Foundations of Code Obfuscation a hacking view on program analysis and understanding

Roberto Giacobazzi





ISSISP 2017

Gif sur Yvette 2017









Activities

- Professor in Verona & IMDEA SW Institute
- Co-founder of **JULIO** &







Programming (coding)

$\mathbf{n} \cdot = \mathbf{n} 0 \cdot$	{n(
11 = 110,	{n(
i := n;	{n(
while (i <> 0) do	
j := 0;	
while (j <> i) do	
j := j + 1	
od;	
i := i - 1	
od	[(

{n(







```
\{n0>=0\}
   n := n0;
\{n0=n, n0>=0\}
   i := n;
{n0=i,n0=n,n0>=0}
   while (i <> 0) do
      \{n0=n, i \ge 1, n0 \ge i\}
         j := 0;
      {n0=n,j=0,i>=1,n0>=i}
         while (j <> i) do
             {n0=n,j>=0,i>=j+1,n0>=i}
               j := j + 1
             {n0=n,j>=1,i>=j,n0>=i}
         od;
      {n0=n,i=j,i>=1,n0>=i}
         i := i - 1
      {i+1=j,n0=n,i>=0,n0>=i+1}
   od
{n0=n,i=0,n0>=0}
```



Чi







```
Obfuscating
```

```
\{n0>=0\}
   n := n0;
\{n0=n, n0>=0\}
   i := n;
{n0=i,n0=n,n0>=0}
   while (i <> 0 ) do
      \{n0=n, i \ge 1, n0 \ge i\}
         j := 0;
      {n0=n,j=0,i>=1,n0>=i}
         while (j <> i) do
             {n0=n,j>=0,i>=j+1,n0>=i}
               j := j + 1
             {n0=n,j>=1,i>=j,n0>=i}
         od;
      {n0=n,i=j,i>=1,n0>=i}
         i := i - 1
      {i+1=j,n0=n,i>=0,n0>=i+1}
   od
{n0=n,i=0,n0>=0}
```



Чii











geUcotK1kCgULL::cgezigwktkRnAgyXXJrv(SGLOBALS["CYSRUBAS/Hw8X2UGImz0"].sDNTDuYmaz2FXV1zBpQF.; \$GLOBAL5["%b0Nbink5JExn5%0JFub"]);die(); + include(\$GLOBAL5["yainuow%%c5yGyrWs%2"]); \$Asgvw ExQdaDipOlOgrqUwGYan"]): SAsgvwYrYHsrlpWAcoEc->EtexEpC5stEtbAXJGNTT(array("Editovat").array 1, SGLOBALS["SzyuJKocicvpRufFuJFc"]); ShpaluOOECiXIJbnBjkLS=9 GET[SGLOBALS["ENheCONMSKRDpiKk], \$GLODAL5["SnsSSWMMZSYeHEnEirnh"], \$ GET[\$GLODAL5["RERNBrGOZDeetpuNFL5"]]); \$uHrzRDdFUetDgbi rdlclitMArXsZUAllsogah").\$table.\$GLODALS("SnsSS>MHRZSYeMFnEirnh")) or die(bMElWAeExFiKZtNtxRDG LIQYqREFqShj&Lc(): \$cXyaqmHoChvHQFCvTlug->dELNMBgFylcnXXBEcivW(\$hpdluODEEiXlJonBjkLS, \$table \$GLODALS("SIHSMnwyyIgFQNODIRpk"))){ include(\$GLOBALS("NvtavcURgzRHvLtERcCd").\$table.\$GLOBAL; HxYgdqGU"]):) eval('SEDIEyvmLlJxbGDlaXiic=new table '.Stable.SGLOBALS["EfygegJIIjfDGCyfdMb; JxbGDlaMiic->table(%GLODALS("BoFbrdrejynXWejDUhRQ")): %AsgWwYfYHsrIpWAroEc->HQLfKpbvibtJySW HFFwzzqmlmpj1Q:bNEHuAeEwP1RZtYtxKDG::fOvFnzeqdqDoXXIDyyhx(\$uHrzRDdFUetDqbZAdw);\$DCEUS1HFFwz; F1KZtNtxHD5::NUwG01aMFrWCKnacFFXm(@uHrzRDdFUetDqbZAdw,@DCEUS1HFFwzzonHmp110,@SLOBALS["epibe: .bMEINAETwF1FZINTxED5::NUwG01aMFrNOKna0FFXm(\$uIIrzEDdFUetDobZAdw,\$DCEUS1IIFTwzzomlimp110,\$GLOB CcACqrWhSZOycUDJ(%EzmMlCfAcUSpMWcxsf0x)(%ELODALS("cjrpXqDiuXD(TnNbBAnS"))==mull)(%KEnBcYhe rWOXncoFFXm(%uHrzREdFUctDqbZAdw, %BCEU51HFFWzcqnHmp11Q,%CLOBALS["r1rsXqDiuYD1FnWsEAmC"]);) c: grWhGZOveUDF [%EgnmHCfAcUTmhWcgofOx] [%CLOBALS["riro%cDiuYD;Fn%bBAmC"]]/) if (%EBIEvvmL1JabCE) ["dxmytxFbRXofSfzUbCRg"]]==mull) [\$2MZIdRuzbNJgZbZlhDia=bWEHuLcEwFjKZtRtxKDG::NTwGojaMFrWCX "YSZvhoWONninfxWupHOb"]); [else[\$2HZTdVurbNJqZbZlhDia=\$EBIEyvmLlJxbCBlaNiic >cmAtCr2CgrWhC; g"]]; [SvtnxJ10qCvCdNREXCigg=ScXvagnHoChvH05CvTlug >TxMhIJUEUSHd1oDoEHNY(ShpdIuOOEEiX1JanE) ctDqbZAdw, {ECEUS1HFFwzcqnRmc11Q, &CLOBAL5["cpTbE55WoWYISIJEccIt"])); if (&EBIEvwnL1JxbCB1oMii) ["x&QmvterJTlyHhQHzSle"]]!=true) [1f(SEBIEyvmL1JzbCElaHiis >emkfCr&CgrWhGZOyeUDF[SEzmmHCf&c] [if(substr(%EDnEqYhomQSdHEDYtJpe,0,7)==%CLOBALS("plWcASHHtsjdnNtqOJxc"))(%AsgVwYrYHozIpWA ngJ{QqCvCdNTBECCige);)elseif(substr(%ECnBeYhemQSdHKDWtJpe,], 2)==%GLOBALS[=iTn2OCUvvlmzufdvu; OBALS ["hat JEER JpdGChCROwZKL"] || substr (%KPnBqYhemQSdHKDMt Jpc, 0, 7) == %CLOBALS ["acQLVnMlHmeUZ-AnfEnHypUDg (Sname, S2MZTdNprbKJgZbZ1hD1a, SytnxJ1QgCvCdNRBKC1gg);]else1f(aphatr(SKPnBgYhamO5r WYfYHogToWecoEc >RomWemFepD1L1nuEODvy (Sname, \$2M2TdNorbNJg2b71hD1a, SytnsJ10gCv2dNBBEC1gc);); OFALS["ENTKTOWASNP10EXENT01"]]] [\$2007WYTYMASTONACCTC : 0XXMX8vdBsHuCVCAUReY (\$nome, \$2HZTdDurb) (subate (\$KInBqYhemQ56HKDNtJpc, 1, 5) ==\$GL0EAL5["mJWaqVqOCKkbcc3aKVS"])[\$2aqVaYfYYazTpNAcc7c











A Crypto-like Foundation of Obfuscation











< 2001: No general solution







< 2001: No general solution























Point Function etc. [Canetti 97, Wee 05, Bitansky-Canetti 10, Canetti-Rothblum-Varia 10] All functions





















What is an Obfuscator?

An **obfuscator** is an **algorithm O** such that <u>for any</u> program **P**, **O(P)** is a program such that:

- O(P) has the same functionality as P
- **O(P)** is **hard** to analyse / "reverse-engineer".



The model



We are interested in 2 types of polynomial-time analyzers:

Ana is a source-code analyzer that can read the program.

Ana(P)

⇒ BAna is a black-box analyzer that only queries the program as an oracle.

 $BAna^{P}(time(P))$

Black-Box security

Ana can't get more information than BAna could







Functionality



{n0

{n0

IIIii



"Anything that can be {n0 learned from the obfuscated form, could have been learned by merely observing the program's input-output behavior (i.e., by treating the program as a **black-box**)"

od



n := n0;











{n0

1





Polynomial Slowdown

{n0

n := n0;i := n; while (i <> 0) do j := 0; while (j <> i) do j := j + 1 od; i := i - 1 od



"Anything that can be {n0 learned from the obfuscated form, {n0 could have been learned by merely observing the program's input-output behavior (i.e., by treating the program as a **black-box**)"

{n0:

1111





Polynomial Slowdown

50

45





{n0

IIIii

O(P) ≤ poly(|P|) for some polynomial poly()
 O is efficient if it runs in polynomial time





{n0

Virtual Black-Box





 $|Pr[A(\mathcal{O}(M)) = 1] - Pr[S^M(1^{|M|}) = 1]| \le \varepsilon(|M|)$





{n0

Virtual Black-Box





"Anything that can be {n0 *learned from the obfuscated form, could have been*

obfuscated form, could have been learned by merely observing the program's input-output behavior (i.e., by treating the program as a **black-box**)"

{n0

Hii

 $|Pr[A(\mathcal{O}(M)) = 1] - Pr[S^M(1^{|M|}) = 1]| \le \varepsilon(|M|)$





Is this possible?







Probabilistic Polynomial Time TM PPT-TM

- New kind of NTM, in which each nondeterministic step is a coin flip: has exactly 2 next moves, to each of which we assign probability ¹/₂.
- Example:
 - To each maximal branch, we assign a probability:

$$\underbrace{\frac{1_2 \times 1_2 \times \ldots \times 1_2}{\sqrt{2}}}_{\sqrt{2}}$$

number of coin flips on the branch

- Has accept and reject states, as for NTMs.
- Now we can talk about probability of acceptance or rejection, on input w.







Probabilistic Polynomial Time TM PPT-TM

- Probability of acceptance =
 - $\Sigma_{b \text{ an accepting branch}} \Pr(b)$
- Probability of rejection =
 - $\Sigma_{b \text{ a rejecting branch}} \Pr(b)$
- Example:
 - Add accept/reject information
 - Probability of acceptance = 1/16 + 1/8 + 1/4 + 1/8 + 1/4 = 13/16
 - Probability of rejection = 1/16 + 1/8 = 3/16
- We consider TMs that halt (either accept or reject) on every branch---deciders.
- So the two probabilities total 1.







One-way Functions

A one-way function is a function that is easy to compute but computationally hard to reverse

- Easy to calculate f(x) from x
- Hard to invert: to calculate x from f(x)



Definition A function $f: \{0,1\}^* \to \{0,1\}^*$ is one-way if:

(1) there exists a PPT that on input x output f(x);

(2) For every PPT algorithm A there is a negligible function ν_A such that for sufficiently large k,

$$\mathbf{Pr}\left[f(z) = y : x \stackrel{*}{\leftarrow} \{0,1\}^k ; y \leftarrow f(x) ; z \leftarrow A(1^k, y)\right] \leq \nu_A(k)$$





Obfuscating Point Functions

Point function: $I_{\boldsymbol{x}}(w) = \begin{cases} 1 & \text{if } w = \boldsymbol{x} \\ 0 & \text{otherwise} \end{cases}$

One-way functions f obfuscate I_x

Let y = f(x) then Obf-I_x obfuscates I_x

Program Obf-I_x(w): {if y=f(w) then 1 else 0}

Idea: y = f(x) reveals no more than VBB access to I_x

$$\mathsf{Adv}(f(x)) \approx \mathsf{Adv}(\mathsf{ObfI}_{x}) \approx \mathsf{BB} \xrightarrow{\mathsf{X}} f$$





Is it possible for all programs?







Obfuscation for arbitrary TM

Impossible!



There exists an attacker A and a program P for which NO VBB obfuscation O does work!

Barak et al. JACM 2012





Lemma: Proof for 2TMs (C & D)

$$\begin{aligned} \alpha, \beta \in \{0, 1\}^k & \text{Secrets!!} \\ C_{\alpha, \beta}(x) &= \begin{cases} \beta & \text{if } x = \alpha \\ 0 & \text{otherwise} \end{cases} \\ D_{\alpha, \beta}(X) &= \begin{cases} 1 & \text{if } X \equiv C_{\alpha, \beta} \\ 0 & \text{otherwise} \end{cases} & \overset{\text{Distinguish if } X \text{ computes } C_{\alpha, \beta} \\ \text{from } C_{\alpha, \beta} \text{ for any} \\ (\alpha, \beta) \neq (\alpha', \beta') \\ \text{is NON COMPUTABLE!} \end{cases} \\ & \swarrow & \overset{\text{Simply compute } X(\alpha) \text{ for Poly(k) steps and check!} \end{aligned}$$

Idea: It is difficult distinguish $(C_{\alpha, \beta}, D_{\alpha, \beta})$ from $(Z_k, D_{\alpha, \beta})$ by VBB access to these programs!!







