Type-theoretic functional semantics

Yves Bertot,

Venanzio Capretta,

Kuntal Das Barman

`FirstName.Surname@sophia.inria.fr`

INRIA Sophia-Antipolis, France

INRIA SOPHIA ANTIPOLIS

How do we proceed?

- A small imperative language - IMP

How do we proceed?

- A small imperative language - IMP
- Operational semantics for IMP

How do we proceed?

- A small imperative language - IMP
- Operational semantics for IMP
- Specification problem with denotational semantics and its solution

INRIA
SOPHIA ANTIPOLIS

How do we proceed?

- A small imperative language - IMP
- Operational semantics for IMP
- Specification problem with denotational semantics and its solution
- Assurance of termination and accessibility predicate

How do we proceed?

- A small imperative language - IMP
- Operational semantics for IMP
- Specification problem with denotational semantics and its solution
- Assurance of termination and accessibility predicate
- Solution in simultaneous induction-recursion

How do we proceed?

- A small imperative language - IMP
- Operational semantics for IMP
- Specification problem with denotational semantics and its solution
- Assurance of termination and accessibility predicate
- Solution in simultaneous induction-recursion
- Disentangle simultaneous induction-recursion for Coq

INRIA SOPHIA ANTIPOLIS

How do we proceed?

- A small imperative language - IMP
- Operational semantics for IMP
- Specification problem with denotational semantics and its solution
- Assurance of termination and accessibility predicate
- Solution in simultaneous induction-recursion
- Disentangle simultaneous induction-recursion for Coq
- Conclusion

IMP - a small imperative language

- arithmetic expressions (AExp)

  $a ::= n \mid X \mid a_0 + a_1 \mid a_0 - a_1 \mid a_0 * a_1,$

IMP - a small imperative language

- arithmetic expressions (AExp)

  $a ::= n \mid X \mid a_0 + a_1 \mid a_0 - a_1 \mid a_0 * a_1,$

- boolean expressions (BExp)

  $b ::= \mathsf{true} \mid \mathsf{false} \mid a_0 = a_1 \mid a_0 \leq a_1 \mid \neg b \mid b_0 \vee b_1 \mid b_0 \wedge b_1,$

IMP - a small imperative language

- arithmetic expressions (AExp)

  $a ::= n \mid X \mid a_0 + a_1 \mid a_0 - a_1 \mid a_0 * a_1,$

- boolean expressions (BExp)

  $b ::= \mathsf{true} \mid \mathsf{false} \mid a_0 = a_1 \mid a_0 \leq a_1 \mid \neg b \mid b_0 \vee b_1 \mid b_0 \wedge b_1,$

- commands (Command)

  $c ::= \mathsf{skip} \mid X \leftarrow a \mid c_0; c_1 \mid \mathsf{if}\ b\ \mathsf{then}\ c_0\ \mathsf{else}\ c_1 \mid \mathsf{while}\ b\ \mathsf{do}\ c,$

INRIA SOPHIA ANTIPOLIS

IMP - a small imperative language

- arithmetic expressions (AExp)

  $a ::= n \mid X \mid a_0 + a_1 \mid a_0 - a_1 \mid a_0 * a_1,$

- boolean expressions (BExp)

  $b ::= \mathsf{true} \mid \mathsf{false} \mid a_0 = a_1 \mid a_0 \leq a_1 \mid \neg b \mid b_0 \vee b_1 \mid b_0 \wedge b_1,$

- commands (Command)

  $c ::= \mathsf{skip} \mid X \leftarrow a \mid c_0; c_1 \mid \mathsf{if}\ b\ \mathsf{then}\ c_0\ \mathsf{else}\ c_1 \mid \mathsf{while}\ b\ \mathsf{do}\ c,$

- states (State)

  Keeps track of contents of locations(X)

Evaluation

- Arithmetic Expressions : $\langle \mathsf{AExp}, \mathsf{State} \rangle_{\mathsf{A}} \rightsquigarrow \texttt{Natural Number}$

Evaluation

- Arithmetic Expressions : $\langle \mathsf{AExp}, \mathsf{State} \rangle_{\mathsf{A}} \leadsto \texttt{Natural Number}$

- Boolean Expressions : $\langle \mathsf{BExp}, \mathsf{State} \rangle_{\mathsf{B}} \leadsto \texttt{Boolean Value}$

INRIA SOPHIA ANTIPOLIS

Evaluation

- Arithmetic Expressions : $\langle \mathsf{AExp}, \mathsf{State}\rangle_{\mathsf{A}} \rightsquigarrow \texttt{Natural Number}$
- Boolean Expressions : $\langle \mathsf{BExp}, \mathsf{State}\rangle_{\mathsf{B}} \rightsquigarrow \texttt{Boolean Value}$
- Commands : $\langle \mathsf{Command}, \mathsf{State}\rangle_{\mathsf{C}} \rightsquigarrow \mathsf{State}$

Where we are?

