# Formal Verification of a Family of Spilling Algorithms

Initial Bachelor Seminar Talk


Julian Rosemann

Advisors: Prof. Gert Smolka, Sigurd Schneider

Saarland University

Department of Computer Science


2016-06-10

# Concept of Spilling

$$\frac{\text{high-level language}}{\text{unbounded number of variables}} \leftrightarrow \frac{\text{assembly language}}{\text{finite number of registers}}$$

Spilling uses the memory to buffer variables

# Spilling example

| code | $r_1$ | $r_2$ | $r_3$ |
|---|---|---|---|
| … | | | |
| let z = x + y in<br>if z $\geq$ y then | | | |
| x + z<br>else<br>z | | | |

- x, y, z variables in register
- X, Y, Z variables in memory
- spill: let X=x in
- load: let x=X in
- memory variables only in loads & spills

# Spilling example

| code | $r_1$ | $r_2$ | $r_3$ |
|---|---|---|---|
| … | x | y | |
| let z = x + y in<br>if z ≥ y then | | | |
| x + z | | | |
| else<br>z | | | |

- x, y, z variables in register
- X, Y, Z variables in memory
- spill: let X=x in
- load: let x=X in
- memory variables only in loads & spills

# Spilling example

| code | $r_1$ | $r_2$ | $r_3$ |
|---|---|---|---|
| ... | x | y | |
| let z = x + y in | x | y | z |
| if z $\geq$ y then | | | |
| x + z | | | |
| else | | | |
| z | | | |

- x, y, z variables in register
- X, Y, Z variables in memory
- spill: let X=x in
- load: let x=X in
- memory variables only in loads & spills

# Spilling example

| code | $r_1$ | $r_2$ | $r_3$ |
|---|---|---|---|
| ... | x | y | |
| | | | |
| let z = x + y in | x | y | z |
| if z ≥ y then | x | y | z |
| | | | |
| x + z | | | |
| else | | | |
| z | | | |

- x, y, z variables in register
- X, Y, Z variables in memory
- spill: let X=x in
- load: let x=X in
- memory variables only in loads & spills

# Spilling example

| code | $r_1$ | $r_2$ | $r_3$ |
|---|---|---|---|
| … | x | y | |
| let X = x in | | | |
| let z = x + y in | x | y | z |
| if z ≥ y then | x | y | z |
| | | | |
| x + z | | | |
| else | | | |
| z | | | |

- x, y, z variables in register
- X, Y, Z variables in memory
- spill: let X=x in
- load: let x=X in
- memory variables only in loads & spills

# Spilling example

| code | $r_1$ | $r_2$ | $r_3$ |
|---|---|---|---|
| ... | x | y | |
| let X = x in | x | y | |
| let z = x + y in | x | y | z |
| if z ≥ y then | x | y | z |
| | | | |
| x + z | | | |
| else | | | |
| z | | | |

- x, y, z variables in register
- X, Y, Z variables in memory
- spill: let X=x in
- load: let x=X in
- memory variables only in loads & spills

# Spilling example

| code | $r_1$ | $r_2$ | $r_3$ |
|---|---|---|---|
| … | x | y | |
| let X = x in | x | y | |
| let z = x + y in | z | y | |
| if z ≥ y then | x | y | z |
| x + z | | | |
| else | | | |
| z | | | |

- x, y, z variables in register
- X, Y, Z variables in memory
- spill: let X=x in
- load: let x=X in
- memory variables only in loads & spills

# Spilling example

| code | $r_1$ | $r_2$ | $r_3$ |
|---|---|---|---|
| ... | x | y | |
| let X = x in | x | y | |
| let z = x + y in | z | y | |
| if z ≥ y then | z | y | |
|   let x = X in | | | |
|   x + z | | | |
| else | | | |
|   z | | | |

- x, y, z variables in register
- X, Y, Z variables in memory
- spill: let X=x in
- load: let x=X in
- memory variables only in loads & spills

# Spilling example

| code | $r_1$ | $r_2$ | $r_3$ |
|---|---|---|---|
| ... | x | y | |
| let X = x in | x | y | |
| let z = x + y in | z | y | |
| if z ≥ y then | z | y | |
| let x = X in | z | x | |
| x + z | | | |
| else | | | |
| z | | | |

- x, y, z variables in register
- X, Y, Z variables in memory
- spill: let X=x in
- load: let x=X in
- memory variables only in loads & spills

# Spilling example

| code | $r_1$ | $r_2$ | $r_3$ |
|---|---|---|---|
| ... | x | y | |
| let X = x in | x | y | |
| let z = x + y in | z | y | |
| if z ≥ y then | z | y | |
| let x = X in | z | x | |
| x + z | | | |
| else | | | |
| z | | | |

| code | $r_1$ | $r_2$ |
|---|---|---|
| ... | x | y |
| | | |
| let z = x + y in | | |
| | | |
| | | |
| if z ≥ y then | | |
| | | |
| x + z | | |
| else | | |
| z | | |

# Spilling example

| code | $r_1$ | $r_2$ | $r_3$ |
|---|---|---|---|
| ... | x | y | |
| let X = x in | x | y | |
| let z = x + y in | z | y | |
| if z ≥ y then | z | y | |
| let x = X in | z | x | |
| x + z | | | |
| else | | | |
| z | | | |

| code | $r_1$ | $r_2$ |
|---|---|---|
| ... | x | y |
| let Y = y in | x | y |
| let z = x + y in | | |
| | | |
| if z ≥ y then | | |
| | | |
| x + z | | |
| else | | |
| z | | |

# Spilling example

| code | $r_1$ | $r_2$ | $r_3$ |
|---|---|---|---|
| ... | x | y | |
| let X = x in | x | y | |
| let z = x + y in | z | y | |
| if z ≥ y then | z | y | |
|    let x = X in | z | x | |
|    x + z | | | |
| else | | | |
|    z | | | |

| code | $r_1$ | $r_2$ |
|---|---|---|
| ... | x | y |
| let Y = y in | x | y |
| let z = x + y in | x | z |
| | | |
| if z ≥ y then | | |
| | | |
|    x + z | | |
| else | | |
|    z | | |

# Spilling example

| code | $r_1$ | $r_2$ | $r_3$ |
|---|---|---|---|
| ... | x | y | |
| let X = x in | x | y | |
| let z = x + y in | z | y | |
| if z ≥ y then | z | y | |
| let x = X in | z | x | |
| x + z | | | |
| else | | | |
| z | | | |

| code | $r_1$ | $r_2$ |
|---|---|---|
| ... | x | y |
| let Y = y in | x | y |
| let z = x + y in | x | z |
| let X = x in | x | z |
| | | |
| if z ≥ y then | | |
| | | |
| x + z | | |
| else | | |
| z | | |

# Spilling example

| code | $r_1$ | $r_2$ | $r_3$ |
|---|---|---|---|
| ... | x | y | |
| let X = x in | x | y | |
| let z = x + y in | z | y | |
| if z ≥ y then | z | y | |
|   let x = X in | z | x | |
|   x + z | | | |
| else | | | |
|   z | | | |

| code | $r_1$ | $r_2$ |
|---|---|---|
| ... | x | y |
| let Y = y in | x | y |
| let z = x + y in | x | z |
| let X = x in | x | z |
| let y = Y in | y | z |
| if z ≥ y then | | |
| | | |
|   x + z | | |
| else | | |
|   z | | |

# Spilling example

| code | $r_1$ | $r_2$ | $r_3$ |
|---|---|---|---|
| … | x | y | |
| let X = x in | x | y | |
| let z = x + y in | z | y | |
| if z ≥ y then | z | y | |
| let x = X in | z | x | |
| x + z | | | |
| else | | | |
| z | | | |

| code | $r_1$ | $r_2$ |
|---|---|---|
| … | x | y |
| let Y = y in | x | y |
| let z = x + y in | x | z |
| let X = x in | x | z |
| let y = Y in | y | z |
| if z ≥ y then | y | z |
| | | |
| x + z | | |
| else | | |
| z | | |