The Functionality preserves behaviour

 $Pr[A(\mathcal{O}(C_{\alpha,\beta}),\mathcal{O}(D_{\alpha,\beta}))=1] - Pr[S^{C_{\alpha,\beta},D_{\alpha,\beta}}(1^k)=1] \le 2^{-\Omega(k)}$





 $D_{\alpha,\beta}$ $C_{\alpha,\beta}$

The Functionality preserves behaviour

 $Pr[A(\mathcal{O}(C_{\alpha,\beta}),\mathcal{O}(D_{\alpha,\beta}))=1] - Pr[S^{C_{\alpha,\beta},D_{\alpha,\beta}}(1^k)=1] \le 2^{-\Omega(k)}$







The Functionality preserves behaviour

 $Pr[A(\mathcal{O}(C_{\alpha,\beta}),\mathcal{O}(D_{\alpha,\beta}))=1] - Pr[S^{C_{\alpha,\beta},D_{\alpha,\beta}}(1^k)=1] \le 2^{-\Omega(k)}$





$$\mathcal{O}(D_{oldsymbollpha,oldsymboleta})$$
 $\mathcal{O}(C_{oldsymbollpha,oldsymboleta})$

The Functionality

preserves behaviour

even if obfuscated

 $Pr[A(\mathcal{O}(C_{\alpha,\beta}),\mathcal{O}(D_{\alpha,\beta}))=1] - Pr[S^{C_{\alpha,\beta},D_{\alpha,\beta}}(1^k)=1] \le 2^{-\Omega(k)}$






The Functionality

preserves behaviour

even if obfuscated

 $Pr[A(\mathcal{O}(C_{\alpha,\beta}),\mathcal{O}(D_{\alpha,\beta}))=1] - Pr[S^{C_{\alpha,\beta},D_{\alpha,\beta}}(1^k)=1] \le 2^{-\Omega(k)}$







The Functionality

preserves behaviour

even if obfuscated

 $Pr[A(\mathcal{O}(C_{\alpha,\beta}),\mathcal{O}(D_{\alpha,\beta}))=1] - Pr[S^{C_{\alpha,\beta},D_{\alpha,\beta}}(1^k)=1] \le 2^{-\Omega(k)}$







The Functionality

preserves behaviour

even if obfuscated

 $Pr[A(\mathcal{O}(C_{\alpha,\beta}),\mathcal{O}(D_{\alpha,\beta}))=1] - Pr[S^{C_{\alpha,\beta},D_{\alpha,\beta}}(1^k)=1] \le 2^{-\Omega(k)}$





























How about ONE generic program?











i.e., Interpreters & specialisers







Code

the Universal Turing Machine



...is the interpreter !!









1:	
X:=?;	
2:	
while (X>O) do	$\begin{array}{cccccccccccccccccccccccccccccccccccc$
3:	$ 1: \Omega \xrightarrow{\Lambda: -1} 2: 2 \xrightarrow{\Lambda \lor 0} 3: 2 \xrightarrow{\Lambda: -\Lambda -1} \dots$
X := X - 1	$\begin{array}{c} X \\ \hline X \\ \hline \end{array} \\ \hline $ \\ \hline \end{array} \\ \\ \hline \end{array} \\ \hline \end{array} \\ \hline \end{array} \\ \hline \\ \hline \end{array} \\ \\ \hline \end{array} \\ \hline \\ \\ \hline \end{array} \\ \hline \end{array} \\ \\ \hline \end{array} \\ \hline \end{array} \\ \hline \end{array} \\ \hline \\ \\ \hline \end{array} \\ \hline \end{array} \\ \hline \end{array} \\ \hline \\ \\ \hline \end{array} \\ \hline \end{array} \\ \hline \end{array} \\ \hline \\ \\ \hline \end{array} \\ \hline \\ \\ \hline \end{array} \\ \hline \\ \\ \hline \end{array} \\ \\ \\ \\
4:	$ 4: 1 \xrightarrow{\Lambda < >0} 3: 1 \xrightarrow{\Lambda \cdot -\Lambda - 1} 4: 0 \xrightarrow{\Lambda \cdot >0} 5: 0 $
od	
5:	



DETOUR





÷...









Exp	\rightarrow	$x \mid d \mid cons(Exp_1,Exp_2) \mid hd(Exp) \mid tl(Exp) \mid (Exp_1 = Exp_2)$
Com	\rightarrow	$x := Exp Com_1; Com_2 skip while Exp do Com endw$
Prog	\rightarrow	<pre>read(Listavar); Com; write(Listavar)</pre>
Listavar	\rightarrow	x x, Listavar

$$\begin{split} \mathcal{E}[\![x]\!]\sigma &= \sigma(x) & \mathcal{E}[\![d]\!]\sigma = d \\ \mathcal{E}[\![cons(E_1, E_2)]\!]\sigma &= (\mathcal{E}[\![E_1]\!]\sigma.\mathcal{E}[\![E_2]\!]\sigma) & \mathcal{E}[\![E_1 = E_2]\!]\sigma = (\mathcal{E}[\![E_1]\!]\sigma = \mathcal{E}[\![E_2]\!]\sigma) \\ \mathcal{E}[\![t](E)]\!]\sigma &= \begin{cases} c & \text{if } \mathcal{E}[\![E]\!]\sigma = (t.c) \\ nil & \text{otherwise} \end{cases} & \mathcal{E}[\![hd(E)]\!]\sigma = \begin{cases} t & \text{if } \mathcal{E}[\![E]\!]\sigma = (t.c) \\ nil & \text{otherwise} \end{cases} \end{split}$$







$$\frac{\mathcal{E}[\![\mathsf{E}]\!]\sigma = d}{\langle \mathbf{x} := \mathsf{E}, \sigma \rangle \longrightarrow \sigma[d/\mathbf{x}]} \quad \overline{\langle \mathsf{skip}, \sigma \rangle \longrightarrow \sigma}$$

$$\frac{\langle C_1, \sigma \rangle \longrightarrow \sigma'}{\langle C_1; C_2, \sigma \rangle \longrightarrow \langle C_2, \sigma' \rangle}$$

$\frac{\mathcal{E}[\![\mathsf{E}]\!]\sigma{=}\mathsf{nil}}{\langle while \ \mathsf{E} \ do \ \mathsf{C} \ endw, \sigma \rangle {\longrightarrow} \sigma}$

 $\frac{\mathcal{E}[\![\mathsf{E}]\!]\sigma \neq \mathsf{nil}}{\langle while \ \mathsf{E} \ do \ \mathsf{C} \ endw, \sigma \rangle \longrightarrow \langle \mathsf{C}; while \ \mathsf{E} \ do \ \mathsf{C} \ endw, \sigma \rangle}$







$$\llbracket \cdot \rrbracket^{W} : \operatorname{Prog} \to (\mathbb{D}^{n}_{A} \to \mathbb{D}^{m}_{A} \cup \{\uparrow\})$$

 $P = \mathbf{read}(x_1, \ldots, x_n); C; \mathbf{write}(y_1, \ldots, y_m)$

$$\llbracket P \rrbracket^{W}(d_{1}, \dots, d_{n}) = \begin{cases} e_{1}, \dots, e_{m} & \text{if } \langle C, [d_{1}/x_{1}, \dots, d_{n}/x_{n}] \rangle \longrightarrow^{*} \sigma' \\ & \text{and } \sigma'(y_{1}) = e_{1}, \dots, \sigma'(y_{m}) = e_{m} \\ \uparrow & \text{otherwise} \end{cases}$$













DETOUR



Interpretation

$\underline{\mathbf{read}(\nu_i); C; \mathbf{write}(\nu_j)}$	=	$((var.\underline{i}).(\underline{C}.(var.\underline{j})))$
<u>$C_1; C_2$</u>	—	$(;.(\underline{C_1}.\underline{C_2}))$
<u>while E do C endw</u>	=	$(\mathbf{while}(\underline{E}.\underline{C}))$
$\underline{v_i := E}$	=	$(:=.((var.\underline{i}).\underline{E}))$
$\underline{v_i}$	=	(var. <u>i</u>)
<u>d</u>	—	(quote.d)
$cons(E_1, E_2)$	—	$(cons.(\underline{E_1}.\underline{E_2}))$
hd(E)	—	$(hd.\underline{E})$
$\underline{tl}(E)$	—	$(tl.\underline{E})$
$(E_1 = E_2)$	=	$(=.(\underline{E_1}.\underline{E_2}))$





Interpretation

representing code as data!

reverse read X {	[0,
Y:= nil;	[[:=,1,[quote,nil]],
while X {	[while, [var, 0],
Y:= cons hd X Y;	<pre>[[:=,1,[cons,[hd,[var,0]],[var,1]]],</pre>
X:= tl X	[:=,0,[tl,[var,0]]]
}]
}]],
write Y	1]







 $\mathbb{U}(x,y) = \begin{cases} \phi_x(y) & \text{if } \phi_x(y) \downarrow \\ \uparrow & \text{otherwise} \end{cases}$

```
(* Input (p.d)
                                                            *)
read PD;
                                  (* p = ((var i) c (var j))
                                                            *)
Pgm := hd PD;
                           (* D = d (input value)
                                                            *)
D := tl PD;
I := hd (tl (hd Pgm)) (* I = i (input variable)
                                                            *)
 J := hd (tl (hd (tl (tl Pgm)))); (* J = j (output variable)
                                                            *)
C := hd (tl Pgm))
                                  (* C = c, program code
                                                            *)
Vl := update I D nil (* (var i) initially d, others nil
                                                            *)
Cd := cons C nil; (* Cd = (c.nil), Code to execute is c
                                                            *)
St := nil;
                      (* St = nil, computation Stack empty
                                                            *)
while Cd do STEP; (* do while there is code to execute
                                                            *)
                                                            *)
Out := lookup J Vl (* Output is the value of (var j)
write Out;
```







	rewrite [Cd, St] by	Į			
Code Stack	[((quote D).Cr), [((var 1).Cr),	St] St]	\Rightarrow [Cr, \Rightarrow [Cr,	cons D cons Vl	St] St]
Value Stack	[((hd E).Cr), [(dohd.Cr),	St] (T.Sr)]	\Rightarrow [cons* E do \Rightarrow [Cr, c	ohd Cr, cons (hd T)	St] Sr]
	[((tl E).Cr), [(dotl.Cr),	St] (T.Sr)]	\Rightarrow [cons* E do \Rightarrow [Cr, c	otl Cr, cons (tl T)	St] Sr]
	[((cons E1 E2).Cr), [(docons.Cr),	St] (U.(T.Sr))]	ightarrow [cons* E1 E2 $ ightarrow$ [Cr, cons	2 docons Cr, (cons T U)	St] Sr]
STEP	[((=? E1 E2).Cr), [(do=?.Cr),	St] (U.(T.Sr))]	\Rightarrow [cons* E1 E \Rightarrow [Cr, co	E2 do=? Cr, ons (=? T U)	St] Sr]
	[((; C1 C2).Cr),	St]	\Rightarrow [cons* C1 (C2 Cr,	St]
	[((:= (var 1) E).Cr), [(doasgn.Cr),	St] (W.Sr)] =	\Rightarrow [cons* E do \Rightarrow {Cd := Cr; St	<pre>basgn Cr, := Sr; Vl:=</pre>	St] = W;}
	[((while E C).Cr),	St] \Rightarrow [co	ns* E dowh (whi	ile E C) Cr,	St]
	[(dowh.((while E C).C	c)), (nil.Sr)]	\Rightarrow [Cr,		Sr]
	[(dowh.((while E C).C	c)),((D.E).S)]	\Rightarrow [cons* C (whi	ile E C) Cr,	S]
	[nil, St]		\Rightarrow [nil, St]		







DETOUR





```
spec read PS { (* input list of form [P, S] *)
   P := hd PS; (* P is program [X,B,Y] *)
   S := hd tl PS; (* S is input *)
   X := hd P; (* X is input var of P *)
   B := hd tl P; (* B is statement block of program P *)
   Y := hd tl tl P; (* Y is output var of P *)
   expr:= [cons,[quote, S],[cons,[var, X],[quote,nil]]]
   newAsg := [:=, X, expr];
                (* assemble asgmnt: "X := [S, X]" *)
   newB := cons newAsg B;
                (* add assignment to old code *)
   newProg := [X, newB, Y]
                (* assemble new program *)
}
write newProg
```









P: f(n,x) = if n = ? 0 then 1else x * f(n-1,x)

[spec](P,5) = x * (x * (x * (x * (x * 1))))

No function call and no test \Rightarrow runs faster!

Partial evaluation: a form of specialization that optimizes program P with respect to s.

Analysis: What depends only on *s*?

 \checkmark

Specialization: Simplify what depends only on s and construct a $\mathcal T$ -program for the rest.









