Verified Compilation of IMP to Linear IMP

Initial Bachelor Seminar Talk

Clara Schneidewind
Advisor: Prof. Dr. Gert Smolka

SAARLAND UNIVERSITY
COMPUTER SCIENCE

May 22nd, 2015

Content

Previous Work

Glynn Winskel
The formal semantics of programming languages
MIT Press, 1993

Benjamin C. Pierce, Chris Casinghino, Marco Gaboardi, Michael Greenberg, Catalin Hritcu, Vilhelm Sjoberg, and Brent Yorgey.
Software Foundations
Electronic textbook, 2015

Tobias Nipkow, Gerwin Klein
Concrete semantics
Springer, 2014

Sigurd Schneider, Gert Smolka, Sebastian Hack
A First-Order Functional Intermediate Language for Verified Compilers
CoRR, abs/1503.08665, 2015

Motivation

Example

$$
\begin{array}{l}
\textbf{if } x < 0 \textbf{ then } x ::= -x \textbf{ else SKIP} \; ; \\
\textbf{while } n > 1 \textbf{ do} \\
\qquad n ::= n - 1 \; ; \\
\qquad x ::= x \cdot x
\end{array}
$$

$$c, d \;::=\; x ::= e \;\mid\; c \,;\, d \;\mid\; \textbf{if } b \textbf{ then } c \textbf{ else } d \;\mid\; \textbf{while } b \textbf{ do } c \;\mid\; \textbf{SKIP}$$

$$\text{where } e \in \text{AExp},\; b \in \text{BExp}$$

- The (terminating) execution of the program changes the state

IMP

Example

if $x < 0$ **then** $x ::= -x$ **else SKIP** ;
while $n > 0$ **do**
$\quad n ::= n - 1$;
$\quad x ::= x \cdot x$

Example

if $b_{x<0}$ **then** $a_{(x ::= -x)}$ **else** $a_{\textbf{SKIP}}$;
while $b_{n>0}$ **do**
$\quad a_{(n ::= n-1)}$;
$\quad a_{(x ::= x \cdot x)}$

IMP: $c, d \ ::= \ a \ | \ c \,;\, d \ | \ \textbf{if } b \textbf{ then } c \textbf{ else } d \ | \ \textbf{while } b \textbf{ do } c$

$$\Sigma = \mathcal{V} \to \mathbb{V} \qquad a : \Sigma \to \Sigma \qquad b : \Sigma \to \mathbb{B}$$

- **SKIP** can be treated as an action that leaves the state unchanged
- Neither arithmetic nor boolean expressions have to be specified

Big-Step Semantics

- relates initial and final state of an execution
- BS $c\ \sigma\ \tau$:= the execution of c in state σ terminates in state τ

$$\frac{a\,\sigma = \tau}{\mathrm{BS}\ a\ \sigma\ \tau} \qquad \frac{\mathrm{BS}\ c\ \sigma_1\ \sigma_2 \quad \mathrm{BS}\ d\ \sigma_2\ \sigma_3}{\mathrm{BS}\ (c\ ;\ d)\ \sigma_1\ \sigma_3} \qquad \frac{b\,\sigma = \mathit{true} \quad \mathrm{BS}\ c\ \sigma\ \tau}{\mathrm{BS}\ (\textbf{if}\ b\ \textbf{then}\ c\ \textbf{else}\ d)\ \sigma\ \tau}$$

$$\frac{b\,\sigma = \mathit{false} \quad \mathrm{BS}\ d\ \sigma\ \tau}{\mathrm{BS}\ (\textbf{if}\ b\ \textbf{then}\ c\ \textbf{else}\ d)\ \sigma\ \tau} \qquad \frac{b\,\sigma = \mathit{false}}{\mathrm{BS}\ (\textbf{while}\ b\ \textbf{do}\ c)\ \sigma\ \sigma}$$

$$\frac{b\,\sigma = \mathit{true} \quad \mathrm{BS}\ c\ \sigma_1\ \sigma_2 \quad \mathrm{BS}\ (\textbf{while}\ b\ \textbf{do}\ c)\ \sigma_2\ \sigma_3}{\mathrm{BS}\ (\textbf{while}\ b\ \textbf{do}\ c)\ \sigma_1\ \sigma_3}$$