# Spilling example

| code | $r_1$ | $r_2$ | $r_3$ |
|---|---|---|---|
| ... | x | y | |
| let X = x in | x | y | |
| let z = x + y in | z | y | |
| if z ≥ y then | z | y | |
| let x = X in | z | x | |
| x + z | | | |
| else | | | |
| z | | | |

| code | $r_1$ | $r_2$ |
|---|---|---|
| ... | x | y |
| let Y = y in | x | y |
| let z = x + y in | x | z |
| let X = x in | x | z |
| let y = Y in | y | z |
| if z ≥ y then | y | z |
| let x = X in | x | z |
| x + z | | |
| else | | |
| z | | |

# Properties of a good spilling algorithm

$$\texttt{spill : } \mathbb{N} \to \texttt{Statement} \to \texttt{Statement}$$

# Properties of a good spilling algorithm

$\texttt{spill :}\quad \mathbb{N} \to \texttt{Statement} \to \texttt{Statement}$

(a) **at most k registers used**

# Properties of a good spilling algorithm

$\texttt{spill :} \quad \mathbb{N} \to \texttt{Statement} \to \texttt{Statement}$

(a) at most k registers used

(b) **every variable is in a register whenever used**

# Properties of a good spilling algorithm

$\texttt{spill} : \quad \mathbb{N} \to \texttt{Statement} \to \texttt{Statement}$

(a) at most k registers used

(b) every variable is in a register whenever used

(c) **equivalence transformation**

---

# Properties of a good spilling algorithm

`spill :` $\mathbb{N} \rightarrow$ `Statement` $\rightarrow$ `Statement`

(a) at most k registers used

(b) every variable is in a register whenever used

(c) equivalence transformation

(d) **smart spilling choices – depend on application**

---

# Properties of a good spilling algorithm

$\texttt{spill} : \mathbb{N} \rightarrow \texttt{Statement} \rightarrow \texttt{Statement}$

(a) at most k registers used

(b) every variable is in a register whenever used

(c) equivalence transformation

(d) smart spilling choices – depend on application

---

Approach:

# Properties of a good spilling algorithm

$\texttt{spill} : \mathbb{N} \to \texttt{Statement} \to \texttt{Statement}$

(a) at most k registers used

(b) every variable is in a register whenever used

(c) equivalence transformation

(d) smart spilling choices – depend on application

---

Approach:

(i) **correctness predicate**

# Properties of a good spilling algorithm

`spill :` $\mathbb{N} \rightarrow$ `Statement` $\rightarrow$ `Statement`

(a) **at most k registers used**

(b) **every variable is in a register whenever used**

(c) **equivalence transformation**

(d) smart spilling choices – depend on application

---

Approach:

(i) **correctness predicate**

- guarantees (a), (b) and (c)

# Properties of a good spilling algorithm

$\texttt{spill :} \quad \mathbb{N} \to \texttt{Statement} \to \texttt{Statement}$

(a) at most k registers used
(b) every variable is in a register whenever used
(c) equivalence transformation
(d) smart spilling choices – depend on application

---

Approach:

(i) correctness predicate
- guarantees (a), (b) and (c)

(ii) **spilling algorithm**

# Properties of a good spilling algorithm

$\texttt{spill :} \quad \mathbb{N} \to \texttt{Statement} \to \texttt{Statement}$

(a) at most k registers used

(b) every variable is in a register whenever used

(c) equivalence transformation

(d) smart spilling choices – depend on application

---

Approach:

(i) correctness predicate
  - guarantees (a), (b) and (c)

(ii) **spilling algorithm**
  - satisfies (i)

# Properties of a good spilling algorithm

$\texttt{spill} : \quad \mathbb{N} \rightarrow \texttt{Statement} \rightarrow \texttt{Statement}$

(a) at most k registers used

(b) every variable is in a register whenever used

(c) equivalence transformation

(d) **smart spilling choices – depend on application**

---

Approach:

(i) correctness predicate
- guarantees (a), (b) and (c)

(ii) **spilling algorithm**
- satisfies (i)
- optimizes (d)

# Related Work – CompCert example

CompCert, a verified optimizing compiler for a C-subset

# Related Work – CompCert example

CompCert, a verified optimizing compiler for a C-subset

- *Leroy, "A formally verified compiler back-end", 2009*

# Related Work – CompCert example

CompCert, a verified optimizing compiler for a C-subset

- *Leroy, "A formally verified compiler back-end", 2009*
    - translation validation for register allocation

# Related Work – CompCert example

CompCert, a verified optimizing compiler for a C-subset

- *Leroy, "A formally verified compiler back-end", 2009*
  - translation validation for register allocation
  - verification of non-optimized spilling

# Related Work – CompCert example

CompCert, a verified optimizing compiler for a C-subset

- *Leroy, "A formally verified compiler back-end", 2009*
  - translation validation for register allocation
  - verification of non-optimized spilling
    - each variable gets either a fixed register or is **always** spilled & loaded

# Related Work – CompCert example

CompCert, a verified optimizing compiler for a C-subset

- *Leroy, "A formally verified compiler back-end", 2009*
  - translation validation for register allocation
  - verification of non-optimized spilling
    - each variable gets either a fixed register or is **always** spilled & loaded

```
...
let x = 2 in
let X = x in
let x = X in
let y = x in
let x = X in
  x
```

# Related Work – CompCert example

CompCert, a verified optimizing compiler for a C-subset

- *Leroy, "A formally verified compiler back-end", 2009*
  - translation validation for register allocation
  - verification of non-optimized spilling
    - each variable gets either a fixed register or is **always** spilled & loaded
- *Blazy, Robillard, and Appel, "Formal Verification of Coalescing Graph-Coloring Register Allocation", 2010*

```
...
let x = 2 in
let X = x in
let x = X in
let y = x in
let x = X in
  x
```

# Related Work – CompCert example

CompCert, a verified optimizing compiler for a C-subset

- *Leroy, "A formally verified compiler back-end", 2009*
  - translation validation for register allocation
  - verification of non-optimized spilling
    - each variable gets either a fixed register or is **always** spilled & loaded
- *Blazy, Robillard, and Appel, "Formal Verification of Coalescing Graph-Coloring Register Allocation", 2010*
  - verification of register allocation

```
...
let x = 2 in
let X = x in
let x = X in
let y = x in
let x = X in
  x
```

# Related Work – CompCert example

CompCert, a verified optimizing compiler for a C-subset

- *Leroy, "A formally verified compiler back-end", 2009*
  - translation validation for register allocation
  - verification of non-optimized spilling
    - each variable gets either a fixed register or is **always** spilled & loaded
- *Blazy, Robillard, and Appel, "Formal Verification of Coalescing Graph-Coloring Register Allocation", 2010*
  - verification of register allocation
- *Rideau and Leroy, "Validating Register Allocation and Spilling", 2010*

```
...
let x = 2 in
let X = x in
let x = X in
let y = x in
let x = X in
  x
```

# Related Work – CompCert example

CompCert, a verified optimizing compiler for a C-subset

- *Leroy, "A formally verified compiler back-end", 2009*
  - translation validation for register allocation
  - verification of non-optimized spilling
    - each variable gets either a fixed register or is **always** spilled & loaded
- *Blazy, Robillard, and Appel, "Formal Verification of Coalescing Graph-Coloring Register Allocation", 2010*
  - verification of register allocation
- *Rideau and Leroy, "Validating Register Allocation and Spilling", 2010*
  - validator for spilling – enables use of sophisticated spilling techniques as in *Braun and Hack, "Register Spilling and Live-Range Splitting for SSA-Form Programs", 2009*

```
...
let x = 2 in
let X = x in
let x = X in
let y = x in
let x = X in
  x
```

# Related Work

- In a functional language in SSA-form spilling can be separated from register allocation
  see: *Hack, Grund, and Goos, "Register Allocation for Programs in SSA-Form", 2006*

# Related Work

- In a functional language in SSA-form spilling can be separated from register allocation
  see: *Hack, Grund, and Goos, "Register Allocation for Programs in SSA-Form", 2006*
- Thesis is continuation of the RIL *Klitzke, "Verification of a Spilling Algorithms in Coq", 2015*

# IL

$$s, \ t \ ::=$$

# IL

$$s,\ t\ ::=\ \mathsf{let}\ x = e\ \mathsf{in}\ s$$

# IL

$$
\begin{aligned}
s,\ t \ ::= &\ \mathsf{let}\ x = e\ \mathsf{in}\ s \\
\mid &\ \mathsf{if}\ e\ \mathsf{then}\ s\ \mathsf{else}\ t
\end{aligned}
$$

