PATTERN-BASED VERIFICATION OF PROGRAMS

Jan Kubíček
Bachelor Degree Programme (1), FIT BIT
E-mail: xkubic28@stud.fit.vutbr.cz

Supervised by: Tomáš Vojnar
E-mail: vojnar@fit.vutbr.cz

ABSTRACT

The work is about complete automation of pattern-based verification. The goal is to make an automatic translation of a subset of the C language to specific internal representation in Prolog. Some new ideas about detection of relevant parts of code are represented here.

1. ÚVOD

Naším cílem je přinést formální verifikaci co nejblíže použití. Konkrétně se jedná o verifikaci správnosti použití ukazatelů při práci s dynamickými strukturami založenými na ukazatelích. Základem, ze kterého vycházíme, je prototypová implementace verifikace založené na ukazatelích (PBV) zpracovaná Pavlem Erlebachem. Zde je potřeba zmínit, že daná implementace je napsána v jazyce Prolog a jejím vstupem je abstrakce programu v Prologu. Verifikace založená na ukazatelích je již automatická. Naším cílem je tedy vytvořit překladač, který zpracuje program, který je napsán v podmnožině jazyka C, a vytvoří z něj abstrakci v Prologu, nad kterou se bude provádět verifikace.

2. VERIFIKACE ZALOŽENÁ NA UKAZATELÍCH (PBV)

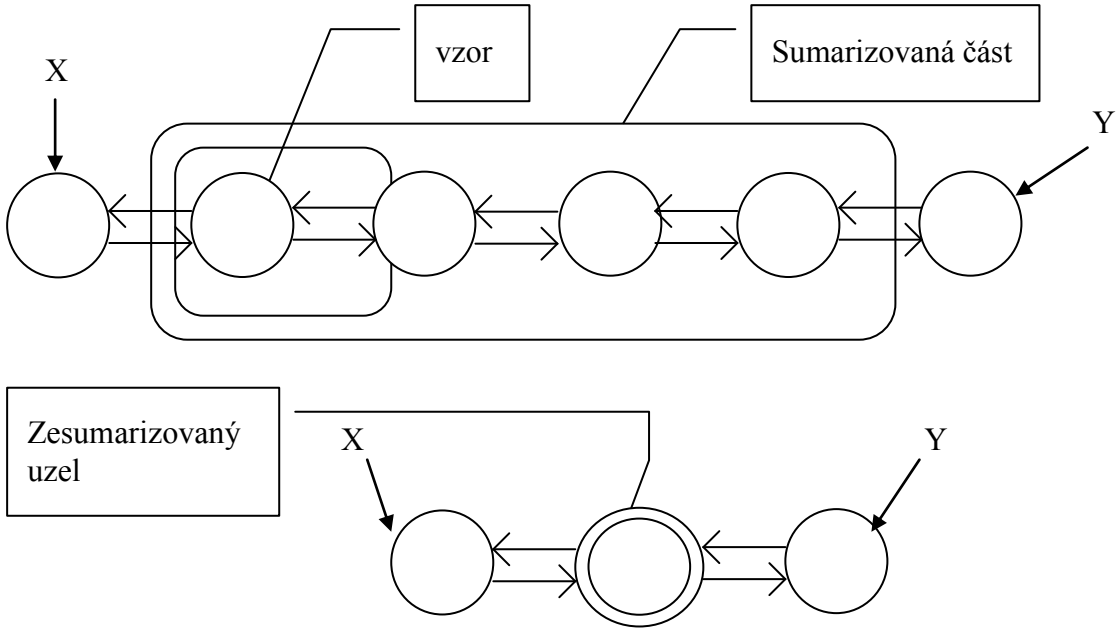
Podle článku [1] se PBV dá použít při verifikaci grafových struktur s lineární páteří. Jako příklad bych uvedl jedno- a obou- směrné seznamy, popřípadě tyto struktury s dodatečnými ukazateli (např. na konec seznamu, na datové položky apod.). Tato forma byla nedávno ještě rozšířena na seznamy seznamů, což je typ struktury používaný často například v linuxovém jádře.

Struktury seznamů jsou obecně neomezené, jejich verifikace je proto nekonečně stavová. Je tedy potřeba se s touto nekonečností vyrovnat. Jednou z možností je abstrakce založená na vzorech.

2.1. VZORY, SUMARIZACE, KONKRETIZACE

Vzor je část grafu, která se v původním grafu opakuje. Podle vzoru se provede sumarizace libovolného množství uzlů do sumarizačního uzlu v abstrakci dané struktury (viz obr 1). Tím se dosáhne reprezentace nekonečného stavového prostoru konečným zápisem. K práci s touto strukturou je zapotřebí konkretizace, kdy se ze sumarizovaného uzlu vydělí

konkrétní uzel, se kterým se bude pracovat. K tomu dochází například při posunu ukazatele na další uzel, přičemž dalším uzlem je právě sumarizovaný uzel.