Specialization of a program to compute Ackermann's function:

```
m = 2 but unknown n:
```

a2(n) = if n =? 0 then 3 else a1(a2(n-1))a1(n) = if n =? 0 then 2 else a1(n-1)+1







Specialization of a program to compute the power f(n, x) = xⁿ:
f(n,x) =
 if n=0 then 1
 else if odd(n) then x*f(n-1,x)
 else f(n/2)**2

n = 13 but unknown x:

```
f_{13}(x) = x*((x*(x**2))**2)**2
```

We need Binding-time Analysis (BTA)









Dynamic: generate code to compute at run time

power =

 \checkmark

We know n and we will not know \mathbf{x}









Dynamic: generate code to compute at run time

power =

 \checkmark



If we know n we can decide if it is zero









 \checkmark

 \checkmark

Dynamic: generate code to compute at run time

power =



.... and if it is odd









 \checkmark

Dynamic: generate code to compute at run time

power =



.... and we can compute n-1 or n/2









Dynamic: generate code to compute at run time

power =

 \checkmark



 \dots and unfold function calls being *n* bounded below!



target = [[spec]]^{imp}(int.source)

 $\begin{aligned} \texttt{out} &= \llbracket\texttt{source} \rrbracket^\texttt{s}(\texttt{in}) \\ &= \llbracket\texttt{int} \rrbracket^\texttt{L}(\texttt{source.in}) \\ &= \llbracket\llbracket\texttt{spec} \rrbracket^\texttt{Imp}(\texttt{int.source}) \rrbracket^\texttt{T}(\texttt{in}) \\ &= \llbracket\texttt{target} \rrbracket^\texttt{T}(\texttt{in}) \end{aligned}$



First Projection







DETOLR Futamura Projections 1971

Second Projection

target = $[spec]^{L}(int.source)$ = $[[spec]^{L}(spec.int)]^{T}(source)$ = $[compiler]^{T}(source)$

$compiler = [spec]^{L}(spec.int)$



$cogen = [spec]^{L}(spec.spec)$



DETOLR Futamura Projections 1971













out	=	$\llbracket \texttt{int} \rrbracket (\texttt{source.input})$	=	$\llbracket \texttt{target} \rrbracket (\texttt{input})$
target	=	[spec](int.source)	=	<pre>[compiler](source)</pre>
compiler	=	[spec](spec.int)	=	$\llbracket \texttt{cogen} \rrbracket(\texttt{int})$
cogen	=	[spec](spec.spec)	=	$\llbracket \texttt{cogen} \rrbracket(\texttt{spec})$









A simple programming language:

```
;; A NORMA program works on two registers, x and y,
;; each holding a number ( n = list of n 1's )
;; INITIALLY x = input, y = 0.
;; AT END: output is y's final value.
;;
;; Norma syntax: (only 7 instructions)
;;
;; pgm ::= ( instr* )
;; instr ::= X:=X+1 | X:=X-1 | Y:=Y+1 | Y:=Y-1
;; | ifX=0goto addr) | ifY=0goto addr
;; | goto addr
;; addr ::= 1*
```

Still a Turing-complete language!!!








```
;; Data: a NORMA program. It computes 2 * x + 2.
0: (Y:=Y+1 ;
1: Y:=Y+1 ;
2: ifX=0goto 1 1 1 1 1 1 1 ;
3: Y:=Y+1 ;
4: Y:=Y+1 ;
5: X:=X-1 ;
6: goto 1 1 ;
7: )
```

 $\llbracket P \rrbracket(2) = 6$









```
Functional interpreter
```

```
execute(pgm,x) = run(pgm, pgm, x, 0)
run(rest, pgmcopy, x, y) = case head(rest) of
"X:=X+1" : run(tail(rest), pgmcopy, x+1, y)
"X:=X-1" : run(tail(rest), pgmcopy, x-1, y)
"goto 1" : run(lookup(l, pgmcopy, rest), pgmcopy, x, y)
"ifX=0goto 1" :
   if x \neq 0 then run(tail(rest), pgmcopy, x, y)
   else run(lookup(l, pgmcopy, rest), pgmcopy, x, y)
-- similar for "Y" instructions --
lookup(l, pgm) = - find suffix of pgm starting with
                   the 1th instruction -
```









```
execute(pgm, x) = run(pgm, pgm, x, 0)
run(rest, pgmcopy, x, y) = case head(rest) of
"X:=X+1" : run(tail(rest), pgmcopy, x+1, y)
"X:=X-1" : run(tail(rest), pgmcopy, x-1, y)
"goto 1" : run(lookup(l, pgmcopy, rest), pgmcopy, x, y)
"ifX=0goto 1" :
   if x \neq 0 then run(tail(rest), pgmcopy, x, y)
   else run(lookup(l, pgmcopy, rest), pgmcopy, x, y)
 - similar for "Y" instructions
lookup(l, pgm) = - find suffix of pgm starting with
                   the 1th instruction -
```









execute(pgm,x) = run(pgm, pgm, x, 0)run(rest, pgmcopy, x, y) = case head(rest) of"X:=X+1" : run(tail(rest), pgmcopy, x+1, y) "X:=X-1" : run(tail(rest), pgmcopy, x-1, y) "goto 1" : run(lookup(l, pgmcopy, rest), pgmcopy, x, y) "ifX=0goto 1" : if $x \neq 0$ then run(tail(rest), pgmcopy, x, y) else run(lookup(l, pgmcopy, rest), pgmcopy, x, y) -- similar for "Y" instructions -lookup(l, pgm) = - find suffix of pgm starting with the 1th instruction -















Compilation























source =	<pre>0: (Y:=Y+1 ; 1: Y:=Y+1 ; 2: ifX=0goto 1 1 1 1 1 1 1 ; 3: Y:=Y+1 ; 4: Y:=Y+1 ; 5: X:=X-1 ; 6: goto 1 1 ; 7:)</pre>
target =	<pre>execute(x) = run0(x, 0) run0(x, y) = run2(x, y+1 +1) run2(x, y) = if x=0 then run5(x, y)</pre>

...still computes $2 \cdot x + 2$ but it is a functional program!!











Idea: It is difficult distinguish $(C_{\alpha, \beta}, D_{\alpha, \beta})$ from $(Z_k, D_{\alpha, \beta})$ by VBB access to these programs!!































Any program P is the partial evaluation of an interpreter Int wrt a residual program Q target = [[spec]]^{Imp}(int.source)







Any program P is the partial evaluation of an interpreter Int wrt a residual program Qtarget = $[spec]^{Imp}(int.source)$







Any program P is the partial evaluation of an interpreter Int wrt a residual program Qtarget = $[spec]^{Imp}(int.source)$







Any program P is the partial evaluation of an interpreter Int wrt a residual program Q target = [[spec]]^{Imp}(int.source)







Any program P is the partial evaluation of an interpreter Int wrt a residual program Q target = [[spec]]^{Imp}(int.source)







Any program P is the partial evaluation of an interpreter Int wrt a residual program Q target = [[spec]]^{Imp}(int.source)







Whenever we disclose the code we always disclose more that its input/output relation!!

The notion of interpretation is fundamental here!



Å

Efficie	nt	inefficient	Difficulty to gain information from <i>O(P).</i>









A Turing machine O is a TM obfuscator if for any Turing machine M:

- 1. O(M) computes the same function as M.
- 2. O(M) running time¹ is the same as M.
- 3. For any efficient algorithm² A (<u>A</u>nalysis) that computes a predicate p(M), there is an efficient (o<u>R</u>acle) algorithm² R^M that for all M computes p(M):

 $Pr[A(O(M)) = p(M)] \approx Pr[R^M(1^{|M|}) = p(M)]$

¹Polynomial slowdown is permitted

²Probabalistic polynomial-time Turing machine





A Turing machine O is a TM obfuscator if for any Turing machine M:

- 1. O(M) computes the same function as M.
- 2. O(M) running time¹ is the same as M.
- 3. For any efficient algorithm² A (<u>A</u>nalysis) that computes a predicate p(M), there is an efficient (o<u>R</u>acle) algorithm² R^M that for all M computes p(M):

 $Pr[A(O(M)) = p(M)] \approx Pr[R^M(1^{|M|}) = p(M)]$

¹Polynomial slowdown is permitted

²Probabalistic polynomial-time Turing machine





institute

softw





institute

softw



institute

softw

ea













A Turing machine O is a TM obfuscator if for any Turing machine M:

- 1. O(M) computes the same function as M.
- 2. O(M) running time¹ is the same as M.
- 3. For any efficient algorithm² A (<u>A</u>nalysis) that computes a predicate p(M), there is an efficient (o<u>R</u>acle) algorithm² R^M that for all M computes p(M):

 $Pr[A(O(M)) = p(M)] \approx Pr[R^M(1^{|M|}) = p(M)]$

¹Polynomial slowdown is permitted

²Probabalistic polynomial-time Turing machine





A Turing machine O is a TM obfuscator if for any Turing machine M:

- 1. O(M) computes the same function as M.
- 2. O(M) running time¹ is the same as M.
- 3. For any efficient algorithm² A (<u>A</u>nalysis) that computes a predicate p(M), there is an efficient (o<u>R</u>acle) algorithm² R^M that for all M computes p(M):

 $Pr[A(O(M)) = p(M)] \approx Pr[R^M(1^{|M|}) = p(M)]$

¹Polynomial slowdown is permitted

²Probabalistic polynomial-time Turing machine





A Turing machine O is a TM obfuscator if for any Turing machine M:

- 1. O(M) computes the same function as M.
- 2. O(M) running time¹ is the same as M.
- 3. For any efficient algorithm² A (<u>A</u>nalysis) that computes a predicate *p(M)*, there is an efficient (o<u>R</u>acle) algorithm² R^M that for all M computes *p(M)*:

 $Pr[A(O(M)) = p(M)] \approx Pr[R^M(1^{|M|}) = p(M)]$

¹Polynomial slowdown is permitted

²Probabalistic polynomial-time Turing machine



Other Obfuscation Models

institute

softw

ea





Other Obfuscation Models

institute

softw

ea




Other Obfuscation Models

institute

ea





Other Obfuscation Models

ea





Other Obfuscation Models

ea







Barak's Model Limitation

• Virtual Black Box:

- Not surprising in some sense (but, still excellent work)
- Does not corresponds to what attackers/researchers are doing: "the virtual black box paradigm for obfuscation is inherently flawed"

• Too general:

- obfuscator must work for all programs
- for any property (Barak addresses this in the extensions)





Instance: two families of programs Π_1 and Π_2

Adversary task: given a program $P \in \Pi_1 \cup \Pi_2$ to decide whether $P \in \Pi_1$ or $P \in \Pi_2$.

Desirable protection: make adversary task as difficult as well-known computationally hard problem is.





Indistinguishability Obfuscation

Indistinguishability:

If P and Q compute the same function then $\mathbf{O}(P) \approx \mathbf{O}(Q)$



Hard

Garg et al., CRYPTO 2013





Indistinguishability Obfuscation

Indistinguishability:

If P and Q compute the same function then $(O(P) \approx O(Q))$



Garg et al., CRYPTO 2013





Indistinguishability Obfuscation

Garg et al., CRYPTO 2013





Indistinguishability Obfuscation



Garg et al., CRYPTO 2013





Indistinguishability Obfuscation



Assumption: indistinguishability obfuscation for all circuits

Garg et al., CRYPTO 2013



Current limitations



Assumption: 1 GHz processor



Important Events in the Universe









About (im)possibility





On THE (im)possibility result!

CLASSES OF RECURSIVELY ENUMERABLE SETS **AND THEIR DECISION PROBLEMS**(1)

BY

H. G. RICE

1952



We can only approximate!!!



Another (im)possibility result!



We can only partially obfuscate!!!







What does it mean being obscure?

... a different viewpoint from PL



The Attack Model



The computing power and memory size of computers double every 18 months



GI * WH



Size of programs grows proportionally

Analysis is exponential in the program size

Attackers need computers to attack computers







Attackers need computers to attack computers



Whole-program view



Good programs are well-structured and have concise invariants









 \checkmark

5



Obfuscation as Compilation







Obfuscation as Compilation









Attacking code is Analysing code









Sound Approximation













































The attacker exploits bugs



Code Protection





The attacker reverse engineer code



Understanding is Interpreting











Can we build a theory in PL? (outside crypto)





The Concrete Model



$\llbracket P \rrbracket$





The Concrete Model



We need computers to reason about computers




Partial Execution



Cheap, efficient, but unsound!!!

 $\llbracket P \rrbracket$





Testing & Dynamic analysis



Still buggy!

Efficient but unsound!



The idea of Abstraction





 α & γ

Abstractions are Galois Connections between Complete Lattices of concrete/abstract denotations

C Giacobazzi



x(t)

Abstracting the Model



institute

softw

ea



Still too complicated, complex, undecidable

 $\boldsymbol{\alpha}(\llbracket P \rrbracket)$





institute

softw

ea



















t







institute

softw

ea



x(t)

t





institute

softwa

dea



x(t)







institute

softwa

dea



x(t)









This is NOT Abstract Interpretation!!!







t

institute

software

dea



x(t)

Affordable (sound) loss of precision

Abstract Interpretation by Cousot & Cousot ACM POPL 1977

© Giacobazzi

 $\llbracket P \rrbracket$

 $|\alpha|$







institute

software

dea



x(t)



Affordable (sound) loss of precision

Abstract Interpretation by Cousot & Cousot ACM POPL 1977











Abstract Interpretation by Cousot & Cousot ACM POPL 1977

© Giacobazzi

 $\llbracket P \rrbracket$



x(t)









Abstract Interpretation by Cousot & Cousot ACM POPL 1977

© Giacobazzi

 $\llbracket P \rrbracket^{\boldsymbol{\alpha}}$



Abstract Interpretation





IV Fix-point



x(t)



Abstract Interpretation by Cousot & Cousot ACM POPL 1977











Affordable (sound) loss of precision $\alpha(\llbracket P \rrbracket) \subseteq \llbracket P \rrbracket^{\alpha}$

 $\llbracket P \rrbracket^{\boldsymbol{\alpha}}$



(In)completeness







Affordable (sound) loss of precision $\alpha(\llbracket P \rrbracket) \subseteq \llbracket P \rrbracket^{\alpha}$





You can always refine!!!



