Formal Verification of a Family of Spilling Algorithms

Second Bachelor Seminar Talk

Julian Rosemann
Advisors: Prof. Gert Smolka, Sigurd Schneider

Saarland University
Department of Computer Science

2016-11-04

Recap: Spilling

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

Recap: Spilling

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

Recap: Spilling

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

Recap: Spilling

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	
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

Recap: Spilling

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

IL

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

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

Outline

Representation of Spilling

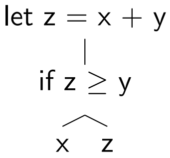
code	S	L
...		
let X = x in		
let z = x + y in		
if z ≥ y then		
let x = X in		
x		
else		
z		

Representation of 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	{}	{}

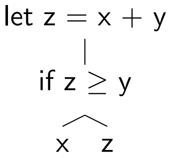
Representation of Spilling

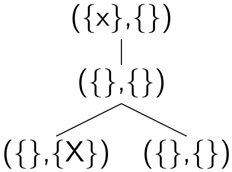
code	S	L
...		
let X = x in		
let z = x + y in	{x}	{}
if z ≥ y then	{}	{}
let x = X in		
x	{}	{X}
else		
z	{}	{}



Representation of Spilling

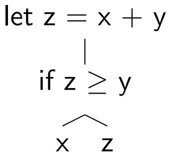
code	S	L
...		
let X = x in		
let z = x + y in	{x}	{}
if z ≥ y then	{}	{}
let x = X in		
x	{}	{X}
else		
z	{}	{}

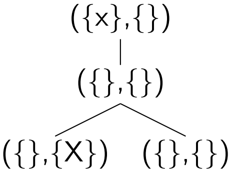




Representation of Spilling

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	{}	{}





\+ additional information at function declaration and application

From a spilling to a spilled program

$$\texttt{do\_spill} : \ \texttt{stmt} \rightarrow \texttt{spilling} \rightarrow \texttt{stmt}$$

From a spilling to a spilled program

`do_spill : stmt → spilling → stmt`

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

Valid spilling

A program s' is called a **spilled program** of s if

Valid spilling

A program s' is called a **spilled program** of s if

1. all variables are in a register when used in s'

Valid spilling

A program s' is called a **spilled program** of s if

1. all variables are in a register when used in s'
2. at most k registers are used in s'

Valid spilling

A program s' is called a **spilled program** of s if

1. all variables are in a register when used in s'
2. at most k registers are used in s'
3. $s \sim s'$.

Valid spilling

A program s' is called a **spilled program** of s if

1. all variables are in a register when used in s'
2. at most k registers are used in s'
3. $s \sim s'$.

A spilling sl **valid** on s if do_spill s sl is a spilled program of s.

Valid spilling

A program s' is called a **spilled program** of s if

1. all variables are in a register when used in s'
2. at most k registers are used in s'
3. $s \sim s'$.

A spilling sl **valid** on s if do_spill s sl is a spilled program of s.

$spill_k$ on s and $sl \Rightarrow sl$ is valid on s.

Inductive Correctness Predicate

$$(ZL, \Lambda); (R, M) \vdash \mathsf{spill_k}\ \ lv\ \ s : sl$$

ZL
Λ
$x \in R$
$x \in M$
k
lv
s
sl

Inductive Correctness Predicate

$$(ZL, \Lambda); (R, M) \vdash \mathsf{spill_k}\ \ lv\ s : sl$$

ZL	list of parameters of defined functions
Λ	
$x \in R$	
$x \in M$	
k	
lv	
s	
sl	

Inductive Correctness Predicate

$$(ZL, \Lambda); (R, M) \vdash \mathsf{spill}_\mathsf{k} \ \ lv \ s : sl$$

ZL	list of parameters of defined functions
Λ	list of expected live variables at function heads
$x \in R$	
$x \in M$	
k	
lv	
s	
sl	

Inductive Correctness Predicate

$$(ZL, \Lambda); (R, M) \vdash \mathsf{spill_k}\ \ lv\ s : sl$$

ZL	list of parameters of defined functions
Λ	list of expected live variables at function heads
$x \in R$	:⇔ current value is in a register
$x \in M$	
k	
lv	
s	
sl	

