Università degli studi di Salerno

TECNICHE AUTOMATICHE PER LA CORRETTEZZA DEL SOFTWARE

Gerardo Sessa

Indice

Bounded Model Checking

Il **BMC** (*Bounded Model Checking*) è una tecnica di verifica automatica usata per trovare errori nei modelli di sistemi.

Si tratta di una variante del model checking tradizionale, ma, anziché esplorare completamente lo spazio degli stati del sistema, BMC esplora solo una porzione limitata degli stati fino a un certo numero di passi (il **bound**, o “limite”).

Bounded Model Checking

Il **bound** quindi è il numero di *k* passi computazionali da esplorare, cioè il numero massimo di transizioni di stato o iterazioni di cicli da considerare.

Per il BMC, l'obiettivo diventa quello di soddisfare una formula booleana. Si considera il modello a stati finiti di un sistema per un numero fissato di passi k, e si controlla se una violazione può verificarsi in k o meno passi.

Bounded Model Checking

BMC converte il problema di verifica in una formula logica booleana che rappresenta il comportamento del sistema fino al limite specificato.

La formula ottenuta sarà quindi l'istanza del problema **SAT**, sottoposto ad un *SAT solver* che riporterà eventualmente un comportamento del sistema che viola la proprietà specificata.

L'obiettivo del SAT solver sarà quello di trovare un controesempio sul cammino di lunghezza k prefissato.

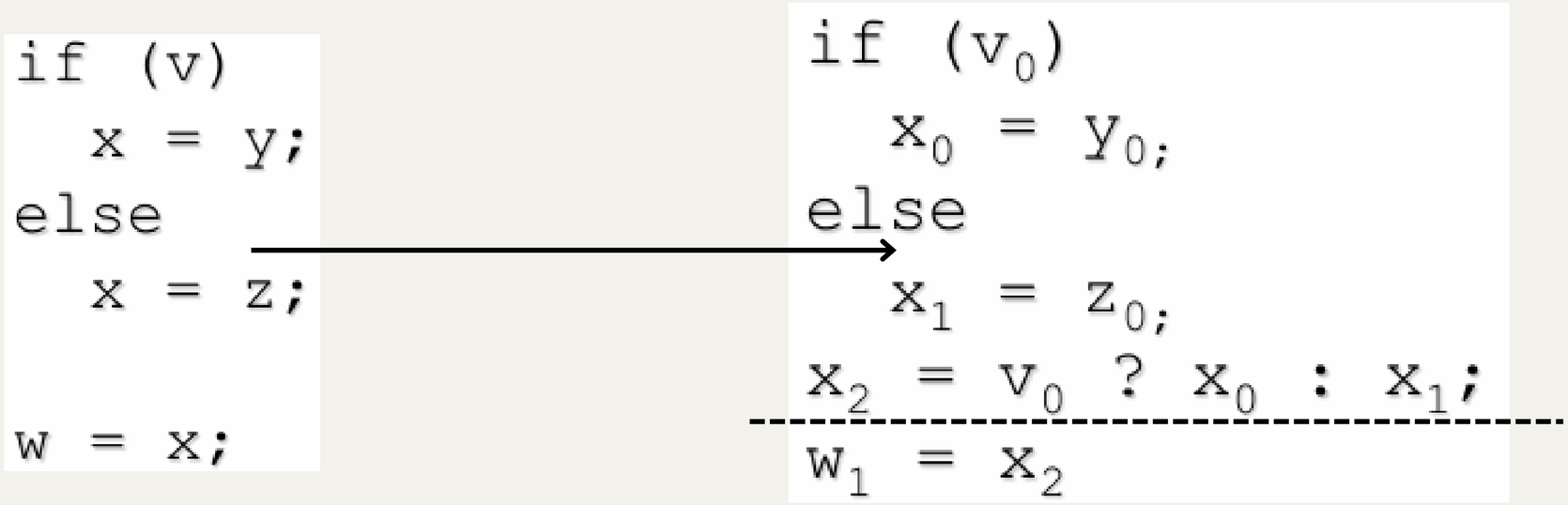
Bounded Model Checking

Per la verifica del software via BMC, il programma viene trasformato fino ad arrivare ad una formula booleana.
Alcune notazioni utili per realizzare la semplificazione del code flow:

Se una variabile viene assegnata più volte, viene utilizzata una *nuova variabile* per ogni assegnazione.