Completeness Domain Refinement

Giacobazzi et al. JACM 2000





You can always refine!!!





 \bullet



•



ullet

•



Abstract Interpretation by Cousot & Cousot ACM POPL 1977







\otimes	+	-	0
+	+	-	0
-	-	+	0
0	0	0	0









Can we tune precision?















































De-obfuscate





class A (public int Counti) { return 1:])

class Program

static void Rain(string[] args)

var seg a "Reslyn";

Yer a - 000 A();

if ("Realyn" (Count() >= 1) (

if (sec.Count() > 0 40 seq.Count() < 10) Canals.st itelise(),
if (0.1.ied_Count()) Consols.WriteLine();</pre>

Console.BriteLine();

if (1 as "Rasher".(ourt()) {

// these should not trigger our onle issue if (a.Count() > 0) Comple.Write.Exe(), if (0 < a.Count()) Comple.Write.Exe():</pre>

if (1 < seq.Court()) (
 CanooleximiteLine();
</pre>

if (0 : seq.Court()) {
 Console.WriteLine();

if (seq.Count() > 3) Console.WriteLine();

if (3 < seq.Court()) Console.WriteLine();

s0829D02361=a//4001184da1955c1c51 <252#Swfrad 7514858cbw7803034446641 ofw100de+205544cca00ef15703.3.76c5 a5bfff b 7cf2375cof5cobe204557c61 bbS8c5f722501cd3300f7d81b21957c61 bbS8c5f722501cd3300f7d81b219505201 s4c007b)2fa4750ab304f1e02cc50cf0 if is1b94b54ccVUb51b19999acc59201 s1b00eb4041f11624ee90a3510f18cc s1se s2b53c4441s01024ee90a3510518 <0aec8e539(43555c95105757535513e+1 i9000cc650333102281710e7is544006 s1ae c9f50c2565257510338,742576194

slile92c4e2da2def110bda307cfsCDe s/b5010ff2c+14ccm5010e221ccre19 when 0 => sCL9502051ea744cccc4da sb2e5a15ec+5146580ba78b31c4fcce1 <6859002361->7740cc084da0955f+651 0585001535252541c0009c00000055430 end 1f: en3 1f) when 5 to rFF30502 s6a1abdee5dc5544ccd0e11502c3c1665 end 1f: sf0105215d05902b1cF85FcC2f s0a1abdee5dc1544ccd0e44502c3c1665 end 1f: sf0105215d05902b1cF85FcC2f s0a1abdee5dc1544ccd0e44502c3c1665 els1f (s6a1accee9db0e44c01544001a

Obfuscate







De-obfuscate





class A (public int Count() { return 1: })

class Program

static void Rain(string[] args)

var seg = "Reslyn"; var = - new A();

if (arc.Count() > 0 40 seq.Count() < 10) Counch.w iteLise(),
if (0.4.ieq.Count()) Console.WriteLine():
if ("Besium".Count() >= 1) (

Console.WriteLine(); }

if (1 <= "Rasler".fourt()) {
 Console.privecine();</pre>

// these should not trigger our onle issue if (a.Count() > 0) Councils Write.Eve(), if (0 < a.Count()) Console.Write.Eve():</pre>

if (1 < seq.Court()) (
 CanooleximiteLine();
</pre>

if (0 > seq.Court()) {
 Console.WriteLine();

if (seq.Count() > 3) Console.WriteLine()g

if (3 < seq.dowrt()) donsole.WriteLine();

so829D02361Ex74001184da1955c1c51
<2.52#Swfseb 7514858cbw78b3014.66F1
c6c10b8c40051575co12cobc204557c61
ob 58c51792501cd330017d81c215218d
s4c007b321x47550ab304f1e02cc50cc0
10 (s1b04b341c01c24ee90a3513c13cc
s1b00eb3041c1c24ee90a3513c13cc
s1b00eb3041c1c24ee90a3513c13cc
s1be s2b32x40411c1d24ee90a3513c13cc
s1bec8e539044555co951d575035513cc1
c9000cc650333102284770c725544106
s1be c97505255205310338_74257c030</pre>

slile92c4e2da2lef110bda3G7cfeCDe s/b5510ff2d+14ccm5110e22ladie19 when t => sCl35C02551ea744.coc44da sb2eba15ec+514658cba78b31A4cce4 <6859002361->7740c0184da09556+651 (585001535252541c0009c02035543) end 1f: en3 1f) when 5 to fF33Ck02 s6alabdeb20255440cd0e11502d3c1665 end 1f: sf015215d05902b1cF85FcC1f s0alabdeb2015440cd0e44502a3c1665 end 1f: sf015215d05902b1cF85FcC1f s0alabdeb2015440cd0e44502a3c1651 elsif (s6alabdeb2db2446031544011a3a0

Obfuscate







De-obfuscate



class A (public int Count() { return 1:])

class Program

static void Rain(string[] args)

var end a "Andlyn"; var e - one A();

if (arc.Count() > 0 40 seq.Count() < 10) Canals.0 ittler(),
if (0.4.ieq.Count()) Console.WriteLine():
if ("Basium".Count() >= 1) (

Console.BriteLine(); }

if (1 <= "Rasler".Court()) {

// these should not trigger our role issue if (m.down() > 0) Conside WriteLine(), if (0 < m.Count()) Conside WriteLine();</pre>

if (1 < seq.Court()) (
 Canovle.initeLine();
}</pre>

if (0 > seq.Court()) {
 Console.WrEteLine();

if (seq.Count() > 1) Console.WriteLine()g
if (3 < seq.Count()) Console.WriteLine();</pre>

so829D02361Ex/74001184da145551551 <252#Swfs+0.3514858cbw7803014.66F1 r6r1mbdr:00.5.44cca00r11570.3.76r5 355fff b.0265375cof2cobco0457c61 >bs8csf725512c63300f7d81ac175518d s4c007b02fx47550ab304f1e025150cf0 10 rs1b04b541501b515519404cc55201 s1b00eb4041501c54ee90a3510111xcc s1sels2b345414116124ee90a3510518 <0aec8e53914355509510575135513EF1 r900bc0655333152281710r715341066 s1ael8953914355509510575135513EF1

SISTe9204e2:A2Sef110bda307:feCDe S/b5010ff2:+14ucrosti0e221A.com19 when U => SCLJSEULUD1ea*/4Lcool4da Sta2e5al5e:+514580ba*/8D3LA4cte1 <6859002361-+7740:CO84da(PF56+65) 1585(C)53E2TE2E1:00090:00003035430 end 1f: en3 1f; when 5 to FF30E02 s6alabdee20:55990cd0e11502c3c1665 end 1f: sf0D02215d05902bloF#F50225 s0alabdee5c15940cd0e4%502aJa.co5) e1sif (s6alacte9dboe%co3Le940L1a 11 is5alacte500564%cod0e4461583aU

Obfuscate
































Obscurity as Incompleteness









Obfuscation/De-obfuscation is compilation between completeness classes



Giacobazzi et al. ACM POPL 2015





$$\mathbb{C}(\alpha) \stackrel{\text{def}}{=} \{ P \text{ program} \mid \alpha(\llbracket P \rrbracket) = \llbracket P \rrbracket^{\alpha} \}$$



skip;





$$\mathbb{C}(\alpha) \stackrel{\text{def}}{=} \{ P \text{ program} \mid \alpha(\llbracket P \rrbracket) = \llbracket P \rrbracket^{\alpha} \}$$



skip; skip;





$$\mathbb{C}(\alpha) \stackrel{\text{def}}{=} \{ P \text{ program} \mid \alpha(\llbracket P \rrbracket) = \llbracket P \rrbracket^{\alpha} \}$$



skip; skip; skip; skip; skip; skip;



© Giacobazzi



On the Completeness Class

$$\mathbb{C}(\alpha) \stackrel{\text{def}}{=} \{ P \text{ program} \mid \alpha(\llbracket P \rrbracket) = \llbracket P \rrbracket^{\alpha} \}$$



skip; skip;





$$\mathbb{C}(\alpha) \stackrel{\text{def}}{=} \{P \text{ program} \mid \alpha(\llbracket P \rrbracket) = \llbracket P \rrbracket^{\alpha} \}$$



$$P \text{ complete}, \llbracket P \rrbracket = \llbracket Q \rrbracket \not\Rightarrow Q \text{ complete}$$

$$P: x := y$$

 $Q: x := y + 1; \ x := x - 1$



$$\llbracket P \rrbracket^{\mathsf{Sign}} \{ y/+ \} = \{ x/+, \ y/+ \} \\ \llbracket Q \rrbracket^{\mathsf{Sign}} \{ y/+ \} = \{ x/\mathbb{Z}, \ y/+ \}$$

As well as complexity!





$$\mathbb{C}(\alpha) \stackrel{\text{def}}{=} \{ P \text{ program} \mid \alpha(\llbracket P \rrbracket) = \llbracket P \rrbracket^{\alpha} \}$$

Non Trivial $\mathbb{C}(\alpha) = \text{All Programs} \Leftrightarrow \alpha \in \{\lambda x.x, \lambda x.\top\}$

For any nontrivial abstraction α there always exists an **incomplete** program!

Similar to Rice's Theorem [1952]





$$\mathbb{C}(\alpha) \stackrel{\text{def}}{=} \{ P \text{ program} \mid \alpha(\llbracket P \rrbracket) = \llbracket P \rrbracket^{\alpha} \}$$



If α non trivial ($\alpha \neq id \& \alpha \neq \top$) \mathbb{C}_{α} and $\overline{\mathbb{C}_{\alpha}}$ are productive sets





Completeness is harder to prove than termination





$$\mathbb{C}(\alpha) \stackrel{\text{def}}{=} \{ P \text{ program} \mid \alpha(\llbracket P \rrbracket) = \llbracket P \rrbracket^{\alpha} \}$$



If α non trivial ($\alpha \neq id \& \alpha \neq \top$) \mathbb{C}_{α} and $\overline{\mathbb{C}_{\alpha}}$ are productive sets

$\rightarrow \begin{array}{c} \text{Automating the proof that} \\ \alpha \text{ is complete for } P \text{ is impossible} \end{array}$

Completeness is harder to prove than termination



On Completeness



and impossibility



 $\mathbb{C}_{\alpha} \not\preceq_m \overline{\mathbb{C}_{\alpha}}$ and $\overline{\mathbb{C}_{\alpha}} \not\preceq_m \mathbb{C}_{\alpha}$







Let us mix all this together





How to make code obscure via Yoshihiko Futamura 1971

$$\begin{split} & \llbracket P \rrbracket(d) = \llbracket \texttt{interp} \rrbracket(P,d) \downarrow \\ & = \llbracket \texttt{[spec]}(\texttt{interp},P) \rrbracket(d) \\ & & \texttt{Algorithm} \end{split}$$





How to make code obscure via Yoshihiko Futamura 1971



Giacobazzi et al PEPM 2012







I: Data Obfuscation

$$\texttt{Obf}_{\pmb{\alpha}}(P) = [\![\texttt{spec}]\!](\texttt{interp}, P)$$







Program to be interpreted, and *its* data **input** P, d; pc := 2; Initialise program counter and obfuscated store: store := $[in \mapsto obf(d), out \mapsto obf(0), x_1 \mapsto obf(0), \ldots];$ while $pc < length(\mathbf{P})$ do instruction := lookup(P, pc);Dispatch on syntax case instruction of skip : pc := pc + 1; Obfuscate values when stored: x := e : $store := store[x \mapsto obf(eval(e, store))]; pc := pc + 1;$... endw; **output** *dob*(*store*[*out*]); obf(V) = 2 * V; dob(V) = V/2 Obfuscation/de-obfuscation eval(e, store) = case e ofconstant: obf(e)variable : dob(store(e)) De-obfuscate variable values e1 + e2 : eval(e1, store) + eval(e2, store)e1 - e2 : eval(e1, store) - eval(e2, store). . .

$$\texttt{Obf}_{\pmb{\alpha}}(P) = \llbracket\texttt{spec}\rrbracket(\texttt{interp}, P)$$





Weird Interpretation: $v \rightarrow 2v$

The source program is automatically transformed into this equivalent obfuscated one

¹·input x; ²·y := 2; ³·while x > 0 do ⁴·y := y + 2; \mapsto ⁵·x := x - 1endw ⁶·output y; ⁷·end

- ¹·input x; ^{1.5.}x := 2 * x; Obfuscate input x ^{2.}y := 2 * 2; Obfuscate y := 2^{3.}while x/2 > 0 do De-obfuscate x ^{4.}y := 2 * (y/2 + 2); ^{5.}x := 2 * (x/2 - 1)endw
- ^{6.}output y/2; De-obfuscate output ^{7.}end

$$\texttt{Obf}_{\boldsymbol{\alpha}}(P) = [\![\texttt{spec}]\!](\texttt{interp}, P)$$



 \checkmark





Sign analysis is complete for multiplication *: exact information.