- A small imperative language - IMP
- Operational semantics for IMP
- Specification problem with denotational semantics and its solution
- Assurance of termination and accessibility predicate
- Solution in simultaneous induction-recursion
- Disentangle simultaneous induction-recursion for Coq
- Conclusion

Example

while b do c

Example

while b do c

$$\Downarrow$$

Example

$$\text{while } b \text{ do } c$$

$$\Downarrow$$

$b = \mathsf{true}$

Example

$$\text{while } b \text{ do } c$$

$$\Downarrow$$

$$\text{if } \; b = \text{true} \;\; \text{then} \;\; \{ \; c$$

Example

$$\text{while } b \text{ do } c$$

$$\Downarrow$$

$$\text{if } \ b = \text{true} \ \ \text{then} \ \ \{ \ c \, ; \ \ \text{while } b \text{ do } c \, \}$$

Operational Semantics

$$\frac{}{\langle \mathsf{while}\ b\ \mathsf{do}\ c, \sigma\rangle_{\mathsf{C}} \rightsquigarrow \sigma''}$$

Operational Semantics

$$\frac{\langle b, \sigma \rangle_{\mathsf{B}} \leadsto \mathsf{true}}{\langle \mathsf{while}\ b\ \mathsf{do}\ c, \sigma \rangle_{\mathsf{C}} \leadsto \sigma''}$$

Operational Semantics

$$\frac{\langle b, \sigma \rangle_{\mathsf{B}} \leadsto \mathsf{true} \quad \langle c, \sigma \rangle_{\mathsf{C}} \leadsto \sigma'}{\langle \mathsf{while}\ b\ \mathsf{do}\ c, \sigma \rangle_{\mathsf{C}} \leadsto \sigma''}$$

INRIA SOPHIA ANTIPOLIS

Operational Semantics

$$\frac{\langle b, \sigma\rangle_{\mathsf{B}} \rightsquigarrow \mathsf{true} \quad \langle c, \sigma\rangle_{\mathsf{C}} \rightsquigarrow \sigma' \quad \langle \mathsf{while}\ b\ \mathsf{do}\ c, \sigma'\rangle_{\mathsf{C}} \rightsquigarrow \sigma''}{\langle \mathsf{while}\ b\ \mathsf{do}\ c, \sigma\rangle_{\mathsf{C}} \rightsquigarrow \sigma''}$$

Operational Semantics

$$\frac{\langle b, \sigma \rangle_{\mathsf{B}} \rightsquigarrow \mathsf{true} \quad \langle c, \sigma \rangle_{\mathsf{C}} \rightsquigarrow \sigma' \quad \langle \mathsf{while}\ b\ \mathsf{do}\ c, \sigma' \rangle_{\mathsf{C}} \rightsquigarrow \sigma''}{\langle \mathsf{while}\ b\ \mathsf{do}\ c, \sigma \rangle_{\mathsf{C}} \rightsquigarrow \sigma''}$$

$$\Downarrow$$

Operational Semantics

$$\frac{\langle b, \sigma\rangle_{\mathsf{B}} \rightsquigarrow \mathsf{true} \quad \langle c, \sigma\rangle_{\mathsf{C}} \rightsquigarrow \sigma' \quad \langle \mathsf{while}\ b\ \mathsf{do}\ c, \sigma'\rangle_{\mathsf{C}} \rightsquigarrow \sigma''}{\langle \mathsf{while}\ b\ \mathsf{do}\ c, \sigma\rangle_{\mathsf{C}} \rightsquigarrow \sigma''}$$

$$\Downarrow$$

$\langle\cdot,\cdot\rangle_{\mathsf{C}} \rightsquigarrow \cdot :$ $\mathsf{Command} \to \mathsf{State} \to \mathsf{State} \to \mathbf{Prop}$

$\vdots$

$\mathsf{eval\_while\_true} :\quad \forall b : \mathsf{BExp}; \forall c : \mathsf{Command}; \forall \sigma, \sigma', \sigma'' : \mathsf{State}$

Operational Semantics

$$\frac{\langle b, \sigma\rangle_{\mathsf{B}} \rightsquigarrow \mathsf{true} \quad \langle c, \sigma\rangle_{\mathsf{C}} \rightsquigarrow \sigma' \quad \langle \mathsf{while}\ b\ \mathsf{do}\ c, \sigma'\rangle_{\mathsf{C}} \rightsquigarrow \sigma''}{\langle \mathsf{while}\ b\ \mathsf{do}\ c, \sigma\rangle_{\mathsf{C}} \rightsquigarrow \sigma''}$$

$$\Downarrow$$

$\langle\cdot,\cdot\rangle_{\mathsf{C}} \rightsquigarrow \cdot : \mathsf{Command} \to \mathsf{State} \to \mathsf{State} \to \mathbf{Prop}$

