

Decompilare fără goto cu DREAM

Adrian Manea

510, SLA

Articolul: No More Gotos: Decompilation Using Pattern-Independent Control Flow Structuring and Semantics-Preserving Transformations, K. Yakdan et al., NDSS 2015

Articolul: No More Gotos: Decompilation Using Pattern-Independent Control Flow Structuring and Semantics-Preserving Transformations, K. Yakdan et al., NDSS 2015

Decompilatoarele, bazate pe analiză structurală pe baza grafului de control (CFG), generează instrucțiuni goto atunci când nu găsesc o continuare așteptată în graf.

Articolul: **No More Gotos: Decompilation Using Pattern-Independent Control Flow Structuring and Semantics-Preserving Transformations**, K. Yakdan et al., NDSS 2015

Decompilatoarele, bazate pe analiză structurală pe baza grafului de control (CFG), generează instrucțiuni goto atunci când nu găsesc o continuare așteptată în graf.

DREAM elimină goto prin transformări care păstrează semantica și alte prelucrări logice pe graf.

Articolul: No More Gotos: Decompilation Using Pattern-Independent Control Flow Structuring and Semantics-Preserving Transformations, K. Yakdan et al., NDSS 2015

Decompilatoarele, bazate pe analiză structurală pe baza grafului de control (CFG), generează instrucțiuni goto atunci când nu găsesc o continuare așteptată în graf.

DREAM elimină goto prin transformări care păstrează semantica și alte prelucrări logice pe graf.

Concurența: Hex-Rays și Phoenix.

Articolul: **No More Gotos: Decompilation Using Pattern-Independent Control Flow Structuring and Semantics-Preserving Transformations**, K. Yakdan et al., NDSS 2015

Decompilatoarele, bazate pe analiză structurală pe baza grafului de control (CFG), generează instrucțiuni goto atunci când nu găsesc o continuare așteptată în graf.

DREAM elimină goto prin transformări care păstrează semantica și alte prelucrări logice pe graf.

Concurența: Hex-Rays și Phoenix.

Benchmark: GNU coreutils.

Eliminarea goto se va putea face ținând cont că:

Eliminarea goto se va putea face ținând cont că:

- Structurile de control din program au un singur punct de intrare și un singur punct succesor \Rightarrow se pot simplifica;

Eliminarea goto se va putea face ținând cont că:

- Structurile de control din program au un singur punct de intrare și un singur punct succesor \Rightarrow se pot simplifica;
- Tipul și posibilele ramificații ale structurilor de control se pot analiza folosind *logică* pe CFG.

Eliminarea goto se va putea face ținând cont că:

- Structurile de control din program au un singur punct de intrare și un singur punct succesor \Rightarrow se pot simplifica;
- Tipul și posibilele ramificații ale structurilor de control se pot analiza folosind *logică* pe CFG.

Etape, în mare:

Eliminarea goto se va putea face ținând cont că:

- Structurile de control din program au un singur punct de intrare și un singur punct succesor \Rightarrow se pot simplifica;
- Tipul și posibilele ramificații ale structurilor de control se pot analiza folosind *logică* pe CFG.

Etape, în mare:

- 1 Se alcătuiește CFG;

Eliminarea goto se va putea face ținând cont că:

- Structurile de control din program au un singur punct de intrare și un singur punct succesori \Rightarrow se pot simplifica;
- Tipul și posibilele ramificații ale structurilor de control se pot analiza folosind *logică* pe CFG.

Etape, în mare:

- 1 Se alcătuieste CFG;
- 2 Se structurează CFG folosind și transformări invariante semantic;

Eliminarea goto se va putea face ținând cont că:

- Structurile de control din program au un singur punct de intrare și un singur punct succesori \Rightarrow se pot simplifica;
- Tipul și posibilele ramificații ale structurilor de control se pot analiza folosind *logică* pe CFG.

Etape, în mare:

- 1 Se alcătuieste CFG;
- 2 Se structurează CFG folosind și transformări invariante semantic;
- 3 Se obține AST (arborele de sintaxă abstractă);

Eliminarea goto se va putea face ținând cont că:

- Structurile de control din program au un singur punct de intrare și un singur punct succesor \Rightarrow se pot simplifica;
- Tipul și posibilele ramificații ale structurilor de control se pot analiza folosind *logică* pe CFG.