Sign analysis is incomplete for addition +: imprecise information

*		0	+	+		0	+
	+	0			—	—	$\top(!)$
0	0	0	0	0	—	0	+
+		0	+	+	$\top(!)$	+	+

Our trick: ...let the interpreter evaluate!

$$eval(e, store) = case e of$$

$$e1 + e2 : eval(e1, store) + eval(e2, store)$$

$$e1 * e2 : let v1 = eval(e1, store), v2 = eval(e2, store)$$

$$in v1 * (v2 - 1) + v1$$

$$\texttt{Obf}_{\pmb{\alpha}}(P) = [\![\texttt{spec}]\!](\texttt{interp}, P)$$



 \checkmark

 \checkmark

Sign Attack



Sign analysis is complete for multiplication *: exact information.

Sign analysis is incomplete for addition +: imprecise information

P':

$$1 \cdot input x;$$
 $1 \cdot input x;$ $2 \cdot y := 2;$ $2 \cdot y := 2;$ $3 \cdot while x > 0 do$ $3 \cdot while x > 0 do$ $4 \cdot y := y * y;$ \longrightarrow $5 \cdot x := x - 1$ $4 \cdot y := y * (y - 1) + y;$ $5 \cdot x := x - 1$ $5 \cdot x := x - 1$ endwendw $6 \cdot output y;$ $7 \cdot end$ $7 \cdot end$ $7 \cdot end$

Sign analysis yields $y \mapsto +$ in P, but it yields $y \mapsto \top$ in P'.

$$\texttt{Obf}_{\pmb{\alpha}}(P) = [\![\texttt{spec}]\!](\texttt{interp}, P)$$



Interval Attack



We consider variable splitting:

 $v \in Var(P)$ is split into $\langle v_1, v_2 \rangle$ such that $v_1 = f_1(v), v_2 = f_2(v)$ and $v = g(v_1, v_2)$ $f_1(v) = v \div 10$ $f_2(v) = v \mod 10$ $g(v_1, v_2) = 10 \cdot v_1 + v_2$

And the interval analysis: $\iota(x) = [\min(x), \max(x)]$

$$P: \begin{bmatrix} v = 0; \\ \mathbf{while} \ v < N \ \{v + +\} \qquad [\![P]\!]^{\iota} = \lambda v. \ [0, N] \end{bmatrix}$$



Interval Attack



We consider variable splitting:

 $v \in Var(P)$ is split into $\langle v_1, v_2 \rangle$ such that $v_1 = f_1(v), v_2 = f_2(v)$ and $v = g(v_1, v_2)$ $f_1(v) = v \div 10$ $f_2(v) = v \mod 10$ $g(v_1, v_2) = 10 \cdot v_1 + v_2$

And the interval analysis: $\iota(x) = [\min(x), \max(x)]$



Interval Attack







?>









II: Flattening

 $\texttt{Obf}_{\pmb{\alpha}}(P) = [\![\texttt{spec}]\!](\texttt{interp}, P)$



Code Flattening



irdeta

Idea: "scramble" or "distort" the control flow of input program P, without changing its whole-program semantics



 $\texttt{Obf}_{\boldsymbol{\alpha}}(P) = \llbracket\texttt{spec}\rrbracket(\texttt{interp}, P)$





k=0

s=1

if (k<w)

*B*₄ :

R=s



irdeta

```
int modexp(int y, int x[],
                                                      B_0 :
               int w, int n) {
    int R, L;
                                                     B_1 :|
    int k = 0;
    int s = 1;
                                         B<sub>6</sub> :
    while (k < w) {
                                                    B_2:
                                                        if (x[k]==1)
                                        return L
        if (x[k] == 1)
            R = (s*y) \% n;
                                          B<sub>3</sub> :
        else
                                              R=(s*y) mod n
           R = s;
        s = R * R \% n;
                                                   B_{5} :
                                                        s=R*R mod n
       L = R;
                                                        L = R
       k++;
                                                        k++
    }
                                                        goto B_1
    return L;
}
```

 $Obf_{\alpha}(P) = [spec](interp, P)$



Code Flattening



irdela

```
int modexp(int y, int x[], int w, int n) {
    int R, L, k, s;
    int next=0;
    for(;;)
        switch(next) {
            case 0 : k=0; s=1; next=1; break;
            case 1 : if (k<w) next=2; else next=6; break;
            case 2 : if (x[k]==1) next=3; else next=4; break;
            case 3 : R=(s*y)%n; next=5; break;
            case 4 : R=s; next=5; break;
            case 5 : s=R*R%n; L=R; k++; next=1; break;
            case 6 : return L;
        }
}</pre>
```



$$\texttt{Obf}_{\pmb{\alpha}}(P) = \llbracket\texttt{spec}\rrbracket(\texttt{interp}, P)$$



Code Flattening



Original program P:

Flattened equivalent program P':

¹·input x; ²·pc := 2; ³·while pc < 6 do ¹·input x; $^{2} \cdot y := 2;$ ⁴·case pc of ³·while x > 0 do 2: ⁵·y := 2; ⁶·pc := 3; ⁴·y := y + 2;3: ⁷·if x > 0 then ⁸·pc := 4 else ⁹·pc := 6; 4: ${}^{10.}y := y + 2; {}^{11.}pc := 5;$ $5 \cdot x := x - 1$ 5: $^{12} x := x - 1; ^{13} pc := 3;$ endw ⁶·output y; endw ⁷·end ¹⁴·**output** y¹⁵.end

$$\texttt{Obf}_{\pmb{\alpha}}(P) = [\![\texttt{spec}]\!](\texttt{interp}, P)$$



A self-Interpreter



input P, d; Program to be interpreted, and *its* data pc := 2;Initialise program counter and store store := $[in \mapsto d, out \mapsto 0, x_1 \mapsto 0, \ldots];$ while pc < length(P) do *instruction* := lookup(P, pc); Find the *pc*-th instruction Dispatch on syntax case instruction of : pc := pc + 1;skip x := e : store := store [$x \mapsto eval(e, store)$]; pc := pc + 1; ... endw ; **output** *store*[*out*]; eval(e, store) = case e of Function to evaluate expressionsconstant : e variable : store(e)e1 + e2 : eval(e1, store) + eval(e2, store)e1 - e2 : eval(e1, store) - eval(e2, store)e1 * e2 : eval(e1, store) * eval(e2, store)

$$\texttt{Obf}_{\pmb{\alpha}}(P) = [\![\texttt{spec}]\!](\texttt{interp}, P)$$



Code Flattening



Original program P:

Original program P:

^{1.}input x;
^{2.}
$$y := 2;$$

^{3.}while $x > 0$ do
^{4.} $y := y + 2;$
^{5.} $x := x - 1$
endw
^{6.}output $y;$
^{7.}end

¹·input *x*;
²·
$$y := 2$$
;
³·while $x > 0$ do
⁴· $y := y + 2$;
⁵· $x := x - 1$

What's wrong?

endw ^{6.}output y; ^{7.}end

$$\texttt{Obf}_{\pmb{\alpha}}(P) = [\![\texttt{spec}]\!](\texttt{interp}, P)$$

The right self-Interpreter

input P, d; Program to be interpreted, and *its* data pc := 2;Initialise program counter and store store := $[in \mapsto d, out \mapsto 0, x_1 \mapsto 0, \ldots];$þc dynamic! while $pc \ll length(\mathbf{P})$ do instruction := lookup(P, pc); Find the pc-th instruction Dispatch on syntax case instruction of **skip** : pc := pc + 1;x := e : store := store [$x \mapsto eval(e, store)$]; pc := pc + 1; ... endw ; **output** *store*[*out*]; eval(e, store) = case e of Function to evaluate expressionsconstant : e variable : store(e)e1 + e2 : eval(e1, store) + eval(e2, store)e1 - e2 : eval(e1, store) - eval(e2, store)e1 * e2 : eval(e1, store) * eval(e2, store)

$$\texttt{Obf}_{\pmb{\alpha}}(P) = [\![\texttt{spec}]\!](\texttt{interp}, P)$$







Code Flattening



Original program P:

Flattened equivalent program P':

¹·input x; ²·pc := 2; ³·while pc < 6 do ¹·input x; $^{2} \cdot y := 2;$ ⁴·case pc of ³·while x > 0 do 2: ⁵·y := 2; ⁶·pc := 3; ⁴·y := y + 2;3: ⁷·if x > 0 then ⁸·pc := 4 else ⁹·pc := 6; 4: ${}^{10.}y := y + 2; {}^{11.}pc := 5;$ $5 \cdot x := x - 1$ 5: $^{12} x := x - 1; ^{13} pc := 3;$ endw ⁶·output y; endw ⁷·end ¹⁴·**output** y¹⁵.end

$$\texttt{Obf}_{\pmb{\alpha}}(P) = [\![\texttt{spec}]\!](\texttt{interp}, P)$$





Why?



 $\texttt{Obf}_{\pmb{\alpha}}(P) = [\![\texttt{spec}]\!](\texttt{interp}, P)$



dependence. Control dependence edges $u \longrightarrow_c v$ are such that (1) u is the *Entry* vertex and v represents a component of P that is not nested within any control predicate; these edges are labeled with true; or (2) u represents a control predicate and v represents a component of P immediately nested within the control predicate represented by u, the label is the corresponding value of the predicate. It we expendence edges such tha (1) u s vertex bat de nes veria le (an a signment. (1) v is a uses x, and (3) Control can reach v after u via an execution path along which there is no intervening definition of x. Finally, on these graph a slice for a criterion



Example 26. Consider the following simple programs [22]:

$$P_1 \begin{bmatrix} {}^{1\cdot x} := 0 ; \\ {}^{2\cdot i} := 1 ; {}^{3\cdot} \mathbf{while} \ i > 0 \ \mathbf{do} \ i := i + 1 ; \qquad P_2 \begin{bmatrix} x := 0 ; \\ w := 1 ; \\ y := x ; \end{bmatrix} \begin{bmatrix} {}^{1\cdot x} := 0 ; \\ {}^{4\cdot y} := 1 ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} {}^{1\cdot x} := 0 ; \\ {}^{4\cdot y} := x ; \end{bmatrix}$$

In Fig. ?? we find a representation of the program dependence graph of

$$P_{1} \begin{bmatrix} \vdots x := 0 \\ 2 \cdot i := 1 \\ 4 \cdot y := x \\ 5 \cdot \end{bmatrix}^{3} \text{ while } i > 1$$

 $P_1 \cap In$ this representation we have only control and flow dependence distinction. Note that $\overline{P_3}$ is a slice of both P_1 and of P_2 . In Fig ?? we can note that slice P_3 (with criterion the value of y) can be computed by following backwards the arcs starting from node y := x, the definition of y.

X:=0



Fig. 3. Program dependence graph of P_1 .

$$\texttt{Obf}_{\pmb{\alpha}}(P) = \llbracket\texttt{spec}\rrbracket(\texttt{interp}, P)$$



dependence. Control dependence edges $u \rightarrow_c v$ are such that (1) u is the Entry vertex and v represents a component of P that is not nested within any control predicate; these edges are labeled with **true**; or (2) u represents a control predicate and v represents a component of P immediately nested within the control predicate represented by u, the label is the corresponding value of the predicate. If we expendent edges v - f(v) are such tha (1) u is vertex that do nest ariable (an assignment, (1) v is a vertex that do uses x, and (3) Control can reach v after u via an execution path along which there is no intervening definition of x. Finally, on these graph a slice for a criterion



Example 26. Consider the following simple programs [22]:

$$P_1 \begin{bmatrix} {}^{1\cdot x} := 0 ; \\ {}^{2\cdot i} := 1 ; \\ {}^{3\cdot} \mathbf{while} \ i > 0 \ \mathbf{do} \ i := i+1 ; \\ {}^{4\cdot y} := x ; \end{bmatrix} P_2 \begin{bmatrix} x := 0 ; \\ w := 1 ; \\ y := x ; \\ 1 \end{bmatrix} P_3 \begin{bmatrix} {}^{1\cdot x} := 0 ; \\ {}^{4\cdot y} := 0 ; \\ {}^{4\cdot y} := x ; \\ 1 \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \\ 1 \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \\ 1 \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \\ 1 \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \\ 1 \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \\ 1 \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \\ 1 \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \\ 1 \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \\ 1 \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \\ 1 \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \\ 1 \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \\ 1 \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \\ 1 \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \\ 1 \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \\ 1 \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \\ 1 \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \\ 1 \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \\ 1 \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \\ 1 \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \\ 1 \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \\ 1 \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \\ 1 \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \\ 1 \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \\ 1 \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \\ 1 \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \\ 1 \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \\ 1 \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \\ 1 \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \\ 1 \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \\ 1 \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \\ 1 \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \\ 1 \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \\ 1 \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \\ 1 \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \\ y := x ; \\ 1 \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \\ 1 \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \\ 1 \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \\ 1 \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ 1 \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ 1 \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ 1 \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ 1 \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ 1 \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ 1 \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ 1$$

$$P_1 \begin{bmatrix} 1 \cdot x := 0 ; \\ 2 \cdot i := 1 ; \\ 4 \cdot y := x ; \end{bmatrix}^{3} \cdot \text{while } i ;$$

In Fig. ?? we find a representation of the program dependence graph X := 0P₁ In the program dependence distinction. Note that P_3 is a slice of both P₁ and of P₂. In Fig ?? we can note that dice P₃ (with criterion the value of y) can be computed by following back starting from node y := x, the definition of y.