$\vdots$

$\mathsf{eval\_while\_true}: \quad \forall b\colon \mathsf{BExp}; \forall c\colon \mathsf{Command}; \forall \sigma, \sigma', \sigma''\colon \mathsf{State}$

$$\langle b, \sigma\rangle_{\mathsf{B}} \rightsquigarrow \mathsf{true} \to$$

Operational Semantics

$$\frac{\langle b, \sigma\rangle_{\mathsf{B}} \rightsquigarrow \mathsf{true} \quad \langle c, \sigma\rangle_{\mathsf{C}} \rightsquigarrow \sigma' \quad \langle \mathsf{while}\ b\ \mathsf{do}\ c, \sigma'\rangle_{\mathsf{C}} \rightsquigarrow \sigma''}{\langle \mathsf{while}\ b\ \mathsf{do}\ c, \sigma\rangle_{\mathsf{C}} \rightsquigarrow \sigma''}$$

$$\Downarrow$$

$\langle\cdot,\cdot\rangle_{\mathsf{C}} \rightsquigarrow \cdot :$ $\mathsf{Command} \to \mathsf{State} \to \mathsf{State} \to \mathbf{Prop}$

$\vdots$

$\mathsf{eval\_while\_true} : \quad \forall b : \mathsf{BExp}; \forall c : \mathsf{Command}; \forall \sigma, \sigma', \sigma'' : \mathsf{State}$

$$\langle b, \sigma\rangle_{\mathsf{B}} \rightsquigarrow \mathsf{true} \to \langle c, \sigma\rangle_{\mathsf{C}} \rightsquigarrow \sigma'$$

INRIA SOPHIA ANTIPOLIS

Operational Semantics

$$\frac{\langle b,\sigma\rangle_{\mathsf{B}} \rightsquigarrow \mathsf{true} \quad \langle c,\sigma\rangle_{\mathsf{C}} \rightsquigarrow \sigma' \quad \langle \mathsf{while}\ b\ \mathsf{do}\ c,\sigma'\rangle_{\mathsf{C}} \rightsquigarrow \sigma''}{\langle \mathsf{while}\ b\ \mathsf{do}\ c,\sigma\rangle_{\mathsf{C}} \rightsquigarrow \sigma''}$$

$$\Downarrow$$

$\langle\cdot,\cdot\rangle_{\mathsf{C}} \rightsquigarrow \cdot :$ $\mathsf{Command} \to \mathsf{State} \to \mathsf{State} \to \mathbf{Prop}$

$\vdots$

$\mathsf{eval\_while\_true} : \quad \forall b : \mathsf{BExp}; \forall c : \mathsf{Command}; \forall \sigma, \sigma', \sigma'' : \mathsf{State}$

$\langle b,\sigma\rangle_{\mathsf{B}} \rightsquigarrow \mathsf{true} \to \langle c,\sigma\rangle_{\mathsf{C}} \rightsquigarrow \sigma'$

$\to \langle \mathsf{while}\ b\ \mathsf{do}\ c,\sigma'\rangle_{\mathsf{C}} \rightsquigarrow \sigma''$

Operational Semantics

$$\frac{\langle b, \sigma\rangle_{\mathsf{B}} \leadsto \mathsf{true} \quad \langle c, \sigma\rangle_{\mathsf{C}} \leadsto \sigma' \quad \langle \mathsf{while}\ b\ \mathsf{do}\ c, \sigma'\rangle_{\mathsf{C}} \leadsto \sigma''}{\langle \mathsf{while}\ b\ \mathsf{do}\ c, \sigma\rangle_{\mathsf{C}} \leadsto \sigma''}$$

$$\Downarrow$$

$\langle \cdot, \cdot\rangle_{\mathsf{C}} \leadsto \cdot :$ Command $\to$ State $\to$ State $\to$ **Prop**

$\vdots$

$$\mathsf{eval\_while\_true}: \ \forall b: \mathsf{BExp}; \forall c: \mathsf{Command}; \forall \sigma, \sigma', \sigma'': \mathsf{State}$$
$$\langle b, \sigma\rangle_{\mathsf{B}} \leadsto \mathsf{true} \to \langle c, \sigma\rangle_{\mathsf{C}} \leadsto \sigma'$$
$$\to \langle \mathsf{while}\ b\ \mathsf{do}\ c, \sigma'\rangle_{\mathsf{C}} \leadsto \sigma''$$
$$\to \langle \mathsf{while}\ b\ \mathsf{do}\ c, \sigma\rangle_{\mathsf{C}} \leadsto \sigma''$$

Where we are?