Etape, în mare:

- 1 Se alcătuieste CFG;
- 2 Se structurează CFG folosind și transformări invariante semantic;
- 3 Se obține AST (arborele de sintaxă abstractă);
- 4 Se fac optimizări post-structurare (redenumiri de variabile, evidențierea funcțiilor pe șiruri de caractere, simplificarea ramificațiilor).

Etapele DREAM

- 1 *Dezasamblează* binarul folosind IDA Pro \Rightarrow CFG;

Etapele DREAM

- 1 *Dezasamblează* binarul folosind IDA Pro \Rightarrow CFG;
- 2 *Analiza fluxului de date*, inclusiv propagarea constantelor și eliminarea codului mort (inaccesibil);

Etapele DREAM

- 1 *Dezasamblează* binarul folosind IDA Pro \Rightarrow CFG;
- 2 *Analiza fluxului de date*, inclusiv propagarea constantelor și eliminarea codului mort (inaccesibil);
- 3 *Inferă tipurile variabilelor*, folosind TIE;

Etapele DREAM

- 1 *Dezasamblează* binarul folosind IDA Pro \Rightarrow CFG;
- 2 *Analiza fluxului de date*, inclusiv propagarea constantelor și eliminarea codului mort (inaccesibil);
- 3 *Inferă tipurile variabilelor*, folosind TIE;
- 4 **Structurează CFG:**

Etapele DREAM

- ① *Dezasamblează* binarul folosind IDA Pro \Rightarrow CFG;
- ② *Analiza fluxului de date*, inclusiv propagarea constantelor și eliminarea codului mort (inaccesibil);
- ③ *Inferă tipurile variabilelor*, folosind TIE;
- ④ **Structurează CFG:**
 - **Nu folosește pattern matching** (“pattern independent”);

Etapele DREAM

- ① *Dezasamblează* binarul folosind IDA Pro \Rightarrow CFG;
- ② *Analiza fluxului de date*, inclusiv propagarea constantelor și eliminarea codului mort (inaccesibil);
- ③ *Inferă tipurile variabilelor*, folosind TIE;
- ④ **Structurează CFG:**
 - **Nu folosește pattern matching** (“pattern independent”);
 - **Folosește DFS** (depth-first search);

Etapele DREAM

- ① *Dezasamblează* binarul folosind IDA Pro \Rightarrow CFG;
- ② *Analiza fluxului de date*, inclusiv propagarea constantelor și eliminarea codului mort (inaccesibil);
- ③ *Inferă tipurile variabilelor*, folosind TIE;
- ④ **Structurează CFG:**
 - **Nu folosește pattern matching** (“pattern independent”);
 - Folosește DFS (depth-first search);
 - **Tratează separat zonele cu cicluri de cele fără cicluri;**

Etapele DREAM

- ❶ *Dezasamblează* binarul folosind IDA Pro \Rightarrow CFG;
- ❷ *Analiza fluxului de date*, inclusiv propagarea constantelor și eliminarea codului mort (inaccesibil);
- ❸ *Inferă tipurile variabilelor*, folosind TIE;
- ❹ **Structurează CFG:**
 - **Nu folosește pattern matching** (“pattern independent”);
 - Folosește DFS (depth-first search);
 - Tratează separat zonele cu cicluri de cele fără cicluri;
 - **Aplică transformări care păstrează semantica;**

Etapele DREAM

- ❶ *Dezasamblează* binarul folosind IDA Pro \Rightarrow CFG;
- ❷ *Analiza fluxului de date*, inclusiv propagarea constantelor și eliminarea codului mort (inaccesibil);
- ❸ *Inferă tipurile variabilelor*, folosind TIE;
- ❹ **Structurează CFG:**
 - **Nu folosește pattern matching** (“pattern independent”);
 - Folosește DFS (depth-first search);
 - Tratează separat zonele cu cicluri de cele fără cicluri;
 - Aplică transformări care păstrează semantica;
 - **Aplică optimizări finale.**

Transferul controlului (*Reaching Condition*)

Fie $G(N, E, n_{in})$ graful de control.

Transferul controlului (*Reaching Condition*)

Fie $G(N, E, n_{in})$ graful de control.