# IL

$$
\begin{aligned}
s,\ t \ ::= \ & \texttt{let } x = e \texttt{ in } s \\
\mid \ & \texttt{if } e \texttt{ then } s \texttt{ else } t \\
\mid \ & e
\end{aligned}
$$

# IL

$$
\begin{aligned}
\mathsf{s},\ \mathsf{t} ::= {} & \mathsf{let}\ \mathsf{x} = \mathsf{e}\ \mathsf{in}\ \mathsf{s} \\
\mid {} & \mathsf{if}\ \mathsf{e}\ \mathsf{then}\ \mathsf{s}\ \mathsf{else}\ \mathsf{t} \\
\mid {} & \mathsf{e} \\
\mid {} & \mathsf{fun}\ \mathsf{f}\ \overline{x} = \mathsf{s}\ \mathsf{in}\ \mathsf{t}
\end{aligned}
$$

# IL

$$
\begin{aligned}
s, t ::= \; & \text{let } x = e \text{ in } s \\
\mid \; & \text{if } e \text{ then } s \text{ else } t \\
\mid \; & e \\
\mid \; & \text{fun } f \; \overline{x} = s \text{ in } t \\
\mid \; & f \; \overline{x}
\end{aligned}
$$

# IL

$$
\begin{aligned}
\mathsf{s},\ \mathsf{t} ::= {} & \mathsf{let}\ \mathsf{x} = \mathsf{e}\ \mathsf{in}\ \mathsf{s} \\
\mid {} & \mathsf{if}\ \mathsf{e}\ \mathsf{then}\ \mathsf{s}\ \mathsf{else}\ \mathsf{t} \\
\mid {} & \mathsf{e} \\
\mid {} & \mathsf{fun}\ \mathsf{f}\ \overline{x} = \mathsf{s}\ \mathsf{in}\ \mathsf{t} \\
\mid {} & \mathsf{f}\ \overline{x}
\end{aligned}
$$

Formally described in *Schneider, Smolka, and Hack, "A First-Order Functional Intermediate Language for Verified Compilers", 2015*

# Liveness

- A variable $x$ is **significant** in statement $s :\Leftrightarrow \exists$ values $v_0, v_1$, where $s$ behaves differently for $x = v_0$ and $x = v_1$.

# Liveness

- A variable $x$ is **significant** in statement $s :\Leftrightarrow \exists$ values $v_0, v_1$, where $s$ behaves differently for $x = v_0$ and $x = v_1$.
- Significance of variables is undecidable

# Liveness

- A variable $x$ is **significant** in statement $s :\Leftrightarrow \exists$ values $v_0, v_1$, where $s$ behaves differently for $x = v_0$ and $x = v_1$.
- Significance of variables is undecidable
- Liveness intuition: variable x is **live** in statement $s$ if its value is used in $s$.

# Liveness

- A variable $x$ is **significant** in statement $s :\Leftrightarrow \exists$ values $v_0, v_1$, where $s$ behaves differently for $x = v_0$ and $x = v_1$.
- Significance of variables is undecidable
- Liveness intuition: variable x is **live** in statement $s$ if its value is used in $s$.
- Live variables overapproximate significant variables

# Liveness

- A variable $x$ is **significant** in statement $s :\Leftrightarrow \exists$ values $v_0, v_1$, where $s$ behaves differently for $x = v_0$ and $x = v_1$.
- Significance of variables is undecidable
- Liveness intuition: variable x is **live** in statement $s$ if its value is used in $s$.
- Live variables overapproximate significant variables
- Live variables are efficiently computable

# Liveness – example

| code | live variables |
|---|---|
| let y = z in<br>if x ≥ 0<br>   then y<br>   else z | |

- A variable $x$ is **significant** in statement $s$ $:\Leftrightarrow$ $\exists$ values $v_0, v_1$, where $s$ behaves differently for $x = v_0$ and $x = v_1$.
- Liveness intuition: variable x is **live** in statement $s$ if its value is used in $s$.

# Liveness – example

| code | live variables |
|---|---|
| let y = z in | |
| if x $\geq$ 0 | |
| then y | |
| else z | $\{z\}$ |

- A variable $x$ is **significant** in statement $s$ $:\Leftrightarrow$ $\exists$ values $v_0, v_1$, where $s$ behaves differently for $x = v_0$ and $x = v_1$.
- Liveness intuition: variable x is **live** in statement $s$ if its value is used in $s$.

# Liveness – example

| code | live variables |
|---|---|
| let y = z in | |
| if x $\geq$ 0 | |
| then y | $\{y\}$ |
| else z | $\{z\}$ |

- A variable $x$ is **significant** in statement $s$ $:\Leftrightarrow \exists$ values $v_0, v_1$, where $s$ behaves differently for $x = v_0$ and $x = v_1$.
- Liveness intuition: variable x is **live** in statement $s$ if its value is used in $s$.

# Liveness – example

| code | live variables |
| --- | --- |
| let y = z in | |
| if x $\geq$ 0 | {x, y, z} |
| then y | {y} |
| else z | {z} |

- A variable $x$ is **significant** in statement $s :\Leftrightarrow \exists$ values $v_0, v_1$, where $s$ behaves differently for $x = v_0$ and $x = v_1$.
- Liveness intuition: variable x is **live** in statement $s$ if its value is used in $s$.

# Liveness – example

| code | live variables |
|---|---|
| let y = z in | $\{x, z\}$ |
| if $x \geq 0$ | $\{x, y, z\}$ |
| then y | $\{y\}$ |
| else z | $\{z\}$ |

- A variable $x$ is **significant** in statement $s$ $:\Leftrightarrow \exists$ values $v_0, v_1$, where $s$ behaves differently for $x = v_0$ and $x = v_1$.
- Liveness intuition: variable x is **live** in statement $s$ if its value is used in $s$.

# Liveness – example

| code | live variables |
|---|---|
| let y = z in | {**x**, z} |
| if x $\geq$ 0 | {x, y, z} |
| then y | {y} |
| else z | {z} |

- A variable $x$ is **significant** in statement $s$ $:\Leftrightarrow \exists$ values $v_0, v_1$, where $s$ behaves differently for $x = v_0$ and $x = v_1$.
- Liveness intuition: variable x is **live** in statement $s$ if its value is used in $s$.

# Liveness – example

| code | live variables |
|---|---|
| let y = z in | $\{\mathbf{x}, z\}$ |
| if $x \geq 0$ | $\{x, y, z\}$ |
| then y | $\{y\}$ |
| else z | $\{z\}$ |

let y = z
|
if $x \geq 0$
y    z

- A variable $x$ is **significant** in statement $s$ $:\Leftrightarrow \exists$ values $v_0, v_1$, where $s$ behaves differently for $x = v_0$ and $x = v_1$.
- Liveness intuition: variable x is **live** in statement $s$ if its value is used in $s$.

# Liveness – example

| code | live variables |
|---|---|
| let y = z in | {**x**, z} |
| if x ≥ 0 | {x, y, z} |
| then y | {y} |
| else z | {z} |


- A variable $x$ is **significant** in statement $s$ :⇔ ∃ values $v_0, v_1$, where $s$ behaves differently for $x = v_0$ and $x = v_1$.
- Liveness intuition: variable x is **live** in statement $s$ if its value is used in $s$.