Bounded Model Checking

Le assegnazioni che possono variare in base al risultato di una *condizione* vengono invece ridefinite in questo modo:



Bounded Model Checking

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

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

```
int main () {
int x0,x1;
y1= 8;
if (x0)
y2= y1 - 1;
else y3= y1+ 1;
y4 = x0 ? y2 : x3;

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

```
(y1 = 8

^ y2 = y1 - 1

^ y3 = y1 + 1

^ y4 = x0 ? y2 : y3)

=> (y4 = 7 ^ y4 = 9)
```

CBMC

CBMC (*C Bounded Model Checker*) è uno strumento di verifica tramite bounded model checking per rilevare errori in programmi scritti in C.

CBMC traduce il codice in un problema di soddisfacibilità booleana (**SAT**), esplorando tutte le possibili combinazioni fino a una profondità scelta, sotto un limite di loop unwinding (srotolamento di cicli) specificato.
Il sat solver di riferimento è **MiniSat**, utilizzato da CBMC per verificare se esistono controesempi (esecuzioni che violano un'asserzione) nei programmi in analisi.

CBMC

CBMC gestisce l'allocazione dinamica simulando il comportamento delle chiamate a funzioni di memoria dinamica, come *malloc*, *calloc* e *realloc*.
La gestione avviene con una rappresentazione simbolica della memoria che consente di verificare la sicurezza di memoria e limiti d’accesso.

CBMC richiede che la dimensione massima della memoria su cui si pratica allocazioni dinamiche sia nota tramite configurazione, altrimenti potrebbe non essere in grado di generare verifiche attendibili.

CBMC

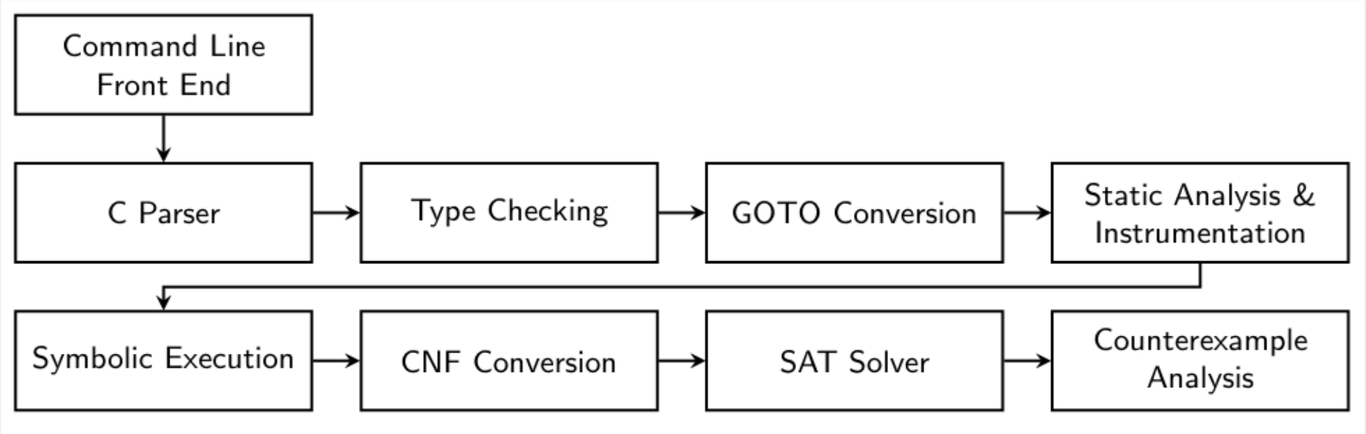
Le verifiche sulla sicurezza per l’accesso alla memoria sono i seguenti:

- **Controllo degli accessi**: Verifica che non ci siano dereferenziazioni di puntatori al di fuori delle aree allocate.
- **Deallocazioni multiple**: Garantisce che un’area di memoria non venga liberata più di una volta.
- **Null pointer**: Controlla che l'allocazione dinamica non generi un puntatore nullo.
- **Memory leaks**: Rileva perdite di memoria quando aree allocate dinamicamente non vengono liberate prima della fine del programma.

CBMC

CBMC genera una formula in forma normale congiuntiva (**CNF**), le cui soluzioni descrivono percorsi del programma che portano a violazioni di asserzioni.

Per fare ciò, CBMC esegue
i seguenti passaggi principali tramite la sua architettura.



CBMC

Front-end:

L’interfaccia a riga di comando configura inizialmente CBMC secondo i parametri forniti dall'utente, come la larghezza in bit.
Il parser di C costruisce un albero di parsing dal codice sorgente preprocessato.

CBMC

Rappresentazione intermedia:

CBMC usa i programmi **GOTO** come rappresentazione intermedia.
In questo linguaggio, tutti i flussi di controllo non lineari, come le istruzioni if o switch, i cicli e i salti, vengono tradotti in istruzioni goto condizionate.

Queste istruzioni includono condizioni opzionali. CBMC genera un programma GOTO per ciascuna funzione C trovata nell'albero di parsing.

CBMC

Middle-end:

CBMC esegue l’esecuzione simbolica srotolando i cicli fino al bound fissato.
Nel corso di questo passaggio, CBMC traduce le istruzioni GOTO in *Static Single Assignment* (**SSA**).

Al termine di questo processo, il programma è rappresentato come un sistema di equazioni su variabili rinominate in istruzioni condizionate, determinando se un'assegnazione è effettivamente eseguita in una data esecuzione concreta del programma.

CBMC

Back-end:

L'equazione risultante viene tradotta in una formula **CNF** attraverso una modellazione bit precisa di tutte le espressioni, più le condizioni booleane.

Un modello calcolato dal risolutore SAT corrisponde a un percorso che viola almeno una delle asserzioni nel programma analizzato, e il modello viene poi tradotto in una sequenza di assegnazioni per fornire un controesempio leggibile dall’utente. Al contrario, se la formula è insoddisfacibile, nessuna asserzione può essere violata entro i limiti di disavvolgimento forniti.

CBMC