Step-Indexed Semantics

- Motivation: executable evaluation function for IMP
- Problem: Possible divergence of programs leads to divergence of the evaluation function
- Solution: decreasing index
  - guarantees termination
  - denotes the depth limit of the recursion tree
- SI : $\mathbb{N} \rightarrow \text{IMP} \rightarrow \Sigma \rightarrow \text{option } \Sigma$
- SI $n\ c\ \sigma = \lceil \tau \rceil$:= at recursion depth of at most n the execution of c in σ terminates in τ
- SI $n\ c\ \sigma = \bot$:= the execution of c in σ does not terminate in n steps
- Relation between Big-Step Semantics and Step-Indexed Semantics :

$$\text{BS } c\ \sigma\ \tau \Leftrightarrow \exists n.\ \text{SI } n\ c\ \sigma = \lceil \tau \rceil$$

Weakest Precondition Semantics

- Motivation: Observation of partial assignments instead of whole states
- Conditions = predicates on states: $\Sigma \to \mathit{Prop}$
- Does the execution of p in σ terminates in a state τ that satisfies $\lambda\tau.\tau\, x = 8$?
- Characterization by a predicate:
  WP $c\ \sigma\ Q$:= the execution of c in state σ terminates in a state that satisfies Q

Weakest Precondition Semantics

$$\frac{a\,\sigma = \tau \quad Q(\tau)}{\text{WP } a\ \sigma\ Q} \qquad \frac{\text{WP } c\ \sigma\ P \quad \text{WP } d\ P\ Q}{\text{WP } (c\ ;\ d)\ \sigma\ Q}$$

$$\frac{b\,\sigma = \mathit{true} \quad \text{WP } c\ \sigma\ Q}{\text{WP } (\textbf{if } b \textbf{ then } c \textbf{ else } d)\ \sigma\ Q} \qquad \frac{b\,\sigma = \mathit{false} \quad \text{WP } d\ \sigma\ Q}{\text{WP } (\textbf{if } b \textbf{ then } c \textbf{ else } d)\ \sigma\ Q}$$

$$\frac{b\,\sigma = \mathit{true} \quad \text{WP } c\ \sigma\ P \quad \text{WP } (\textbf{while } b \textbf{ do } c)\ P\ Q}{\text{WP } (\textbf{while } b \textbf{ do } c)\ \sigma\ Q}$$

$$\frac{b\,\sigma = \mathit{false} \quad Q\,(\sigma)}{\text{WP } (\textbf{while } b \textbf{ do } c)\ \sigma\ Q}$$

$$\text{WP } c\ P\ Q := \forall \sigma, P(\sigma) \rightarrow \text{WP } c\ \sigma\ Q$$

Weakest Preconditions

- Subsumption of all states σ satisfying WP c σ Q as the weakest precondition of c and Q:

$$\mathrm{wp}(c, Q) := \lambda\sigma.\mathrm{WP}\ c\ \sigma\ Q$$

- Definition of weakest preconditions via BS :

$$\mathrm{wp}_C(c, Q) := \lambda\sigma.\exists\tau, \mathrm{BS}\ c\ \sigma\ \tau \wedge Q\ \tau$$

- Coincidence of Big-Step and Weakest Precondition Semantics:

$$\mathrm{WP}\ c\ \sigma\ Q \leftrightarrow \mathrm{wp}_C(c, Q)\,\sigma$$
$$\mathrm{BS}\ c\ \sigma\ \tau \leftrightarrow \mathrm{WP}\ c\ \sigma\ (\lambda\tau'.\tau = \tau')$$

LIMP

- IMP not linear due to sequences and while-loops (needs a stack)
- Goal of compilation: Translation of IMP to a register transfer language
  - Sequences have to be linearized (no nesting)
  - While-loops have to be translated to blocks and calls

Example

while $b_{n>1}$ **do**

$a_{(n ::= n-1)}$;

$a_{(x ::= x \cdot x)}$

Example

block l : **if** $b_{n>1}$ **then** $a_{(n ::= n-1)}$; $(a_{(x ::= x \cdot x)}$; call $l)$

else halt ;

call l

LIMP

Alternative syntax for blocks and calls:

Example

$$\textbf{while } b_{n>1} \textbf{ do}$$
$$a_{(n ::= n-1)} \ ;$$
$$a_{(x ::= x \cdot x)}$$