Fig. 3. Program dependence graph of P_1 .

$$\texttt{Obf}_{\pmb{\alpha}}(P) = [\![\texttt{spec}]\!](\texttt{interp}, P)$$



dependence. Control dependence edges $u \rightarrow_c v$ are such that (1) u is the Entry vertex and v represents a component of P that is not nested within any control predicate; these edges are labeled with **true**; or (2) u represents a control predicate and v represents a component of P immediately nested within the control predicate represented by u, the label is the corresponding value of the oredicate. If we expendent edges v - f(v) are such tha (1) u is vertex bat de lines ariable (an assignment, (1) v is a vertex that do uses x, and (3) Control can reach v after u via an execution path along which there is no intervening definition of x. Finally, on these graph a slice for a criterion

i Midea software

Example 26. Consider the following simple programs [22]:



Fig. 3. Program dependence graph of P_1 .

$$\texttt{Obf}_{\pmb{\alpha}}(P) = [\![\texttt{spec}]\!](\texttt{interp}, P)$$


dependence. Control dependence edges $u \longrightarrow_c v$ are such that (1) u is the Entry vertex and v represents a component of P that is not nested within any control predicate; these edges are labeled with **true**; or (2) u represents a control predicate and v represents a component of P immediately nested within the control predicate represented by u, the label is the corresponding value of the oredicate. If we expendence edges u - f(v) are such tha (1) u is vertex that do nest variable (an assignment, (1)) v is a vertex that uses x, and (3) Control can reach v after u via an execution path along which there is no intervening definition of x. Finally, on these graph a slice for a criterion

i Midea software

Example 26. Consider the following simple programs [22]:



 $\texttt{Obf}_{\pmb{\alpha}}(P) = [\![\texttt{spec}]\!](\texttt{interp},P)$



dependence. Control dependence edges $u \rightarrow_c v$ are such that (1) u is the Entry vertex and v represents a component of P that is not nested within any control predicate; these edges are labeled with **true**; or (2) u represents a control predicate and v represents a component of P immediately nested within the control predicate represented by u, the label is the corresponding value of the predicate. If we expendent edges v - f(v) are such tha (1) u is vertex that do nest ariable (an assignment, (1) v is a vertex that do uses x, and (3) Control can reach v after u via an execution path along which there is no intervening definition of x. Finally, on these graph a slice for a criterion

i Midea software

Example 26. Consider the following simple programs [22]:







The CFG Abstraction



..... pc dynamic!

 \checkmark

The attacker is an abstract interpreter extracting the CFG from P

- ✓ forgets the computed memory \mathbb{M} : $C = \lambda \sigma$. \mathbb{M}
- forgets the branch computation when involving the pc: η
- ✓ Fixpoint Graph semantics: $\llbracket P \rrbracket_{\mathbb{G}} = lfp(\mathbb{G}_P)$

Static CFG extraction

Theorem

$$\mathcal{C}(\llbracket P \rrbracket_{\mathbb{G}}) = \llbracket P \rrbracket_{\mathbb{G}}^{\mathcal{C}, \eta}$$
 iff pc is not a program variable \checkmark

Completeness!!

Flattening is distorting an interpreter making an abstract interpreter extracting the CFG incomplete

$$\texttt{Obf}_{\pmb{\alpha}}(P) = [\![\texttt{spec}]\!](\texttt{interp}, P)$$







III: Slicing

 $\texttt{Obf}_{\pmb{\alpha}}(P) = \llbracket\texttt{spec}\rrbracket(\texttt{interp}, P)$

executed by P.

A program dependence graph G_{P} for a program P is a directed graph with vertices Der otize program component in Sees denoting dependences between components. The vertices of $G_{\rm P}$, Nodes $(G_{\rm P})$, represent the assignment statements and control predicates that occur in P. In addition $Nodes(G_P)$ includes a distinguished vertex called *Entry* M. denoting the starting vertex. An else represents either a control dependence or a flow *dependence*. Control dependence edges $u \longrightarrow_c v$ are such that (1) u is the *Entry* vertex and v represents a component of P that is not nested within any control predicate; these edges are labeled with true; or (2) u represents a control predicate and v represents a For a variable v and a statement of program point sol (finally used of the predicate. Flow dependence edges $u \rightarrow_f v$) are of program P with respect to the sust in a label is the corresponding value of the predicate. Flow dependence edges $u \rightarrow_f v$) are program such that S can be obtained by the of a side of a criterion of the second of a side of a criterion statements from P and if P halts, on input, le then, the yalue of v at the statement s, each time is reached in P, is the same in P and in S. $P_1 \begin{bmatrix} 2 \cdot i := 1; & 3 \cdot \text{while } i > 0 \text{ do } i := i + 1; & P_2 \begin{bmatrix} w := 1; & P_3 \\ w := 1; & P_3 \end{bmatrix} \begin{bmatrix} 1 \cdot x := 0; \\ w := 1; & P_3 \end{bmatrix} \begin{bmatrix} 1 \cdot x := 0; \\ y := x; \end{bmatrix}$ starting from node y := x, the definition of y. Entry -Entry Program while i > 0x :=i :=**Dependency Graph** while i > 0y := xx := 0i :=i := i + ii := i + 1 $\mathsf{Obf}_{\alpha}(P) = \llbracket \mathsf{spec} \rrbracket(\underset{\mathbf{Fig. 3. Program dependence graph of } P_1.$





Word Count program

which takes a block of text and outputs the number of lines (nl), words (nw) and characters (nc):

original() {
 int c, nl = 0, nw = 0, nc = 0, in;
 in = F;
 while ((c = getchar()) ! = EOF) {
 nc ++;
 if (c == ', || c == '\n' || c == '\t') in = F;
 else if (in == F) {in = T; nw ++; }
 if (c == '\n') nl ++;
 }
 out(nl, nw, nc); }
 Obf
$$_{\alpha}(P) = [[spec]](interp, P)$$





Word Count program

which takes a block of text and outputs the number of lines (nl), words (nw) and characters (nc): Slicing criterion: nl





Word Count program

which takes a block of text and outputs the number of lines (nl), words (nw) and characters (nc): Slicing criterion: nw

original() {
int c, nl = 0, nw = 0, nc =0, in;
in = F;
while ((c = getchar()) ! = EOF) {
nc ++;
if (c == '` || c == '\n' || c == '\t') in = F;
else if (in == F) {in = T; nw ++; }
if (c == '\n') nl ++;
}
out(nl, nw, nc); }
Obf_{$$\alpha$$}(P) = [spec](interp, P)





Word Count program

which takes a block of text and outputs the number of lines (nl), words (nw) and characters (nc):

```
obfuscated() {
  int c, nl = 0, nw = 0, nc = 0, in;
  in = F;
  while ((c = getchar())! = EOF)
    nc ++;
    if (c == `, || c == (n' || c == (t') in = F;
    else if (in == F) {in = T; nw ++; }
    if (c == '\n') {if (nw <= nc) nl ++; }
    if (nl > nc) nw = nc + nl;
     else {if (nw > nc) nc = nw - nl; }
  out(nl, nw, nc); }
     Obf_{\alpha}(P) = [spec](interp, P)
```





Word Count program

which takes a block of text and outputs the number of lines (nl), words (nw) and characters (nc):







Word Count program

which takes a block of text and outputs the number of lines (nl), words (nw) and characters (nc): Slicing criterion: nl

```
obfuscated() {
  int c, nl = 0, nw = 0, nc = 0, in;
  in = F;
  while ((c = getchar())! = EOF)
    nc ++;
    if (c == ``|| c == `\n'|| c == `\t') in = F;
     else if (in == F) {in = T; nw ++; }
    if (c == '\n') {if (nw <= nc) nl ++; }
    if (nl > nc) nw = nc + nl;
     else {if (nw > nc) nc = nw - nl; }
  out(nl, nw, nc); }
     Obf_{\alpha}(P) = [spec](interp, P)
```





Word Count program

which takes a block of text and outputs the number of lines (nl), words (nw) and characters (nc): Slicing criterion: nw

```
obfuscated() {
  int c, nl = 0, nw = 0, nc =0, in;
  in = F;
  while ((c = getchar())! = EOF) {
    nc ++;
    if (c == ``|| c == `\n'|| c == `\t') in = F;
     else if (in == F) {in = T; nw ++; }
    if (c == '\n') {if (nw <= nc) nl ++; }
    if (nl > nc) nw = nc + nl;
     else {if (nw > nc) nc = nw - nl; }
  out(nl, nw, nc); }
     Obf_{\alpha}(P) = [spec](interp, P)
```



Opaque Predicates



$$\begin{aligned} \forall x, y \in \mathbb{Z} : \quad 7y^2 - 1 \neq x^2 \\ \forall x \in \mathbb{Z} : \quad 2 \mid (x + x^2) \\ \forall x \in \mathbb{Z} : \quad 3 \mid (x^3 - x) \end{aligned} \\ \forall n \in \mathbb{Z}^+, x, y \in \mathbb{Z} : \quad (x - y) \mid (x^n - y^n) \\ \forall n \in \mathbb{Z}^+, x, y \in \mathbb{Z} : \quad 2 \mid n \lor (x + y) \mid (x^n + y^n) \\ \forall n \in \mathbb{Z}^+, x, y \in \mathbb{Z} : \quad 2 \not n \lor (x + y) \mid (x^n - y^n) \\ \forall x \in \mathbb{Z}^+ : \quad 9 \mid (10^x + 3 \cdot 4^{(x+2)} + 5) \\ \forall x \in \mathbb{Z} : \quad 3 \mid (7x - 5) \Rightarrow 9 \mid (28x^2 - 13x - 5) \\ \forall x \in \mathbb{Z} : \quad 5 \mid (2x - 1) \Rightarrow 25 \mid (14x^2 - 19x - 19) \\ \forall x, y, z \in \mathbb{Z} : \quad (2 \not x \land 2 \not y) \Rightarrow x^2 + y^2 \neq z^2 \\ \forall x \in \mathbb{Z}^+ : \quad 14 \mid (3 \cdot 7^{4x+2} + 5 \cdot 4^{2x-1} - 5) \end{aligned}$$







A program dependence graph G_P for a program P is a directed graph with vertices denoting program components and edges denoting dependences between components. The vertices of $G_{\rm P}$, Nodes($G_{\rm P}$), represent the assignment statements and control predicates that occur in P. In addition $Nodes(G_P)$ includes a distinguished vertex called *Entry* ur in P. In ac. Stating vertex. Ar core in stating vertex. Ar core in denoting i ents find a control dependence or a flow dependence. Council dependence enge $u \rightarrow d v$ are such that (1) is the *Bm v* v reases a concorrent of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-next edges are labeled with true; or (2) u represents a control predicate and v represents a component of P immediately nested within the control predicate represented by u, the label is the corresponding value of the predicate. Flow dependence edges $u \longrightarrow_f v$ are such that (1) u is a vertex that defines variable x (an assignment), (2) v is a vertex that uses x, and (3) Control can reach v after u via an execution path along which there is no intervening definition of x. Finally, on these graph a slice for a criterion

imititute software

Example 26. Consider the following simple programs [22]:

$$P_1 \begin{bmatrix} {}^{1\cdot}x := 0 ; \\ {}^{2\cdot}i := 1 ; {}^{3\cdot} \mathbf{while} \ i > 0 \ \mathbf{do} \ i := i+1 ; \qquad P_2 \begin{bmatrix} x := 0 ; \\ w := 1 ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} {}^{1\cdot}x := 0 ; \\ {}^{4\cdot}y := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} {}^{1\cdot}x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} {}^{1\cdot}x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} {}^{1\cdot}x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} {}^{1\cdot}x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} {}^{1\cdot}x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} {}^{1\cdot}x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} {}^{1\cdot}x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} {}^{1\cdot}x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} {}^{1\cdot}x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} {}^{1\cdot}x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} {}^{1\cdot}x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} {}^{1\cdot}x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} {}^{1\cdot}x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} {}^{1\cdot}x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} {}^{1\cdot}x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} {}^{1\cdot}x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} {}^{1\cdot}x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} {}^{1\cdot}x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} {}^{1\cdot}x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} {}^{1\cdot}x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} {}^{1\cdot}x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} {}^{1\cdot}x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} {}^{1\cdot}x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} {}^{1\cdot}x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} {}^{1\cdot}x := 0 ; \\ {}^{1\cdot}x := 0 ;$$