```
int x = 0;
if (y > 0) {
    x = x + 1;
} else {
    x = x - 1;
}
while (x < 10) {
    x = x + 2;
}
```

GOTO

```
L1: x = 0;
if (y > 0) goto L2;
goto L3;
L2: x = x + 1;
goto L4;
L3: x = x - 1;
L4: if (x >= 10) goto L5;
x = x + 2;
goto L4;
L5: // end of program
```

SSA

```
x_1 = 0;
if (y > 0) {
    x_2 = x_1 + 1;
} else {
    x_2 = x_1 - 1;
}
while (x_2 < 10) {
    x_3 = x_2 + 2;
    x_2 = x_3;
}
```

CBMC

CNF:

Otteniamo dall’espressione logica proposta una formula CNF, mettendo "AND" tra le clausole:

```
(x1 = 0) ∧
(¬B1 ∨ x2 = x1 + 1) ∧
(B1 ∨ x2 = x1 - 1) ∧
(¬B2 ∨ x3 = x2 + 2) ∧
(¬B2 ∨ x2' = x3) ∧
(B2 ∨ x2 ≥ 10)
```

CBMC

Contro:

Il bounded model checker CBMC non è in grado di fornire prove di correttezza per programmi con cicli dal bound non definito, a meno che non vengano applicate trasformazioni aggiuntive sui cicli.

Pro:

I punti di forza del bounded model checking, invece, sono rappresentati dalle sue prestazioni prevedibili e dalla sua capacità di coprire l'intero spettro delle tipologie di verifica.

ESBMC

ESBMC è un context bounded model checker progettato per la verifica di programmi in C a singolo thread e multi-thread.

ESBMC è progettato principalmente per aiutare gli sviluppatori di software a individuare bug nel codice per le proprietà di sicurezza predefinite, come controlli su *overflow*, *deadlock* e *race condition*, oltre che consentire agli utenti di aggiungere proprie asserzioni e verificare possibili violazioni di queste.

ESBMC permette l’accesso alle strutture dati interne, consentendo ispezione ed estensioni in ogni fase del processo di verifica.

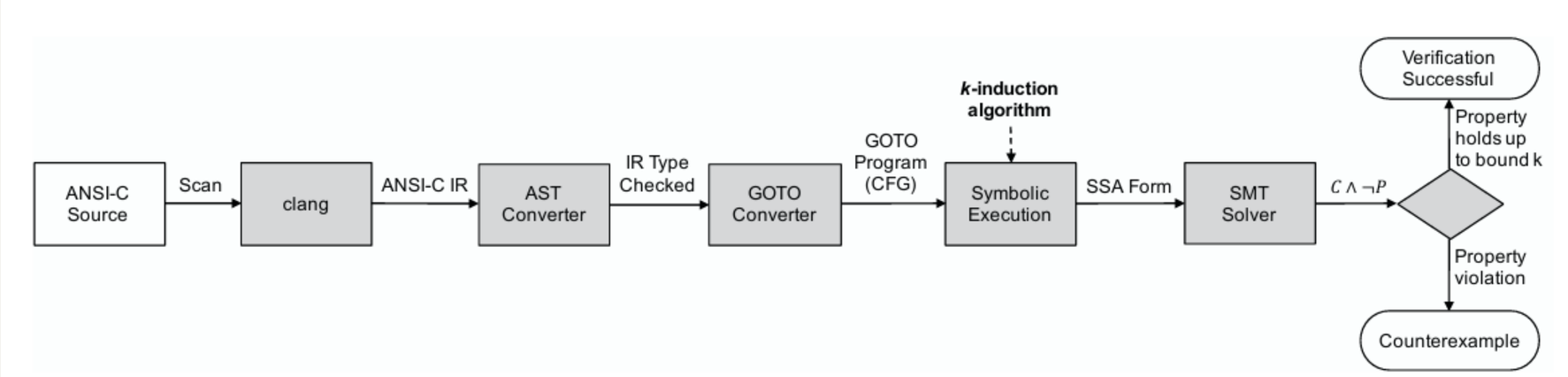
ESBMC

ESBMC implementa lo schema di **k-induction**, che può essere utilizzato per dimostrare l'assenza di violazioni di proprietà o la validità delle asserzioni definite dall'utente per k passi del programma da verificare.