Inductive Correctness Predicate

$$(ZL, \Lambda); (R, M) \vdash \mathsf{spill_k}\ \ lv\ s : sl$$

ZL	list of parameters of defined functions
Λ	list of expected live variables at function heads
$x \in R$	$:\Leftrightarrow$ current value is in a register
$x \in M$	$:\Leftrightarrow$ current value is in the memory
k	
lv	
s	
sl	

Inductive Correctness Predicate

$$(ZL, \Lambda); (R, M) \vdash \mathrm{spill_k}\ \ lv\ s : sl$$

ZL	list of parameters of defined functions
Λ	list of expected live variables at function heads
$x \in R$	$:\Leftrightarrow$ current value is in a register
$x \in M$	$:\Leftrightarrow$ current value is in the memory
k	register bound
lv	
s	
sl	

Inductive Correctness Predicate

$$(ZL, \Lambda); (R, M) \vdash \mathsf{spill}_\mathsf{k}\ \ lv\ s : sl$$

ZL	list of parameters of defined functions
Λ	list of expected live variables at function heads
$x \in R$	$:\Leftrightarrow$ current value is in a register
$x \in M$	$:\Leftrightarrow$ current value is in the memory
k	register bound
lv	liveness information
s	
sl	

Inductive Correctness Predicate

$$(ZL, \Lambda); (R, M) \vdash \mathsf{spill}_\mathsf{k}\ \ lv\ s : sl$$

ZL	list of parameters of defined functions
Λ	list of expected live variables at function heads
$x \in R$	:⇔ current value is in a register
$x \in M$	:⇔ current value is in the memory
k	register bound
lv	liveness information
s	source program
sl	

Inductive Correctness Predicate

$$(ZL, \Lambda); (R, M) \vdash \mathsf{spill_k}\ \ lv\ s : sl$$

ZL	list of parameters of defined functions
Λ	list of expected live variables at function heads
$x \in R$	$:\Leftrightarrow$ current value is in a register
$x \in M$	$:\Leftrightarrow$ current value is in the memory
k	register bound
lv	liveness information
s	source program
sl	spill/load information

Inductive Correctness Predicate

- $(ZL, \Lambda); (R, M) \vdash \mathrm{spill_k}\ (LV \cdot lv_s)\ (\mathrm{let}\, x := e\, \mathrm{in}\, s) : (Sp, L, None) \cdot sl_s$
- $(ZL, \Lambda); (R, M) \vdash \mathrm{spill_k}\ (LV \cdot lv_s, lv_t)\ (\mathrm{if}\, e\, \mathrm{then}\, s\, \mathrm{else}\, t)$
  $: (Sp, L, None) \cdot sl_s, sl_t$
- $(ZL, \Lambda); (R, M) \vdash \mathrm{spill_k}\ \ LV\ \ e : (Sp, L, None)$
- $(ZL, \Lambda); (R, M) \vdash \mathrm{spill_k}\ \ LV\ \ (\mathrm{f}\ Y) : (Sp, L, \mathrm{Some}(\mathrm{inr}\ Sl))$
- $(ZL, \Lambda); (R, M) \vdash \mathrm{spill_k}\ (LV \cdot lv_s, lv_t)\ (\mathrm{fun}\, f\ \overline{x} := s\, \mathrm{in}\, t)$
  $: (Sp, L, \mathrm{Some}(\mathrm{inl}(R_f, M_f))) \cdot sl_s, sl_t$

Inductive Correctness Predicate

- $(ZL, \Lambda); (R, M) \vdash \mathsf{spill}_\mathsf{k}\ (LV \cdot lv_s)\ (\mathsf{let}\, x := e\, \mathsf{in}\, s) : (Sp, L, None) \cdot sl_s$
- $(ZL, \Lambda); (R, M) \vdash \mathsf{spill}_\mathsf{k}\ (LV \cdot lv_s, lv_t)\ (\mathsf{if}\, e\, \mathsf{then}\, s\, \mathsf{else}\, t)$
  $: (Sp, L, None) \cdot sl_s, sl_t$
- $(ZL, \Lambda); (R, M) \vdash \mathsf{spill}_\mathsf{k}\ \ LV\ \ e : (Sp, L, None)$
- $(ZL, \Lambda); (R, M) \vdash \mathsf{spill}_\mathsf{k}\ \ LV\ \ (\mathsf{f}\ Y) : (Sp, L, \mathsf{Some}(\mathsf{inr}\ Sl))$
- $(ZL, \Lambda); (R, M) \vdash \mathsf{spill}_\mathsf{k}\ (LV \cdot lv_s, lv_t)\ (\mathsf{fun}\, f\ \overline{x} := s\, \mathsf{in}\, t)$
  $: (Sp, L, \mathsf{Some}(\mathsf{inl}(R_f, M_f))) \cdot sl_s, sl_t$