Example

$$\textbf{fix } l.\, \textbf{if } b_{n>1} \textbf{ then } a_{(n ::= n-1)} \ ; \ a_{(x ::= x \cdot x)} \ ; \ l$$
$$\textbf{else halt}$$

$$s, t \ ::= \ \textbf{halt} \ \mid \ a \ ; \ s \ \mid \ \textbf{if } b \textbf{ then } c \textbf{ else } d \ \mid \ \textbf{fix } l.\, s \ \mid \ l = s \ ; \ t \ \mid \ l$$

Construct for non-recursive blocks helps to linearize conditionals (omittable)

LIMP

Example (p)

$$\begin{array}{l}\textbf{if } b_{x<0} \textbf{ then } a_{(x ::= -x)} \textbf{ else } a_{\textbf{SKIP}} \ ; \\ \textbf{while } b_{n>1} \textbf{ do} \\ \qquad a_{(n ::= n-1)} \ ; \\ \qquad a_{(x ::= x \cdot x)}\end{array}$$

Example

$$\begin{array}{l}k = \textbf{fix}\, l.\, \textbf{if } b_{n>1} \textbf{ then } a_{(n ::= n-1)} \ ; \ a_{(x ::= x \cdot x)} \ ; \ l \\ \qquad\qquad \textbf{else halt} \ ; \\ \textbf{if } b_{x<0} \textbf{ then } a_{(x ::= -x)} \ ; \ k \textbf{ else } k\end{array}$$

Weakest Precondition Semantics

$$\frac{Q(\sigma)}{\mathrm{WP}\ \mathbf{halt}\ \sigma\ Q} \qquad \frac{a\,\sigma = \tau \quad \mathrm{WP}\ s\ \sigma\ Q}{\mathrm{WP}\ (a\,;\,s)\ \sigma\ Q} \qquad \frac{b\,\sigma = \mathit{true} \quad \mathrm{WP}\ s\ \sigma\ Q}{\mathrm{WP}\ (\mathbf{if}\ b\ \mathbf{then}\ s\ \mathbf{else}\ t)\ \sigma\ Q}$$

$$\frac{b\,\sigma = \mathit{false} \quad \mathrm{WP}\ t\ \sigma\ Q}{\mathrm{WP}\ (\mathbf{if}\ b\ \mathbf{then}\ s\ \mathbf{else}\ t)\ \sigma\ Q} \qquad \frac{\mathrm{WP}\ s^{x}_{\mathbf{fix}\,x.\,s}\ \sigma\ Q}{\mathrm{WP}\ (\mathbf{fix}\,x.\,s)\ \sigma\ Q} \qquad \frac{\mathrm{WP}\ t^{x}_{s}\ \sigma\ Q}{\mathrm{WP}\ (x = s\,;\ t)\ \sigma\ Q}$$

- At most one recursive premise per rule
- No interpolants are needed
- Substitution semantics for fix and remember makes it unnecessary to keep track of introduced blocks

Compiler

- Problem: IMP-commands cannot be translated isolatedly as there is no sequence operator in LIMP to compose them

Example

$$\mathcal{C}\ (a_1) \rightsquigarrow a_1\ ;\ \textbf{halt} \qquad \mathcal{C}\ (a_2) \rightsquigarrow a_2\ ;\ \textbf{halt} \qquad \mathcal{C}\ (a_1\ ;\ a_2) \rightsquigarrow ?$$

- Solution: Translation of IMP-commands with respect to a continuation

$$\begin{aligned}
\mathcal{C}(_,_) &: \mathrm{IMP} \to \mathrm{LIMP} \to \mathrm{LIMP} \\
\mathcal{C}(a, s) &= a\ ;\ s \\
\mathcal{C}(c\ ;\ d, s) &= \mathcal{C}(c, \mathcal{C}(d, s)) \\
\mathcal{C}(\textbf{if}\ b\ \textbf{then}\ c\ \textbf{else}\ d, s) &= x = s\ ;\ \textbf{if}\ b\ \textbf{then}\ \mathcal{C}(c, x)\ \textbf{else}\ \mathcal{C}(d, x) && x\ \textit{fresh} \\
\mathcal{C}(\textbf{while}\ b\ \textbf{do}\ c, s) &= \textbf{fix}\, x.\ \textbf{if}\ b\ \textbf{then}\ \mathcal{C}(c, x)\ \textbf{else}\ s && x\ \textit{fresh}
\end{aligned}$$