ESBMC si basa su solver **SMT** (*Satisfiability Modulo Theory*).
Mentre SAT si occupa esclusivamente di formule logiche booleane, SMT estende questa capacità integrando teorie come aritmetica (numeri interi o reali), insiemi, array, o strutture algebriche più complesse.

ESBMC

L’architettura di ESBMC è composta da:



- **Front-end**: ESBMC utilizza l'API di Clang per accedere e attraversare l'*albero sintattico astratto* (AST) del programma, che semplifica il programma in in input facilitando così l'analisi del codice.

ESBMC

- **Generatore di Control-Flow Graph**: prende l'albero sintattico astratto (AST) del programma e lo trasforma in un equivalente programma *GOTO*, con eventuale loop unwinding se consideriamo la modalità di k-induction in presenza di cicli, assegnando valori non deterministici a tutte le variabili scritte all'interno del ciclo.

- **Motore di Esecuzione Simbolica**: esecuzione del programma GOTO ottenuto, svolgendo k volte i cicli e generando un *Static Single Assignment* (SSA).

ESBMC

- **SMT back-end**: utilizzato per codificare la forma SSA del programma in una formula priva di quantificatori e verifica la soddisfacibilità di $C \wedge \neg P$, con C che è l'insieme di vincoli e P l'insieme di proprietà.

Se la formula è SAT (soddisfacibile), significa che il programma contiene un bug, e in tal caso, verrà mostrato un controesempio, fornendo il set di assegnazioni che porta alla violazione della proprietà. SMT, infatti, estende SAT con eventuali vincoli più complessi ma non elimina la sua centralità nella verifica.

ESBMC

- API Python: l’aggiunta di questo componente permette di aggiungere una funzione intrinseca per modellare una nuova funzione di libreria o per sfruttare una teoria SMT diversa intercettando il processo di verifica.