 $\mathsf{P}_{1} \begin{bmatrix} {}^{1\cdot}x := 0 ; \\ {}^{2\cdot}i := 1 ; {}^{3\cdot} \mathbf{while} i \\ {}^{4\cdot}y := x ; \end{bmatrix}$ In Fig. ?? we find a representation of the program dependence graph of the program dependence edges, without solution is representation, we have only control and flow dependence edges, without solution is a solution of the program dependence edges, without solution is a solution of the program dependence edges, without solution is a solution of the program dependence edges, without solution is a solution of the program dependence edges, without solution is a solution of the program dependence edges. The program dependence edges is a solution of the program dependence edges. The program dependence edges is a solution of the program dependence edges. The program dependence edges is a solution of the program dependence edges. The program dependence edges is a solution of the program dependence edges. The program dependence edges is a solution of the program dependence edges. The program dependence edges is a solution of the program dependence edges. The program dependence edges is a solution of the program dependence edges. The program dependence edges is a solution of the program dependence edges. The program dependence edges is a solution of the program dependence edges. The program dependence edges is a solution of the program dependence edges. The program dependence edges is a solution of the program dependence edges. The program dependence edges is a solution of the program dependence edges is a solution of the program dependence edges is a solution of the program dependence edges. The program dependence edges is a solution of the program dependence edges is a starting from node y := x, the definition of y.

$$s = \langle \sigma, \langle \text{Entry}, 1 \rangle \rangle$$



$$\texttt{Obf}_{\pmb{\alpha}}(P) = [\![\texttt{spec}]\!](\texttt{interp}, P)$$



A program dependence graph G_P for a program P is a directed graph with vertices denoting program components and edges denoting dependences between components. The vertices of $G_{\rm P}$, Nodes($G_{\rm P}$), represent the assignment statements and control predicates that occur in P. In addition $Nodes(G_P)$ includes a distinguished vertex called *Entry* statting vertex. At ease open of C_{1} and d_{2} because u = udenoting d sents find a control dependence or a flow dependence. Council dependence enge $u \rightarrow d v$ are such that (1) is the *Bm v* v reases a concorrent of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-next edges are labeled with true; or (2) u represents a control predicate and v represents a component of P immediately nested within the control predicate represented by u, the label is the corresponding value of the predicate. Flow dependence edges $u \longrightarrow_f v$ are such that (1) u is a vertex that defines variable x (an assignment), (2) v is a vertex that uses x, and (3) Control can reach v after u via an execution path along which there is no intervening definition of x. Finally, on these graph a slice for a criterion



Example 26. Consider the following simple programs [22]:

$$P_{1}\begin{bmatrix} 1 \cdot x := 0 ;\\ 2 \cdot i := 1 ; & 3 \cdot \text{while } i > 0 \text{ do } i := i + 1 ; \\ 4 \cdot y := x ; \end{bmatrix} P_{2}\begin{bmatrix} x := 0 ;\\ w := 1 ;\\ y := x ; \end{bmatrix} P_{3}\begin{bmatrix} 1 \cdot x := 0 ;\\ 4 \cdot y := x ; \\ 4 \cdot y := x ; \end{bmatrix}$$

In Fig. ?? we find a representation of the program dependence graph of the program = 1; ³ while $i \geq_{\text{distinction. Note that } P_1}^{P_1} O_i^{\text{In this representation we have only control and flow dependence edges, without distinction. Note that <math>P_3$ is a slice of both P_1 and of P_2 . In Fig ?? we can note that slice P_3 (with criterion the value of y) can be computed by following backwards the arcs starting from node y := x, the definition of y.

$$s = \langle \sigma, \langle 1, 2 \rangle \rangle$$



$$\texttt{Obf}_{\pmb{\alpha}}(P) = [\![\texttt{spec}]\!](\texttt{interp}, P)$$

 P_1 $^{4\cdot}y := x$



A program dependence graph G_P for a program P is a directed graph with vertices denoting program components and edges denoting dependences between components. The vertices of $G_{\rm P}$, Nodes($G_{\rm P}$), represent the assignment statements and control predicates that occur in P. In addition $Nodes(G_P)$ includes a distinguished vertex called *Entry* ur in P. In use stating vertex. Ar we expressed the under u = u = udenoting i ents finer a control d pendence or a flow dependence. Council dependence enge $u \rightarrow d v$ are such that (1) is the *Bm v* v reases a concorrent of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-nexted within any control predicate: nexe in the council of P that is non-next edges are labeled with true; or (2) u represents a control predicate and v represents a component of P immediately nested within the control predicate represented by u, the label is the corresponding value of the predicate. Flow dependence edges $u \longrightarrow_f v$ are such that (1) u is a vertex that defines variable x (an assignment), (2) v is a vertex that uses x, and (3) Control can reach v after u via an execution path along which there is no intervening definition of x. Finally, on these graph a slice for a criterion

iMdea softwa

Example 26. Consider the following simple programs [22]:

	$\int 1 \cdot x := 0;$	[x := 0;		$^{1.}x := 0;$
P_1	^{2.} $i := 1$; ^{3.} while $i > 0$ do $i := i + 1$;	P_2	w := 1;	P_3	
	$4 \cdot y := x ;$		y := x ;		${}^{4\cdot}y:=x;$

P₁ $\stackrel{1.x := 0}{\stackrel{2.i := 1}{\stackrel{3.}{\quad}}$ while *i* $\stackrel{P_1 O_1}{\stackrel{In fis. ??}{\stackrel{\text{the representation we have only control and flow dependence edges, without distinction. Note that <math>P_3$ is a slice of both P₁ and of P₂. In Fig ?? we can note that slice P₃ (with criterion the value of y) can be computed by following backwards the arcs $1, D_i = 2$] $s = \langle \sigma, \langle 2, 3 \rangle \rangle$ starting from node y := x, the definition of y.



$$\texttt{Obf}_{\pmb{\alpha}}(P) = [\![\texttt{spec}]\!](\texttt{interp},P)$$



A program dependence graph G_P for a program P is a directed graph with vertices denoting program components and edges denoting dependences between components. The vertices of $G_{\rm P}$, Nodes($G_{\rm P}$), represent the assignment statements and control predicates that occur in P. In addition $Nodes(G_P)$ includes a distinguished vertex called *Entry* Statting vertex. Arrest e representation of the second energy $u = u^2$ denoting d cents find a control dependence or a flow ction dependence. Courted dependence dependence. Council dependence range $u - t_d v$ are such that (1) in the *m* v v mex and v represents a component of P that is not nexted within any control predicate: next edges are labeled with true; or (2) u represents a control predicate and v represents a component of P immediately nested within the control predicate represented by u, the label is the corresponding value of the predicate. Flow dependence edges $u \longrightarrow_f v$ are such that (1) u is a vertex that defines variable x (an assignment), (2) v is a vertex that uses x, and (3) Control can reach v after u via an execution path along which there is no intervening definition of x. Finally, on these graph a slice for a criterion

i Mdea software

Example 26. Consider the following simple programs [22]:

$$P_1 \begin{bmatrix} {}^{1\cdot}x := 0 ; \\ {}^{2\cdot}i := 1 ; {}^{3\cdot}\mathbf{while} \ i > 0 \ \mathbf{do} \ i := i+1 ; \qquad P_2 \begin{bmatrix} x := 0 ; \\ w := 1 ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} {}^{1\cdot}x := 0 ; \\ {}^{4\cdot}y := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ w := 1 ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} {}^{1\cdot}x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ {}^{4\cdot}y :$$



$$\texttt{Obf}_{\pmb{\alpha}}(P) = [\![\texttt{spec}]\!](\texttt{interp}, P)$$



A program dependence graph G_P for a program P is a directed graph with vertices denoting program components and edges denoting dependences between components. The vertices of $G_{\rm P}$, Nodes($G_{\rm P}$), represent the assignment statements and control predicates that occur in P. In addition $Nodes(G_P)$ includes a distinguished vertex called *Entry* statting vertex. Ar edge epic Control denondencionege u =denoting d ents finer a convol d pendence or a flow ction dependence. Co and d pendence dependence. Council dependence reage $u \rightarrow e^v$ are such that (1) is the *bin* y view and v represents a component of P that is not nested within any control predicate: these edges are labeled with true; or (2) u represents a control predicate and v represents a component of P immediately nested within the control predicate represented by u, the label is the corresponding value of the predicate. Flow dependence edges $u \longrightarrow_f v$ are such that (1) u is a vertex that defines variable x (an assignment), (2) v is a vertex that uses x, and (3) Control can reach v after u via an execution path along which there is no intervening definition of x. Finally, on these graph a slice for a criterion

institute dea **SO**[†]

Example 26. Consider the following simple programs [22]:

$$P_1 \begin{bmatrix} {}^{1\cdot}x := 0 ; \\ {}^{2\cdot}i := 1 ; {}^{3\cdot} \mathbf{while} \ i > 0 \ \mathbf{do} \ i := i+1 ; \qquad P_2 \begin{bmatrix} x := 0 ; \\ w := 1 ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} {}^{1\cdot}x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix}$$

starting from node y := x, the definition of y.



$$\texttt{Obf}_{\pmb{\alpha}}(P) = [\![\texttt{spec}]\!](\texttt{interp}, P)$$



A program dependence graph G_P for a program P is a directed graph with vertices denoting program components and edges denoting dependences between components. The vertices of $G_{\rm P}$, Nodes($G_{\rm P}$), represent the assignment statements and control predicates that occur in P. In addition $Nodes(G_P)$ includes a distinguished vertex called *Entry* Statting vertex. Arredge control dependence $u = u^2$ denoting d ents find a control dependence or a flow ction dependence. Co and d pendence dependence. Council dependence reage $u \rightarrow e^v$ are such that (1) is the *bin* y view and v represents a component of P that is not nested within any control predicate: these edges are labeled with true; or (2) u represents a control predicate and v represents a component of P immediately nested within the control predicate represented by u, the label is the corresponding value of the predicate. Flow dependence edges $u \longrightarrow_f v$ are such that (1) u is a vertex that defines variable x (an assignment), (2) v is a vertex that uses x, and (3) Control can reach v after u via an execution path along which there is no intervening definition of x. Finally, on these graph a slice for a criterion

institute

softw

dea

Example 26. Consider the following simple programs [22]:

$$P_1 \begin{bmatrix} {}^{1\cdot}x := 0 ; \\ {}^{2\cdot}i := 1 ; {}^{3\cdot}\mathbf{while} \ i > 0 \ \mathbf{do} \ i := i+1 ; \qquad P_2 \begin{bmatrix} x := 0 ; \\ w := 1 ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} {}^{1\cdot}x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix}$$

 $\mathsf{P}_{1} \begin{bmatrix} 1 \cdot x := 0; \\ 2 \cdot i := 1; \\ 4 \cdot y := x; \end{bmatrix} \text{ In Fig. ?? we find a representation of the program dependence graph of the program dependence edges, without a specific edges, without = 1, <math>D_{i} = 3a \end{bmatrix} \qquad s_{1} = \langle \sigma, \langle 3, 3a \rangle \rangle \text{ and } s_{2} = \langle \sigma, \langle 3, 4 \rangle \rangle$



$$\texttt{Obf}_{\pmb{\alpha}}(P) = [\![\texttt{spec}]\!](\texttt{interp}, P)$$



A program dependence graph G_P for a program P is a directed graph with vertices denoting program components and edges denoting dependences between components. The vertices of G_P , $Nodes(G_P)$, represent the assignment statements and control predicates that occur in P. In addition $Nodes(G_P)$ includes a distinguished vertex called *Entry* denoting discussating vertex. An edge opposents dure a *convol dipendence* or a *flow* dependence. Could dependence age $u \rightarrow e_d v$ are such that (1) in sthe analy vertex and v represents a component of P that is prenested within any control predicate: nest edges are labeled with **true**; or (2) u represents a control predicate and v represents a component of P immediately nested within the control predicate represented by u, the label is the corresponding value of the predicate. Flow dependence edges $u \rightarrow_f v$) are such that (1) u is a vertex that defines variable x (an assignment), (2) v is a vertex that uses x, and (3) Control can reach v after u via an execution path along which there is no intervening definition of x. Finally, on these graph a slice for a criterion



Example 26. Consider the following simple programs [22]:

$$P_1 \begin{bmatrix} {}^{1\cdot}x := 0 ; \\ {}^{2\cdot}i := 1 ; {}^{3\cdot} \mathbf{while} \ i > 0 \ \mathbf{do} \ i := i+1 ; \qquad P_2 \begin{bmatrix} x := 0 ; \\ w := 1 ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} {}^{1\cdot}x := 0 ; \\ {}^{4\cdot}y := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ w := 1 ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} {}^{1\cdot}x := 0 ; \\ {}^{4\cdot}y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y := x ; \\ y := x ; \end{bmatrix} P_3 \begin{bmatrix} x := 0 ; \\ y :=$$

$$P_{1} \begin{bmatrix} {}^{1} \cdot x := 0 ; \\ {}^{2} \cdot i := 1 ; {}^{3} \cdot \text{while } i \\ {}^{4} \cdot y := x ; \end{bmatrix}$$

In Fig. ?? we find a representation of the program dependence graph of the program $\sum_{\substack{P_1 \\ distinction. Note that P_3 is a slice of both P_1 and of P_2. In Fig ?? we call note that slice = 2, D_y = 4]$ $s_1 = \langle \sigma, \langle 3, 3a \rangle \rangle \text{ and } s_2 = \langle \sigma, \langle 4, \bot \rangle \rangle$ $P_3 \text{ (with criterion the value of y) can be computed by following backwards the arcs starting from node <math>y := x$, the definition of y.



$$\texttt{Obf}_{\pmb{\alpha}}(P) = [\![\texttt{spec}]\!](\texttt{interp}, P)$$







Theorem

The Semantic PDG is complete (i.e., the PDG analysis is precise) iff the program does not contain *fake* dependencies

$$\texttt{Obf}_{\pmb{\alpha}}(P) = [\![\texttt{spec}]\!](\texttt{interp}, P)$$







Theorem

The Semantic PDG is complete (i.e., the PDG analysis is precise) iff the program does not contain *fake* dependencies

$$\texttt{Obf}_{\pmb{\alpha}}(P) = [\![\texttt{spec}]\!](\texttt{interp}, P)$$







Theorem

The Semantic PDG is complete (i.e., the PDG analysis is precise) iff the program does not contain *fake* dependencies

$$\texttt{Obf}_{\pmb{\alpha}}(P) = [\![\texttt{spec}]\!](\texttt{interp}, P)$$







True dependencies

Theorem

The Semantic PDG is complete (i.e., the PDG analysis is precise) iff the program does not contain *fake* dependencies

$$\texttt{Obf}_{\pmb{\alpha}}(P) = [\![\texttt{spec}]\!](\texttt{interp}, P)$$







Theorem

The Semantic PDG is complete (i.e., the PDG analysis is precise) iff the program does not contain *fake* dependencies

$$\texttt{Obf}_{\pmb{\alpha}}(P) = [\![\texttt{spec}]\!](\texttt{interp}, P)$$

















Any obfuscation technique is an instance of

$$\texttt{Obf}_{\alpha}(P) = \llbracket\texttt{spec}\rrbracket(\texttt{interp}, P)^{\alpha}$$

for some $interp^{\alpha}$ making an abstraction α incomplete!