# Spilling

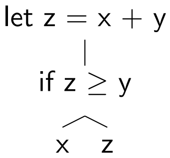
| code | $S$ | $L$ |
|---|---|---|
| ... | | |
| let X = x in | | |
| let z = x + y in | | |
| if z ≥ y then | | |
|    let x = X in | | |
|    x | | |
| else | | |
|    z | | |

# Spilling

| code | $S$ | $L$ |
|---|---|---|
| ... | | |
| let X = x in | | |
| let z = x + y in | {x} | {} |
| if z ≥ y then | {} | {} |
| let x = X in | | |
| x | {} | {X} |
| else | | |
| z | {} | {} |

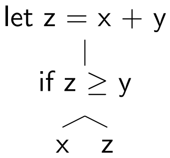
# Spilling

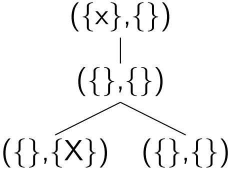
| code | $S$ | $L$ |
|---|---|---|
| ... | | |
| let X = x in | | |
| let z = x + y in | {x} | {} |
| if z ≥ y then | {} | {} |
|    let x = X in | | |
|    x | {} | {X} |
| else | | |
|    z | {} | {} |



# Spilling

| code | $S$ | $L$ |
|---|---|---|
| ... | | |
| let X = x in | | |
| let z = x + y in | {x} | {} |
| if z ≥ y then | {} | {} |
|    let x = X in | | |
|    x | {} | {X} |
| else | | |
|    z | {} | {} |



# doSpill

$$\text{doSpill} : \text{Statement} \rightarrow \text{Tree} \ (\text{Set } \mathcal{V} * \text{Set } \mathcal{V}) \rightarrow \text{Statement}$$

# doSpill

$$\text{doSpill} : \text{Statement} \to \text{Tree } (\text{Set } \mathcal{V} * \text{Set } \mathcal{V}) \to \text{Statement}$$

$$(s, (\underbrace{\{x_1, ..., x_n\}}_{\text{spills}}, \underbrace{\{Y_1, ... Y_m\}}_{\text{loads}})) \mapsto \begin{array}{c} \textit{let } X_1 = x_1 \textit{ in} \ldots \textit{let } X_n = x_n \textit{ in} \\ \textit{let } y_1 = Y_1 \textit{ in} \ldots \textit{let } y_m = Y_m \textit{ in} \\ s \end{array}$$

# Properties of a good spilling algorithm

`spill :` $\mathbb{N} \rightarrow$ `Statement` $\rightarrow$ `Statement`

(a) **at most k registers used**

(b) **every variable is in a register whenever used**

(c) **equivalence transformation**

(d) smart spilling choices – depend on application

---

Approach:

(i) **correctness predicate**
- guarantees (a), (b) and (c)

(ii) spilling algorithm
- satisfies (i)
- optimizes (d)

# Inductive Correctness Predicate

$$(R, M) \vdash spill_k\ s\ lv\ \ : sl$$

| |
|---|
| $x \in R$ |
| $x \in M$ |
| $s$ |
| $lv$ |
| $sl$ |

# Inductive Correctness Predicate

$$(R, M) \vdash spill_k \ s \ lv \ : sl$$

| | |
|---|---|
| $x \in R$ | :⇔ current value is in a register |
| $x \in M$ | |
| $s$ | |
| $lv$ | |
| $sl$ | |

# Inductive Correctness Predicate

$$(R, M) \vdash spill_k \ s \ lv \ \ : sl$$

| | |
|---|---|
| $x \in R$ | :⇔ current value is in a register |
| $x \in M$ | :⇔ current value is in the memory |
| $s$ | |
| $lv$ | |
| $sl$ | |

# Inductive Correctness Predicate

$$(R, M) \vdash spill_k \; s \; lv \;\; : sl$$

| | |
|---|---|
| $x \in R$ | $:\Leftrightarrow$ current value is in a register |
| $x \in M$ | $:\Leftrightarrow$ current value is in the memory |
| $s$ | statement |
| $lv$ | |
| $sl$ | |

# Inductive Correctness Predicate

$$(R, M) \vdash spill_k \ s \ lv \ : sl$$

| | |
|---|---|
| $x \in R$ | :⇔ current value is in a register |
| $x \in M$ | :⇔ current value is in the memory |
| $s$ | statement |
| $lv$ | liveness information |
| $sl$ | |

# Inductive Correctness Predicate

$$(R, M) \vdash spill_k\ s\ lv\ \ : sl$$

| | |
|---|---|
| $x \in R$ | :⇔ current value is in a register |
| $x \in M$ | :⇔ current value is in the memory |
| $s$ | statement |
| $lv$ | liveness information |
| $sl$ | spill/load information |

# Inductive Correctness Predicate

$$(R, M) \vdash \mathit{spill}_k\ s\ \mathit{lv}\ : \mathit{sl}$$

| | |
|---|---|
| $x \in R$ | $:\Leftrightarrow$ current value is in a register |
| $x \in M$ | $:\Leftrightarrow$ current value is in the memory |
| $s$ | statement |
| $\mathit{lv}$ | liveness information |
| $\mathit{sl}$ | spill/load information |

- read: $\mathit{sl}$ is a correct $k$-spilling of $s$ with liveness $\mathit{lv}$ on $R$ and $M$

# Inductive Correctness Predicate

$$(R, M) \vdash spill_k\ s\ lv\ \ : sl$$

| | |
|---|---|
| $x \in R$ | :⇔ current value is in a register |
| $x \in M$ | :⇔ current value is in the memory |
| $s$ | statement |
| $lv$ | liveness information |
| $sl$ | spill/load information |

- read: $sl$ is a correct $k$-spilling of $s$ with liveness $lv$ on $R$ and $M$
- **not** neccessarily $R \cap M = \emptyset$

# Inductive Correctness Predicate

$$(R, M) \vdash spill_k\ s\ lv\ \ : sl$$

| | |
|---|---|
| $x \in R$ | :⇔ current value is in a register |
| $x \in M$ | :⇔ current value is in the memory |
| $s$ | statement |
| $lv$ | liveness information |
| $sl$ | spill/load information |

- read: $sl$ is a correct $k$-spilling of $s$ with liveness $lv$ on $R$ and $M$
- **not** neccessarily $R \cap M = \emptyset$
- $lv$, $sl$ and the abstract syntax tree of $s$ have the same shape

# Correctness predicate for `let x=e in`

| execution step | register state | memory state |
|---|---|---|
| ... | $R$ | $M$ |

$$(let\ x = e\ in\ s, (\underbrace{\{x_1, ..., x_n\}}_{\text{spills}}, \underbrace{\{Y_1, ...Y_m\}}_{\text{loads}})) \mapsto \begin{array}{l} let\ X_1 = x_1\ in \ldots \\ let\ y_1 = Y_1\ in \ldots \\ let\ x = e\ in\ s \end{array}$$

# Correctness predicate for `let x=e in`

| execution step | register state | memory state |
|---|---|---|
| ... | $R$ | $M$ |
| spill S | | |

$$(let\ x = e\ in\ s, (\underbrace{\{x_1, ..., x_n\}}_{\text{spills}}, \underbrace{\{Y_1, ...Y_m\}}_{\text{loads}})) \mapsto \begin{array}{l} let\ X_1 = x_1\ in \ldots \\ let\ y_1 = Y_1\ in \ldots \\ let\ x = e\ in\ s \end{array}$$

# Correctness predicate for `let x=e in`

| execution step | register state | memory state |
|---|---|---|
| ... | $R$ | $M$ |
| spill S | | |
| load L | | |

$$(let\ x = e\ in\ s, (\underbrace{\{x_1, ..., x_n\}}_{\text{spills}}, \underbrace{\{Y_1, ...Y_m\}}_{\text{loads}})) \mapsto \begin{array}{l} let\ X_1 = x_1\ in \ldots \\ let\ y_1 = Y_1\ in \ldots \\ let\ x = e\ in\ s \end{array}$$

# Correctness predicate for `let x=e in`

| execution step | register state | memory state |
|---|---|---|
| ... | $R$ | $M$ |
| spill S | | |
| load L | | |
| eval e | | |

$$(let\ x = e\ in\ s, (\underbrace{\{x_1, ..., x_n\}}_{\text{spills}}, \underbrace{\{Y_1, ...Y_m\}}_{\text{loads}})) \mapsto \begin{array}{l} let\ X_1 = x_1\ in \ldots \\ let\ y_1 = Y_1\ in \ldots \\ let\ x = e\ in\ s \end{array}$$

# Correctness predicate for `let x=e in`

| execution step | register state | memory state |
|---|---|---|
| ... | $R$ | $M$ |
| spill S | | |
| load L | | |
| eval e | | |
| store x | | |

$$(let\ x = e\ in\ s, (\underbrace{\{x_1, ..., x_n\}}_{\text{spills}}, \underbrace{\{Y_1, ...Y_m\}}_{\text{loads}})) \mapsto \begin{array}{l} let\ X_1 = x_1\ in \ldots \\ let\ y_1 = Y_1\ in \ldots \\ let\ x = e\ in\ s \end{array}$$