- A small imperative language - IMP
- Operational semantics for IMP
- Specification problem with denotational semantics and its solution
- Assurance of termination and accessibility predicate
- Solution in simultaneous induction-recursion
- Disentangle simultaneous induction-recursion for Coq
- Conclusion

Denotational Semantics

$$[\![\cdot]\!] : \mathsf{Command} \rightarrow \mathsf{State} \rightarrow \mathsf{State}$$

Denotational Semantics

$[\![\cdot]\!]$: $\mathsf{Command} \rightarrow \mathsf{State} \rightarrow \mathsf{State}$

$\vdots$

$[\![\mathsf{while}\ b\ \mathsf{do}\ c]\!]_\sigma :=$

Denotational Semantics

$[\![\cdot]\!]$: Command $\rightarrow$ State $\rightarrow$ State

$\vdots$

$[\![\text{while } b \text{ do } c]\!]_\sigma :=$ $\qquad$ $[\![b]\!]_\sigma$

Denotational Semantics

$[\![\cdot]\!]$: Command $\to$ State $\to$ State

$\vdots$

$[\![\text{while } b \text{ do } c]\!]_\sigma :=$ $\quad [\![c]\!]_\sigma \quad$ if $\quad [\![b]\!]_\sigma = \text{true}$

Denotational Semantics

$$[\![\cdot]\!] : \textsf{Command} \to \textsf{State} \to \textsf{State}$$

$$\vdots$$

$$[\![\textsf{while}\ b\ \textsf{do}\ c]\!]_\sigma := \quad [\![\textsf{while}\ b\ \textsf{do}\ c]\!]_{[\![c]\!]_\sigma} \quad \textsf{if} \quad [\![b]\!]_\sigma \ = \textsf{true}$$

Denotational Semantics

$$[\![\cdot]\!] : \mathsf{Command} \to \mathsf{State} \to \mathsf{State}$$

$$\vdots$$

$$[\![\mathsf{while}\ b\ \mathsf{do}\ c]\!]_\sigma := [\![\mathsf{while}\ b\ \mathsf{do}\ c]\!]_{[\![c]\!]_\sigma} \quad \text{if} \quad [\![b]\!]_\sigma = \mathsf{true}$$

Note: Recursive call is not structurally smaller

Second Order

$f : \mathsf{Command} \to \mathsf{State} \to \mathsf{State}$

Second Order

$f :$ Command $\rightarrow$ State $\rightarrow$ State

$\vdots$

$(\quad f \text{ (while } b \text{ do } c) \; \sigma) :=$

Second Order

$f:$ Command $\to$ State $\to$ State

$\vdots$

$$(\quad f \; (\text{while } b \text{ do } c) \; \sigma) := \left\{ \begin{array}{l} \\ \qquad [\![b]\!]_\sigma \end{array} \right.$$

Second Order

$f :$ Command $\rightarrow$ State $\rightarrow$ State

$\vdots$

$$(\quad f \text{ (while } b \text{ do } c) \ \sigma) := \begin{cases} & (f\ c\ \sigma) \\ \text{if } [\![b]\!]_\sigma = \text{true} & \end{cases}$$

Second Order

$f :$ Command $\to$ State $\to$ State

$\vdots$

$$(\quad f \;(\text{while } b \text{ do } c)\; \sigma) := \begin{cases} (f \;(\text{while } b \text{ do } c)\; (f\; c\; \sigma)) \\ \text{if } \;[\![b]\!]_\sigma = \text{true} \end{cases}$$

Second Order

- Using second order function

$$\vdots$$

$$(\textcolor{red}{\mathsf{F}}\ f\ (\mathsf{while}\ b\ \mathsf{do}\ c)\ \sigma) := \begin{cases} (f\ (\mathsf{while}\ b\ \mathsf{do}\ c)\ (f\ c\ \sigma)) \\ \mathsf{if}\ \ [\![b]\!]_\sigma = \mathsf{true} \end{cases}$$

Second Order

- Using second order function

$$\mathsf{F}: (\mathsf{Command} \to \mathsf{State} \to \mathsf{State})$$

$$\vdots$$

$$(\mathsf{F}\ f\ (\mathsf{while}\ b\ \mathsf{do}\ c)\ \sigma) := \begin{cases} (f\ (\mathsf{while}\ b\ \mathsf{do}\ c)\ (f\ c\ \sigma)) \\ \mathsf{if}\ \ [\![b]\!]_\sigma = \mathsf{true} \end{cases}$$

Second Order

- Using second order function

$$\mathsf{F} : (\mathsf{Command} \rightarrow \mathsf{State} \rightarrow \mathsf{State}) \rightarrow \mathsf{Command}$$

$$\vdots$$

$$(\mathsf{F}\ f\ (\mathsf{while}\ b\ \mathsf{do}\ c)\ \sigma) := \begin{cases} (f\ (\mathsf{while}\ b\ \mathsf{do}\ c)\ (f\ c\ \sigma)) \\ \mathsf{if}\ \ [\![b]\!]_\sigma = \mathsf{true} \end{cases}$$

Second Order

- Using second order function

$$\mathsf{F}: (\mathsf{Command} \to \mathsf{State} \to \mathsf{State}) \to \mathsf{Command} \to \mathsf{State}$$

$$\vdots$$

$$(\mathsf{F}\ f\ (\mathsf{while}\ b\ \mathsf{do}\ c)\ \sigma) := \begin{cases} (f\ (\mathsf{while}\ b\ \mathsf{do}\ c)\ (f\ c\ \sigma)) \\ \mathsf{if}\ \ [\![b]\!]_\sigma = \mathsf{true} \end{cases}$$

