Překladače (10. přednáška)

Jan Hubička

Katedra aplikované matematiky
Univerzita Karlova
Praha

18. května 2020

Alias analysis

Alias analysis

Alias analýza je orákulum zodpovídající otázku, jestli dva přistupy do paměti mohou přistupovat na stejná data.

Alias analysis

Alias analysis

Alias analýza je orákulum zodpovídající otázku, jestli dva přistupy do paměti mohou přistupovat na stejná data.

Používá se například při value numberingu, mazání mrtvého kódu, schedulingu, otimalizaci práce s pamětí (store sinking and merging, load hoisting), optimalizaci smyček (vektorizaci, transformaci vnořených smyček atd.)

Alias analysis

Alias analysis

Alias analýza je orákulum zodpovídající otázku, jestli dva přistupy do paměti mohou přistupovat na stejná data.

Používá se například při value numberingu, mazání mrtvého kódu, schedulingu, otimalizaci práce s pamětí (store sinking and merging, load hoisting), optimalizaci smyček (vektorizaci, transformaci vnořených smyček atd.)

Typy závoslostí

1. True dependence

Alias analysis

Alias analysis

Alias analýza je orákulum zodpovídající otázku, jestli dva přistupy do paměti mohou přistupovat na stejná data.

Používá se například při value numberingu, mazání mrtvého kódu, schedulingu, otimalizaci práce s pamětí (store sinking and merging, load hoisting), optimalizaci smyček (vektorizaci, transformaci vnořených smyček atd.)

Typy závoslostí

1. True dependence
2. Anti dependence

Alias analysis

Alias analysis

Alias analýza je orákulum zodpovídající otázku, jestli dva přistupy do paměti mohou přistupovat na stejná data.

Používá se například při value numberingu, mazání mrtvého kódu, schedulingu, otimalizaci práce s pamětí (store sinking and merging, load hoisting), optimalizaci smyček (vektorizaci, transformaci vnořených smyček atd.)

Typy závoslostí

1. True dependence
2. Anti dependence
3. Output dependence

Alias analysis

Alias analysis

Alias analýza je orákulum zodpovídající otázku, jestli dva přistupy do paměti mohou přistupovat na stejná data.

Používá se například při value numberingu, mazání mrtvého kódu, schedulingu, otimalizaci práce s pamětí (store sinking and merging, load hoisting), optimalizaci smyček (vektorizaci, transformaci vnořených smyček atd.)

Typy závoslostí

1. True dependence
2. Anti dependence
3. Output dependence
4. Input dependence

Alias analysis

Alias analysis

Alias analýza je orákulum zodpovídající otázku, jestli dva přistupy do paměti mohou přistupovat na stejná data.

Používá se například při value numberingu, mazání mrtvého kódu, schedulingu, otimalizaci práce s pamětí (store sinking and merging, load hoisting), optimalizaci smyček (vektorizaci, transformaci vnořených smyček atd.)

Typy závoslostí

1. True dependence
2. Anti dependence
3. Output dependence
4. Input dependence

Závislosti si můžeme představovat jako DAG, který ovšem není tranzitivní. (Tedy se na něj úplně nehodí SSA forma. O různých formách memory SSA existuje mnoho publikací, ale v praxi jsme nenalezli žádnou dobře fungující.)

Base+offset+size

Base+offset+size oracle

Každý přístup do paměti se rozdělí na

1. Base pointer
2. Offset
3. Min size
4. max size

Pokud dva přístupy mají stejný base, lze snadno otestovat jestli se dva přístupy protínají.

Type based alias analysis

V memory modelu ANSI-C (a modernějších jazyků) maji data uložené v paměti typy. Ty se přidělují při zápisu a čtení lze provádět pouze “kompatibilním typem”

Alias set

Každý (kanonický) typ má přidělený alias set. Ke každému typu se pak určí množina všech alias setů, ze kterých je zkonstruován

Přístupová cesta (access path oracle)

```
struct C {struct C c};
struct A {struct B b} *a;
...
a->b->c
```

Points-to

Poinst-to množina ukazatele obsahuje všechny pamětové lokace na které může ukazovat

Implementování ve dvou fázích.
1. sbírání podmínek (address-of, copy, assign, dereference),
2. propagace.

```
int main(void)
{
        struct { int *f1; }x, *z;
        int y[70], w;

        z = &x;         /* [(z,x,D)] */
        (*z).f1 = &w; /* [(<x.f1>,w,D), (z,x,D)] */
        x.f1 = &y[0]; /* [(<x.f1>,y,P), (z,x,D)] */
}
```

Varianty points-to analýzy

- Intraprocedurální / interprocedurální

Varianty points-to analýzy

- Intraprocedurální / interprocedurální
- Reprezentace (Andresen / Steensgaard)

