Topici speciale în logică și securitate I

Interacțiunea inter-domenii

Paul Irofti

Master anul II, Sem. I, 2019-2020

Accesul la memorie

Ce se întâmplă cu domeniul *Pts*?

```
int A[4][8] = {...};
int i, j;
int sum = 0;

for (i = 0; i < 4; i++)
    for (j = 0; j < 8; j++)
        sum += A[i][j];

printf("sum = %d\n", sum);
```

Accesul la memorie

Ce se întâmplă cu domeniul *Pts*?

```
int A[4][8] = {...};
int i, j;
int sum = 0;

for (i = 0; i < 4; i++)
    for (j = 0; j < 8; j++)
        sum += A + 8*sizeof(*A)*i + sizeof(*A)*j;

printf("sum = %d\n", sum);
```

Accesul la memorie

Ce se întâmplă cu domeniul *Pts*?

```
for (i = 0; i < 4; i++)
    for (j = 0; j < 8; j++)
        sum += A + 8*sizeof(*A)*i + sizeof(*A)*j;
```

Pe o arhitectură unde `int` are 64-biți

```
for (i = 0; i < 4; i++)
    for (j = 0; j < 8; j++)
        sum += A + 8*8*i + 8*j
```

Dacă A începe la *byte*-ul 0, rezultă șirul de acces la memorie:

$$\underbrace{0, 8, 16, 32, \ldots, 56}_{\texttt{A[0]}} \quad \underbrace{64, 72, \ldots, 120}_{\texttt{A[1]}} \quad \underbrace{128, 136, \ldots, 184}_{\texttt{A[2]}} \quad \underbrace{192, 200, \ldots, 248}_{\texttt{A[3]}}$$

Accesul la memorie

$$\underbrace{0, 8, 16, 32, \ldots, 56}_{\texttt{A[0]}} \quad \underbrace{64, 72, \ldots, 120}_{\texttt{A[1]}} \quad \underbrace{128, 136, \ldots, 184}_{\texttt{A[2]}} \quad \underbrace{192, 200, \ldots, 248}_{\texttt{A[3]}}$$

Putem rescrie

```
for (i = 0; i < 4; i++)
    for (j = 0; j < 8; j++)
        sum += A + 64*i + 8*j
```

în domeniul *Lin* unde ***ax*** sunt

$$\boldsymbol{a} = [64\ 8], \quad \boldsymbol{x} = [i\ j]$$

ceea ce duce la două semi-plane din domeniul *Ineq*

$$0 \leq i \leq 3, \quad 0 \leq j \leq 7 \quad , [\![64i \leq 192]\!], \quad [\![8j \leq 56]\!]$$

ce definesc o suprafață convexă în domeniul poliedrelor *Poly*.
Dar ce putem spune despre domeniul *Pts*, ce putem spune despre valoriile luate de *pointerii* ce accesează tabloul `A`?

Accesul aliniat vs. nealiniat

Modul în care declarăm tabloul

```
int A[4][8];
A[i][j] == *(A + 8*sizeof(*A)*i + sizeof(*A)*j)
```

oferă informații privind la cum este intenționat accesul la memorie: 32 de întregi așezați în 4 zone de memorie contiguă.

Dar ce se întâmplă când scriem

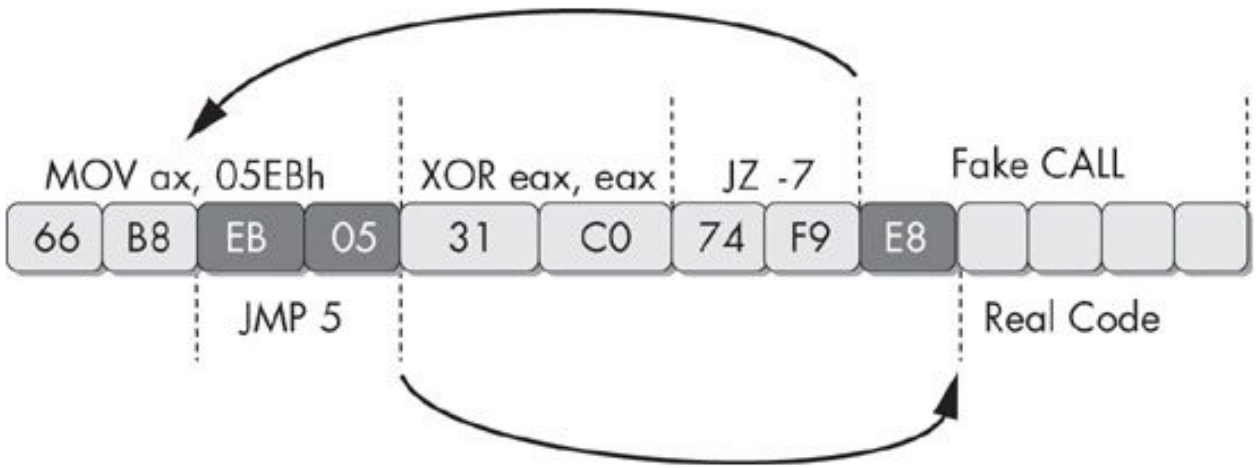
```
u = *(A + 2);
v = *(A + 111);
t = *(A + 254);
```

ce valori iau `u`, `v` și `t`?