Second Order

- Using second order function

$$\mathsf{F} : (\mathsf{Command} \to \mathsf{State} \to \mathsf{State}) \to \mathsf{Command} \to \mathsf{State} \to \mathsf{State}$$

$$\vdots$$

$$(\mathsf{F}\ f\ (\mathsf{while}\ b\ \mathsf{do}\ c)\ \sigma) := \begin{cases} (f\ (\mathsf{while}\ b\ \mathsf{do}\ c)\ (f\ c\ \sigma)) \\ \mathsf{if}\ \ [\![b]\!]_\sigma = \mathsf{true} \end{cases}$$

Second Order

- Using second order function

$$\mathsf{F} : (\mathsf{Command} \to \mathsf{State} \to \mathsf{State}) \to \mathsf{Command} \to \mathsf{State} \to \mathsf{State}$$

$$\vdots$$

$$(\mathsf{F}\ f\ (\mathsf{while}\ b\ \mathsf{do}\ c)\ \sigma) := \begin{cases} (f\ (\mathsf{while}\ b\ \mathsf{do}\ c)\ (f\ c\ \sigma)) \\ \mathsf{if}\ \ [\![b]\!]_\sigma = \mathsf{true} \end{cases}$$

- Apply F as many times as needed

$$\mathsf{F}^k = \lambda g.\underbrace{(\mathsf{F}\ (\mathsf{F}\ \cdots\ (\mathsf{F}}_{k\ \text{times}}\ g)\ \cdots\ ))$$

Equivalence

$$\forall \sigma : \mathsf{State}. \forall a : \mathsf{AExp}. \forall n : \mathbb{N}. \qquad \langle a, \sigma \rangle \rightsquigarrow n \Leftrightarrow [\![a]\!]_\sigma = n.$$

Equivalence

$\forall \sigma : \text{State}. \forall a : \text{AExp}. \forall n : \mathbb{N}. \qquad \langle a, \sigma \rangle \rightsquigarrow n \Leftrightarrow [\![a]\!]_\sigma = n.$

$\forall \sigma : \text{State}. \forall b : \text{BExp}. \forall t : \mathbb{B}. \qquad \langle b, \sigma \rangle \rightsquigarrow t \Leftrightarrow [\![b]\!]_\sigma = t.$

Equivalence

$\forall \sigma : \mathsf{State}. \forall a : \mathsf{AExp}. \forall n : \mathbb{N}. \qquad \langle a, \sigma \rangle \rightsquigarrow n \Leftrightarrow [\![a]\!]_\sigma = n.$

$\forall \sigma : \mathsf{State}. \forall b : \mathsf{BExp}. \forall t : \mathbb{B}. \qquad \langle b, \sigma \rangle \rightsquigarrow t \Leftrightarrow [\![b]\!]_\sigma = t.$

$\forall c : \mathsf{Command}. \forall \sigma_1, \sigma_2 : \mathsf{State}.$

$\langle c, \sigma_1 \rangle \rightsquigarrow \sigma_2 \Rightarrow$

$\exists k : \mathbb{N}.\ \ \forall g : \mathsf{Command} \to \mathsf{State} \to \mathsf{State}. (\mathsf{F}^k\ g\ c\ \sigma_1) = \sigma_2$

INRIA SOPHIA ANTIPOLIS

Where we are?

- A small imperative language - IMP
- Operational semantics for IMP
- Specification problem with denotational semantics and its solution
- Assurance of termination and accessibility predicate
- Solution in simultaneous induction-recursion
- Disentangle simultaneous induction-recursion for Coq
- Conclusion

Characterizing terminating programs

$$D\colon \mathsf{Command} \to \mathsf{State} \to \mathbf{Prop}$$

$$\begin{aligned}(D\ c\ \sigma) := \quad & \exists k\colon \mathbb{N}. \quad \forall g_1, g_2\colon \mathsf{Command} \to \mathsf{State} \to \mathsf{State}.\\ & (\mathsf{F}^k\ g_1\ c\ \sigma) = (\mathsf{F}^k\ g_2\ c\ \sigma).\end{aligned}$$

Characterizing terminating programs

D: Command $\to$ State $\to$ **Prop**

$$(D\ c\ \sigma) := \ \exists k : \mathbb{N}. \quad \forall g_1, g_2 : \text{Command} \to \text{State} \to \text{State}.$$
$$(\mathsf{F}^k\ g_1\ c\ \sigma) = (\mathsf{F}^k\ g_2\ c\ \sigma).$$

Let's try:

$c :=$ while true do skip; exception.