Cu DFS între n_{src} și n_{exit} fixate obținem un subgraf aciclic $S_G(n_s, n_e)$ (“slice”) și drumurile *simple* între n_s și n_e (similar: Cifuentes).

Transferul controlului (*Reaching Condition*)

Fie $G(N, E, n_{in})$ graful de control.

Cu DFS între n_{src} și n_{exit} fixate obținem un subgraf aciclic $S_G(n_s, n_e)$ (“slice”) și drumurile *simple* între n_s și n_e (similar: Cifuentes).

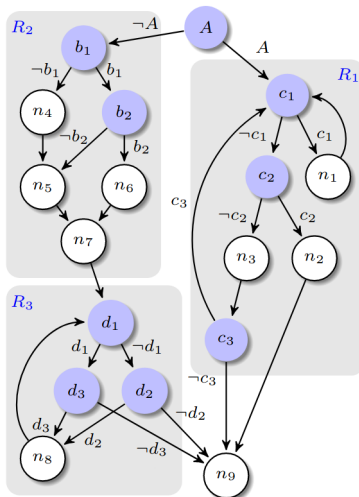
Algorithm 1: *Graph Slice*

Input : Graph $G = (N, E, h)$; source node n_s ; sink node n_e

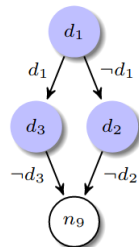
Output: $S_G(n_s, n_e)$

```
1  $S_G \leftarrow \emptyset$ ;  
2  $dfsStack \leftarrow \{n_s\}$ ;  
3 while  $E$  has unexplored edges do  
4    $e := DFSNextEdge(G)$ ;  
5    $n_t := target(e)$ ;  
6   if  $n_t$  is unvisited then  
7      $dfsStack.push(n_t)$ ;  
8     if  $n_t = n_e$  then  
9        $AddPath(S_G, dfsStack)$   
10    end  
11  else if  $n_t \in S_G \wedge n_t \notin dfsStack$   
12    then  
13       $AddPath(S_G, dfsStack)$   
14    end  
15   $RemoveVisitedNodes()$   
15 end
```

Exemplu

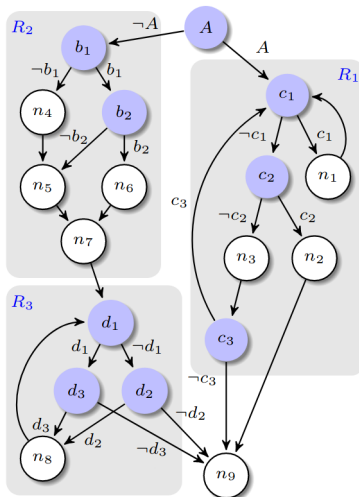


(a) CFG – exemplu de lucru

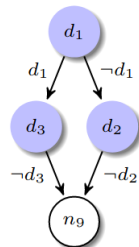


(b) $SC(d_1, n_9)$

Exemplu



(a) CFG – exemplu de lucru



(b) $SC(d_1, n_9)$

$$d_1 \rightarrow n_9 \Leftrightarrow (d_1 \wedge \neg d_3) \vee (\neg d_1 \wedge \neg d_2)$$

Ideea: Putem ordona un graf orientat aciclic inversând ordinea parcurgerii DF (ordine topologică);

Structurarea zonelor aciclice

Ideea: Putem ordona un graf orientat aciclic inversînd ordinea parcurgerii DF (ordine topologică);

Calculăm condiția de accesibilitate de la intrare la fiecare nod;

Structurarea zonelor aciclice

Ideea: Putem ordona un graf orientat aciclic inversînd ordinea parcurgerii DF (ordine topologică);

Calculăm condiția de accesibilitate de la intrare la fiecare nod;

Obținem un AST al nodurilor în ordine topologică, *ne-optim*;

Structurarea zonelor aciclice

Ideea: Putem ordona un graf orientat aciclic inversînd ordinea parcurgerii DF (ordine topologică);

Calculăm condiția de accesibilitate de la intrare la fiecare nod;

Obținem un AST al nodurilor în ordine topologică, *ne-optim*;

Rafinăm acest AST folosind logică (condiții complementare, `switch` pentru clustere de control) și eventuale `if-then-else` în cascadă.