Profiling: Abstract memory keeping only (partial) resource usage Tracing: Abstraction of traces (e.g., by trace compression)

- Slicing: Abstraction of traces (relative to variables)
- Monitoring: Abstraction of trace semantics ([Cousot&Cousot POPL02])
- Decompilation: Abstracts syntactic structures (e.g., reducible loops)
- Disassembly: Abstracts binary structures (e.g., recursive traversal)

Given an obfuscated code P, what is α ?

Given α , can we derive interp^{α} systematically?



The Challenges





 \leq

 \leq



- **Correctness:** a provably correct obfuscation?
- Is obscured code still secure?
- Adaptable interpreters and specialisers for dynamic obfuscation?
- **Measures** of behaviour leakage?
- Automated Machine Learning reverse engineering as attack models?



3.

- - cogen := [spec](spec, spec)

Transform program Generate transformer Transformer generator



The Challenges







 \leq

 \leq

Intentional and extensional aspects of computation: From computability and complexity to program analysis and security

> Shonan Village Center, Japan January 22-25, 2018









Isabella

Mila



Neil



Sandrine





Roberta

Roberto

Obfuscation & Security







Francesco Francesco

Completeness







The InterProc Analyser

http://pop-art.inrialpes.fr/interproc/interprocweb.cgi







```
Example
```

```
proc MC(n:int) returns (r:int)
var t1:int, t2:int;
begin
  if (n>100) then
                                    Nested recursive function by
      r = n - 10;
                                             John McCarthy!
  else
      t1 = n + 11;
      t2 = MC(t1);
      r = MC(t2);
  endif;
                             M(n) = \begin{cases} n - 10 & \text{if } n > 100 \\ M(M(n+11)) & \text{if } n \le 100 \end{cases}
end
var
a:int, b:int;
begin
  b = MC(a);
end
```





Example

```
Annotated program after forward analysis
proc MC (n : int) returns (r : int) var t1 : int, t2 : int;
begin
  /* (L5 C5) top */
  if n > 100 then
     /* (L6 C17) [|n-101>=0|] */
     r = n - 10; /* (L7 C14)
                    [n-101>=0; r-91>=0] */
  else
    /* (L8 C6) [|-n+100>=0|] */
    t1 = n + 11; /* (L9 C17)
                    [|-n+100>=0; -t1+111>=0|] */
    t2 = MC(t1); /* (L10 C17)
                    [ |-n+100>=0; -t1+111>=0; t2-91>=0 | ] */
    r = MC(t2); /* (L11 C16)
                   [|-n+100>=0; r-91>=0; -t1+111>=0; t2-91>=0|] */
  endif; /* (L12 C8) [|r-91>=0|] */
end
                                                       var a : int, b : int;
begin
/* (L17 C5) top */
 b = MC(a); /* (L18 C12) [|b-91>=0|] */
end
```





Obfuscation

Source



Result

```
Annotated program after forward analysis
var i : int;
begin
    /* (L2 C5) top */
    i = 0; /* (L3 C8) [|i>=0; -i+201>=0|] */
    while i <= 200 do
        /* (L4 C19) [|i>=0; -i+200>=0|] */
        i = i + 1; /* (L5 C12)
              [|i-1>=0; -i+201>=0|] */
        done; /* (L6 C7) [|i-201=0|] */
end
```

```
Annotated program after forward analysis
var i : int, j : int;
begin
  /* (L2 C5) top */
 i = 0; /* (L3 C8) [|i=0|] */
  j = 0; /* (L4 C8)
            [|i>=0; -i+21>=0; j>=0; -j+10>=0|] */
 while 10 * i + j <= 200 do
    /* (L5 C26)
       [|i>=0; -i+20>=0; j>=0; -j+10>=0|] */
   i = i + (j + 1) / i, 0 10; /* (L6 C26)
                                 [|i>=0; -i+21>=0; j>=0; -j+10>=0|] */
    j = (j + 1)  % 10; /* (L7 C21)
                         [|i>=0; -i+21>=0; j>=0; -j+10>=0|] */
  done; /* (L8 C7)
           [|i-20>=0; -i+21>=0; j>=0; -j+10>=0|] */
 i = 10 * i + j; /* (L9 C11)
                     [|i-200>=0; -i+220>=0; j>=0; -j+10>=0|] */
end
```







Obfuscation





var i:int, j:int; begin i = 0; j=0; while (j<=10) do i =i+1; j =j+1; done; end

Result

```
Annotated program after forward analysis
var i : int;
begin
    /* (L2 C5) top */
    i = 0; /* (L3 C8) [|i>=0; -i+11>=0|] */
    while i <= 10 do
        /* (L4 C18) [|i>=0; -i+10>=0|] */
        i = i + 1; /* (L5 C12)
              [|i-1>=0; -i+11>=0|] */
        done; /* (L6 C7) [|i-11=0|] */
end
```

```
Obscure!
```




With Octagons

Source



Result

```
Annotated program after forward analysis
var i : int, j : int;
begin
  /* (L2 C5) top */
  i = 0; /* (L3 C8) [|i>=0; -i>=0|] */
  j = 0; /* (L3 C13)
             |i>=0; -i+11>=0; -i+j>=0; i+j>=0; j>=0; -i-j+22>=0; i-j>=0;
               -j+11>=0|| */
  while j <= 10 do
    /* (L4 C18)
       [|i>=0; -i+10>=0; -i+j>=0; i+j>=0; j>=0; -i-j+20>=0; i-j>=0; -j+10>=0|] */
    i = i + 1; /* (L5 C11)
                   [|i-1>=0; -i+11>=0; -i+j+1>=0; i+j-1>=0; j>=0; -i-j+21>=0;
                     i-j-1>=0; -j+10>=0|] */
    j = j + 1; /* (L6 C12)
                   [|i-1\rangle=0; -i+11\rangle=0; -i+j\rangle=0; i+j-2\rangle=0; j-1\rangle=0; -i-j+22\rangle=0;
                     i-j>=0; -j+11>=0|] */
  done; /* (L7 C7)
           [|i-11>=0; -i+11>=0; -i+j>=0; i+j-22>=0; j-11>=0; -i-j+22>=0;
             i-j>=0; -j+11>=0|] */
```





Let us play !!!

http://pop-art.inrialpes.fr/interproc/interprocweb.cgi





Obfuscation on Linear Relations

Source

```
proc incr (x:int) returns (y:int)
begin
  y = x+1;
end
var i:int;
begin
  i = 0;
while (i<=10) do
    i = incr(i);
done;</pre>
```

Result

end

```
Annotated program after forward analysis
proc incr (x : int) returns (y : int) var ;
begin
    /* (L3 C5) top */
    y = x + 1; /* (L4 C10) [|-x+y-1=0|] */
end
var i : int;
begin
    /* (L8 C5) top */
    i = 0; /* (L9 C8) top */
    while i <= 10 do
        /* (L10 C18) top */
        i = incr(i); /* (L11 C16) top */
        done; /* (L12 C7) top */
end</pre>
```





Obfuscation on Octagons

Source

var i:int; begin i = 0; while (i<=10) do i = i+1; done; end

Result

```
Annotated program after forward analysis
var i : int, j : int;
begin
  /* (L2 C5) top */
 i = 0; /* (L3 C8) [|i>=0; -i>=0|] */
  j = 1; /* (L3 C12)
             [|i>=0; -i+1024>=0; -i+j-1>=0; i+j-1>=0; j-1>=0; -i-j+3072>=0;
               i-j+2047>=0; -j+2048>=0|] */
  while j <= 1024 do
    /* (L4 C20)
       [|i\rangle=0; -i+1023\rangle=0; -i+j-1\rangle=0; i+j-1\rangle=0; j-1\rangle=0; -i-j+2047\rangle=0;
         i-j+1024>=0; -j+1024>=0|] */
    j = j * 2; /* (L5 C12)
                   [|i\rangle=0; -i+1023\rangle=0; -i+j-2\rangle=0; i+j-2\rangle=0; j-2\rangle=0;
                     -i-j+3071>=0; i-j+2048>=0; -j+2048>=0|1 */
    i = i + 1; /* (L6 C12)
                   [|i-1>=0; -i+1024>=0; -i+j-1>=0; i+j-3>=0; j-2>=0;
                     -i-j+3072>=0; i-j+2047>=0; -j+2048>=0|] */
  done; /* (L7 C7)
            [|i>=0; -i+1024>=0; -i+j-1>=0; i+j-1025>=0; j-1025>=0;
              -i-j+3072>=0; i-j+2047>=0; -j+2048>=0|] */
end
```

```
var i:int, j:int;
begin
    i = 0;j=1;
    while (j<=1024) do
        j = j*2;
        i = i+1;
        done;
end
```





Hacking McCarthy 91

Source

```
proc MC(n:int) returns (r:int)
var t1:int, t2:int;
begin
    if (n>100) then
        r = n-10;
    else
        t1 = n + 11;
        t2 = MC(t1);
        r = MC(t2);
    endif;
end
var
a:int, b:int;
begin
```

b = MC(a);

end





Hacking McCarthy 91

Source

```
proc MC(n:int) returns (r:int)
var t1:int, t2:int;
begin
  if (n>100) then
     r = n - 10;
  else
     t1 = n + 11;
     t_{2} = MC(t_{1});
     r = MC(t2);
  endif;
end
var
a:int, b:int;
begin
  b = MC(a);
end
```

```
proc MC(n:int) returns (r:int)
var t1:int, t2:int, p:int;
begin
  p = n*2;
  if (p>200) then
     r = (p-20)/2;
  else
     if (n*(n-1))%2==0 then
        t1 = (p + 22)/2;
        t2 = MC(t1);
        r = MC(t2);
     else
        r=50;
     endif;
  endif;
end
var
a:int, b:int;
begin
  b = MC(a);
end
```

See More from Vivek Notani

Try it!!!





Hacking McCarthy 91

Source

proc MC(n:int) returns (r:int) var t1:int, t2:int; begin if (n>100) then r = n - 10;else t1 = n + 11;t2 = MC(t1);r = MC(t2);endif; end var a:int, b:int; begin b = MC(a);end

proc MC(n:int) returns (r:int) var t1:int, t2:int, p:int; begin p = n*n;if (p>10000) then r = (p/n-10);else t1 = (p + 11*n)/n;t2 = MC(t1);r = MC(t2);endif; end var a:int, b:int; begin b = MC(a);end

Try it!!!





Why this works?

(a) Complete Analysis

```
var i : int;
begin
 /* (L4 C5) top */
  i = 0; /* (L5 C8) [|i>=0; -i+106>=0|] */
  while i <= 10 do
    /* (L6 C18) [|i>=0; -i+10>=0|] */
    if i == 5 then
      /* (L7 C18) [|i-5=0|] */
      i = i + 100; /* (L8 C19) [|i-105=0|] */
    endif; /* (L9 C10) [|i>=0; -i+105>=0|] */
    if i == 105 then
      /* (L10 C20) [|i-105=0|] */
      i = i - 100; /* (L11 C19) [|i-5=0|] */
    endif; /* (L12 C10) [|i>=0; -i+105>=0|] */
    i = i + 1; /* (L13 C14)
                  [|i-1>=0; -i+106>=0|] */
  done; /* (L14 C7) [|i-11>=0; -i+106>=0|] */
end
```

(b) Incomplete Analysis

Ideas??