Varianty points-to analýzy

- Intraprocedurální / interprocedurální
- Reprezentace (Andresen / Steensgaard)
- Field sensitivita

Varianty points-to analýzy

- Intraprocedurální / interprocedurální
- Reprezentace (Andresen / Steensgaard)
- Field sensitivita
- Flow sensitivita

Varianty points-to analýzy

- Intraprocedurální / interprocedurální
- Reprezentace (Andresen / Steensgaard)
- Field sensitivita
- Flow sensitivita
- Kontextová sensitivita

Varianty points-to analýzy

- Intraprocedurální / interprocedurální
- Reprezentace (Andresen / Steensgaard)
- Field sensitivita
- Flow sensitivita
- Kontextová sensitivita
- Sbírání may a must informace

Varianty points-to analýzy

- Intraprocedurální / interprocedurální
- Reprezentace (Andresen / Steensgaard)
- Field sensitivita
- Flow sensitivita
- Kontextová sensitivita
- Sbírání may a must informace
- Modelování hlady

Statistiky z LTO-optimalizace gcc:

```
Alias oracle query stats:
  refs_may_alias_p: 39 227 004 disambiguations, 47 344 870 queries
  ref_maybe_used_by_call_p: 57 815 disambiguations, 39 808 081 queries
  call_may_clobber_ref_p: 5 511 disambiguations, 8 287 queries
  aliasing_component_ref_p: 90 654 disambiguations, 269 895 queries
  TBAA oracle: 11 130 153 disambiguations 34 368 560 queries
               14 199 938 are in alias set 0
               5 193 428 queries asked about the same object
               147 queries asked about the same alias set
               0 access volatile
               1 665 979 are dependent in the DAG
               2 178 915 are aritificially in conflict with void *

PTA query stats:
  pt_solution_includes: 464 074 disambiguations, 7 105 649 queries
  pt_solutions_intersect: 355 139 disambiguations, 6 952 492 queries
```

Profile feedback

1. BB profile

Profile feedback

1. BB profile
2. Edge profile

Profile feedback

1. BB profile
2. Edge profile
3. Path profile

Profile feedback

1. BB profile
2. Edge profile
3. Path profile
4. Value profile

Auto-FDO

Static profile estimation

1. Odhadnutí pravděpodobností hran pomocí heuristik (Ball, Larus: Branch prediction for free 1993)

Static profile estimation

1. Odhadnutí pravděpodobností hran pomocí heuristik (Ball, Larus: Branch prediction for free 1993)
2. Sloučení násobných heuristik podle first match nebo Dempster-Shafer theory: $\frac{p_1 p_2}{p_1 p_2 + (1-p_1)(1-p_2)}$

Static profile estimation

1. Odhadnutí pravděpodobností hran pomocí heuristik (Ball, Larus: Branch prediction for free 1993)
2. Sloučení násobných heuristik podle first match nebo Dempster-Shafer theory: $\frac{p_1 p_2}{p_1 p_2 + (1-p_1)(1-p_2)}$
3. Propagace profilu (Wu, Larus: Static Branch Frequency and Program Profile Analysis)

Static profile estimation

1. Odhadnutí pravděpodobností hran pomocí heuristik (Ball, Larus: Branch prediction for free 1993)
2. Sloučení násobných heuristik podle first match nebo Dempster-Shafer theory: $\frac{p_1 p_2}{p_1 p_2 + (1-p_1)(1-p_2)}$
3. Propagace profilu (Wu, Larus: Static Branch Frequency and Program Profile Analysis)

Heuristiky:

1. Loop: Smyčky se točí (a pokud vím kolikrát o to lépe)
2. Continue: Smyčky uzavřené pomocí continue mívají menší počet iterací než ostatní

Static profile estimation

1. Odhadnutí pravděpodobností hran pomocí heuristik (Ball, Larus: Branch prediction for free 1993)
2. Sloučení násobných heuristik podle first match nebo Dempster-Shafer theory: $\frac{p_1 p_2}{p_1 p_2 + (1-p_1)(1-p_2)}$
3. Propagace profilu (Wu, Larus: Static Branch Frequency and Program Profile Analysis)

Heuristiky:

1. Loop: Smyčky se točí (a pokud vím kolikrát o to lépe)
2. Continue: Smyčky uzavřené pomocí continue mívají menší počet iterací než ostatní
3. Pointer: Ukazatele nebývají NULL