In the first talk for `let`:

$$\frac{\begin{array}{cc} Sp \subseteq R & L \subseteq Sp \cup M \\ |(R \setminus K \cup L) \setminus K_x \cup \{x\}| \leq k & fv(e) \subseteq R \setminus K \cup L \\ |R \setminus K \cup L| \leq k & (ZL, \Lambda); ((R \setminus K \cup L) \setminus K_x \cup \{x\}, Sp \cup M) \vdash \mathsf{spill}_\mathsf{k}\ \ lv_s\ \ s : sl \end{array}}{(ZL, \Lambda); (R, M) \vdash \mathsf{spill}_\mathsf{k}\ (LV \cdot lv_s)\ (\mathsf{let}\, x := e\, \mathsf{in}\, s) : (Sp, L, None) \cdot sl}$$

Inductive Correctness Predicate

- $(ZL, \Lambda); (R, M) \vdash \mathrm{spill_k}\ (LV \cdot lv_s)\ (\mathrm{let}\, x := e \,\mathrm{in}\, s) : (\emptyset, \emptyset, None) \cdot sl_s$
- $(ZL, \Lambda); (R, M) \vdash \mathrm{spill_k}\ (LV \cdot lv_s, lv_t)\ (\mathrm{if}\, e \,\mathrm{then}\, s \,\mathrm{else}\, t)$ $: (\emptyset, \emptyset, None) \cdot sl_s, sl_t$
- $(ZL, \Lambda); (R, M) \vdash \mathrm{spill_k}\ \ LV\ \ e : (\emptyset, \emptyset, None)$
- $(ZL, \Lambda); (R, M) \vdash \mathrm{spill_k}\ \ LV\ \ (\mathrm{f}\ Y) : (\emptyset, \emptyset, \mathrm{Some}(\mathrm{inr}\, Sl))$
- $(ZL, \Lambda); (R, M) \vdash \mathrm{spill_k}\ (LV \cdot lv_s, lv_t)\ (\mathrm{fun}\, f\ \overline{x} := s \,\mathrm{in}\, t)$ $: (\emptyset, \emptyset, \mathrm{Some}(\mathrm{inl}(R_f, M_f))) \cdot sl_s, sl_t$
- $(ZL, \Lambda); (R, M) \vdash \mathrm{spill_k}\ \ lv\ \ s : (Sp, L, rm) \cdot \_$
- $(ZL, \Lambda); (R, M) \vdash \mathrm{spill_k}\ \ lv\ \ s : (\emptyset, L, rm) \cdot \_$

In the first talk for `let`:

$$\frac{\begin{array}{cc} Sp \subseteq R & L \subseteq Sp \cup M \\ |(R \setminus K \cup L) \setminus K_x \cup \{x\}| \leq k & fv(e) \subseteq R \setminus K \cup L \\ |R \setminus K \cup L| \leq k & (ZL, \Lambda); ((R \setminus K \cup L) \setminus K_x \cup \{x\}, Sp \cup M) \vdash \mathrm{spill_k}\ \ lv_s\ \ s : sl \end{array}}{(ZL, \Lambda); (R, M) \vdash \mathrm{spill_k}\ (LV \cdot lv_s)\ (\mathrm{let}\, x := e \,\mathrm{in}\, s) : (Sp, L, None) \cdot sl}$$

