Formální Metody a Specifikace (LS 2011)
Přednáška 6:
Ověření omezené správnosti programů

Stefan Ratschan

Katedra číslicového návrhu
Fakulta informačních technologií
České vysoké učení technické v Praze

18. březen 2011

Dnešní příklad

1: $x \leftarrow 2x$
2: **goto** 1

Dnes: Metoda pro automatické důkazy správnosti.

Funguje pro mnohem větší příklady (ukázka na konci)

Ale: Vyrábí obrovské formule které
můžeme rozumným způsobem zobrazovat jen pro tak malé příklady.

Operační sémantika: zopakování

Stav programu je valuací (tj. funkcí která přiřazuje proměnným hodnoty) která

- přiřazuje speciální proměnné pc číslo řádku,
- každé proměnné programu hodnotu odpovídajícího typu.

Množina všech stavů S.

Pro stavy $s, s' \in S$, $s \rightarrow s'$
pokud program může dělat krok ze stavu s do stavu s'
(*přechodovou relace*)

Přechodová relace: souhrn

$s \to s'$ když přechodová podmínka platí na s, s'.

V našem příkladě:

$$\mathcal{I}, s \circ \pi(s') \models \begin{array}{l} pc = 1 \Rightarrow [pc' = 2 \wedge x' = 2x] \wedge \\ pc = 2 \Rightarrow [pc' = 1 \wedge x' = x] \end{array}$$

Obecně: $s \to s'$ přesně když

$$\mathcal{I}, s \circ \pi(s') \models \bigwedge_{i \in \{1,\dots,l\}} pc = i \Rightarrow \Phi_{P,i}$$

přičemž $\Phi_{P,i}$ je formule odpovídající řádku i v programu P
(viz. minulá přednáška)

Celkový programový průběh

Program může dělat libovolný počet kroků podle $\rightarrow$:

$r \rightarrow^* r'$ přesně když

existuje posloupnost $s_1, \ldots, s_n$ tak, že $r = s_1 \rightarrow \cdots \rightarrow s_n = r'$

Tj. $\rightarrow^*$ je *tranzitivním uzávěrem* relace $\rightarrow$

Např. $\{pc \mapsto 1, x \mapsto 1\} \rightarrow^* \{pc \mapsto 2, x \mapsto 4\}$ kvůli

$$\begin{array}{l}
\{pc \mapsto 1, x \mapsto 1\} \rightarrow \\
\{pc \mapsto 2, x \mapsto 2\} \rightarrow \\
\{pc \mapsto 1, x \mapsto 2\} \rightarrow \\
\{pc \mapsto 2, x \mapsto 4\}
\end{array}$$

Ověření správnosti programů

Formule popisující správnost:

$$T \Rightarrow \forall x, x' \,.\, [I(x) \wedge x' = f(x) x' = f(x)] \Rightarrow O(x, x')$$

Chování programů:

- Přechodová relace $\rightarrow$,
- tranzitivní uzávěr $\rightarrow^*$,
- sémantika $[\![P]\!]$.

Jak zakódovat chování určitého programu do formule?

Cíl: Dokázat správnost programů, pokud možné automaticky!

Tj., pokud všechny datové struktury jsou ve rozhodnutelných teoriích (např. lineární aritmetika celých čísel)

Logická formule pro programovou sémantiku

$\rightarrow$ je definováno na základě logické formule (přechodové podmínky Φ_P) $\surd$

Podíváme se na definici $\rightarrow^*$:

$r \rightarrow^* r'$ přesně když

existuje posloupnost stavů $s_1, \ldots, s_n$ tak,

že $r = s_1 \rightarrow \cdots \rightarrow s_n = r'$

Logická formule?

Pro definici posloupnosti potřebujeme teorii množin,

Gödel (neúplnost) a Turing (nerozhodnutelnost)

předpovídají nepříjemné výsledky

Prostě škrtněme slovo "posloupnost":

existují stavy $s_1, \ldots, s_n$ tak, že $r = s_1 \rightarrow \cdots \rightarrow s_n = r'$

Rozdíl?

Omezená dosažitelnost stavů

$r \rightarrow^n r'$ přesně když
existují stavy $s_1, \dots, s_n$ tak, že $r = s_1 \rightarrow \cdots \rightarrow s_n = r'$

Intuice: Stav r' je dosažitelný ze stavu r během n kroků, $\rightarrow^n \neq \rightarrow^*$!

Po substituci definice $\rightarrow$
existují stavy $s_1, \dots, s_n$ tak, že

- $r = s_1$,
- pro každý $i \in \{1, \dots, n-1\}$, $s_i \circ \pi(s_{i+1}) \models \Phi_P$,
- $s_n = r'$.

Existují vs. $\exists$?

s_i: Valuace (např. $\{pc \mapsto 2, x \mapsto 7\}$)

$$\Phi_P = \begin{array}{l} pc = 1 \Rightarrow [pc' = 2 \wedge x' = 2x] \wedge \\ pc = 2 \Rightarrow [pc' = 1 \wedge x' = x] \end{array}$$

Podmínka omezené dosažitelnosti

Pro náš program: $r \to^n r'$ přesně když

$$r \circ \pi(r') \models \exists pc_1, x_1, \ldots, pc_n, x_n \,.\, pc = pc_1 \wedge x = x_1 \wedge$$
$$\begin{aligned} & pc_1 = 1 \Rightarrow [pc_2 = 2 \wedge x_2 = 2x_1] \wedge \\ & pc_1 = 2 \Rightarrow [pc_2 = 1 \wedge x_2 = x_1] \wedge \\ & \ldots \\ & pc_{n-1} = 1 \Rightarrow [pc_n = 2 \wedge x_n = 2x_{n-1}] \wedge \\ & pc_{n-1} = 2 \Rightarrow [pc_n = 1 \wedge x_n = x_{n-1}] \wedge \end{aligned}$$
$$pc' = pc_n \wedge x' = x_n.$$

Obecně: $r \to^n r'$ přesně když

$$r \circ \pi(r') \models \exists v_1, \ldots, v_n \,.$$
$$r = v_1 \wedge \bigwedge_{i=1,\ldots,n-1} \Phi_P[v \leftarrow v_i, v' \leftarrow v_{i+1}] \wedge v_n = r'.$$

přičemž v stojí pro všechny proměnné v programu,
popř. proměnné s určitým indexem, s čárkou.