Structurarea zonelor aciclice

Ideea: Putem ordona un graf orientat aciclic inversînd ordinea parcurgerii DF (ordine topologică);

Calculăm condiția de accesibilitate de la intrare la fiecare nod;

Obținem un AST al nodurilor în ordine topologică, *ne-optim*;

Rafinăm acest AST folosind logică (condiții complementare, switch pentru clustere de control) și eventuale if-then-else în cascadă.

Exemplu: În regiunea R_2 , avem *condițiile complementare*

if (b1 AND b2) then n6 și if ($\tilde{b}1$ OR $\tilde{b}2$) then n4 or n5.

Structurarea zonelor aciclice

Ideea: Putem ordona un graf orientat aciclic inversînd ordinea parcurgerii DF (ordine topologică);

Calculăm condiția de accesibilitate de la intrare la fiecare nod;

Obținem un AST al nodurilor în ordine topologică, *ne-optim*;

Rafinăm acest AST folosind logică (condiții complementare, switch pentru clustere de control) și eventuale if-then-else în cascadă.

Exemplu: În regiunea R_2 , avem *condițiile complementare*

if (b1 AND b2) then n6 și if ($\tilde{b}1$ OR $\tilde{b}2$) then n4 or n5.

Putem structura sub forma if (b1 AND b2) then n6 else n5.

Structurarea zonelor cu cicluri

- 1 Găsim nodurile care conduc la bucle;

Structurarea zonelor cu cicluri

- 1 Găsim nodurile care conduc la bucle;
- 2 Refacem zonele ciclice \Rightarrow o singură intrare, un singur succesor;

Structurarea zonelor cu cicluri

- 1 Găsim nodurile care conduc la bucle;
- 2 Refacem zonele ciclice \Rightarrow o singură intrare, un singur succesor;
- 3 Obținem AST pentru buclă;

Structurarea zonelor cu cicluri

- 1 Găsim nodurile care conduc la bucle;
- 2 Refacem zonele ciclice \Rightarrow o singură intrare, un singur succesor;
- 3 Obținem AST pentru buclă;
- 4 Determinăm tipul buclei și condiția de intrare prin analiza AST;

Structurarea zonelor cu cicluri

- 1 Găsim nodurile care conduc la bucle;
- 2 Refacem zonele ciclice \Rightarrow o singură intrare, un singur succesori;
- 3 Obținem AST pentru buclă;
- 4 Determinăm tipul buclei și condiția de intrare prin analiza AST;

Algorithm 2: *Loop Successor Refinement*

Input : Initial sets of loop nodes N_{loop} and successor nodes N_{succ} ; loop header n_h

Output: Refined N_{loop} and N_{succ}

```
1  $N_{new} \leftarrow N_{succ}$ ;  
2 while  $|N_{succ}| > 1 \wedge N_{new} \neq \emptyset$  do  
3    $N_{new} \leftarrow \emptyset$ ;  
4   forall the  $n \in N_{succ}$  do  
5     if  $\text{preds}(n) \subseteq N_{loop}$  then  
6        $N_{loop} \leftarrow N_{loop} \cup \{n\}$ ;  
7        $N_{succ} \leftarrow N_{succ} \setminus \{n\}$ ;  
8        $N_{new} \leftarrow N_{new} \cup$   
          $\{u : u \in [\text{succs}(n) \setminus N_{loop}] \wedge \text{dom}(n_h, u)\}$ ;  
9     end  
10  end  
11   $N_{succ} \leftarrow N_{succ} \cup N_{new}$   
12 end
```