Inductive Correctness Predicate

- $(ZL, \Lambda); (R, M) \vdash \text{spill}_\text{k}\ (LV \cdot lv_s)\ (\text{let}\, x := e \,\text{in}\, s) : (\emptyset, \emptyset, None) \cdot sl_s$
- $(ZL, \Lambda); (R, M) \vdash \text{spill}_\text{k}\ (LV \cdot lv_s, lv_t)\ (\text{if}\, e \,\text{then}\, s \,\text{else}\, t)$
  $: (\emptyset, \emptyset, None) \cdot sl_s, sl_t$
- $(ZL, \Lambda); (R, M) \vdash \text{spill}_\text{k}\ \ LV\ \ e : (\emptyset, \emptyset, None)$
- $(ZL, \Lambda); (R, M) \vdash \text{spill}_\text{k}\ \ LV\ \ (\text{f}\ Y) : (\emptyset, \emptyset, \text{Some}(\text{inr}\ Sl))$
- $(ZL, \Lambda); (R, M) \vdash \text{spill}_\text{k}\ (LV \cdot lv_s, lv_t)\ (\text{fun}\, f\ \overline{x} := s \,\text{in}\, t)$
  $: (\emptyset, \emptyset, \text{Some}(\text{inl}(R_f, M_f))) \cdot sl_s, sl_t$
- $(ZL, \Lambda); (R, M) \vdash \text{spill}_\text{k}\ \ lv\ \ s : (Sp, L, rm) \cdot \_$
- $(ZL, \Lambda); (R, M) \vdash \text{spill}_\text{k}\ \ lv\ \ s : (\emptyset, L, rm) \cdot \_$

In the first talk for `let`:

$$\frac{\begin{array}{cc} Sp \subseteq R & L \subseteq Sp \cup M \\ |(R \setminus K \cup L) \setminus K_x \cup \{x\}| \le k & fv(e) \subseteq R \setminus K \cup L \\ |R \setminus K \cup L| \le k & (ZL, \Lambda); ((R \setminus K \cup L) \setminus K_x \cup \{x\}, Sp \cup M) \vdash \text{spill}_\text{k}\ \ lv_s\ \ s : sl \end{array}}{(ZL, \Lambda); (R, M) \vdash \text{spill}_\text{k}\ (LV \cdot lv_s)\ (\text{let}\, x := e \,\text{in}\, s) : (Sp, L, None) \cdot sl}$$

spill$_k$

$$\frac{\begin{array}{cc} Sp \subseteq R & L \subseteq Sp \cup M \\ |(R \setminus K \cup L) \setminus K_x \cup \{x\}| \leq k & fv(e) \subseteq R \setminus K \cup L \\ |R \setminus K \cup L| \leq k & (ZL, \Lambda); ((R \setminus K \cup L) \setminus K_x \cup \{x\}, Sp \cup M) \vdash \mathrm{spill_k}\ \ lv_s\ \ s : sl \end{array}}{(ZL, \Lambda); (R, M) \vdash \mathrm{spill_k}\ (LV \cdot lv_s)\ (\mathrm{let}\, x := e \,\mathrm{in}\, s) : (Sp, L, None) \cdot sl}$$

spill_k

$$\frac{\begin{array}{l} Sp \subseteq R \\ (ZL, \Lambda); (R, M \cup Sp) \vdash \text{spill}_\text{k}\ \ lv\ \ s : (\emptyset, L, rm) \cdot \_ \end{array} \quad \begin{array}{|l} \text{spilled variables available} \\ \text{induction} \end{array}}{(ZL, \Lambda); (R, M) \vdash \text{spill}_\text{k}\ \ lv\ \ s : (Sp, L, rm) \cdot \_}$$

$$\frac{\begin{array}{c} Sp \subseteq R \\ |(R \setminus K \cup L) \setminus K_x \cup \{x\}| \leq k \\ |R \setminus K \cup L| \leq k \end{array} \quad \begin{array}{c} L \subseteq Sp \cup M \\ fv(e) \subseteq R \setminus K \cup L \\ (ZL, \Lambda); ((R \setminus K \cup L) \setminus K_x \cup \{x\}, Sp \cup M) \vdash \text{spill}_\text{k}\ \ lv_s\ \ s : sl \end{array}}{(ZL, \Lambda); (R, M) \vdash \text{spill}_\text{k}\ (LV \cdot lv_s)\ (\text{let}\, x := e \,\text{in}\, s) : (Sp, L, None) \cdot sl}$$