Accessibility predicate

- For simple recursion:

$$\left\{ \begin{array}{l} f(e) := \cdots f(e_1) \cdots f(e_2) \cdots \\ \\ \end{array} \right.$$

Accessibility predicate

- For simple recursion:

$$\begin{cases} f(e) := \cdots f(e_1) \cdots f(e_2) \cdots \\ \hfill \mathsf{Acc}(e) \end{cases}$$

Accessibility predicate

- For simple recursion:

$$\begin{cases} f(e) := \cdots f(e_1) \cdots f(e_2) \cdots \\ \mathsf{Acc}(e_1) \rightarrow \qquad\qquad\qquad \mathsf{Acc}(e) \end{cases}$$

Accessibility predicate

- For simple recursion:

$$\begin{cases} f(e) := \cdots f(e_1) \cdots f(e_2) \cdots \\ \mathsf{Acc}(e_1) \to \mathsf{Acc}(e_2) \to \mathsf{Acc}(e) \end{cases}$$

Accessibility predicate

- For simple recursion:

$$\begin{cases} f(e) := \cdots f(e_1) \cdots f(e_2) \cdots \\ \mathsf{Acc}(e_1) \to \mathsf{Acc}(e_2) \to \mathsf{Acc}(e) \end{cases}$$

- What will happen for nested recursion?

$$\begin{cases} f(e) := \cdots f(f(e')) \cdots \\ \end{cases}$$

Accessibility predicate

- For simple recursion:

$$\begin{cases} f(e) := \cdots f(e_1) \cdots f(e_2) \cdots \\ \mathsf{Acc}(e_1) \to \mathsf{Acc}(e_2) \to \mathsf{Acc}(e) \end{cases}$$

- What will happen for nested recursion?

$$\begin{cases} f(e) := \cdots f(f(e')) \cdots \\ \qquad\qquad\qquad\qquad\qquad \mathsf{Acc}(e) \end{cases}$$

Accessibility predicate

- For simple recursion:

$$\begin{cases} f(e) := \cdots f(e_1) \cdots f(e_2) \cdots \\ \mathsf{Acc}(e_1) \to \mathsf{Acc}(e_2) \to \mathsf{Acc}(e) \end{cases}$$

- What will happen for nested recursion?

$$\begin{cases} f(e) := \cdots f(f(e')) \cdots \\ \mathsf{Acc}(e') \to \qquad\qquad\qquad\qquad \mathsf{Acc}(e) \end{cases}$$

INRIA SOPHIA ANTIPOLIS

Accessibility predicate

- For simple recursion:

$$\begin{cases} f(e) := \cdots f(e_1) \cdots f(e_2) \cdots \\ \mathsf{Acc}(e_1) \to \mathsf{Acc}(e_2) \to \mathsf{Acc}(e) \end{cases}$$

- What will happen for nested recursion?

$$\begin{cases} f(e) := \cdots f(f(e')) \cdots \\ \mathsf{Acc}(e') \to \color{red}{\mathsf{Acc}(f(e'))} \to \mathsf{Acc}(e) \end{cases}$$

Where we are?

- A small imperative language - IMP
- Operational semantics for IMP
- Specification problem with denotational semantics and its solution
- Assurance of termination and accessibility predicate
- Solution in simultaneous induction-recursion
- Disentangle simultaneous induction-recursion for Coq
- Conclusion

Example

$$\text{while } b \text{ do } c$$

$$\Downarrow$$

$$\text{if } \boxed{b = \text{true}} \text{ then } \{ \boxed{\boxed{c} \;;\; \text{while } b \text{ do } c} \}$$

Simultaneous induction-recursion

comAcc: $\mathsf{Command} \to \mathsf{State} \to \mathbf{Prop}$

Simultaneous induction-recursion

$\mathsf{comAcc}\colon \mathsf{Command} \to \mathsf{State} \to \mathbf{Prop}$

$\vdots$

$\mathsf{accWhile\_true}\colon \quad \forall b\colon \mathsf{BExp};\ \forall c\colon \mathsf{Command};\ \forall \sigma\colon \mathsf{State}.$

Simultaneous induction-recursion

$\mathsf{comAcc}\colon \mathsf{Command} \to \mathsf{State} \to \mathbf{Prop}$

$\vdots$

$\mathsf{accWhile\_true}\colon\ \ \forall b\colon \mathsf{BExp};\ \forall c\colon \mathsf{Command};\ \forall \sigma\colon \mathsf{State}.\ \ [\![b]\!] = \mathsf{true}$

Simultaneous induction-recursion

$\mathsf{comAcc}\colon \mathsf{Command} \to \mathsf{State} \to \mathbf{Prop}$

$\vdots$

$\mathsf{accWhile\_true}\colon \quad \forall b\colon \mathsf{BExp};\ \forall c\colon \mathsf{Command};\ \forall \sigma\colon \mathsf{State}.\ \ [\![b]\!] = \mathsf{true}$
$\qquad \to (h\colon (\mathsf{comAcc}\ c\ \sigma))$

Simultaneous induction-recursion

$\mathsf{comAcc}\colon \mathsf{Command} \to \mathsf{State} \to \mathbf{Prop}$

$\vdots$

$\mathsf{accWhile\_true}\colon\ \forall b\colon \mathsf{BExp};\ \forall c\colon \mathsf{Command};\ \forall \sigma\colon \mathsf{State}.\ [\![b]\!] = \mathsf{true}$

