Live variables analysis for JavaScript

End-of-internship talk (1/2)

Jacques-Henri Jourdan

9/8/2010

Example 1

```
function f(a) {
  for(var i = 1; i < a.length - 1; i++) {
    var tmp = a[i];
    a[i] = a[i + 1];
    a[i + 1] = tmp;
  }

}
```

- Do we have to save `tmp`?

Example 1

```
function f(a) {
  for(var i = 1; i < a.length - 1; i++) {
    var tmp = a[i];
    a[i] = a[i + 1];
    a[i + 1] = tmp;
  }
  print(tmp);
}
```

- Do we have to save `tmp`?

Example 1

```
function f(a) {
  for(var i = 1; i < a.length - 1; i++) {
    var tmp = a[i];
    a[i] = a[i + 1];
    a[i + 1] = tmp;
  }
  print(tmp);
}
```

- Do we have to save `tmp`?
- All variables are hoisted to function scope.

Example 2

```
function f() {
  var i;

  for(i = 0; i < 10000000; i++);
}
```

- Can `i` be considered an integer?

Example 2

```
function f() {
  var i;
  print(i);
  for(i = 0; i < 10000000; i++);
}
```

- Can `i` be considered an integer?

Example 2

```
function f() {
  var i;
  print(i);
  for(i = 0; i < 10000000; i++);
}
```

- Can `i` be considered an integer?
- All variables are initialized to `undefined`.

Example 3

```
function f(a) {
  try {
    for(var i = 0; i < a.length; i++) {
      var tmp = a.length;
      a[i] = tmp + 1;
    }
  } catch(e) {
    print(e);
    print(tmp);
  }
}
```

- Do we have to save tmp?

Example 3

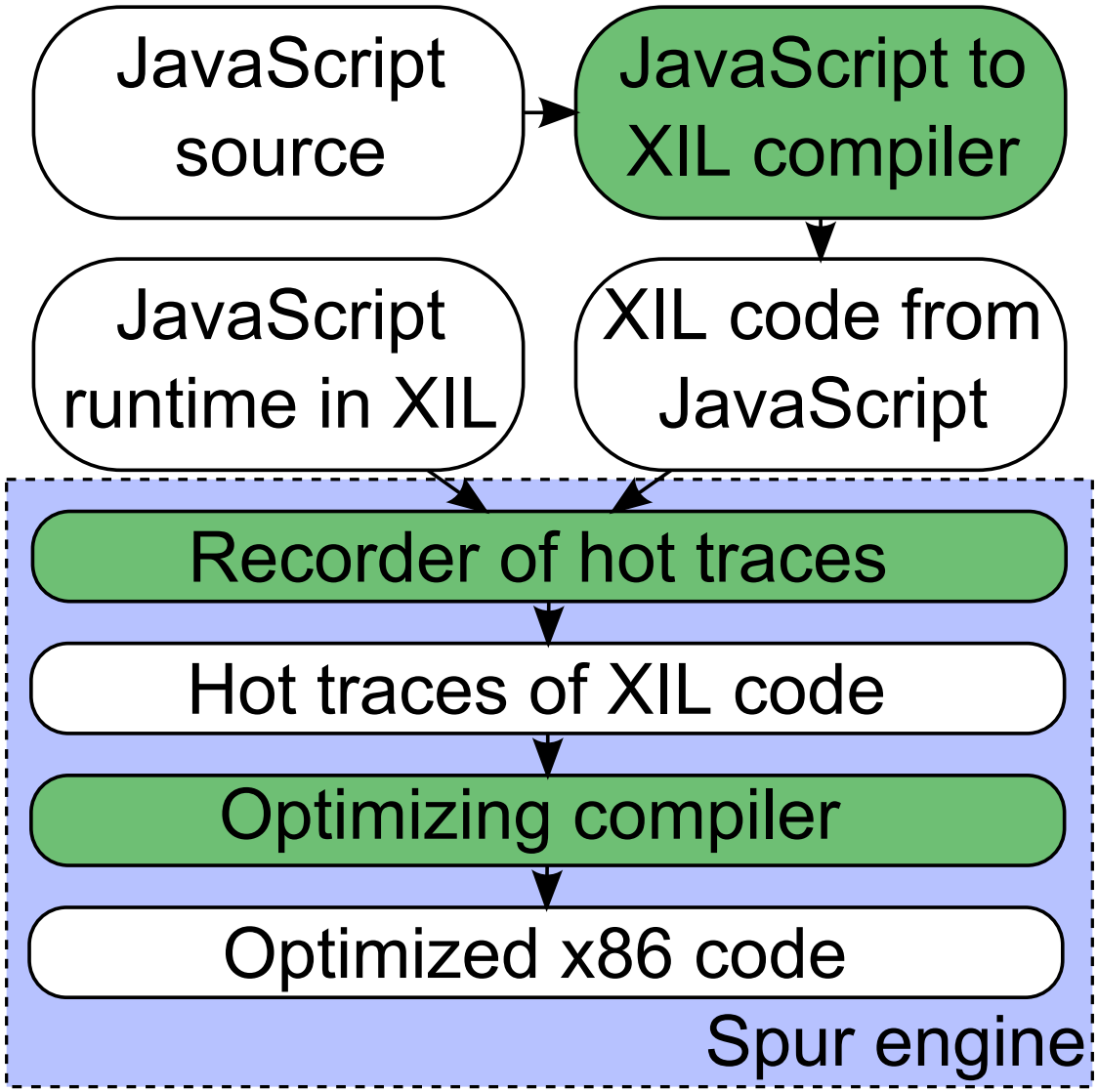
```
function f(a) {
  try {
    for(var i = 0; i < a.length; i++) {
      var tmp = a.length;
      a[i] = tmp.valueOf() + 1;
    }
  } catch(e) {
    print(e);
    print(tmp);
  }
}
```