spill_k

$$\frac{\begin{array}{l|l} Sp \subseteq R & \text{spilled variables available} \\ (ZL, \Lambda); (R, M \cup Sp) \vdash \text{spill}_\text{k}\ \ lv\ \ s : (\emptyset, L, rm) \cdot \_ & \text{induction} \end{array}}{(ZL, \Lambda); (R, M) \vdash \text{spill}_\text{k}\ \ lv\ \ s : (Sp, L, rm) \cdot \_}$$

$$\frac{\begin{array}{l|l} L \subseteq M & \text{loaded variables available} \\ |R \setminus K \cup L| \leq k & \text{don't load too much} \\ (ZL, \Lambda); (R \setminus K \cup L, M) \vdash \text{spill}_\text{k}\ \ lv\ \ s : (\emptyset, \emptyset, rm) \cdot \_ & \text{induction} \end{array}}{(ZL, \Lambda); (R, M) \vdash \text{spill}_\text{k}\ \ lv\ \ s : (\emptyset, L, rm) \cdot \_}$$

$$\frac{\begin{array}{cc} Sp \subseteq R & L \subseteq Sp \cup M \\ |(R \setminus K \cup L) \setminus K_x \cup \{x\}| \leq k & fv(e) \subseteq R \setminus K \cup L \\ |R \setminus K \cup L| \leq k & (ZL, \Lambda); ((R \setminus K \cup L) \setminus K_x \cup \{x\}, Sp \cup M) \vdash \text{spill}_\text{k}\ \ lv_s\ \ s : sl \end{array}}{(ZL, \Lambda); (R, M) \vdash \text{spill}_\text{k}\ (LV \cdot lv_s)\ (\text{let}\, x := e\, \text{in}\, s) : (Sp, L, None) \cdot sl}$$

spill$_k$

$$\frac{\begin{array}{ll} Sp \subseteq R & \text{spilled variables available} \\ (ZL, \Lambda); (R, M \cup Sp) \vdash \mathsf{spill_k}\ \ lv\ \ s : (\emptyset, L, rm) \cdot \_ & \text{induction} \end{array}}{(ZL, \Lambda); (R, M) \vdash \mathsf{spill_k}\ \ lv\ \ s : (Sp, L, rm) \cdot \_}$$

$$\frac{\begin{array}{ll} L \subseteq M & \text{loaded variables available} \\ |R \setminus K \cup L| \leq k & \text{don't load too much} \\ (ZL, \Lambda); (R \setminus K \cup L, M) \vdash \mathsf{spill_k}\ \ lv\ \ s : (\emptyset, \emptyset, rm) \cdot \_ & \text{induction} \end{array}}{(ZL, \Lambda); (R, M) \vdash \mathsf{spill_k}\ \ lv\ \ s : (\emptyset, L, rm) \cdot \_}$$

$$\frac{\begin{array}{ll} fv(e) \subseteq R & \text{variables available} \\ |R \setminus K_x \cup \{x\}| \leq k & \text{register bound afterwards} \\ (ZL, \Lambda); (R \setminus K_x \cup \{x\}, M) \vdash \mathsf{spill_k}\ \ lv_s\ \ s : sl_s & \text{induction} \end{array}}{(ZL, \Lambda); (R, M) \vdash \mathsf{spill_k}\ (LV \cdot lv_s)\ (\mathrm{let}\, x := e \,\mathrm{in}\, s) : (\emptyset, \emptyset, None) \cdot sl_s}$$

variables in register & register bound

$$\frac{Sp \subseteq R \qquad (ZL, \Lambda); (R, M \cup Sp) \vdash \mathsf{spill_k}\ \ lv\ \ s : (\emptyset, L, rm) \cdot \_}{(ZL, \Lambda); (R, M) \vdash \mathsf{spill_k}\ \ lv\ \ s : (Sp, L, rm) \cdot \_}$$

spilled variables available
induction

$$\frac{L \subseteq M \qquad |R \setminus K \cup L| \leq k \qquad (ZL, \Lambda); (R \setminus K \cup L, M) \vdash \mathsf{spill_k}\ \ lv\ \ s : (\emptyset, \emptyset, rm) \cdot \_}{(ZL, \Lambda); (R, M) \vdash \mathsf{spill_k}\ \ lv\ \ s : (\emptyset, L, rm) \cdot \_}$$

