Bounded model-checking

Approccio simbolico

- approccio basato sui BDD consente di analizzare sistemi con un numero elevato di stati
- BDD forniscono una rappresentazione compatta degli insiemi di stati eliminando ridondanze
- questa tecnica è sensibile alla determinazione di un ordinamento delle variabili ottimale
- determinare un buon ordinamento delle variabili è un problema NP-completo e non sempre l'ordinamento ottimale permette di evitare il problema dello state explosion

Approccio simbolico alternativo

- Bounded model-checking (BMC)
  - Idea: ridurre il model-checking a verificare la soddisfacibilità di una formula proposizionale (SAT)
  - si costruisce una formula booleana β_k che è soddisfacibile se e solo se esiste un cammino di lunghezza k che certifica soddisfacimento di φ (contro-esempio di lunghezza k)
- prestazioni dipendono dal SAT-solver sottostante
  - la ricerca sui SAT-solvers è molto attiva e le prestazioni dei tool realizzati hanno subito negli ultimi anni notevoli miglioramenti
- BMC è molto rapido nell'individuare controesempi con k piccolo

Formula per catturare esecuzioni bounded

- M: struttura di Kripke
- f: formula LTL
- *k: intero positivo (bound)*

- formula che cattura la relazione di transizione:

$$[[\, M \,]]_{\boldsymbol{k}} := I\,(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1})$$

dove I (s) è vero sse s iniziale e

T(s,s') è vero sse s' è un successore di s in M

ovvero, (s,a,s') è una transizione di M

- $[[\, M \,]]_{\boldsymbol{k}}$ viene messa in congiunzione con una formula che cattura semantica di f su M con bound k

Esempio

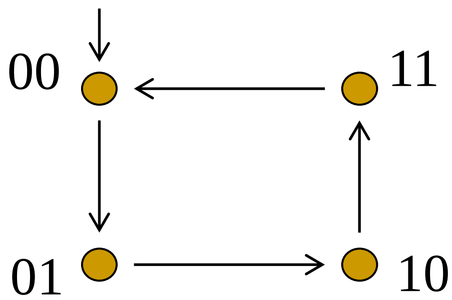
- supponiamo di voler verificare ◊p in due passi (k=2)

  la formula è:

$[[\, M, f\,]]_2 := I(s_0) \wedge T(s_0, s_1) \wedge T\,(s_1,s_2) \wedge (\, p\,(s_0) \vee\, p\,(s_1) \vee p\,(s_2)\,)$

  dove p(s) è vero sse la proposizione p vale nello stato s

- predicati del modello nella formula contatore a due bit (ρ è usato per T, e I_0 per I):



$I_0 \colon \neg l_0 \wedge \neg r_0$

$\rho(s_i\,, s_{i+1})\colon\; l_{i+1} = l_i \otimes r_i \,\wedge\, r_{i+1} = \neg r_i$

Specifica: safety property

$f = \square(\neg l \vee \neg r)$.

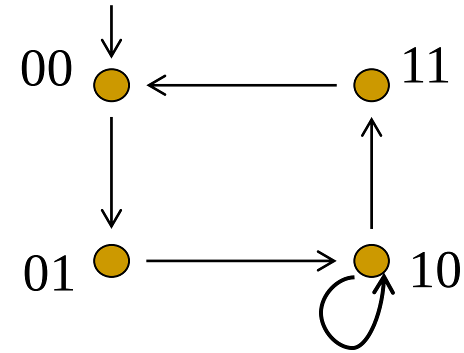
$k = 2$

Contro-esempio: uno stato raggiungibile entro k passi che soddisfa $(l \wedge r)$

- $I(s_0) : (\neg l_0 \wedge \neg r_0) \wedge$
- $T(s_0, s_1) : ((l_1 \boxed{\leftrightarrow} l_0 \otimes r_0) \wedge (r_1 \boxed{\leftrightarrow} \neg r_0)) \wedge$
- $T(s_1, s_2) : ((l_2 \boxed{\leftrightarrow} l_1 \otimes r_1) \wedge (r_2 \boxed{\leftrightarrow} \neg r_1)) \wedge$
- $p(s_0) : ((l_0 \wedge r_0) \vee$
- $p(s_1) : (l_1 \wedge r_1) \vee$
- $p(s_2) : (l_2 \wedge r_2))$

è facile verificare che $[[M, f]]_2$ non è soddisfacibile mentre $[[M, f]]_3$ lo è

Specifica: liveness property



f = ◊ $(l \wedge r)$.

k = 3

- aggiungiamo auto-ciclo su stato 10
- denota $\quad inc\ (s,s') = (l' \boxed{\leftrightarrow} l \otimes r) \wedge (r' \boxed{\leftrightarrow} \neg r)$