Identificarea logică a buclelor

$$\begin{array}{c}
 \frac{n_\ell = \text{Loop}[\tau_{\text{endless}}, -, \text{Seq}[n_i]^{i \in 1..k}] \quad n_1 = \mathcal{B}_r^c}{n_\ell \rightsquigarrow \text{Loop}[\tau_{\text{while}}, \neg c, \text{Seq}[n_i]^{i \in 2..k}]} \text{ WHILE} \qquad \frac{n_\ell = \text{Loop}[\tau_{\text{endless}}, -, \text{Seq}[n_i]^{i \in 1..k}] \quad n_k = \mathcal{B}_r^c}{n_\ell \rightsquigarrow \text{Loop}[\tau_{\text{dowhile}}, \neg c, \text{Seq}[n_i]^{i \in 1..k-1}]} \text{ DOWHILE} \\
 \\
 \frac{n_\ell = \text{Loop}[\tau_{\text{endless}}, -, \text{Seq}[n_i]^{i \in 1..k}] \quad \forall i \in 1..k-1 : \mathcal{B}_r \notin \sum[n_i] \quad n_k = \text{Cond}[c, n_t, -]}{n_\ell \rightsquigarrow \text{Loop}[\tau_{\text{endless}}, -, \text{Seq}[\text{Loop}[\tau_{\text{dowhile}}, \neg c, \text{Seq}[n_i]^{i \in 1..k-1}], n_t]]} \text{ NESTEDDOWHILE} \\
 \\
 \frac{n_\ell = \text{Loop}[\tau_{\text{endless}}, -, \text{Seq}[n_i]^{i \in 1..k}] \quad n_k = n'_k \Downarrow \mathcal{B}_r}{n_\ell \rightsquigarrow \text{Seq}[n_1, \dots, n_{k-1}, n'_k]} \text{ LOOPTOSEQ} \\
 \\
 \frac{n_\ell = \text{Loop}[\tau_{\text{endless}}, -, \text{Cond}[c, n_t, n_f]] \quad \mathcal{B}_r \notin \sum[n_t] \quad \mathcal{B}_r \in \sum[n_f]}{n_\ell \rightsquigarrow \text{Loop}[\tau_{\text{endless}}, -, \text{Seq}[\text{Loop}[\tau_{\text{while}}, c, n_t], n_f]]} \text{ CONDTOSEQ} \\
 \\
 \frac{n_\ell = \text{Loop}[\tau_{\text{endless}}, -, \text{Cond}[c, n_t, n_f]] \quad \mathcal{B}_r \in \sum[n_t] \quad \mathcal{B}_r \notin \sum[n_f]}{n_\ell \rightsquigarrow \text{Loop}[\tau_{\text{endless}}, -, \text{Seq}[\text{Loop}[\tau_{\text{while}}, \neg c, n_f], n_t]]} \text{ CONDTOSEQNEG}
 \end{array}$$

Identificarea logică a buclelor

$$\begin{array}{c}
 \frac{n_\ell = \text{Loop}[\tau_{\text{endless}}, -, \text{Seq}[n_i]^{i \in 1..k}] \quad n_1 = \mathcal{B}_r^c}{n_\ell \rightsquigarrow \text{Loop}[\tau_{\text{while}}, \neg c, \text{Seq}[n_i]^{i \in 2..k}]} \text{ WHILE} \quad \frac{n_\ell = \text{Loop}[\tau_{\text{endless}}, -, \text{Seq}[n_i]^{i \in 1..k}] \quad n_k = \mathcal{B}_r^c}{n_\ell \rightsquigarrow \text{Loop}[\tau_{\text{dowhile}}, \neg c, \text{Seq}[n_i]^{i \in 1..k-1}]} \text{ DOWHILE} \\
 \\
 \frac{n_\ell = \text{Loop}[\tau_{\text{endless}}, -, \text{Seq}[n_i]^{i \in 1..k}] \quad \forall i \in 1..k-1 : \mathcal{B}_r \notin \sum[n_i] \quad n_k = \text{Cond}[c, n_t, -]}{n_\ell \rightsquigarrow \text{Loop}[\tau_{\text{endless}}, -, \text{Seq}[\text{Loop}[\tau_{\text{dowhile}}, \neg c, \text{Seq}[n_i]^{i \in 1..k-1}], n_t]]} \text{ NESTEDDOWHILE} \\
 \\
 \frac{n_\ell = \text{Loop}[\tau_{\text{endless}}, -, \text{Seq}[n_i]^{i \in 1..k}] \quad n_k = n'_k \Downarrow \mathcal{B}_r}{n_\ell \rightsquigarrow \text{Seq}[n_1, \dots, n_{k-1}, n'_k]} \text{ LOOPTOSEQ} \\
 \\
 \frac{n_\ell = \text{Loop}[\tau_{\text{endless}}, -, \text{Cond}[c, n_t, n_f]] \quad \mathcal{B}_r \notin \sum[n_t] \quad \mathcal{B}_r \in \sum[n_f]}{n_\ell \rightsquigarrow \text{Loop}[\tau_{\text{endless}}, -, \text{Seq}[\text{Loop}[\tau_{\text{while}}, c, n_t], n_f]]} \text{ CONDTOSEQ} \\
 \\
 \frac{n_\ell = \text{Loop}[\tau_{\text{endless}}, -, \text{Cond}[c, n_t, n_f]] \quad \mathcal{B}_r \in \sum[n_t] \quad \mathcal{B}_r \notin \sum[n_f]}{n_\ell \rightsquigarrow \text{Loop}[\tau_{\text{endless}}, -, \text{Seq}[\text{Loop}[\tau_{\text{while}}, \neg c, n_f], n_t]]} \text{ CONDTOSEQNEG}
 \end{array}$$