- Do we have to save `tmp`?
- Exceptions can arise (almost) everywhere.

Live variables analysis : What? Why?

- What?
  - At that point of the program, will the value of this variable be used?
- Why?
  - Direct use in Spur:
    - ★ Local variables are often copied from a type-specialised location (registers) to unspecialised locations.
    - ★ Dead variables: no need to copy.
    - ★ Decrease stack memory consumption.
  - Helping type inference
    - ★ At the beginning of a function, each local is `undefined`
    - ★ `undefined` is neither integer nor float
    - ★ Must we take the initial value into account?

Spur: a tracing JIT compiler for eXtended IL



Dynamic optimizations vs. static optimizations

- Spur uses dynamic information (traces) to optimize hot code.
- Some information are destroyed by the JavaScript to XIL compiler.
  - Code structure replaced by a control flow graph.
  - XIL level: difficult to know when a local won't be used
    - ★ Especially when we take the address of a local.
- Spur is not able to do some optimizations.
- Analyzes at Javascript level:
  - Type inference for local variables.
  - Liveness analysis for local variables.

Example

```
function foo(a) {
    var start = new Date();
    var i = 0;
    while (i < a.length - 1) {
      var tmp = a[i];
      a[i] = a[i + 1];
      a[i + 1] = tmp;
      i++;
    }
    var end = new Date();
    return end - start;
}
```

a	Dead
start	Live
end	Live
i	Dead
tmp	Dead

Example

```
function foo(a) {
    var start = new Date();
    var i = 0;
    while (i < a.length - 1) {
      var tmp = a[i];
      a[i] = a[i + 1];
      a[i + 1] = tmp;
      i++;
    }
▶   var end = new Date();
    return end - start;
}
```

a	Dead
start	Live
end	Dead
i	Dead
tmp	Dead

Example

```
function foo(a) {
    var start = new Date();
    var i = 0;
    while (i < a.length - 1) {
      var tmp = a[i];
      a[i] = a[i + 1];
      a[i + 1] = tmp;
      i++;
    }
    var end = new Date();
    return end - start;
}
```

a	Dead
start	Live
end	Dead
i	Dead
tmp	Dead

Example

```
function foo(a) {
    var start = new Date();
    var i = 0;
    while (i < a.length - 1) {
      var tmp = a[i];
      a[i] = a[i + 1];
      a[i + 1] = tmp;
      i++;
    }
    var end = new Date();
    return end - start;
}
```

a	Dead
start	Live
end	Dead
i	Live
tmp	Dead

Example

```
function foo(a) {
    var start = new Date();
    var i = 0;
    while (i < a.length - 1) {
      var tmp = a[i];
      a[i] = a[i + 1];
      a[i + 1] = tmp;
      i++;
    }
    var end = new Date();
    return end - start;
}
```

a	Live
start	Live
end	Dead
i	Live
tmp	Live

Example

```
function foo(a) {
    var start = new Date();
    var i = 0;
    while (i < a.length - 1) {
      var tmp = a[i];
      a[i] = a[i + 1];
      a[i + 1] = tmp;
      i++;
    }
    var end = new Date();
    return end - start;
}
```

a	Live
start	Live
end	Dead
i	Live
tmp	Live

Example

```
function foo(a) {
    var start = new Date();
    var i = 0;
    while (i < a.length - 1) {
      var tmp = a[i];
      a[i] = a[i + 1];
      a[i + 1] = tmp;
      i++;
    }
    var end = new Date();
    return end - start;
}
```