loaded variables available
don't load too much
induction

$$\frac{fv(e) \subseteq R \qquad |R \setminus K_x \cup \{x\}| \leq k \qquad (ZL, \Lambda); (R \setminus K_x \cup \{x\}, M) \vdash \mathsf{spill_k}\ \ lv_s\ \ s : sl_s}{(ZL, \Lambda); (R, M) \vdash \mathsf{spill_k}\ (LV \cdot lv_s)\ (\mathsf{let}\, x := e\, \mathsf{in}\, s) : (\emptyset, \emptyset, None) \cdot sl_s}$$

variables available
register bound afterwards
induction

variables in register & register bound

$$\frac{\begin{array}{l|l} Sp \subseteq R & \text{spilled variables available} \\ (ZL, \Lambda); (R, M \cup Sp) \vdash \mathsf{spill}_k\ \ lv\ \ s : (\emptyset, L, rm) \cdot \_ & \text{induction} \end{array}}{(ZL, \Lambda); (R, M) \vdash \mathsf{spill}_k\ \ lv\ \ s : (Sp, L, rm) \cdot \_}$$

$$\frac{\begin{array}{l|l} L \subseteq M & \text{loaded variables available} \\ |R \setminus K \cup L| \leq k & \text{don't load too much} \\ (ZL, \Lambda); (R \setminus K \cup L, M) \vdash \mathsf{spill}_k\ \ lv\ \ s : (\emptyset, \emptyset, rm) \cdot \_ & \text{induction} \end{array}}{(ZL, \Lambda); (R, M) \vdash \mathsf{spill}_k\ \ lv\ \ s : (\emptyset, L, rm) \cdot \_}$$

$$\frac{\begin{array}{l|l} fv(e) \subseteq R & \text{variables available} \\ |R \setminus K_x \cup \{x\}| \leq k & \text{register bound afterwards} \\ (ZL, \Lambda); (R \setminus K_x \cup \{x\}, M) \vdash \mathsf{spill}_k\ \ lv_s\ \ s : sl_s & \text{induction} \end{array}}{(ZL, \Lambda); (R, M) \vdash \mathsf{spill}_k\ (LV \cdot lv_s)\ (\mathsf{let}\, x := e\, \mathsf{in}\, s) : (\emptyset, \emptyset, None) \cdot sl_s}$$

variables in register & register bound

$$\frac{\begin{array}{l} Sp \subseteq R \\ (ZL, \Lambda); (R, M \cup Sp) \vdash \mathsf{spill_k}\ \ lv\ \ s : (\emptyset, L, rm) \cdot \_ \end{array}}{(ZL, \Lambda); (R, M) \vdash \mathsf{spill_k}\ \ lv\ \ s : (Sp, L, rm) \cdot \_}$$

spilled variables available
induction

$$\frac{\begin{array}{l} L \subseteq M \\ |R \setminus K \cup L| \leq k \\ (ZL, \Lambda); (R \setminus K \cup L, M) \vdash \mathsf{spill_k}\ \ lv\ \ s : (\emptyset, \emptyset, rm) \cdot \_ \end{array}}{(ZL, \Lambda); (R, M) \vdash \mathsf{spill_k}\ \ lv\ \ s : (\emptyset, L, rm) \cdot \_}$$

loaded variables available
don't load too much
induction

$$\frac{\begin{array}{l} fv(e) \subseteq R \\ |R \setminus K_x \cup \{x\}| \leq k \\ (ZL, \Lambda); (R \setminus K_x \cup \{x\}, M) \vdash \mathsf{spill_k}\ \ lv_s\ \ s : sl_s \end{array}}{(ZL, \Lambda); (R, M) \vdash \mathsf{spill_k}\ (LV \cdot lv_s)\ (\mathsf{let}\, x := e\, \mathsf{in}\, s) : (\emptyset, \emptyset, None) \cdot sl_s}$$

variables available
register bound afterwards
induction

Reconstructing Liveness

Liveness after spilling needed by:

Reconstructing Liveness

Liveness after spilling needed by:

- proof of register bound *and*

Reconstructing Liveness

Liveness after spilling needed by:

- proof of register bound *and*
- register allocator

Reconstructing Liveness

Liveness after spilling needed by:

- proof of register bound *and*
- register allocator

Approach:

Reconstructing Liveness

Liveness after spilling needed by:

- proof of register bound *and*
- register allocator

Approach:

- use original liveness algorithm *or*

Reconstructing Liveness

Liveness after spilling needed by:

- proof of register bound *and*
- register allocator

Approach:

- use original liveness algorithm *or*
- develop own algorithm using
  - original liveness *and*
  - register and memory states at function heads

Reconstructing Liveness

original algorithm:

new algorithm:

Reconstructing Liveness

original algorithm:

- runs in $\mathcal{O}(n^3)$

new algorithm:

Reconstructing Liveness

original algorithm:

- runs in $\mathcal{O}(n^3)$
- complicated fixpoint algorithm is unpractical in proof

new algorithm:

Reconstructing Liveness

original algorithm:

- runs in $\mathcal{O}(n^3)$
- complicated fixpoint algorithm is unpractical in proof

new algorithm:

- using
  - original liveness and
  - register and memory states at function heads

  it runs in $\mathcal{O}(n^2)$

Reconstructing Liveness

original algorithm:

- runs in $\mathcal{O}(n^3)$
- complicated fixpoint algorithm is unpractical in proof

new algorithm:

- using
  - original liveness and
  - register and memory states at function heads

  it runs in $\mathcal{O}(n^2)$
- challenge: kill sets are not available computationally

Liveness

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

Liveness intuition: variable x is **live** in statement s if either

- its value is used in s

or

- x was defined in the preceding statement

Liveness

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

Liveness intuition: variable x is **live** in statement s if either

- its value is used in s

or

- x was defined in the preceding statement

Liveness

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

Liveness intuition: variable x is **live** in statement s if either

- its value is used in s

or

- x was defined in the preceding statement

Liveness

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

Liveness intuition: variable x is **live** in statement s if either

- its value is used in s

or

- x was defined in the preceding statement

Liveness

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

Liveness intuition: variable x is **live** in statement s if either

- its value is used in s

or

- x was defined in the preceding statement

$$\frac{fv(e) \subseteq X \qquad \Lambda \vdash \mathit{live}\ s : (X \setminus K \cup \{x\})}{\Lambda \vdash \mathit{live}\ (\mathit{let}\ x := e\ \mathit{in}\ s) : X}$$

Liveness

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

Liveness intuition: variable x is **live** in statement s if either

- its value is used in s

or

- x was defined in the preceding statement

$$\frac{fv(e) \subseteq X \qquad \Lambda \vdash \mathit{live}\ s : (X \setminus K \cup \{x\})}{\Lambda \vdash \mathit{live}\ (\mathit{let}\ x := e\ \mathit{in}\ s) : X}$$

Liveness

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

let y = z

if x ≥ 0

x z

Liveness intuition: variable x is **live** in statement s if either

- its value is used in s

or

- x was defined in the preceding statement

$$\frac{fv(e) \subseteq X \qquad \Lambda \vdash \mathit{live}\ s : (X \setminus K \cup \{x\})}{\Lambda \vdash \mathit{live}\ (\mathit{let}\ x := e\ \mathit{in}\ s) : X}$$

Liveness

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

Liveness intuition: variable x is **live** in statement s if either

- its value is used in s

or

- x was defined in the preceding statement

$$\frac{fv(e) \subseteq X \qquad \Lambda \vdash \mathit{live}\ s : (X \setminus K \cup \{x\})}{\Lambda \vdash \mathit{live}\ (\mathit{let}\ x := e\ \mathit{in}\ s) : X}$$

StupSpill:

SimplSpill:

StupSpill:

- at any statement:

SimplSpill:

StupSpill:

- at any statement:
  - load everything

SimplSpill:

StupSpill:

- at any statement:
  - load everything
  - spill everything

SimplSpill:

StupSpill:

- at any statement:
  - load everything
  - spill everything
- satisfies $spill_k$

SimplSpill:

StupSpill:

- at any statement:
  - load everything
  - spill everything
- satisfies $spill_k$
- not efficient

SimplSpill:

StupSpill:

- at any statement:
  - load everything
  - spill everything
- satisfies $spill_k$
- not efficient
- originally used in Compcert, by now it has been replaced

SimplSpill:

StupSpill:

- at any statement:
  - load everything
  - spill everything
- satisfies $spill_k$
- not efficient
- originally used in Compcert, by now it has been replaced

SimplSpill:

- at any statement:

StupSpill:

- at any statement:
  - load everything
  - spill everything
- satisfies $spill_k$
- not efficient
- originally used in Compcert, by now it has been replaced

SimplSpill:

- at any statement:
  - load as little as possible

StupSpill:

- at any statement:
  - load everything
  - spill everything
- satisfies $spill_k$
- not efficient
- originally used in Compcert, by now it has been replaced

SimplSpill:

- at any statement:
  - load as little as possible
  - spill as little as possible

StupSpill:

- at any statement:
  - load everything
  - spill everything
- satisfies $spill_k$
- not efficient
- originally used in Compcert, by now it has been replaced

SimplSpill:

- at any statement:
  - load as little as possible
  - spill as little as possible
- satisfies $spill_k$

StupSpill:

- at any statement:
  - load everything
  - spill everything
- satisfies $spill_k$
- not efficient
- originally used in Compcert, by now it has been replaced

SimplSpill:

- at any statement:
  - load as little as possible
  - spill as little as possible
- satisfies $spill_k$
- spills are selected arbitrarily

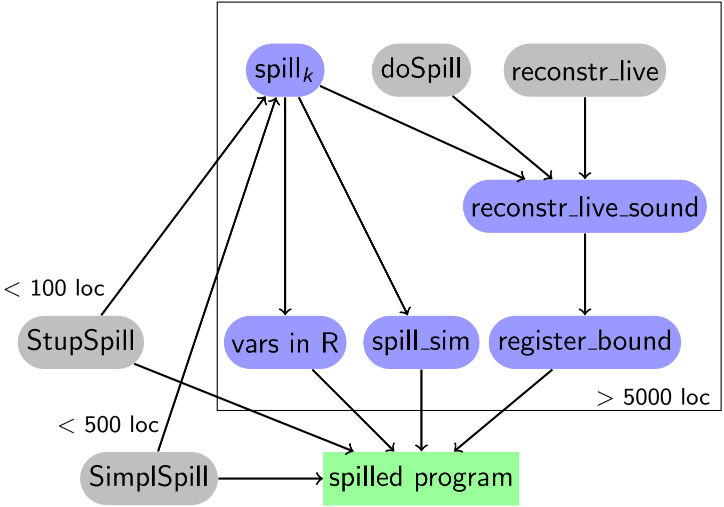
StupSpill:

- at any statement:
  - load everything
  - spill everything
- satisfies $spill_k$
- not efficient
- originally used in Compcert, by now it has been replaced

SimplSpill:

- at any statement:
  - load as little as possible
  - spill as little as possible
- satisfies $spill_k$
- spills are selected arbitrarily
- possible factorization: (unproven) oracle specifies priorities on selection of spills

Overview


spill$_k$
doSpill
reconstr_live
reconstr_live_sound
< 100 loc
StupSpill
vars in R
spill_sim
register_bound
> 5000 loc
< 500 loc
SimplSpill
spilled program

Conclusion

Conclusion

- first fully-verified optimizing spilling algorithm supporting arbitrary live-range-splitting

Conclusion

- first fully-verified optimizing spilling algorithm supporting arbitrary live-range-splitting
- `reconstr_live` will be used by register allocator

Conclusion

- first fully-verified optimizing spilling algorithm supporting arbitrary live-range-splitting
- `reconstr_live` will be used by register allocator
- $\texttt{spill}_k$ can be used to proof spilling algorithms

Conclusion

- first fully-verified optimizing spilling algorithm supporting arbitrary live-range-splitting
- `reconstr_live` will be used by register allocator
- $\texttt{spill}_k$ can be used to proof spilling algorithms
- `SimplSpill` can be factorized, such that efficient spilling is possible