  $\quad stutt_{10}(s,s') = (l \wedge \neg r \wedge l' \wedge \neg r')$
- allora, $\quad T\ (s,s') = inc\ (s,s') \vee stutt_{10}\ (s,s')$
- controesempio: esiste un cammino infinito su cui □$(\neg l \vee \neg r)$
- dobbiamo verificare due cose:
  - s_0, s_1, s_2, s_3 sono un prefisso di un cammino a partire dallo stato iniziale
  - s_0, s_1, s_2, s_3 contengono un ciclo ($s_0 = s_3$ opp. $s_1 = s_3$ opp. $s_2 = s_3$)

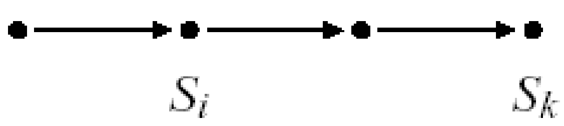
Specifica: liveness property

- $inc\ (s,s') = (l' \boxed{\leftrightarrow} l \otimes r) \wedge (r' \boxed{\leftrightarrow} \neg r)$ $\quad$ $\mathrm{stutt}_{10}\ (s,s') = (l \wedge \neg r \wedge l' \wedge \neg r')$
- $I\ (s_0) : \quad (\neg l_0 \wedge \neg r_0) \quad \wedge$
- $T\ (s_0, s_1) : \quad (\mathrm{inc}\ (s_0, s_1) \vee \mathrm{stutt}_{10}\ (s_0, s_1)) \quad \wedge$
- $T\ (s_1, s_2) : \quad (\mathrm{inc}\ (s_1, s_2) \vee \mathrm{stutt}_{10}\ (s_1, s_2)) \quad \wedge$
- $T\ (s_2, s_3) : \quad (\mathrm{inc}\ (s_2, s_3) \vee \mathrm{stutt}_{10}\ (s_2, s_3)) \quad \wedge$
- $s_3 = s_0 \ : \quad (\ (\ (l_3 \boxed{\leftrightarrow} l_0) \wedge (r_3 \boxed{\leftrightarrow} r_0) \ ) \quad \vee$
- $s_3 = s_1 \ : \quad (\ (l_3 \boxed{\leftrightarrow} l_1) \wedge (r_3 \boxed{\leftrightarrow} r_1) \ ) \quad \vee$
- $s_3 = s_2 \ : \quad (\ (l_3 \boxed{\leftrightarrow} l_2) \wedge (r_3 \boxed{\leftrightarrow} r_2) \ ) \ ) \quad \wedge$
- $p\ (s_0) : \quad (\ \neg l_0 \vee \neg r_0 \ ) \quad \wedge$
- $p\ (s_1) : \quad (\ \neg l_1 \vee \neg r_1 \ ) \quad \wedge$
- $p\ (s_2) : \quad (\ \neg l_2 \vee \neg r_2 \ )$

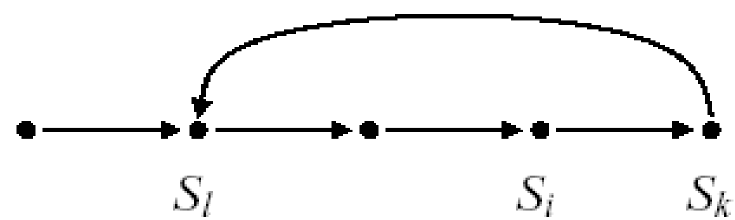
Semantica bounded

- Obiettivo analisi: cercare un controesempio su un ***prefisso finito*** di un cammino di lunghezza k
  - rappresenta cammino infinito se contiene un back loop dall'ultimo stato ad uno degli stati precedenti

- Se non è presente un back loop non si può dire nulla sui comportamenti infiniti
  - Ad esempio: per la formula □p, anche se p vale da s_0 a s_k, in assenza di loop non possiamo dire nulla riguardo al suo soddisfacimento



(a) no loop



(b) (k, l)-loop

Semantica k-bounded (no back-loop)

$\pi \models_k^i p \quad \text{iff} \quad p \in \ell(\pi(i))$

$\pi \models_k^i \neg p \quad \text{iff} \quad p \notin \ell(\pi(i))$

$\pi \models_k^i f \wedge g \quad \text{iff} \quad \pi \models_k^i f \text{ and } \pi \models_k^i g$

$\pi \models_k^i f \vee g \quad \text{iff} \quad \pi \models_k^i f \text{ or } \pi \models_k^i g$

$\pi \models_k^i \mathbf{G} f \quad \text{is always false}$

$\pi \models_k^i \mathbf{F} f \quad \text{iff} \quad \exists j, i \leq j \leq k . \pi \models_k^j f$

$\pi \models_k^i \mathbf{X} f \quad \text{iff} \quad i < k \text{ and } \pi \models_k^{i+1} f$