Obrázek 1: Sumarizace obousměrně vázaného seznamu.

2.2. VERIFIKACE S VYUŽITÍM VZORŮ

Při verifikaci se postupně berou vstupní příkazy a jsou aplikovány na již vzniklé paměťové konfigurace, čímž vznikají další paměťové konfigurace. Přitom je potřeba si uvědomit, že abstrakce programu často obsahuje nedeterminismus, který vede ke vzniku více paměťových konfigurací. Dojde-li při aplikaci nějakého příkazu k chybě (např. reference přes NULL ukazatel), je předána zpráva o tom, kde a za jakých okolnosti se to může stát a verifikace končí. Pokud nebyla nalezena žádná chyba, verifikace končí, když již nejsou žádné nezpracovaná paměťové konfigurace.

3. ŘEŠENÍ VSTUPU DO PBV

3.1. ANALÝZA JAZYKA C

Konstrukce v jazyce C lze podle relevantnosti vzhledem k ukazatelovým manipulacím a možnosti překladu rozdělit do několika větších celků.

- Části, které se mapují 1:1 s případným preprocesingem. Jako příklad se zde dá uvést přiřazení do proměnné nebo ukazatele *x = y -> z; x -> y = NULL;*
- Části, které se dají rozložit na posloupnost predikátů v Prologu. Typickým příkladem jsou řídící instrukce *while(){}* nebo *if(){} else{}*, které jsou zpracovávány samostatně tak, aby byla zachována logika těchto příkazů. Přitom spojené podmínky je potřeba řešit vytvořením kaskády jednoduchých podmínek. Podmínky jsou dvojího druhu. Podporované ve verifikaci, které jsou zachovány, a nepodporované, které jsou nahrazeny nedeterminismem. To znamená, že konkrétní podmínka nebude při verifikaci brána v potaz vůbec, a obě cesty (přeskočení podmínky, nebo průchod tělem) jsou stejně pravděpodobné. Jako příklad podmínky nahrazené nedeterminismem bych uvedl *(x>5)*.

- Skupiny příkazů, které se dají označit jako zbytečné, protože nemění ukazatele, ani strukturu paměti. Tyto příkazy se dají bezpečně nadaproximovat. Jako příklad vypuštěného příkazu lze uvést x=5; printf(“mňam”); deklarace struktury a podobně. Je potřeba se ovšem zamyslet nad vedlejšími efekty některých operací, aby do této skupiny nebyly zařazeny příkazy, které jsou důležité.
- Nepřeložitelné příkazy. Jedním z těchto příkazů je rekurzivní volání funkce. Přitom je třeba rozlišovat, zdali je ve funkci alespoň jeden relevantní příkaz, protože jinak je možné tuto rekurzi zahrnout do skupiny vypuštěných příkazů. Toto omezení je dáno nemožností pojmenovávat části kódu ve verifikačním prototypu. Druhé omezení je zakázání pointerové aritmetiky.

3.2. CÍLOVÝ KÓD

Program je překládán na množinu predikátů *stat* vytvářejících graf toku řízení programu. Každý predikát má 3 části. První část vyjadřuje výchozí stav toku řízení, druhá část je akce řízení programu, a třetí část je konečný stav po přechodu. Příklad překladu z jazyka C do Prologu:

```
Pseznam * seznam;
x=10;
while (x-- > 0){
  printf("%d", seznam->data);
  seznam = seznam ->dalsi;
}
seznam =NULL;
```

```
stat(s0,skip,s2).
stat(s0,(setToField, seznam, seznam,dalsi),s0).
stat(s2,(setNull, seznam),s3).
```

Zde je vidět jak lze přejít do stavu *s0* dvěma různými způsoby (návrat v cyklu, nebo příchod původní výpočetní větví). Toto se stává v případě řídících konstrukcí, jejichž podmínka byla nahrazena nedeterminismem.

4. ZÁVĚR

Jakožto největší vlastní přínos považuji to, že PBV je úplně automatizována do té míry, že je možné ji použít i na jednoduché programy napsané v určité podmnožině běžného programovacího jazyka. Program byl úspěšně otestován na několika předpřipravených příkladech s omezeným počtem příkazů. V dalším vývoji bude dále rozšiřována podporovaná podmnožina jazyka C. Dlouhodobým cílem do budoucnosti by mohlo být vyřešení problému s rekurzí, dořešení problémů s pointerovou aritmetikou a kombinace datových a ukazatelových složek.

LITERATURA

[1] M. Ceska, P. Erlebach, and T. Vojnar. Pattern-Based Verification of Programs with Extended Linear Linked Data Structures. *Electronic Notes in Theoretical Computer Science*, 145:113--130, 2006. *Proc. of 5th International Workshop on Automated Verification of Critical Systems*---AVOCS'05, Warwick, UK, pages 101--117, 2005.