a	Live
start	Live
end	Dead
i	Live
tmp	Dead

Example

```
function foo(a) {
    var start = new Date();
    var i = 0;
    while (i < a.length - 1) {
      var tmp = a[i];
      a[i] = a[i + 1];
      a[i + 1] = tmp;
      i++;
    }
    var end = new Date();
    return end - start;
}
```

a	Live
start	Live
end	Dead
i	Live
tmp	Dead

Example

```
function foo(a) {
    var start = new Date();
    var i = 0;
    while (i < a.length - 1) {
      var tmp = a[i];
      a[i] = a[i + 1];
      a[i + 1] = tmp;
      i++;
    }
    var end = new Date();
    return end - start;
}
```

a	Live
start	Live
end	Dead
i	Live
tmp	Dead

Example

```
function foo(a) {
    var start = new Date();
    var i = 0;
    while (i < a.length - 1) {
      var tmp = a[i];
      a[i] = a[i + 1];
      a[i + 1] = tmp;
      i++;
    }
    var end = new Date();
    return end - start;
}
```

a	Live
start	Live
end	Dead
i	Live
tmp	Live

Example

```
function foo(a) {
    var start = new Date();
    var i = 0;
    while (i < a.length - 1) {
      var tmp = a[i];
      a[i] = a[i + 1];
      a[i + 1] = tmp;
      i++;
    }
    var end = new Date();
    return end - start;
}
```

a	Live
start	Live
end	Dead
i	Live
tmp	Live

Example

```
function foo(a) {
    var start = new Date();
    var i = 0;
    while (i < a.length - 1) {
      var tmp = a[i];
      a[i] = a[i + 1];
      a[i + 1] = tmp;
      i++;
    }
    var end = new Date();
    return end - start;
}
```

a	Live
start	Live
end	Dead
i	Live
tmp	Dead

Example

```
function foo(a) {
    var start = new Date();
    var i = 0;
    while (i < a.length - 1) {
      var tmp = a[i];
      a[i] = a[i + 1];
      a[i + 1] = tmp;
      i++;
    }
    var end = new Date();
    return end - start;
}
```

a	Live
start	Live
end	Dead
i	Live
tmp	Dead

Example

```
function foo(a) {
    var start = new Date();
    var i = 0;
    while (i < a.length - 1) {
      var tmp = a[i];
      a[i] = a[i + 1];
      a[i + 1] = tmp;
      i++;
    }
    var end = new Date();
    return end - start;
}
```

a	Live
start	Live
end	Dead
i	Dead
tmp	Dead

Example

```
function foo(a) {
    var start = new Date();
    var i = 0;
    while (i < a.length - 1) {
      var tmp = a[i];
      a[i] = a[i + 1];
      a[i + 1] = tmp;
      i++;
    }
    var end = new Date();
    return end - start;
}
```

a	Live
start	Dead
end	Dead
i	Dead
tmp	Dead

Example

```
function foo(a) {
    var start = new Date();
    var i = 0;
    while (i < a.length - 1) {
      var tmp = a[i];
      a[i] = a[i + 1];
      a[i + 1] = tmp;
      i++;
    }
    var end = new Date();
    return end - start;
}
```

a	Live
start	Dead
end	Dead
i	Dead
tmp	Dead

- We won't have to save `tmp`.
- We have to do only 2 iterations to reach fixpoint!

The analysis

- We visit the program backward.
- End of the program: all the variable are dead.
- Variable read: becomes live.
- Variable write: becomes dead.
- Branch: is the variable live in any branch?
- Loop: fixpoint, only 2 iterations necessary.
- High-level constructs: `break`, `continue`, exceptions...
  - We maintain the set of live variables after a `break`/`continue`/exception.