Anumite arhitecturi constrâng hardware accesul aliniat la memorie.

Intel, AMD și ARM permit acces nealiniat. Trebuie luat în considerare întreg intervalul $[0, 255]$, nu doar setul $\{0, 8, 16, \ldots, 248\}$.

Exemplu acces nealiniat



https://stackoverflow.com/questions/30192694/jump-to-the-middle-of-an-instruction

Domeniul Granger

Domeniul *Granger* este folosit pentru analiza și deducția variabilelor ce iau valori de forma $c + m\mathbb{Z}$

$$\{\ldots, -2m + c, -m + c, c, c + m, c + 2m, \ldots\}$$

ce este suficient de generic pentru a analiza construcții de tipul

```
int i = 5;
struct {
    int a;
    char v[100];
    short f;
} a[10];
a[i].f = 0;
*(a + 13) = 2;
```

Exercițiu: Arătați cum putem modela cele două apeluri.

Domeniul de Multiplicitate

Propoziție: Pentru analiza vectorilor și tablourilor simple, ce folosesc tipuri de bază (ex. `int`, `char`), este suficient să arătăm că o variabilă este multiplu de 2^n pentru a deduce că accesul la memorie este aliniat.

Exercițiu: Demonstrați propoziția pentru exemplul anterior cu `A[4][8]`.

Definiție

Fie $Mult = \mathcal{X} \to \{0, \ldots, 64\}$ spațiul funcțional ce înregistrează numărul de biți cel mai puțin semnificativi (*least significant bits (LSB)*) ce sunt tot timpul zero, nuli.

Transformarea liniară $M \in Mult$ atribuie o valoare $n = M(x)$ tuturor variabilelor $x \in \mathcal{X}$. Presupunem că variabilele sunt reprezentate pe cel mult 64-biți.

Tip	Var.	Mult. max.
`int`	`x`	$M(x) = 63$
`int32_t`	`y`	$M(y) = 31$
`short`	`f`	$M(f) = 15$
`char`	`c`	$M(c) = 7$

Atunci, indiferent de tip, `x=0` poate fi reprezentat ca $M(x) = 64$.

Laticea $(Mult, \subseteq_M, \vee_M, \wedge_M)$

Fie $M, M', M_1, M_2 \in Mult$.

Actualizare: $M \to M' = M[x \to n'] \implies M'(x) = n' \wedge M'(y) = M(y), \forall y \neq x$.

***Join*:** $M' = M_1 \vee_M M_2$ a.î. $M'(x) = \min(M_1(x), M_2(x)), \forall x \in \mathcal{X}$.

Incluziune: $M_1 \subseteq_M M_2 \iff M_1(x) \geq M_2(x), \forall x \in \mathcal{X}$.

Exercițiu: Găsiți elementul $\top$: cel mai mare element al laticei. Motivați.

Fie $Equ = Lin \times \mathbb{Z}$ setul ecuațiilor liniare de tipul $e = c$, unde $e \in Lin, c \in \mathbb{Z}$.

***Meet*:** Operatorul de intersecție adăuga informația oferită de o nouă ecuație: $M' = M \wedge_M (e = c)$.

Definiție

$\wedge_M : Mult \times Equ \to (Mult \cup \{\bot_M\})$, unde $\bot_M$ denotă o stare nesatifăcătoare, imposibilă.

Operația $\wedge_M$

Definiție

Fie $\delta : \mathbb{Z} \to \{0, \ldots, 64\}$ a.î. $\delta(c)$ reprezintă numărul de *LSB* nefolosiți (zero) din c.

Fie $e \equiv a_1x_1 + \cdots + a_nx_n$ a.î. $a_i \neq 0, \forall i = 1, \ldots, n$. Recalculăm multiplicitatea variabilei x_j rescriind $e = c$

$$-a_jx_j = a_1x_1 + \cdots + a_{j-1}x_{j-1} + a_{j+1}x_{j+1} + \cdots + a_nx_n - c$$

Observație: Multiplicitatea lui a_ix_i este $\delta(a_i) + M(x_i)$, iar multiplicitatea lui c este pur și simplu $\delta(c)$.

Intuiție: *Mult* este similar operațiilor cu exponenți: $2^m2^n = 2^{m+n}$.

Propoziție: Operația $\wedge_M$ adaugă informație, deci numărul de *LSB* nuli din x_j nu poate descreste. Dimpotrivă, acest număr poate crește datorită apariției lui c.