Omezená správnost programů

Porovnání:

$$T \Rightarrow \forall x, x' \,.\, [I(x) \wedge x' = f(x)] \Rightarrow O(x, x')$$

$$\Phi_P^n := \exists v_1, \ldots, v_n \,.\, r = v_1 \wedge \bigwedge_{i=1,\ldots,n-1} \Phi_P[v \leftarrow v_i, v' \leftarrow v_{i+1}] \wedge v_n = r'$$

Místo podmínek na vstupy a výstupy,
podmínky $I(r)$, $O(r)$ na proměnné programu včetně pc.

Každý program se specifikací vstupů a výstupů lze do toho přeložit.

Potom můžeme substituovat Φ_P^n:

$$T \Rightarrow \forall r, r' \,.\, \left[I \wedge \Phi_P^n\right] \Rightarrow O[r \leftarrow r']$$

Jsme ve predikátové logice prvního řádu! Pokud používáme jen datové typy ve rozhodnutelných teoriích můžeme automaticky ověřit platnost

Nicméně to zkusíme dále zjednodušit!

Omezená správnost programů

Výsledek zjednodušení *BMC*(n) ("bounded model checking") :=

$$\neg\exists v_1, \dots, v_n \,.\, I[v \leftarrow v_1] \wedge \bigwedge_{i=1,\dots,n-1} \Phi_P[v \leftarrow v_i, v' \leftarrow v_{i+1}] \wedge \neg O[v \leftarrow v_n]$$

Posloupnost stavů $s_1, \dots, s_n$ tak, že

$$\pi^1(s_1) \circ \dots \circ \pi^n(s_n) \models$$
$$I[v \leftarrow v_1] \wedge \bigwedge_{i=1,\dots,n-1} \Phi_P[v \leftarrow v_i, v' \leftarrow v_{i+1}] \wedge \neg O[v \leftarrow v_n]$$

přičemž pro valuaci s, $\pi^i(s)$ je valuací která
přiřazuje stejné hodnoty do proměnných s indexem i,

se jmenuje *protipříklad* (*counter-example*), *chybná stopa* (*error trace*), *chybná trajektorie* (error trajectory).

Ověřování omezeného počtu kroků

Do $O(r)$ můžeme zakódovat požadavky na
libovolný řádek v programu, např.: array overflow, divize nulou, atd.

Správnost během $1, 2, \ldots$ kroků:

$$\begin{gathered} \models T \Rightarrow BMC(1) \\ \models T \Rightarrow BMC(2) \\ \models T \Rightarrow BMC(3) \\ \ldots \end{gathered}$$

Diskuse

Protipříklady vždy mají omezenou délku:
Můžeme ho je vždy najít, pokud máme dostatečně mnoho času!

Tudíž: Nalezení chyb
v programech s datovými typy v rozhodnutelné teorii
je částečně rozhodnutelný.

V praxi: Programy můžou dělat obrovský počet kroků,
tím pádem musíme ověřit $BMC(n)$ pro obrovská n.

Ale: v určitých aplikacích se to většinou nestává

Např.: Vestavěné systémy:
Reakce na určité události smí trvat jen omezenou dobu

Hardware: BMC se už používá běžně
(viz. MI-MAS: Modelování a analýza systémů)

CBMC Demo

`http://www.cprover.org/cbmc/`

`cbmc demo.c --bounds-check`

`cbmc no_bound.c --bounds-check`:

`cbmc bubble_sort.c --bounds-check`:

Existuje k tak, že

$$\bigcup_{i \in \{0,\ldots,k\}} \to^i \; = \; \to^*$$

`cbmc --help`

Další aplikace: Kombinace s testováním

Pro software v aplikacích kde bezpečnost je kritická,
existují standardy které žádají určitou úplnost testů.

Většinou jde o kriteria pokrytí zdrojového kódu (*coverage criteria*).

Např.: Testy musí provést každý řádek v programu aspoň jednou.

Problém: Jak najít test který provede řádek l?

Spustíme $BMC(1)$, $BMC(2)$, $\ldots$ pro vlastnost $O :\Leftrightarrow pc \neq l$.

European Train Control System (ETCS) [Angeletti et al., 2010]

Další aplikace: Odstranění chyb

Často známe chybu, ale neznáme důvod.

Např.: Může $x \leq 12$ v řádku 2643 způsobit dělení nulou v řádku 752?

Spustíme $BMC(1)$, $BMC(2)$, ... pro vlastnosti

- $I :\Leftrightarrow pc = 2643 \wedge x \leq 12$,
- $O :\Leftrightarrow pc = 752 \Rightarrow y \neq 0$.

Závěr

BMC může dokázat správnost programů
během omezeného počtu kroku.

Přespříští přednáška:
Ověřování neomezeného počtu kroků.

Příští přednáška:
Formální metody pro vestavěné systémy.

Literature I

Damiano Angeletti, Enrico Giunchiglia, Massimo Narizzano, Alessandra Puddu, and Salvatore Sabina. Using bounded model checking for coverage analysis of safety-critical software in an industrial setting. *Journal of Automated Reasoning*, 45:397–414, 2010. ISSN 0168-7433.