$\pi \models_k^i f \,\mathbf{U}\, g \quad \text{iff} \quad \exists j, i \leq j \leq k [\pi \models_k^j g \;\text{ and }\; \forall n, i \leq n < j . \pi \models_k^n f]$

$\pi \models_k^i f \,\mathbf{R}\, g \quad \text{iff} \quad \exists j, i \leq j \leq k [\pi \models_k^j f \;\text{ and }\; \forall n, i \leq n \leq j . \pi \models_k^n g]$

Soundness e completeness di LTL BMC

- f: formula LTL M: struttura di kripke k>0: bound
- **Remark.** Semantica LTL rispetto a cammini con back loop equivalente a semantica su cammini infiniti
- **Lemma.**

  Dato un cammino π di M. Se $\pi \models_k f$ allora $\pi \models f$
- **Lemma.**

  Se $M \models f$ allora esiste k tale che $M \models_k f$

  (per formule che richiedono soddisfacimento su cammini infiniti, semantica è con back loop)
- **Teorema.** $M \models f$ sse esiste k tale che $M \models_k f$

Traduzione

- formula cammini di lunghezza k in M:

$$[[\, M \,]]_{k} := I\,(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1})$$

- formula LTL ha due traduzioni a seconda se si considera semantica
  - con loop, denotato $_{l}[[\varphi]]^{i}_{k}$ oppure
  - senza loop, denotato $[[\varphi]]^{i}_{k}$

Traduzione formula (senza loop)

Inductive Case: $\forall i \leq k$

$$[\![p]\!]_k^i := p(s_i)$$

$$[\![\neg p]\!]_k^i := \neg p(s_i)$$

$$[\![f \vee g]\!]_k^i := [\![f]\!]_k^i \vee [\![g]\!]_k^i$$

$$[\![f \wedge g]\!]_k^i := [\![f]\!]_k^i \wedge [\![g]\!]_k^i$$

$$[\![\mathbf{G} f]\!]_k^i := [\![f]\!]_k^i \wedge [\![\mathbf{G} f]\!]_k^{i+1}$$

$$[\![\mathbf{F} f]\!]_k^i := [\![f]\!]_k^i \vee [\![\mathbf{F} f]\!]_k^{i+1}$$

$$[\![f \mathbf{U} g]\!]_k^i := [\![g]\!]_k^i \vee ([\![f]\!]_k^i \wedge [\![f \mathbf{U} g]\!]_k^{i+1})$$

$$[\![f \mathbf{R} g]\!]_k^i := [\![g]\!]_k^i \wedge ([\![f]\!]_k^i \vee [\![f \mathbf{R} g]\!]_k^{i+1})$$

$$[\![\mathbf{X} f]\!]_k^i := [\![f]\!]_k^{i+1}$$

Base Case:

$$[\![f]\!]_k^{k+1} := 0$$

Traduzione formula (con loop)

- **Def successore in loop**.
  In un (k,l)-loop, succ(i)=i+1 per i < k e succ(i) = l per i = k

$$
\begin{aligned}
{}_l[\![p]\!]_k^i &:= p(s_i) \\
{}_l[\![\neg p]\!]_k^i &:= \neg p(s_i) \\
{}_l[\![f \vee g]\!]_k^i &:= {}_l[\![f]\!]_k^i \vee {}_l[\![g]\!]_k^i \\
{}_l[\![f \wedge g]\!]_k^i &:= {}_l[\![f]\!]_k^i \wedge {}_l[\![g]\!]_k^i
\end{aligned}
$$

$$
\begin{aligned}
{}_l[\![\mathbf{G} f]\!]_k^i &:= {}_l[\![f]\!]_k^i \wedge {}_l[\![\mathbf{G} f]\!]_k^{\mathrm{succ}(i)} \\
{}_l[\![\mathbf{F} f]\!]_k^i &:= {}_l[\![f]\!]_k^i \vee {}_l[\![\mathbf{F} f]\!]_k^{\mathrm{succ}(i)} \\
{}_l[\![f \mathbf{U} g]\!]_k^i &:= {}_l[\![g]\!]_k^i \vee \left({}_l[\![f]\!]_k^i \wedge {}_l[\![f \mathbf{U} g]\!]_k^{\mathrm{succ}(i)} \right) \\
{}_l[\![f \mathbf{R} g]\!]_k^i &:= {}_l[\![g]\!]_k^i \wedge \left({}_l[\![f]\!]_k^i \vee {}_l[\![f \mathbf{R} g]\!]_k^{\mathrm{succ}(i)} \right) \\
{}_l[\![\mathbf{X} f]\!]_k^i &:= {}_l[\![f]\!]_k^{\mathrm{succ}(i)}
\end{aligned}
$$