#*LSB* nuli din x_j nu poate descreşte

$$-a_j x_j = a_1 x_1 + \cdots + a_{j-1} x_{j-1} + a_{j+1} x_{j+1} + \cdots + a_n x_n - c$$

Multiplicitatea părții din dreapta a ecuației trebuie să fie mai mare sau egală cu cea a fiecărui termen individual:

$$\min(\delta(c), \min_{i, i \neq j} \delta(a_i) + M(x_i))$$

Exemplu (`A[i][j]=*(A + 8*8*i + 8*j)`)

`64*i + 8*j` $\Longrightarrow \min(\delta(64) + M(i), \delta(8) + M(j)) = \min(5 + M(i), 2 + M(j))$.
Fie $i = 2, j = 4$, atunci $\min(5 + 1, 3 + 2) = 5$, iar $64i + 8j = 160_2 = 1010\ 0000$.

Dacă $a_j > 1 \vee a_j < -1$, atunci numărul din ecuația de mai sus trebuie redus cu $\delta(a_j)$ pentru a afla noul $M' = M(x_j \to n')$

$$M' = M\left[x_j \to \max\left(M(x_j), \min(\delta(c), \min_{i, i \neq j} \delta(a_i) + M(x_i)) - \delta(a_j)\right)\right]$$

Exemplu actualizare muliplicități

Fie M starea inițială a multiplicităților a trei variabile x, y, z, unde x este multiplu de 8, iar y și z sunt multiplii de 2.

Adăugăm ecuația $x + y + 2z = 0 \in \mathit{Equ}$ rezultată dintr-o problemă de aritmetică de *pointeri* domeniului M

$$M' = M \wedge_M \{x + y + 2z = 0\}$$

pe care o rezolvăm actualizând iterativ multiplicitatea fiecărei variabile.

	$M(x)$	$M(y)$	$\delta(2) + M(z)$	$\delta(0)$
M	3	1	$1+1$	64
$M'(x)$	3	1	$1+1$	64
$M'(y)$	3	2	$1+1$	64
$M'(z)$	3	2	$1+1$	64

Exercițiu: Ce se întâmplă dacă adăugăm ecuația $x + y + 2z = 1$?

Proprietăți

Propoziție: Operația $\wedge_M$ duce la starea invalidă $\perp_M$ dacă

$$\min_{i=1,\ldots,n} \delta(a_i) + M(x_i) > \delta(c)$$

Complexitate: Actualizarea unei variabile trebuie să ia în calcul toate variabile: cost pătratic. În practică avem de a face cu cel mult 2–3 variabile.

Observație: Operațiile speciale de la *Poly* se rezolvă similar:

- $M' = M \triangleright x := e \implies M(x) = M(t) = 0$ deci actualizarea termenilor din e nu aduce informație nouă
- $M \triangleright x := y \gg n \implies M(x)$ este cel puțin $(M(y) - n)$, $M(y)$ nu se schimbă

Aliniere: Putem verifica dacă accesul este aliniat prin operația $M \wedge_M \{x = 2^n\}$. Dacă rezultatul este $\perp_M$ atunci avem o eroare de acces.

Proiecție: Fie funcția $\exists_x : Mult \to Mult$ și $M' = \exists_x(M)$. Atunci $M'(x) = 0$ și $M'(y) = M(y), \forall y \neq x$.

Teoremă: $(Mult, \subseteq_M, \wedge_M, \vee_M)$ formează o latice completă.

Interacțiunea *Poly* și *Mult*

Fie $Num = (Poly \times Mult) \cup \{\bot_N\}$, unde $\bot_N$ reprezintă o stare *unreachable*, imposibil de atins, în execuția programului. Definim:

- $(P, M) \subseteq_N (P', M') \iff (P \subseteq_P P') \wedge (M \subseteq_M M')$
- $(P', M') = (P_1, M_1) \vee_N (P_2, M_2) \iff (P' = P_1 \vee_P P_2) \wedge (M' = M_1 \vee_M M_2)$
- $(P', M') = (P, M) \triangleright x := e \iff (P' = P \triangleright x := e) \wedge (M' = M \triangleright x := e)$
- $(P', M') = (P, M) \triangleright x := e \gg n \iff (P' = P \triangleright x := e \gg n) \wedge (M' = M \triangleright x := e \gg n)$
- $(P', M') = \exists_x(P, M) \iff (P' = \exists_x(P)) \wedge (M' = \exists_x(M))$
- $(P, M) \wedge_N \{e = c\} = \begin{cases} \bot_N & \text{dacă } P' = \emptyset \vee M' = \bot_M \\ (P', M') & \text{altfel} \end{cases}$, unde $P' = P \wedge_P [\![\{e = c\}]\!]$ și $M' = M \wedge_M \{e = c\}$.