Compiler in practice

- More convenient to use De Bruijn indices instead of names
- Makes it unnecessary to carry a counter as third argument

$$s, t \ ::= \ \textbf{halt} \ \mid \ a \,;\, s \ \mid \ \textbf{if } b \textbf{ then } c \textbf{ else } d \ \mid \ \textbf{fix } s \ \mid \ \textbf{rem } s \,;\, t \ \mid \ l$$

$$\begin{aligned}
\mathcal{C}(_,_) &: \mathsf{IMP} \to \mathsf{LIMP} \to \mathsf{LIMP} \\
\mathcal{C}(a, s) &= a \,;\, s \\
\mathcal{C}(c \,;\, d, s) &= \mathcal{C}(c, \mathcal{C}(d, s)) \\
\mathcal{C}(\textbf{if } b \textbf{ then } c \textbf{ else } d, s) &= \textbf{rem } s \,;\, \textbf{if } b \textbf{ then } \mathcal{C}(c, 0) \textbf{ else } \mathcal{C}(d, 0) \\
\mathcal{C}(\textbf{while } b \textbf{ do } c, s) &= \textbf{fix}\,\textbf{if } b \textbf{ then } \mathcal{C}(c, 0) \textbf{ else } s\uparrow^1
\end{aligned}$$

shift-operation $s\uparrow^1$ increases indices in s by 1 and prevents missreferencing

Compiler Correctness

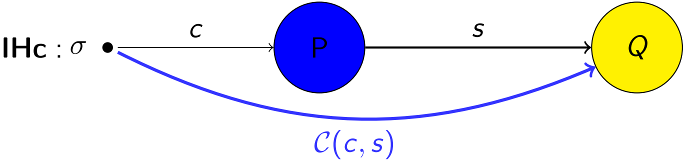
Goal: show correspondence of IMP-command c and LIMP-command $\mathcal{C}(c, \mathit{halt})$ with respect to weakest precondition semantics:

$$\text{WP } c\ \sigma\ Q \leftrightarrow \text{WP } \mathcal{C}(c, \mathit{halt})\ \sigma\ Q$$

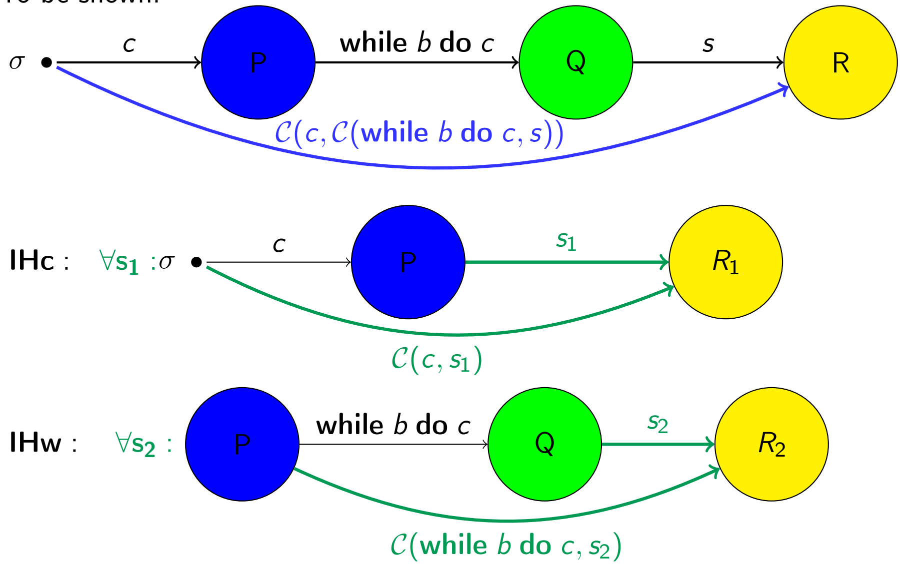
If-part:

- Generalization of the lemma for arbitrary continuations
- Effect of the continuation has to be taken into account
- Idea: $\mathcal{C}(c, s)$ first executes c and then continues with s

$$\text{WP } c\ \sigma\ P \rightarrow (\forall \sigma. P(\sigma) \rightarrow \text{WP } s\ \sigma\ Q) \rightarrow \text{WP } \mathcal{C}(c, s)\ \sigma\ Q$$



To be shown:



σ
c
P
while b do c
Q
s
R
$\mathcal{C}(c, \mathcal{C}(\textbf{while } b \textbf{ do } c, s))$
IHc :
$\forall s_1 : \sigma$
c
P
s_1
R_1
$\mathcal{C}(c, s_1)$
IHw :
$\forall s_2$:
P
while b do c
Q
s_2
R_2
$\mathcal{C}(\textbf{while } b \textbf{ do } c, s_2)$

