**ψ] */
local var W, X, Y
begin
    $W := SAT(\phi)$;
    $X := \mathcal{S}$;
    $Y := SAT(\psi)$;
    repeat until $X = Y$
    begin
        $X := Y$;
        $Y := Y \cup (W \cap \text{pre}_\exists(Y))$
    end
    return Y
end

SAT_{EG}

function $SAT_{EG}(\phi)$
/* determina l'insieme di stati che soddisfano **EG**ϕ */
local var X, Y
begin
$Y := SAT(\phi)$;
$X := \{\}$;
repeat until $X = Y$
begin
$X := Y$;
$Y := Y \cap \mathsf{pre}_\exists(Y)$
end
return Y
end

Proprietà di SAT

Questo algoritmo si dimostra essere corretto e terminante (chi è interessato può vedere il paragrafo 3.7)

CTL con fairness

In LTL la fairness si esprime a livello di formule
(e.g. **FG**$\neg c \rightarrow \phi$, i.e. tutte le tracce che soddisfano infinitamente spesso $\neg c$ soddisfano anche ϕ)

In CTL non si può esprimere.

SMV permette di esprimere dei **vincoli di fairness** semplici.

- vincoli semplici: ϕ *è vero infinite volte*
- vincoli "complessi": *Se* ϕ *è vero infinite volte allora* ψ *è vero infinite volte*

CTL con fairness (cont)

Si considerano solo le tracce che soddisfano i vincoli di fairness.

Sia $\mathcal{C} = \{\psi_1, \psi_2, \cdots, \psi_n\}$ un insieme di vincoli di fairness. Una traccia soddisfa $\mathcal{C}$ sse per ogni $i = 1, \cdots, n$ esistono infiniti j t.c. $s_j \models \psi_i$

CTL con fairness (cont)

Introduciamo $\mathbf{A}_{\mathcal{C}}$ ed $\mathbf{E}_{\mathcal{C}}$ per denotare **A** ed **E** ristretta ai cammini fair rispetto a $\mathcal{C}$.

Si può provare che $\mathbf{E}_{\mathcal{C}}\mathbf{U}, \mathbf{E}_{\mathcal{C}}\mathbf{X}$ e $\mathbf{E}_{\mathcal{C}}\mathbf{G}$ sono un insieme adeguato di connettivi.

Ma si può anche provare che

$$\begin{aligned} \mathbf{E}_{\mathcal{C}}[\phi \mathbf{U} \psi] &\equiv \mathbf{E}[\phi \mathbf{U} (\psi \wedge \mathbf{U}\mathbf{E}_{\mathcal{C}}\mathbf{G}\top)] \\ \mathbf{E}_{\mathcal{C}}\mathbf{X}\phi &\equiv \mathbf{E}\mathbf{X}(\phi \wedge \mathbf{E}_{\mathcal{C}}\mathbf{G}\top) \end{aligned}$$

Quindi basta dare un algoritmo per $\mathbf{E}_{\mathcal{C}}\mathbf{G}\phi$.

Algoritmo per $\mathbf{E}_{\mathcal{C}}\mathbf{G}\phi$

• restringere il grafo agli stati che soddisfano ϕ; del grafo risultante vogliamo sapere da quali stati parte una traccia fair

• trovare le CFCM del grafo risultante

• rimuovere una CFCM sse per qualche vincolo ψ_i non contiene uno stato che soddisfa ψ_i; le CFCM restanti sono fair rispetto a $\mathcal{C}$. Ogni nodo del grafo restante che può raggiungere una delle CFCM ha una traccia fair che parte da esso.

Complessità

$$O(n \times f \times (\mathcal{V} + \mathcal{E}))$$

cioè è lineare nel numero di vincoli, nella formula e nel modello

LTL

Costruzione più "articolata":

Un automa rappresenta il modello
Si costruisce un automa $A_{\neg\phi}$ che rappresenta le stringhe accettate da $\neg\phi$

Si combinano i due.

Se nel sistema risultante c'è una traccia da s, allora s non soddisfa ϕ. (Noi ci fermiamo qui)

Tecniche di implementazione

- OBDD (capitolo 6 del libro)