```
def symex_step(self, art):
  # Boilerplate accessing instruction 'insn' omitted
  if insn.type == gptypes.FUNCTION_CALL:
    call = esbmc.downcast_expr(insn.code)
    sym = esbmc.downcast_expr(call.function)
    if sym.name.as_string() == 'c::isnan':
      # Interpretation of call here
      return
  # Otherwise call through to rest of ESBMC
  super(ThisClass, self).symex_step(art)
```

ESBMC

Grazie a SMT che permette lo svolgimento di problemi più complessi, per la prova per k-induzione è possibile verificare la correttezza di programmi con cicli, utilizzando una tecnica basata sull'induzione per dimostrare che un certo comportamento del programma è vero per tutte le iterazioni del ciclo, superando il limite definito per l’unwinding del loop.

ESBMC

$$kind(P,k) = \begin{cases} P \text{ contains a bug}, & \text{if } B(k) \text{ is SAT} \\ P \text{ is correct}, & \text{if } B(k) \wedge [F(k) \vee I(k)] \text{ is UNSAT} \\ kind(P,k+1), & \text{otherwise.} \end{cases}$$

La formula *B(k)* è la formula standard del bounded model checking (BMC), che è soddisfacibile e e solo se il programma ha un controesempio di lunghezza k o inferiore.
Se tutti gli stati sono raggiungibili per il valore corrente di k, sappiamo che il programma deve essere corretto senza bisogno di controllare il passo induttivo.

ESBMC

La condizione forward *F(k)* può essere derivata dal programma inserendo asserzioni di srotolamento (unwinding) dopo ogni ciclo.

Questo passaggio è particolarmente utile per dimostrare la sicurezza in presenza di cicli limitati.

Il passo induttivo *I(k)* verifica che, se una proprietà di sicurezza è valida nei primi k passi, allora è valida anche per il passo k+1.

Il deepening iterativo implica che ESBMC trova sempre il valore minimo di k per dimostrare la correttezza o individuare una violazione di proprietà.

ESBMC

Nel confronto con CBMC, ESBMC si distingue per gestire l'intero processo di verifica con k-induction in un'unica chiamata, mentre CBMC richiede tre passaggi separati: generare il CFG, annotare il programma e verificare.

Inoltre, ESBMC include una forward condition per verificare se tutti gli stati sono stati raggiunti, una funzionalità assente in CBMC, che invece si basa su un'iterazione limitata dei cicli (loop unwinding).

Il contro di questa modalità di verifica è può diventare costosa in termini di tempo e memoria per cicli con logica complessa o per programmi di grandi dimensioni.

Riferimenti

- **CBMC: CBounded Model Checker (Competition Contribution)** - Daniel Kroening, Michael Tautschnig

- **ESBMC 5.0: An Industrial-Strength C Model Checker** - Mikhail R. Gadelha, Felipe R. Monteiro, Jeremy Morse, Lucas C. Cordeiro, Bernd Fischer, Denis A. Nicole

Fine presentazione

Grazie per l'attenzione