Compiler Correctness

$$b\,\sigma = \mathit{true} \rightarrow$$
$$\mathrm{WP}\ (\mathcal{C}(c, \mathcal{C}(\textbf{while}\ b\ \textbf{do}\ c, s)))\ \sigma\ R \leftrightarrow \mathrm{WP}\ (\mathcal{C}(\textbf{while}\ b\ \textbf{do}\ c, s))\ \sigma\ R$$

Proof.

$$\begin{aligned}
& \mathrm{WP}\ (\mathcal{C}(\textbf{while}\ b\ \textbf{do}\ c, s))\ \sigma\ R \\
\overset{\mathit{Def.}\mathcal{C}}{\Longleftrightarrow}\ & \mathrm{WP}\ (\textbf{fix}\, x.\, \textbf{if}\ b\ \textbf{then}\ \mathcal{C}(c, x)\ \textbf{else}\ s)\ \sigma\ R \\
\overset{\mathit{subst.}}{\Longleftrightarrow}\ & \mathrm{WP}\ (\textbf{if}\ b\ \textbf{then}\ \mathcal{C}(c, \textbf{fix}\, x.\, \textbf{if}\ b\ \textbf{then}\ \mathcal{C}(c, x)\ \textbf{else}\ s)\ \textbf{else}\ s)\ \sigma\ R \\
\overset{b\,\sigma = \mathit{true}}{\Longleftrightarrow}\ & \mathrm{WP}\ (\mathcal{C}(c, \textbf{fix}\, x.\, \textbf{if}\ b\ \textbf{then}\ \mathcal{C}(c, s)\ \textbf{else}\ s))\ \sigma\ R \\
\overset{\mathit{Def.}\mathcal{C}}{\Longleftrightarrow}\ & \mathrm{WP}\ (\mathcal{C}(c, \mathcal{C}(\textbf{while}\ b\ \textbf{do}\ s, s)))\ \sigma\ R
\end{aligned}$$

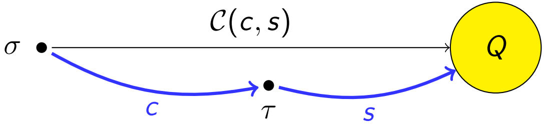
Substitution Lemma

$$\mathcal{C}(c, s)^x_t = \mathcal{C}(c, s^x_t)$$

Compiler Correctness

Idea: split up the execution of $\mathcal{C}(c, s)$ into the executions of c and s

$$\text{WP } \mathcal{C}(c, s)\ \sigma\ Q \to \exists \tau, \text{WP } c\ \sigma\ (\tau) \land \text{WP } s\ \tau\ Q$$



Proof by induction on c.

Problem: Inductive hypothesis in the while-true-case too weak.

Outlook

- Adding system calls to IMP and LIMP
  - Weakest Precondition Semantics with traces
  - Allows to compare non-terminating programs
- Weakest Precondition Semantics for functional IL

5 Appendix

Step-indexed Semantics

$$
\begin{aligned}
\text{SI} \quad &: \mathbb{N} \to \text{IMP} \to \Sigma \to \text{option } \sigma \\
\text{SI } 0\ c\ \sigma &= \bot \\
\text{SI } n\ (a)\ \sigma &= \lceil a\,\sigma \rceil \\
\text{SI } n\ (c\ ;\ d)\ \sigma &= \text{SI' } (n-1)\ d\ (\text{SI } (n-1)\ c\ \sigma) \\
\text{SI } n\ (\textbf{if } b \textbf{ then } c \textbf{ else } d)\ \sigma &= \textit{if } (b\,\sigma)\ \textit{then } \text{SI } (n-1)\ c\ \sigma\ \textit{else } \text{SI } (n-1)\ d\ \sigma \\
\text{SI } n\ (\textbf{while } b \textbf{ do } c)\ \sigma &= \textit{if } (b\,\sigma)\ \textit{then} \\
&\quad\ \text{SI' } (n-1)\ (\textbf{while } b \textbf{ do } c)\ (\text{SI } (n-1)\ c\ \sigma) \\
&\quad\ \textit{else } \lceil \sigma \rceil
\end{aligned}
$$

$$
\begin{aligned}
\text{SI'} \quad &: \mathbb{N} \to \text{IMP} \to \text{option } \Sigma \to \text{option } \Sigma \\
\text{SI' } n\ c\ \lceil \sigma \rceil &= \text{SI } n\ c\ \sigma \\
\text{SI' } n\ c\ \bot &= \bot
\end{aligned}
$$