$\to (h\colon (\mathsf{comAcc}\ c\ \sigma))(h'\colon (\mathsf{comAcc}\ (\mathsf{while}\ b\ \mathsf{do}\ c)\ [\![c]\!]^h_\sigma))$

Simultaneous induction-recursion

$\mathsf{comAcc}\colon \mathsf{Command} \to \mathsf{State} \to \mathbf{Prop}$

$\vdots$

$$\begin{aligned}\mathsf{accWhile\_true}\colon\ & \forall b\colon \mathsf{BExp};\ \forall c\colon \mathsf{Command};\ \forall \sigma\colon \mathsf{State}.\ \ [\![b]\!] = \mathsf{true} \\ & \to (h\colon (\mathsf{comAcc}\ c\ \sigma))(h'\colon (\mathsf{comAcc}\ (\mathsf{while}\ b\ \mathsf{do}\ c)\ [\![c]\!]^h_\sigma)) \\ & \to (\mathsf{comAcc}(\mathsf{while}\ b\ \mathsf{do}\ c)\ \sigma)\end{aligned}$$

Simultaneous induction-recursion

$\mathsf{comAcc} \colon \mathsf{Command} \to \mathsf{State} \to \mathbf{Prop}$

$[\![\cdot]\!] : (c \colon \mathsf{Command}; \sigma \colon \mathsf{State})(\mathsf{comAcc}\ c\ \sigma) \to \mathsf{State}$

$\vdots$

$$\begin{aligned}\mathsf{accWhile\_true} \colon\ & \forall b \colon \mathsf{BExp};\ \forall c \colon \mathsf{Command};\ \forall \sigma \colon \mathsf{State}.\ [\![b]\!] = \mathsf{true} \\ & \to (h \colon (\mathsf{comAcc}\ c\ \sigma))(h' \colon (\mathsf{comAcc}\ (\mathsf{while}\ b\ \mathsf{do}\ c)\ [\![c]\!]^h_\sigma)) \\ & \to (\mathsf{comAcc}(\mathsf{while}\ b\ \mathsf{do}\ c)\ \sigma)\end{aligned}$$

Simultaneous induction-recursion

$\mathsf{comAcc}\colon \mathsf{Command} \to \mathsf{State} \to \mathbf{Prop}$

$[\![\cdot]\!] : (c\colon \mathsf{Command}; \sigma\colon \mathsf{State})(\mathsf{comAcc}\ c\ \sigma) \to \mathsf{State}$

$\vdots$

$$\begin{aligned}\mathsf{accWhile\_true}\colon\ & \forall b\colon \mathsf{BExp};\ \forall c\colon \mathsf{Command};\ \forall \sigma\colon \mathsf{State}.\ \ [\![b]\!] = \mathsf{true}\\ & \to (h\colon (\mathsf{comAcc}\ c\ \sigma))(h'\colon (\mathsf{comAcc}\ (\mathsf{while}\ b\ \mathsf{do}\ c)\ [\![c]\!]_\sigma^h))\\ & \to (\mathsf{comAcc}(\mathsf{while}\ b\ \mathsf{do}\ c)\ \sigma)\end{aligned}$$

$\vdots$

$$[\![\mathsf{while}\ b\ \mathsf{do}\ c]\!]_\sigma^{(\mathsf{accWhile\_true}\ b\ c\ \sigma\ p\ h\ h')} := [\![\mathsf{while}\ b\ \mathsf{do}\ c]\!]_{[\![c]\!]_\sigma^h}^{h'}$$

$\vdots$

Where we are?

- A small imperative language - IMP
- Operational semantics for IMP
- Specification problem with denotational semantics and its solution
- Assurance of termination and accessibility predicate
- Solution in simultaneous induction-recursion
- Disentangle simultaneous induction-recursion for Coq
- Conclusion

INRIA
SOPHIA ANTIPOLIS

Extending . . .

comAcc:

$\mathsf{Command} \to \mathsf{State} \to \mathbb{N} \to (\mathsf{Command} \to \mathsf{State} \to \mathsf{State}) \to \mathbf{Prop}$

Extending . . .

comAcc :

$\mathsf{Command} \to \mathsf{State} \to \mathbb{N} \to (\mathsf{Command} \to \mathsf{State} \to \mathsf{State}) \to \mathbf{Prop}$

$\vdots$

accWhile_true : $\forall b\colon \mathsf{BExp};\ \forall c\colon \mathsf{Command};\ \forall \sigma\colon \mathsf{State};$
$\forall k\colon \mathbb{N};\ \forall f\colon \mathsf{Command} \to \mathsf{State} \to \mathsf{State}.$

$\vdots$

Extending ...

comAcc:

$\text{Command} \to \text{State} \to \mathbb{N} \to (\text{Command} \to \text{State} \to \text{State}) \to \mathbf{Prop}$