Traduzione

- **(Loop Condition)**: ${}_hL_k = T(s_k, s_h)$ e $L_k = \bigvee_h {}_hL_k$

- Formula final

$$[\![M, f]\!]_k := [\![M]\!]_k \wedge \left(\left(\neg \mathrm{L}_k \wedge [\![f]\!]_k^0 \right) \vee \bigvee_{l=0}^{k} \left({}_l\mathrm{L}_k \wedge {}_l[\![f]\!]_k^0 \right) \right)$$

- **Teorema.** $[\![M,f]\!]_k$ è soddisfacibile sse $M \models_k f$

Commenti

- ogni formula LTL può essere verificata con BMC
- BMC può essere applicato alla verifica del software
  - il programma viene trasformato

How does it work

Transform a programs into a set of equations

- Simplify control flow
- Unwind all of the loops
- Convert into Static Single Assignment (SSA)
- Convert into equations
- Bit-blast
- Solve with a SAT Solver
- Convert SAT assignment into a counterexample

Control Flow Simplifications

- All side effect are removed
  - e.g., j=i++ becomes j=i; i=i+1
- Control Flow is made explicit
  - continue, break replaced by goto
- All loops are simplified into one form
  - for, do, while replaced by while

Loop Unwinding

```
void f(...) {
  ...
  while(cond) {
    Body;
  }
  Remainder;
}
```

- while() loops are unwound iteratively
- Break / continue replaced by goto

Loop Unwinding

```
void f(...) {
  ...
  if(cond) {
    Body;
    while(cond) {
      Body;
    }
  }
  Remainder;
}
```

while() loops are unwound iteratively

Break / continue replaced by goto

Loop Unwinding

```
void f(...) {
  ...
  if(cond) {
    Body;
    if(cond) {
      Body;
      while(cond) {
        Body;
      }
    }
  }
  Remainder;
}
```

while() loops are unwound iteratively

Break / continue replaced by goto

Unwinding assertion

```
void f(...) {
  ...
  if(cond) {
    Body;
    if(cond) {
      Body;
      if(cond) {
        Body;
        while(cond) {
          Body;
        }
      }
    }
  }
  Remainder;
}
```

while() loops are unwound iteratively

Break / continue replaced by goto

Assume statements inserted after last iteration: block execution if program runs longer than bound permits

Unwinding assertion

```
void f(...) {
  ...
  if(cond) {
    Body;
    if(cond) {
      Body;
      if(cond) {
        Body;
        assume(!cond);
      }
    }
  }
  Remainder;
}
```

Unwinding assume

while() loops are unwound iteratively

Break / continue replaced by goto

Assume statements inserted after last iteration: block execution if program runs longer than bound permits

Transforming Loop-Free Programs Into Equations (1)

- Easy to transform when every variable is only assigned once!

Program

```
x = a;
y = x + 1;
z = y - 1;
```

Constraints

x = a &&
y = x + 1 &&
z = y - 1 &&

Transforming Loop-Free Programs Into Equations (2)

- When a variable is assigned multiple times,
- use a new variable for the RHS of each assignment

Program

```
x=x+y;
x=x*2;
a[i]=100;
```

$\xrightarrow{\rho}$

SSA Program

$x_1 = x_0 + y_0;$

$x_2 = x_1 * 2;$

$a_1[i_0] = 100;$

What about conditionals?

Program

```
if (v)
   x = y;
else
   x = z;

w = x;
```

ρ

SSA Program

```
if (v_0)
   x_0 = y_0;
else
   x_1 = z_0;

w_1 = x??;
```

What should 'x' be?

What about conditionals?

Program

```
if (v)
   x = y;
else
   x = z;

w = x;
```

ρ

SSA Program

$\texttt{if (}v_0\texttt{)}$

$\quad x_0 = y_0;$

$\texttt{else}$

$\quad x_1 = z_0;$

$x_2 = v_0 \; ? \; x_0 : x_1;$

$w_1 = x_2$

For each join point, add new variables with selectors

Example

```
int main() {
   int x, y;
   y=8;
   if(x)
     y--;
   else
     y++;

   assert
       (y==7 ||
       y==9);
}
```

ρ

```
int main() {
   int x0, y0;
   y1=8;
   if(x0)
     y2=y1-1;
   else
     y3=y1+1;
   y4= x0 ?y2:y3;
   assert
       (y4==7 ||
       y4==9);
}
```

$$
\begin{array}{ll}
(& y_1 = 8 \\
\wedge & y_2 = y_1 - 1 \\
\wedge & y_3 = y_1 + 1 \\
\wedge & y_4 = x_0 \,?\, y_2 : y_3 \;) \\
\Longrightarrow & (y_4 = 7 \vee y_4 = 9)
\end{array}
$$

CBMC: Bounded Model Checker for C

A tool by D. Kroening/Oxford and Ed Clarke/CMU