Nested While-loop

Example

$$
\begin{aligned}
&\mathcal{C}(\textbf{while } b_1 \textbf{ do } (\textbf{while } b2 \textbf{ do } a), \textbf{halt}) \\
&= \textbf{fix}\,(\textbf{if } b_1 \textbf{ then } \mathcal{C}(\textbf{while } b_2 \textbf{ do } a, 0) \textbf{ else halt} \uparrow^1) \\
&= \textbf{fix}\,(\textbf{if } b_1 \textbf{ then fix}\,(\textbf{if } b_2 \textbf{ then } \mathcal{C}(a, 0) \textbf{ else } 0 \uparrow^1) \textbf{ else halt}) \\
&= \textbf{fix}\,(\textbf{if } b_1 \textbf{ then fix}\,(\textbf{if } b_2 \textbf{ then } a\,;\,0 \textbf{ else } 1) \textbf{ else halt})
\end{aligned}
$$

Compiler Correctness

- Idea: split up the execution of $\mathcal{C}(c, s)$ in the execution of c and s
- τ denotes the state after the execution of c in σ

$$\text{WP } \mathcal{C}(c, s)\ \sigma\ Q \rightarrow \exists \tau, \text{WP } c\ \sigma\ (\tau) \wedge \text{WP } s\ \tau\ Q$$

- we try to prove by induction on c, but fail in the while-case:

$$\begin{aligned}
\textbf{IHc} &: \text{WP } \mathcal{C}(c, s)\ \sigma\ Q \rightarrow \exists \tau, \text{WP } c\ \sigma\ (\tau) \wedge \text{WP } s\ \tau\ Q \\
\textbf{A} &: \text{WP } (\textbf{fix}\, x.\, \textbf{if } b \textbf{ then } \mathcal{C}(c, x) \textbf{ else } s)\ \sigma\ Q \\
&\overset{\textit{subst.}}{\Longrightarrow} \text{WP } (\textbf{if } b \textbf{ then } \mathcal{C}(c, \mathcal{C}(\textbf{while } b \textbf{ do } c, s)) \textbf{ else } s)\ \sigma\ Q \\
&\overset{\textit{IHc}}{\Longrightarrow} \exists \tau, \text{WP } c\ \sigma\ \tau \wedge \text{WP } \mathcal{C}(\textbf{while } c \textbf{ do } d, s)\ \tau\ Q \\
&\lightning\ \textit{Inductive hypothesis for while would be needed}
\end{aligned}$$

- Solution: Introduction a new step-indexed predicate for WP that keeps track of the number of substitutions in the fix-case
- Nested induction n the substitution depth in the while-case

Compiler Correctness

$$\text{WP}\ c\ \sigma\ P \rightarrow (\forall \sigma . P(\sigma) \rightarrow \text{WP}\ s\ \sigma\ Q) \rightarrow \text{WP}\ \mathcal{C}(c, s)\ \sigma\ Q$$

Proof by induction on WP $c\ \sigma\ P$:

Proof.

Case: **while** b **do** c (true):

$\mathbf{A_1}$: WP $c\ \sigma\ P$ $\quad$ $\mathbf{A_2}$: WP (**while** b **do** c) $P\ Q$ $\quad$ $\mathbf{A_3}$: WP $s\ Q\ R$

$$\mathbf{IHc} : \text{WP}\ s'\ P\ Q' \rightarrow \text{WP}\ (\mathcal{C}(c, s'))\ \sigma\ Q'$$

$$\mathbf{IHw} : \text{WP}\ s'\ Q\ Q' \rightarrow \text{WP}\ (\mathcal{C}(\textbf{while}\ b\ \textbf{do}\ c, s'))\ P\ Q'$$

$$\mathbf{A_3} : \text{WP}\ s\ Q\ R$$

$$\stackrel{IHw}{\Longrightarrow} \text{WP}\ (\mathcal{C}(\textbf{while}\ b\ \textbf{do}\ c, s))\ P\ R$$

$$\stackrel{IHc}{\Longrightarrow} \text{WP}\ (\mathcal{C}(c, \mathcal{C}(\textbf{while}\ b\ \textbf{do}\ c, s)))\ \sigma\ R$$

$\square$