Static profile estimation

1. Odhadnutí pravděpodobností hran pomocí heuristik (Ball, Larus: Branch prediction for free 1993)
2. Sloučení násobných heuristik podle first match nebo Dempster-Shafer theory: $\frac{p_1 p_2}{p_1 p_2 + (1-p_1)(1-p_2)}$
3. Propagace profilu (Wu, Larus: Static Branch Frequency and Program Profile Analysis)

Heuristiky:

1. Loop: Smyčky se točí (a pokud vím kolikrát o to lépe)
2. Continue: Smyčky uzavřené pomocí continue mívají menší počet iterací než ostatní
3. Pointer: Ukazatele nebývají NULL
4. Opcode positive: Hodnoty jsou většinou kladné

Static profile estimation

1. Odhadnutí pravděpodobností hran pomocí heuristik (Ball, Larus: Branch prediction for free 1993)
2. Sloučení násobných heuristik podle first match nebo Dempster-Shafer theory: $\frac{p_1 p_2}{p_1 p_2 + (1-p_1)(1-p_2)}$
3. Propagace profilu (Wu, Larus: Static Branch Frequency and Program Profile Analysis)

Heuristiky:

1. Loop: Smyčky se točí (a pokud vím kolikrát o to lépe)
2. Continue: Smyčky uzavřené pomocí continue mívají menší počet iterací než ostatní
3. Pointer: Ukazatele nebývají NULL
4. Opcode positive: Hodnoty jsou většinou kladné
5. Opcode nonequal: Hodnoty jsou většinou různé

Static profile estimation

1. Odhadnutí pravděpodobností hran pomocí heuristik (Ball, Larus: Branch prediction for free 1993)
2. Sloučení násobných heuristik podle first match nebo Dempster-Shafer theory: $\frac{p_1 p_2}{p_1 p_2 + (1-p_1)(1-p_2)}$
3. Propagace profilu (Wu, Larus: Static Branch Frequency and Program Profile Analysis)

Heuristiky:

1. Loop: Smyčky se točí (a pokud vím kolikrát o to lépe)
2. Continue: Smyčky uzavřené pomocí continue mívají menší počet iterací než ostatní
3. Pointer: Ukazatele nebývají NULL
4. Opcode positive: Hodnoty jsou většinou kladné
5. Opcode nonequal: Hodnoty jsou většinou různé
6. loop header: Duplikované hlavičky smyček jsou většinou true

Static profile estimation

1. Odhadnutí pravděpodobností hran pomocí heuristik (Ball, Larus: Branch prediction for free 1993)
2. Sloučení násobných heuristik podle first match nebo Dempster-Shafer theory: $\frac{p_1 p_2}{p_1 p_2 + (1-p_1)(1-p_2)}$
3. Propagace profilu (Wu, Larus: Static Branch Frequency and Program Profile Analysis)

Heuristiky:

1. Loop: Smyčky se točí (a pokud vím kolikrát o to lépe)
2. Continue: Smyčky uzavřené pomocí continue mívají menší počet iterací než ostatní
3. Pointer: Ukazatele nebývají NULL
4. Opcode positive: Hodnoty jsou většinou kladné
5. Opcode nonequal: Hodnoty jsou většinou různé
6. loop header: Duplikované hlavičky smyček jsou většinou true
7. loop header: Duplikované hlavičky smyček jsou většinou true

Static profile estimation

1. Odhadnutí pravděpodobností hran pomocí heuristik (Ball, Larus: Branch prediction for free 1993)
2. Sloučení násobných heuristik podle first match nebo Dempster-Shafer theory: $\frac{p_1 p_2}{p_1 p_2 + (1-p_1)(1-p_2)}$
3. Propagace profilu (Wu, Larus: Static Branch Frequency and Program Profile Analysis)

Heuristiky:

1. Loop: Smyčky se točí (a pokud vím kolikrát o to lépe)
2. Continue: Smyčky uzavřené pomocí continue mívají menší počet iterací než ostatní
3. Pointer: Ukazatele nebývají NULL
4. Opcode positive: Hodnoty jsou většinou kladné
5. Opcode nonequal: Hodnoty jsou většinou různé
6. loop header: Duplikované hlavičky smyček jsou většinou true
7. loop header: Duplikované hlavičky smyček jsou většinou true
8. goto: podmínky rozhodující o provedení goto jsou většinou false

Static profile estimation

1. Odhadnutí pravděpodobností hran pomocí heuristik (Ball, Larus: Branch prediction for free 1993)
2. Sloučení násobných heuristik podle first match nebo Dempster-Shafer theory: $\frac{p_1 p_2}{p_1 p_2 + (1-p_1)(1-p_2)}$
3. Propagace profilu (Wu, Larus: Static Branch Frequency and Program Profile Analysis)

Heuristiky:

1. Loop: Smyčky se točí (a pokud vím kolikrát o to lépe)
2. Continue: Smyčky uzavřené pomocí continue mívají menší počet iterací než ostatní
3. Pointer: Ukazatele nebývají NULL
4. Opcode positive: Hodnoty jsou většinou kladné
5. Opcode nonequal: Hodnoty jsou většinou různé
6. loop header: Duplikované hlavičky smyček jsou většinou true
7. loop header: Duplikované hlavičky smyček jsou většinou true
8. goto: podmínky rozhodující o provedení goto jsou většinou false
9. return: Záporné hodnoty jsou často chybové stavy. NULL se většinou nevrací. Konstatní návratové hodnoty jsou méně časté než nekonstatní

Static profile estimation (SPEC2000)

HEURISTICS	BRANCHES	(REL)	HITRATE	COVERAGE	(REL)
combined	12643	100.0%	74.78%/ 92.52%	7203748247	100.0%
DS theory	8101	64.1%	80.11%/ 94.05%	3453275146	47.9%
call	4143	32.8%	82.65%/ 96.30%	2130740141	29.6%
first match	1370	10.8%	82.45%/ 90.29%	1904508787	26.4%
no prediction	3172	25.1%	56.89%/ 91.95%	1845964314	25.6%
loop branch	556	4.4%	88.11%/ 89.99%	1379795296	19.1%
opcode values positive	429	3.4%	88.34%/ 89.09%	304001983	4.2%
opcode values nonequal	3034	24.0%	66.94%/ 90.08%	811368857	11.3%
loop header	954	7.5%	79.77%/ 93.88%	410490743	5.7%
loop exit	812	6.4%	67.56%/ 91.09%	524736878	7.3%
pointer	1242	9.8%	93.06%/ 98.11%	587903278	8.2%
early return	117	0.9%	90.55%/ 94.79%	33862048	0.5%
goto	170	1.3%	79.02%/ 94.22%	19885611	0.3%
nil return	95	0.8%	96.44%/ 99.63%	37988786	0.5%
continue	94	0.7%	9.38%/ 90.67%	17406599	0.2%
negative return	194	1.5%	99.97%/ 99.99%	6983805	0.1%
const return	144	1.1%	80.19%/ 93.11%	6485456	0.1%
loop iterations	20	0.2%	98.60%/ 98.60%	12201	0.0%
noreturn call	8	0.1%	100.00%/100.00%	9	0.0%

Static profile estimation (SPEC2006)

HEURISTICS	BRANCHES	(REL)	HITRATE		COVERAGE	COVERAGE	(REL)
combined	53398	100.0%	70.31% /	80.36%	989164856862	989.16G	100.0%
first match	16607	31.1%	78.00% /	78.42%	702435244516	702.44G	71.0%
loop iterations	2689	5.0%	67.99% /	67.99%	408309517405	408.31G	41.3%
loop exit	9909	18.6%	91.80% /	92.81%	282927773783	282.93G	28.6%
DS theory	26385	49.4%	68.62% /	85.44%	146974369890	146.97G	14.9%
no prediction	10406	19.5%	33.41% /	84.76%	139755242456	139.76G	14.1%
early return (on trees)	6328	11.9%	54.20% /	86.48%	33569991740	33.57G	3.4%
opcode values positive	4266	8.0%	64.30% /	91.28%	16931889792	16.93G	1.7%
opcode values nonequal	6600	12.4%	66.23% /	80.60%	71483051282	71.48G	7.2%
continue	507	0.9%	66.66% /	82.85%	10086808016	10.09G	1.0%
call	11351	21.3%	67.16% /	92.24%	34680666103	34.68G	3.5%
pointer (on trees)	6230	11.7%	69.59% /	87.18%	16667735314	16.67G	1.7%
null return	393	0.7%	91.47% /	93.08%	3268678197	3.27G	0.3%
guess loop iv compare	178	0.3%	97.81% /	97.85%	4375086453	4.38G	0.4%
negative return	277	0.5%	97.94% /	99.23%	1062119028	1.06G	0.1%
noreturn call	2372	4.4%	100.00% /	100.00%	8356562323	8.36G	0.8%
guessed loop iterations	112	0.2%	62.06% /	64.49%	958458522	958.46M	0.1%
const return	271	0.5%	69.39% /	87.09%	301566712	301.57M	0.0%
overflow	1282	2.4%	100.00% /	100.00%	175074177	175.07M	0.0%
zero-sized array	677	1.3%	100.00% /	100.00%	112723803	112.72M	0.0%
unconditional jump	103	0.2%	100.00% /	100.00%	491001	491.00K	0.0%
fail alloc	595	1.1%	62.18% /	100.00%	595	595.00	0.0%