# Correctness predicate for `let x=e in`

| execution step | register state | memory state |
|---|---|---|
| ... | $R$ | $M$ |
| spill S | | |
| load L | | |
| eval e | | |
| store x | | |

(a) size of R is bounded by k

(b) variables are in R when needed

(c) program equivalence

# Correctness predicate for `let x=e in`

| execution step | register state | memory state |
|---|---|---|
| ... | $R$ | $M$ |
| spill S | | |
| load L | | |
| eval e | | |
| store x | | |

$$\frac{}{(R, M) \vdash spill_k\ (let\ x = e\ in\ s)\ (lv \cdot lv_s)\ \ : (S, L) \cdot sl_s}$$

# Correctness predicate for `let x=e in`

| execution step | register state | memory state |
|---|---|---|
| ... | $R$ | $M$ |
| spill S | $R$ | $M \cup S$ |
| load L | | |
| eval e | | |
| store x | | |

$$\frac{}{(R, M) \vdash spill_k \ (let\ x = e\ in\ s)\ (lv \cdot lv_s)\ \ : (S, L) \cdot sl_s}$$

# Correctness predicate for `let x=e in`

| execution step | register state | memory state |
|---|---|---|
| ... | $R$ | $M$ |
| spill S | $R$ | $M \cup S$ |
| load L | | $M \cup S$ |
| eval e | | $M \cup S$ |
| store x | | $M \cup S$ |

$$\frac{}{(R, M) \vdash spill_k \ (let \ x = e \ in \ s) \ (lv \cdot lv_s) \ \ : (S, L) \cdot sl_s}$$

# Correctness predicate for `let x=e in`

| execution step | register state | memory state |
|---|---|---|
| ... | $R$ | $M$ |
| spill S | $R$ | $M \cup S$ |
| load L | $R \setminus K \cup L =: R_e$ | $M \cup S$ |
| eval e | | $M \cup S$ |
| store x | | $M \cup S$ |

$$\frac{}{(R, M) \vdash spill_k \; (let \; x = e \; in \; s) \; (lv \cdot lv_s) \;\; : (S, L) \cdot sl_s}$$

# Correctness predicate for `let x=e in`

| execution step | register state | memory state |
|---|---|---|
| ... | $R$ | $M$ |
| spill S | $R$ | $M \cup S$ |
| load L | $R \setminus K \cup L =: R_e$ | $M \cup S$ |
| eval e | | $M \cup S$ |
| store x | | $M \cup S$ |

$$\frac{|R_e| \leq k}{(R, M) \vdash spill_k \ (let \ x = e \ in \ s) \ (lv \cdot lv_s) \ : (S, L) \cdot sl_s}$$

# Correctness predicate for `let x=e in`

| execution step | register state | memory state |
| --- | --- | --- |
| ... | $R$ | $M$ |
| spill S | $R$ | $M \cup S$ |
| load L | $R \setminus K \cup L =: R_e$ | $M \cup S$ |
| eval e | $R \setminus K \cup L$ | $M \cup S$ |
| store x | | $M \cup S$ |

$$\frac{|R_e| \leq k}{(R, M) \vdash spill_k \ (let \ x = e \ in \ s) \ (lv \cdot lv_s) \ : (S, L) \cdot sl_s}$$

# Correctness predicate for `let x=e in`

| execution step | register state | memory state |
| --- | --- | --- |
| ... | $R$ | $M$ |
| spill S | $R$ | $M \cup S$ |
| load L | $R \setminus K \cup L =: R_e$ | $M \cup S$ |
| eval e | $R \setminus K \cup L$ | $M \cup S$ |
| store x | | $M \cup S$ |

$$\frac{|R_e| \leq k \qquad\qquad fv(e) \subseteq R_e}{(R, M) \vdash spill_k\ (let\ x = e\ in\ s)\ (lv \cdot lv_s)\ : (S, L) \cdot sl_s}$$

# Correctness predicate for `let x=e in`

| execution step | register state | memory state |
|---|---|---|
| ... | $R$ | $M$ |
| spill S | $R$ | $M \cup S$ |
| load L | $R \setminus K \cup L =: R_e$ | $M \cup S$ |
| eval e | $R \setminus K \cup L$ | $M \cup S$ |
| store x | $(R \setminus K \cup L) \setminus K_x \cup \{x\} =: R_s$ | $M \cup S$ |

$$\frac{|R_e| \leq k \qquad\qquad fv(e) \subseteq R_e}{(R, M) \vdash spill_k\ (let\ x = e\ in\ s)\ (lv \cdot lv_s)\ \ : (S, L) \cdot sl_s}$$

# Correctness predicate for `let x=e in`

| execution step | register state | memory state |
|---|---|---|
| ... | $R$ | $M$ |
| spill S | $R$ | $M \cup S$ |
| load L | $R \setminus K \cup L =: R_e$ | $M \cup S$ |
| eval e | $R \setminus K \cup L$ | $M \cup S$ |
| store x | $(R \setminus K \cup L) \setminus K_x \cup \{x\} =: R_s$ | $M \cup S$ |

$$\frac{\begin{array}{ll} |R_e| \leq k & fv(e) \subseteq R_e \\ |R_s| \leq k & \end{array}}{(R, M) \vdash spill_k \ (let\ x = e\ in\ s)\ (lv \cdot lv_s) \ : (S, L) \cdot sl_s}$$

# Correctness predicate for `let x=e in`

| execution step | register state | memory state |
|---|---|---|
| ... | $R$ | $M$ |
| spill S | $R$ | $M \cup S$ |
| load L | $R \setminus K \cup L =: R_e$ | $M \cup S$ |
| eval e | $R \setminus K \cup L$ | $M \cup S$ |
| store x | $(R \setminus K \cup L) \setminus K_x \cup \{x\} =: R_s$ | $M \cup S$ |

$$\frac{\begin{array}{cc} |R_e| \leq k & fv(e) \subseteq R_e \\ |R_s| \leq k & (R_s, M \cup S) \vdash spill_k\ s\ lv_s\ : sl_s \end{array}}{(R, M) \vdash spill_k\ (let\ x = e\ in\ s)\ (lv \cdot lv_s)\ : (S, L) \cdot sl_s}$$

# Correctness predicate for `let x=e in` – example

| code | $S$ | $L$ | $K$ | $K_z$ |
|---|---|---|---|---|
| let W = w in<br>let y = Y in<br>let z = x + y in<br>let w = W in<br>  w + z | | | | |

# Correctness predicate for `let x=e in` – example

| code | $S$ | $L$ | $K$ | $K_z$ |
|---|---|---|---|---|
| let W = w in | | | | |
| let y = Y in | | | | |
| let z = x + y in | $\{w\}$ | $\{y\}$ | | |
| let w = W in | | | | |
| w + z | $\{\}$ | $\{w\}$ | | |

# Correctness predicate for `let x=e in` – example

$(R,M) := (\{w,x\}, \{y\})$

| code | $S$ | $L$ | $K$ | $K_z$ |
|---|---|---|---|---|
| let W = w in | | | | |
| let y = Y in | | | | |
| let z = x + y in | {w} | {y} | | |
| let w = W in | | | | |
| w + z | {} | {w} | | |

# Correctness predicate for `let x=e in` – example

$$(R,M) := (\{w,x\}, \{y\})$$

| code | $S$ | $L$ | $K$ | $K_z$ |
|---|---|---|---|---|
| let W = w in | | | | |
| let y = Y in | | | | |
| **let z = x + y in** | {w} | {y} | | |
| let w = W in | | | | |
| w + z | {} | {w} | | |

$$\frac{\begin{array}{ll} |\ R_e\ | \leq 2 & fv(x+y) \subseteq R_e \\ |\ R_s\ | \leq 2 & (\ R_s\ , \{w,y\}) \vdash spill_2\ (w+z)\ lv_s\ : sl_s \end{array}}{(\{w,x\},\{y\}) \vdash spill_2\ (let\ z = x+y\ in\ w+z)\ (lv \cdot lv_s)\ : (\{w\},\{y\}) \cdot sl_s}$$