Jitted codes

```
B16@46:  SOURCEREGION     MicroModified\array-int-swap.js:9,50 i+1
B16@47:  SOURCEREGION     MicroModified\array-int-swap.js:9,50 i
B16@48:  SOURCEREGION     MicroModified\array-int-swap.js:9,52 1
B16@49:  BINARY           $L11 <- i Add 1D
B16@50:  SOURCEREGION     MicroModified\array-int-swap.js:9,57 tmp
B16@51:  LOADADDRESS      $L5 <- tmp
B16@52:  CALLSTATIC        <- Microsoft.JScript.XILHelper::SetValue(
B16@53:  FREE             tmp,$L5,$L11,$L12
B16@54:  UNCONDBRANCH     B20
B17:
B17@0:   FREE             i
B17@1:   UNCONDBRANCH     B15
B18:
B18@0:   SOURCEREGION     MicroModified\array-int-swap.js:8,45 i++
B18@1:   BINARY           i <- i Add 1D
B18@2:   UNCONDBRANCH     B16
```

```
L#17:
        mov     EBX, dword ptr [EBP - 0x00000200]
        cmp     EBX, 2147483645
        jnl     L#18
        lea     ESI, dword ptr [EBX + 0x01]
        push    ESI
        fld     dword ptr [ESP]
        add     ESP, 4
        fld     qword ptr [EBP - 0x00000260]
        fucomip st(1)
        jp      L#19
        jbe     L#19
        mov     EDI, dword ptr [EBP - 0x00000264]
        cmp     EBX, EDI
        jnb     L#21
        mov     EBX, dword ptr [EBP - 0x00000248]
        mov     EDX, dword ptr [EBP - 0x00000200]
        mov     EAX, dword ptr [EBX + EDX*4 + 0x08]
        mov     dword ptr [EBP - 0x00000288], EAX
        cmp     ESI, EDI
        jnb     L#23
        mov     EDI, dword ptr [EBX + ESI*4 + 0x08]
        mov     ECX, dword ptr [EBP - 0x00000200]
        mov     dword ptr [EBX + ECX*4 + 0x08], EDI
        mov     EAX, dword ptr [EBP - 0x00000288]
        mov     dword ptr [EBX + ESI*4 + 0x08], EAX
        mov     EDI, dword ptr [EBP - 0x00000200]
        add     EDI, 2
        cmp     EDI, dword ptr [EBP - 0x00000264]
        jnb     L#25
        mov     dword ptr [EBP - 0x00000278], ESI
        mov     ESI, dword ptr [EBX + EDI*4 + 0x08]
        mov     ECX, dword ptr [EBP - 0x00000278]
        mov     dword ptr [EBX + ECX*4 + 0x08], ESI
        mov     ESI, dword ptr [EBP - 0x00000288]
        mov     dword ptr [EBX + EDI*4 + 0x08], ESI
        lea     EBX, dword ptr [EBP - 0x48]
        mov     dword ptr [EBX + 0x04], 0x00000001
        mov     dword ptr [EBX + 0x08], ESI
        xor     EAX, EAX
        mov     dword ptr [EBX], EAX
        mov     dword ptr [EBP - 0x00000200], EDI
        fstp    st(0)
        add     dword ptr [EBP - 0x00000204], 2
        jmp     L#17
```

Benchmarks

Micro benchmarks	
array-double-swap.js	8.86%
array-double-times-bigtree.js	-3.82%
array-double-times.js	2.24%
array-int-copy-const-bound.js	-0.98%
array-int-copy-len-bound.js	-0.98%
array-int-swap.js	-0.96%
ffunction-closure.js	-1.80%
ffunction-empty.js	0.00%
ffunction-missing-args.js	0.00%
ffunction-sum.js	0.00%
floop-empty.js	-0.28%
function-closure.js	0.00%
function-empty.js	0.16%
function-missing-args.js	-0.74%
function-sum.js	0.32%
invoke-empty-closure.js	-1.13%
invoke-empty-function.js	4.06%
invoke-empty-method.js	-0.26%
jsvar-int-add.js	-0.22%
jsvar-int-sub.js	-0.24%
jsvar-loop-ind.js	0.67%
loop-empty-prop.js	0.19%
loop-empty-resolve.js	-0.08%
loop-empty.js	-2.38%
loop-sum.js	-0.48%
obj-field-dyn-acc.js	-0.33%
obj-field-inferred-acc-double.js	2.47%
obj-field-inferred-acc.js	0.33%
obj-field-inferred-this-acc-double.js	-0.71%

SunSpider benchmarks	
3d-cube.js	1.39%
3d-morph.js	0.00%
3d-raytrace.js	5.56%
access-binary-trees.js	2.06%
access-fannkuch.js	-15.38%
access-nbody.js	0.00%
access-nsieve.js	9.09%
bitops-3bit-bits-in-byte.js	N/A
bitops-bits-in-byte.js	0.00%
bitops-bitwise-and.js	0.00%
bitops-nsieve-bits.js	-5.88%
controlflow-recursive.js	0.00%
crypto-aes.js	11.59%
crypto-md5.js	0.00%
crypto-sha1.js	0.00%
date-format-tofte.js	-3.13%
date-format-xparb.js	-1.30%
math-cordic.js	-18.92%
math-partial-sums.js	1.64%
math-spectral-norm.js	0.00%
regexp-dna.js	0.07%
string-base64.js	0.00%
string-fasta.js	0.00%
string-tagcloud.js	-1.38%
string-unpack-code.js	0.68%
string-validate-input.js	1.69%