$\vdots$

accWhile_true: $\forall b\colon \text{BExp};\ \forall c\colon \text{Command};\ \forall \sigma\colon \text{State};$

$\forall k\colon \mathbb{N};\ \forall f\colon \text{Command} \to \text{State} \to \text{State}.$

$\langle b, \sigma \rangle \rightsquigarrow \text{true}$

$\vdots$

Extending . . .

comAcc:

$\text{Command} \to \text{State} \to \mathbb{N} \to (\text{Command} \to \text{State} \to \text{State}) \to \mathbf{Prop}$

$\vdots$

accWhile_true: $\forall b\colon \text{BExp};\ \forall c\colon \text{Command};\ \forall \sigma\colon \text{State};$

$\forall k\colon \mathbb{N};\ \forall f\colon \text{Command} \to \text{State} \to \text{State}.$

$\langle b, \sigma \rangle \rightsquigarrow \text{true}$

$\to (\text{comAcc}\ c\ \sigma\ k\ f)$

$\vdots$

Extending . . .

comAcc :

$\text{Command} \to \text{State} \to \mathbb{N} \to (\text{Command} \to \text{State} \to \text{State}) \to \mathbf{Prop}$

$\vdots$

accWhile_true : $\forall b : \text{BExp};\ \forall c : \text{Command};\ \forall \sigma : \text{State};$

$\forall k : \mathbb{N};\ \forall f : \text{Command} \to \text{State} \to \text{State}.$

$\langle b, \sigma \rangle \rightsquigarrow \text{true}$

$\to (\text{comAcc}\ c\ \sigma\ k\ f)$

$\to (\text{comAcc}\ (\text{while}\ b\ \text{do}\ c)\ (F_f^k\ c\ \sigma))$

$\vdots$

Extending . . .

comAcc :

$\mathsf{Command} \to \mathsf{State} \to \mathbb{N} \to (\mathsf{Command} \to \mathsf{State} \to \mathsf{State}) \to \mathbf{Prop}$

$\vdots$

accWhile_true : $\forall b \colon \mathsf{BExp};\ \forall c \colon \mathsf{Command};\ \forall \sigma \colon \mathsf{State};$

$\forall k \colon \mathbb{N};\ \forall f \colon \mathsf{Command} \to \mathsf{State} \to \mathsf{State}.$

$\langle b, \sigma \rangle \rightsquigarrow \mathsf{true}$

$\to (\mathsf{comAcc}\ c\ \sigma\ k\ f)$

$\to (\mathsf{comAcc}\ (\mathsf{while}\ b\ \mathsf{do}\ c)\ (F_f^k\ c\ \sigma))$

$\to (\mathsf{comAcc}(\mathsf{while}\ b\ \mathsf{do}\ c)\ \sigma\ (k+1)\ f)$

$\vdots$

INRIA SOPHIA ANTIPOLIS

Using Accessibility Predicate

- Defining domain

$$\mathsf{comDom}\colon \mathsf{Command} \to \mathsf{State} \to \mathbf{Set}$$

$$(\mathsf{comDom}\ c\ \sigma) = \Sigma k\colon \mathbb{N}.\forall f\colon \mathsf{Command} \to \mathsf{State} \to \mathsf{State}.(\mathsf{comAcc}\ c\ \sigma\ k\ f)$$

Using Accessibility Predicate

- Defining domain

$$\mathsf{comDom} \colon \mathsf{Command} \to \mathsf{State} \to \mathbf{Set}$$

$$(\mathsf{comDom}\ c\ \sigma) = \Sigma k \colon \mathbb{N}.\forall f \colon \mathsf{Command} \to \mathsf{State} \to \mathsf{State}.(\mathsf{comAcc}\ c\ \sigma\ k\ f)$$

- Defining execution function

$$[\![\cdot]\!] \colon \quad (c \colon \ \ldots; \quad \sigma \colon \ \ldots; \quad f \colon \ \ldots)\ (\mathsf{comDom}\ c\ \sigma) \to \mathsf{State}$$

Finally ...

$\forall c\colon \mathsf{Command}.\forall \sigma, \sigma' : \mathsf{State}.$

$\langle c, \sigma \rangle \leadsto \sigma' \Leftrightarrow$

$\exists H : (\mathsf{comDom}\ c\ \sigma).\ \ \forall f : \mathsf{Command} \to \mathsf{State} \to \mathsf{State}.\quad [\![c]\!]^{H}_{\sigma,f} = \sigma'.$

Where we are?

- A small imperative language - IMP
- Operational semantics for IMP
- Specification problem with denotational semantics and its solution.
- Assurance of termination and accessibility predicate
- Solution in simultaneous induction-recursion
- Disentangle simultaneous induction-recursion for Coq
- Conclusion

Conclusion

- Developed operational and denotational semantics inside type theory(Coq)

Conclusion

- Developed operational and denotational semantics inside type theory(Coq)

- Have a soundness and completeness theorem stating operational and denotational semantics agree

Merci!