# Correctness predicate for `let x=e in` – example

$$(R,M) := (\{w,x\}, \{y\})$$

| code | $S$ | $L$ | $K$ | $K_z$ |
|---|---|---|---|---|
| let W = w in | | | | |
| let y = Y in | | | | |
| **let z = x + y in** | {w} | {y} | | |
| let w = W in | | | | |
| w + z | {} | {w} | | |

$$\frac{\begin{array}{ll} |\ R_e\ | \leq 2 & fv(x+y) \subseteq R_e \\ |\ R_s\ | \leq 2 & (\ R_s\ , \{w,y\}) \vdash spill_2\ (w+z)\ lv_s\ : sl_s \end{array}}{(\{w,x\},\{y\}) \vdash spill_2\ (let\ z = x+y\ in\ w+z)\ (lv \cdot lv_s)\ : (\{w\},\{y\}) \cdot sl_s}$$

$$R_e := R \setminus K \cup L$$

$$R_s := R_e \setminus K_z \cup \{z\}$$

# Correctness predicate for `let x=e in` – example

(R,M) := ({w,x}, {y})

| code | $S$ | $L$ | $K$ | $K_z$ |
|---|---|---|---|---|
| let W = w in | | | | |
| let y = Y in | | | | |
| **let z = x + y in** | {w} | {y} | | |
| let w = W in | | | | |
| w + z | {} | {w} | | |

$$\frac{\begin{array}{ll} |\ R_e\ | \leq 2 & fv(x+y) \subseteq R_e \\ |\ R_s\ | \leq 2 & (\ R_s\ , \{w,y\}) \vdash spill_2\ (w+z)\ lv_s\ : sl_s \end{array}}{(\{w,x\},\{y\}) \vdash spill_2\ (let\ z = x+y\ in\ w+z)\ (lv \cdot lv_s)\ : (\{w\},\{y\}) \cdot sl_s}$$

$$R_e := R \setminus K \cup L = \{w,x\} \setminus K \cup \{y\}$$

$$R_s := R_e \setminus K_z \cup \{z\}$$

# Correctness predicate for `let x=e in` – example

(R,M) := ({w,x}, {y})

| code | $S$ | $L$ | $K$ | $K_z$ |
|---|---|---|---|---|
| let W = w in | | | | |
| let y = Y in | | | | |
| **let z = x + y in** | {w} | {y} | {w} | |
| let w = W in | | | | |
| w + z | {} | {w} | | |

$$\frac{|\ R_e\ | \leq 2 \qquad fv(x+y) \subseteq R_e \qquad |\ R_s\ | \leq 2 \quad (\ R_s\ , \{w,y\}) \vdash spill_2\ (w+z)\ lv_s\ : sl_s}{(\{w,x\},\{y\}) \vdash spill_2\ (let\ z = x+y\ in\ w+z)\ (lv \cdot lv_s)\ : (\{w\},\{y\}) \cdot sl_s}$$

$$R_e := R \setminus K \cup L = \{w,x\} \setminus K \cup \{y\}$$

$$R_s := R_e \setminus K_z \cup \{z\}$$

# Correctness predicate for `let x=e in` – example

(R,M) := ({w,x}, {y})

| code | $S$ | $L$ | $K$ | $K_z$ |
|---|---|---|---|---|
| let W = w in | | | | |
| let y = Y in | | | | |
| **let z = x + y in** | {w} | {y} | {w} | |
| let w = W in | | | | |
| w + z | {} | {w} | | |

$$\frac{\begin{array}{l} |\ R_e\ | \leq 2 \qquad\qquad fv(x+y) \subseteq R_e \\ |\ R_s\ | \leq 2 \quad (\ R_s\ , \{w,y\}) \vdash spill_2\ (w+z)\ lv_s\ : sl_s \end{array}}{(\{w,x\},\{y\}) \vdash spill_2\ (let\ z = x+y\ in\ w+z)\ (lv \cdot lv_s)\ : (\{w\},\{y\}) \cdot sl_s}$$

$$R_e := R \setminus K \cup L = \{w,x\} \setminus K \cup \{y\} = \{x,y\}$$

$$R_s := R_e \setminus K_z \cup \{z\}$$

# Correctness predicate for `let x=e in` – example

(R,M) := ({w,x}, {y})

| code | $S$ | $L$ | $K$ | $K_z$ |
|---|---|---|---|---|
| let W = w in | | | | |
| let y = Y in | | | | |
| **let z = x + y in** | {w} | {y} | {w} | |
| let w = W in | | | | |
| w + z | {} | {w} | | |

$$\frac{|\{x,y\}| \leq 2 \qquad fv(x+y) \subseteq \{x,y\} \qquad | \quad R_s \quad | \leq 2 \qquad ( \quad R_s \quad , \{w,y\}) \vdash spill_2 \; (w+z) \; lv_s \;\; : sl_s}{(\{w,x\},\{y\}) \vdash spill_2 \; (let\ z = x + y\ \ in\ w + z) \; (lv \cdot lv_s) \;\; : (\{w\},\{y\}) \cdot sl_s}$$

$$R_e := R \setminus K \cup L = \{w,x\} \setminus K \cup \{y\} = \{x,y\}$$

$$R_s := R_e \setminus K_z \cup \{z\}$$

# Correctness predicate for `let x=e in` – example

(R,M) := ({w,x}, {y})

| code | $S$ | $L$ | $K$ | $K_z$ |
|---|---|---|---|---|
| let W = w in | | | | |
| let y = Y in | | | | |
| **let z = x + y in** | {w} | {y} | {w} | |
| let w = W in | | | | |
| w + z | {} | {w} | | |

$$\dfrac{|\{x,y\}| \leq 2 \qquad fv(x+y) \subseteq \{x,y\} \qquad | \ R_s \ | \leq 2 \quad ( \ R_s \ , \{w,y\}) \vdash spill_2 \ (w+z) \ lv_s \ : sl_s}{(\{w,x\},\{y\}) \vdash spill_2 \ (let \ z = x+y \ in \ w+z) \ (lv \cdot lv_s) \ : (\{w\},\{y\}) \cdot sl_s}$$

$$R_e := R \setminus K \cup L = \{w,x\} \setminus K \cup \{y\} = \{x,y\}$$

$$R_s := R_e \setminus K_z \cup \{z\}$$

# Correctness predicate for `let x=e in` – example

(R,M) := ({w,x}, {y})

| code | $S$ | $L$ | $K$ | $K_z$ |
|---|---|---|---|---|
| let W = w in | | | | |
| let y = Y in | | | | |
| **let z = x + y in** | {w} | {y} | {w} | |
| let w = W in | | | | |
| w + z | {} | {w} | | |

$$\dfrac{|\{x,y\}| \leq 2 \qquad\qquad fv(x+y) \subseteq \{x,y\} \\ |\ R_s\ | \leq 2 \quad (\ R_s\ , \{w,y\}) \vdash spill_2\ (w+z)\ lv_s\ : sl_s}{(\{w,x\},\{y\}) \vdash spill_2\ (let\ z = x+y\ in\ w+z)\ (lv \cdot lv_s)\ : (\{w\},\{y\}) \cdot sl_s}$$

$$R_e := R \setminus K \cup L = \{w,x\} \setminus K \cup \{y\} = \{x,y\}$$

$$R_s := R_e \setminus K_z \cup \{z\}$$

# Correctness predicate for `let x=e in` – example

(R,M) := ({w,x}, {y})

| code | $S$ | $L$ | $K$ | $K_z$ |
|---|---|---|---|---|
| let W = w in | | | | |
| let y = Y in | | | | |
| **let z = x + y in** | {w} | {y} | {w} | |
| let w = W in | | | | |
| w + z | {} | {w} | | |

$$\frac{\overset{|\{x,y\}| \leq 2}{|\quad R_s \quad| \leq 2} \qquad \overset{fv(x+y) \subseteq \{x,y\}}{(\quad R_s \quad, \{w,y\}) \vdash spill_2\ (w+z)\ lv_s\ \ : sl_s}}{(\{w,x\},\{y\}) \vdash spill_2\ (let\ z = x+y\ \ in\ w+z)\ (lv \cdot lv_s)\ \ : (\{w\},\{y\}) \cdot sl_s}$$