+ Transformări care *păstrează semantica* buclei prin înregistrarea punctelor de intrare.

Optimizări finale (pentru lizibilitate)

- Simplificarea structurilor de control:

Optimizări finale (pentru lizibilitate)

- Simplificarea structurilor de control:

- $\text{if } (c) \text{ then } (x = v) \text{ else } (x = w) \rightsquigarrow x = c ? v : w;$

Optimizări finale (pentru lizibilitate)

- Simplificarea structurilor de control:
 - $\text{if } (c) \text{ then } (x = v) \text{ else } (x = w) \rightsquigarrow x = c ? v : w;$
 - Se transformă **while** în **for** oricât de des este posibil;

Optimizări finale (pentru lizibilitate)

- Simplificarea structurilor de control:
 - $\text{if } (c) \text{ then } (x = v) \text{ else } (x = w) \rightsquigarrow x = c ? v : w;$
 - Se transformă while în for oricât de des este posibil;
- Identificarea funcțiilor pe șiruri de caractere:

Optimizări finale (pentru lizibilitate)

- Simplificarea structurilor de control:
 - `if (c) then (x = v) else (x = w) \rightsquigarrow x = c ? v : w;`
 - Se transformă `while` în `for` oricât de des este posibil;
- Identificarea funcțiilor pe șiruri de caractere:
 - `strcpy`, `strlen`, `strcmp` sînt înlocuite de definițiile lor de compilator, așa că DREAM le evită sau le evidențiază în mod special;

Optimizări finale (pentru lizibilitate)

- Simplificarea structurilor de control:
 - `if (c) then (x = v) else (x = w) \rightsquigarrow x = c ? v : w;`
 - Se transformă `while` în `for` oricât de des este posibil;
- Identificarea funcțiilor pe șiruri de caractere:
 - `strcpy`, `strlen`, `strcmp` sînt înlocuite de definițiile lor de compilator, așa că DREAM le evită sau le evidențiază în mod special;
- Redenumirea variabilelor, e.g. cele corespunzătoare API-urilor folosite.

Teste și comparații

Considered Functions F	$ F $	Number of goto Statements			Lines of Code			Compact Functions		
		DREAM	Phoenix	Hex-Rays	DREAM	Phoenix	Hex-Rays	DREAM	Phoenix	Hex-Rays
coreutils functions with duplicates										
$T_1 : F_p^r \cap F_h^r$	8,676	0	40	47	93k	243k	120k	81.3%	0.3%	32.1%
$T_2 : F_d \cap F_p \cap F_h$	10,983	0	4,505	3,166	196k	422k	264k	81%	0.2%	30.4%
coreutils functions without duplicates										
$T_3 : F_p^r \cap F_h^r$	785	0	31	28	15k	30k	18k	74.9%	1.1%	36.2%
$T_4 : F_d \cap F_p \cap F_h$	1,821	0	4,231	2,949	107k	164k	135k	75.2%	0.7%	31.3%
Malware Samples										
ZeusP2P	1,021	0	N/A	1,571	42k	N/A	53k	82.9%	N/A	14.5%
SpyEye	442	0	N/A	446	24k	N/A	28k	69.9%	N/A	25.7%
Cridex	167	0	N/A	144	7k	N/A	9k	84.8%	N/A	12.3%

TABLE III: Structuredness and compactness results. For the *coreutils* benchmark, we denote by F_x the set of functions decompiled by compiler x . F_x^r is the set of recompilable functions decompiled by compiler x . d represents DREAM, p represents Phoenix, and h represents Hex-Rays.