$$R_e := R \setminus K \cup L = \{w,x\} \setminus K \cup \{y\} = \{x,y\}$$

$$R_s := R_e \setminus K_z \cup \{z\} = \{x,y\} \setminus K_z \cup \{z\}$$

# Correctness predicate for `let x=e in` – example

$$(R,M) := (\{w,x\}, \{y\})$$

| code | $S$ | $L$ | $K$ | $K_z$ |
|---|---|---|---|---|
| let W = w in | | | | |
| let y = Y in | | | | |
| **let z = x + y in** | {w} | {y} | {w} | {y} |
| let w = W in | | | | |
| w + z | {} | {w} | | |

$$\frac{\overset{|\{x,y\}| \leq 2}{|\ \ R_s\ \ | \leq 2} \qquad \overset{fv(x+y) \subseteq \{x,y\}}{(\ \ R_s\ \ , \{w,y\}) \vdash spill_2\ (w+z)\ lv_s\ \ : sl_s}}{(\{w,x\},\{y\}) \vdash spill_2\ (let\ z = x+y\ in\ w+z)\ (lv \cdot lv_s)\ \ : (\{w\},\{y\}) \cdot sl_s}$$

$$R_e := R \setminus K \cup L = \{w,x\} \setminus K \cup \{y\} = \{x,y\}$$

$$R_s := R_e \setminus K_z \cup \{z\} = \{x,y\} \setminus K_z \cup \{z\}$$

# Correctness predicate for `let x=e in` – example

(R,M) := ({w,x}, {y})

| code | $S$ | $L$ | $K$ | $K_z$ |
|---|---|---|---|---|
| let W = w in | | | | |
| let y = Y in | | | | |
| **let z = x + y in** | {w} | {y} | {w} | {y} |
| let w = W in | | | | |
| w + z | {} | {w} | | |

$$\frac{\overset{|\{x,y\}| \leq 2}{|\ R_s\ | \leq 2} \quad \overset{fv(x+y) \subseteq \{x,y\}}{(\ R_s\ , \{w,y\}) \vdash spill_2\ (w+z)\ lv_s\ : sl_s}}{(\{w,x\},\{y\}) \vdash spill_2\ (let\ z = x+y\ in\ w+z)\ (lv \cdot lv_s)\ : (\{w\},\{y\}) \cdot sl_s}$$

$$R_e := R \setminus K \cup L = \{w,x\} \setminus K \cup \{y\} = \{x,y\}$$

$$R_s := R_e \setminus K_z \cup \{z\} = \{x,y\} \setminus K_z \cup \{z\} = \{x,z\}$$

# Correctness predicate for `let x=e in` – example

(R,M) := ({w,x}, {y})

| code | $S$ | $L$ | $K$ | $K_z$ |
|---|---|---|---|---|
| let W = w in | | | | |
| let y = Y in | | | | |
| **let z = x + y in** | {w} | {y} | {w} | {y} |
| let w = W in | | | | |
| w + z | {} | {w} | | |

$$\frac{|\{x,y\}| \leq 2 \qquad fv(x+y) \subseteq \{x,y\} \qquad |\{x,z\}| \leq 2 \quad (\{x,z\},\{w,y\}) \vdash spill_2\ (w+z)\ lv_s\ :sl_s}{(\{w,x\},\{y\}) \vdash spill_2\ (let\ z = x+y\ in\ w+z)\ (lv \cdot lv_s)\ :(\{w\},\{y\}) \cdot sl_s}$$

$$R_e := R \setminus K \cup L = \{w,x\} \setminus K \cup \{y\} = \{x,y\}$$

$$R_s := R_e \setminus K_z \cup \{z\} = \{x,y\} \setminus K_z \cup \{z\} = \{x,z\}$$

# Correctness predicate for `let x=e in` – example

$$(R,M) := (\{w,x\}, \{y\})$$

| code | $S$ | $L$ | $K$ | $K_z$ |
|---|---|---|---|---|
| let W = w in | | | | |
| let y = Y in | | | | |
| **let z = x + y in** | {w} | {y} | {w} | {y} |
| let w = W in | | | | |
| w + z | {} | {w} | | |

$$\frac{|\{x,y\}| \leq 2 \qquad fv(x+y) \subseteq \{x,y\} \qquad |\{x,z\}| \leq 2 \qquad (\{x,z\},\{w,y\}) \vdash spill_2\ (w+z)\ lv_s\ : sl_s}{(\{w,x\},\{y\}) \vdash spill_2\ (let\ z = x+y\ in\ w+z)\ (lv \cdot lv_s)\ : (\{w\},\{y\}) \cdot sl_s}$$

$$R_e := R \setminus K \cup L = \{w,x\} \setminus K \cup \{y\} = \{x,y\}$$

$$R_s := R_e \setminus K_z \cup \{z\} = \{x,y\} \setminus K_z \cup \{z\} = \{x,z\}$$

# Correctness predicate for `let x=e in` – example

(R,M) := ({w,x}, {y})

| code | $S$ | $L$ | $K$ | $K_z$ |
|---|---|---|---|---|
| let W = w in | | | | |
| let y = Y in | | | | |
| **let z = x + y in** | {w} | {y} | {w} | {y} |
| let w = W in | | | | |
| w + z | {} | {w} | {x} | |

$$\frac{|\{x,y\}| \leq 2 \qquad fv(x+y) \subseteq \{x,y\} \qquad |\{x,z\}| \leq 2 \qquad (\{x,z\},\{w,y\}) \vdash spill_2\ (w+z)\ lv_s\ : sl_s}{(\{w,x\},\{y\}) \vdash spill_2\ (let\ z = x+y\ in\ w+z)\ (lv \cdot lv_s)\ : (\{w\},\{y\}) \cdot sl_s}$$

$$R_e := R \setminus K \cup L = \{w,x\} \setminus K \cup \{y\} = \{x,y\}$$

$$R_s := R_e \setminus K_z \cup \{z\} = \{x,y\} \setminus K_z \cup \{z\} = \{x,z\}$$

# Correctness predicate for `let x=e in` – example

$$(R,M) := (\{w,x\}, \{y\})$$

| code | $S$ | $L$ | $K$ | $K_z$ |
|---|---|---|---|---|
| let W = w in | | | | |
| let y = Y in | | | | |
| **let z = x + y in** | {w} | {y} | {w} | {y} |
| let w = W in | | | | |
| w + z | {} | {w} | {x} | |

$$\frac{|\{x,y\}| \leq 2 \qquad fv(x+y) \subseteq \{x,y\} \qquad |\{x,z\}| \leq 2 \qquad (\{x,z\},\{w,y\}) \vdash spill_2\ (w+z)\ lv_s\ : sl_s}{(\{w,x\},\{y\}) \vdash spill_2\ (let\ z = x+y\ in\ w+z)\ (lv \cdot lv_s)\ : (\{w\},\{y\}) \cdot sl_s}$$

$$R_e := R \setminus K \cup L = \{w,x\} \setminus K \cup \{y\} = \{x,y\}$$

$$R_s := R_e \setminus K_z \cup \{z\} = \{x,y\} \setminus K_z \cup \{z\} = \{x,z\}$$

# Soundness

**Conjecture 1.** If $(\emptyset, \emptyset) \vdash \mathit{spill}_k\ s\ \mathit{lv}\ : \mathit{sl}$ holds then:

# Soundness

**Conjecture 1.** If $(\emptyset, \emptyset) \vdash \mathit{spill}_k\ s\ \mathit{lv}\ : \mathit{sl}$ holds then:

(a) *doSpill s sl* uses at most $k$ registers

# Soundness

**Conjecture 1.** If $(\emptyset, \emptyset) \vdash \mathit{spill}_k\ s\ \mathit{lv}\ : \mathit{sl}$ holds then:

(a) *doSpill s sl* uses at most $k$ registers

---

$$\text{doSpill} : (s, (\underbrace{\{x_1, ..., x_n\}}_{\text{spills}}, \underbrace{\{Y_1, ... Y_m\}}_{\text{loads}})) \mapsto \begin{array}{l} \mathit{let}\ X_1 = x_1\ \mathit{in} \ldots \\ \mathit{let}\ y_1 = Y_1\ \mathit{in} \ldots \\ s \end{array}$$

# Soundness

**Conjecture 1.** If $(\emptyset, \emptyset) \vdash \mathit{spill}_k\ s\ \mathit{lv}\ : \mathit{sl}$ holds then:

(a) *doSpill s sl* uses at most $k$ registers

(b) For any Expression $e$ in *doSpill s sl* , every variable used in $e$ is in a register.

---

$$\text{doSpill} : (s, (\underbrace{\{x_1, ..., x_n\}}_{\text{spills}}, \underbrace{\{Y_1, ... Y_m\}}_{\text{loads}})) \mapsto \begin{array}{l} \mathit{let}\ X_1 = x_1\ \mathit{in} \ldots \\ \mathit{let}\ y_1 = Y_1\ \mathit{in} \ldots \\ s \end{array}$$

# Soundness

**Conjecture 1.** If $(\emptyset, \emptyset) \vdash \mathit{spill}_k\ s\ \mathit{lv}\ : \mathit{sl}$ holds then:

(a) *doSpill s sl* uses at most $k$ registers

(b) For any Expression $e$ in *doSpill s sl* , every variable used in $e$ is in a register.

(c) *doSpill s sl* and $s$ are semantically equivalent.

---

$$\text{doSpill} : (s, (\underbrace{\{x_1, ..., x_n\}}_{\text{spills}}, \underbrace{\{Y_1, ... Y_m\}}_{\text{loads}})) \mapsto \begin{array}{l} \mathit{let}\ X_1 = x_1\ \mathit{in} \ldots \\ \mathit{let}\ y_1 = Y_1\ \mathit{in} \ldots \\ s \end{array}$$

# Soundness

**Conjecture 1.** If $(\emptyset, \emptyset) \vdash \mathit{spill}_k\ s\ \mathit{lv}\ : \mathit{sl}$ holds then:

(a) *doSpill s sl* uses at most $k$ registers

(b) For any Expression $e$ in *doSpill s sl* , every variable used in $e$ is in a register.

(c) *doSpill s sl* and $s$ are semantically equivalent.

---

- has to be generalized for inductive proof

---

$$\text{doSpill} : (s, (\underbrace{\{x_1, ..., x_n\}}_{\text{spills}}, \underbrace{\{Y_1, ...Y_m\}}_{\text{loads}})) \mapsto \begin{array}{l} \mathit{let}\ X_1 = x_1\ \mathit{in} \ldots \\ \mathit{let}\ y_1 = Y_1\ \mathit{in} \ldots \\ s \end{array}$$

# Soundness

**Conjecture 1.** If $(\emptyset, \emptyset) \vdash \mathit{spill}_k\ s\ \mathit{lv}\ : \mathit{sl}$ holds then:

(a) *doSpill s sl* uses at most $k$ registers

(b) For any Expression $e$ in *doSpill s sl* , every variable used in $e$ is in a register.

(c) *doSpill s sl* and $s$ are semantically equivalent.

---

- has to be generalized for inductive proof
- judgement also needs information about functions

---

$$\text{doSpill} : (s, (\underbrace{\{x_1, ..., x_n\}}_{\text{spills}}, \underbrace{\{Y_1, ... Y_m\}}_{\text{loads}})) \mapsto \begin{array}{l} \mathit{let}\ X_1 = x_1\ \mathit{in} \ldots \\ \mathit{let}\ y_1 = Y_1\ \mathit{in} \ldots \\ s \end{array}$$

# Conclusion

- concentrate on spilling – independant to register allocation

# Conclusion

- concentrate on spilling – independant to register allocation
- verification of a family of spilling algorithms

# Conclusion

- concentrate on spilling – independant to register allocation
- verification of a family of spilling algorithms
  - helpful in the construction of a translation validator

# Conclusion

- concentrate on spilling – independant to register allocation
- verification of a family of spilling algorithms
  - helpful in the construction of a translation validator
  - should simplify verification of concrete spilling algorithms

# References


- *Blazy, Robillard, and Appel, "Formal Verification of Coalescing Graph-Coloring Register Allocation", 2010*
- *Braun and Hack, "Register Spilling and Live-Range Splitting for SSA-Form Programs", 2009*
- *Hack, Grund, and Goos, "Register Allocation for Programs in SSA-Form", 2006*
- *Klitzke, "Verification of a Spilling Algorithms in Coq", 2015*
- *Leroy, "A formally verified compiler back-end", 2009*
- *Rideau and Leroy, "Validating Register Allocation and Spilling", 2010*
- *Schneider, Smolka, and Hack, "A First-Order Functional Intermediate Language for Verified Compilers", 2015*

$$\frac{\begin{array}{c}[a_e] \subseteq R_e \\ max\{R_e, R_s\} \leq k \quad (R_s, M \cup S) \vdash spill_k\ s\ a_s\ : b_s\end{array}}{(R, M) \vdash spill_k\ (let\ x = e\ in\ s)\ (\lambda \cdot a_e, a_s)\ : (S, L) \cdot b_s}$$

where $R_e := (R \setminus K \cup L) \qquad R_s := (R_e \setminus K_x) \cup \{x\}$

$spill\ k\ \Lambda\ R\ M\ (Let\ x\ e\ s)\ (Node\ Y\ [y_e, y_s]) =$
    let $L = [y_e] \setminus R$ in
    let $K \subseteq R \setminus [y_e] \wedge |K| = |L|$ in
    let $S = [y_s] \cap K$ in
    let $R_e = (R \setminus K \cup L)$ in
    if $[y_e] \setminus [y_s] \neq \emptyset$ then $K_x = \{\in [y_e] \setminus [y_s]\}$ else $K_x = \emptyset$ in
    let $R_s = (R_e \cup \{x\}) \setminus K_x$ in
    if $|R_s| \leq k$
        then $Node\ (S, L)\ [spill\ k\ \Lambda\ R_s\ (M \cup S)\ s\ y_s]$
        else $\bot$

$$\frac{\begin{array}{cc} R_f \subseteq R_t \cup X_R & \\ M_f \subseteq M_t \cup X_S & f \mapsto (R_f, M_f), \Lambda; (R_f, M_f) \vdash spill_k\ s\ a_s\ : b_s \\ max\{|R_f|, |R_t|\} \leq k & f \mapsto (R_f, M_f), \Lambda; (R_t, M_t) \vdash spill_k\ t\ a_t\ : b_t \end{array}}{\Lambda; (R, M) \vdash spill_k\ (fun\ f\ X = s\ in\ t)\ (\lambda \cdot a_s, a_t)\ : (S, L) \cdot b_s, b_t}$$

| step | register | memory | function env. |
|---|---|---|---|
| ... | $R$ | $M$ | $\Lambda$ |
| def f | $R \setminus K \cup L =: R_t$ | $M \cup S$ | $f \mapsto (R_f, M_f), \Lambda$ |
| ... | $R'$ | $M'$ | $f \mapsto (R_f, M_f), \Lambda'$ |
| apply f | $R' \setminus K' \cup L'$ | $M' \cup S'$ | $f \mapsto (R_f, M_f), \Lambda'$ |

$$\frac{\begin{array}{cc} & R_f \subseteq R' \setminus K' \cup L' \\ |R' \setminus K' \cup L'| \leq k & M_f \subseteq M' \cup S \end{array}}{\Lambda; (R', M') \vdash spill_k\ (f\overline{x})\ \lambda\ : (S', L')}\ \Lambda f = (R_f, M_f)$$

# doSpill

```
doSpill    s      (Node (xs,y::ys) sl)
  = doSpill    (let y = Y in s)     (Node (xs,ys) sl)
doSpill    s      (Node (x::xs,[]) sl)
  =
      doSpill    (let X = x in s)           (Node (xs, []) sl)


doSpill    s      (Node ([],[]) sl)
  = match s, sl with
      Let x e s', [sl'] ⇒ Let x e (doSpill    s'     sl')
      E e, [] ⇒ E e
      If e s' t, [sl1, sl2]
          ⇒ If e (doSpill    s'     sl1) (doSpill    t     sl2)
```