Proof Styles in Operational Semantics

Sandip Ray and J Strother Moore

Department of Computer Science
University of Texas at Austin

Email: {sandip,moore}@cs.utexas.edu
web: http://www.cs.utexas.edu/users/{sandip,moore}

Contributions

- We study two styles of proofs used in deductive verification of sequential programs modeled with operational semantics.
  - Inductive Invariants
  - Clock functions

Contributions

- We study two styles of proofs used in deductive verification of sequential programs modeled with operational semantics.
  - Inductive Invariants
  - Clock functions
- We show by mechanical proof that the two styles are equivalent in logical strength (in a sufficiently expressive logic).

Contributions

- We study two styles of proofs used in deductive verification of sequential programs modeled with operational semantics.
  - Inductive Invariants
  - Clock functions
- We show by mechanical proof that the two styles are equivalent in logical strength (in a sufficiently expressive logic).
- The results hold for both total correctness and partial correctness.

Contributions

- We study two styles of proofs used in deductive verification of sequential programs modeled with operational semantics.
  - Inductive Invariants
  - Clock functions
- We show that the two styles are equivalent in logical strength (in a sufficiently expressive logic).
- The results hold for both total correctness and partial correctness.
- The equivalence allows us to use different styles in proofs of different program components.

Outline

- Operational Semantics
- Proof Styles
  - Inductive Invariants
  - Clock Functions
- Equivalence Theorems
- Proof Composition
- Discussions

Operational Semantics

“The meaning of a program is defined by its effect on the state vector.” – John McCarthy, 1962.

Operational Semantics

"The meaning of a program is defined by its effect on the state vector." – John McCarthy, 1962.

- We model states of the machine executing the program as objects (n -tuples) in some mathematical logic.
  - Two special state components are the pc and the program being executed.

Operational Semantics

"The meaning of a program is defined by its effect on the state vector." – John McCarthy, 1962.

- We model states of the machine executing the program as objects (n -tuples) in some mathematical logic.
  - Two special state components are the pc and the program being executed.
- A program is an object, e.g., a sequence of instructions.
  - The semantics of a program is given by defining a language interpreter, which is a function on states.

Operational Semantics

$effect(s, i)$ specifies the state obtained by executing instruction i on state s.

Operational Semantics

$\mathit{effect}(s, i)$ specifies the state obtained by executing instruction i on state s.

Let I be the instruction in $\mathit{program}(s)$ pointed to by $\mathit{pc}(s)$. Then $\mathit{step}(s)$ is defined to be $\mathit{effect}(s, I)$.

Operational Semantics

$effect(s, i)$ specifies the state obtained by executing instruction i on state s.

Let I be the instruction in $program(s)$ pointed to by $pc(s)$. Then $step(s)$ is defined to be $effect(s, I)$.

We define the concept of running the machine for n steps from state s by the function run:

$$run(s, n) \triangleq \begin{cases} s & \text{if } n = 0 \\ run(step(s), n-1) & \text{otherwise} \end{cases}$$

Operational Semantics

$effect(s, i)$ specifies the state obtained by executing instruction i on state s.

Let I be the instruction in $program(s)$ pointed to by $pc(s)$. Then $step(s)$ is defined to be $effect(s, I)$.

We define the concept of running the machine for n steps from state s by the function run:

$$run(s, n) \triangleq \begin{cases} s & \text{if } n = 0 \\ run(step(s), n-1) & \text{otherwise} \end{cases}$$

Predicate $halting(s) \triangleq (step(s) = s)$ specifies termination.

Correctness in Operational Semantics

Informal correctness statement:

If the machine starts from a state p satisfying some desired precondition, then when it reaches a halting state q, the state q satisfies the desired postcondition.

Formally, there are two notions of correctness:

- Partial Correctness
- Total correctness

Partial Correctness

For a state p satisfying the precondition if a $halting$ state q is reachable from p, then such $halting$ state must satisfy the postcondition.

Formally:

$$\forall s, n : pre(s) \land halting(run(s, n)) \Rightarrow post(run(s, n)).$$

Total Correctness

Total correctness is partial correctness along with a guarantee that a $halting$ state is eventually reached.

Termination

$\forall s : pre(s) \Rightarrow (\exists n : halting(run(s, n)))$.

Outline

- Operational Semantics
- Proof Styles
  - Inductive Invariants
  - Clock Functions
- Equivalence Theorems
- Proof Composition
- Discussions

Inductive Invariants

Define the function inv such that the following are theorems:

1. $\forall s : pre(s) \Rightarrow inv(s)$

2. $\forall s : inv(s) \Rightarrow inv(step(s))$

3. $\forall s : inv(s) \wedge halting(s) \Rightarrow post(s)$

The three theorems above guarantee partial correctness.

Inductive Invariants

For total correctness, we define in addition a function m that maps the machine states to a well-founded set with some ordering relation $\prec$.

4. $\forall s : inv(s) \wedge \neg halting(s) \Rightarrow m(step(s)) \prec m(s)$

Well-foundedness guarantees that some $halting$ state is eventually reached.

Outline

- Operational Semantics
- Proof Styles
  - Inductive Invariants
  - Clock Functions
- Equivalence Theorems
- Proof Composition
- Discussions

Clock Functions

Define a function $clock$ that maps machine states to natural numbers. The number tells us how many steps are left before termination.

Formally, we prove the following theorems:

1. $\forall s : pre(s) \Rightarrow halting(run(s, clock(s)))$

2. $\forall s : pre(s) \Rightarrow post(run(s, clock(s)))$

The two theorems above guarantee total correctness.

Clock Functions

For partial correctness, we weaken the theorems a little bit.

1. $\forall s, n : pre(s) \wedge halting(run(s, n)) \Rightarrow halting(run(s, clock(s)))$

2. $\forall s, n : pre(s) \wedge halting(run(s, n)) \Rightarrow post(run(s, clock(s)))$

Proof Style Issues

Both inductive invariants and clock functions guarantee correctness.

But the approaches of the two styles are different.

Are the theorems proved in one style stronger or weaker than the theorems in the other?

Proof Style Issues

Clock functions require that we define a function $clock$ so that $clock(s)$ is the number of steps required to reach a $halting$ state from state s.

Proof Style Issues

Clock functions require that we define a function $clock$ so that $clock(s)$ is the number of steps required to reach a $halting$ state from state s.

But this number is precisely the time complexity of the program.

Proof Style Issues

Clock functions require that we define a function $clock$ so that $clock(s)$ is the number of steps required to reach a $halting$ state from state s.

But this number is precisely the time complexity of the program.

On the other hand, inductive invariants require us to define inv so that $inv(s)$ holds for every reachable state.

Outline

- Operational Semantics
- Proof Styles
  - Inductive Invariants
  - Clock Functions
- Equivalence Theorems
- Proof Composition
- Discussions

Inductive Invariants to Clock Functions

Assume we have been given an inductive invariant proof of total correctness.

1. $\forall s : pre(s) \Rightarrow inv(s)$
2. $\forall s : inv(s) \Rightarrow inv(step(s))$
3. $\forall s : inv(s) \wedge halting(s) \Rightarrow post(s)$
4. $\forall s : inv(s) \wedge \neg halting(s) \Rightarrow m(step(s)) \prec m(s)$

Inductive Invariants to Clock Functions

Assume we have been given an inductive invariant proof of total correctness.

1. $\forall s : pre(s) \Rightarrow inv(s)$

2. $\forall s : inv(s) \Rightarrow inv(step(s))$

3. $\forall s : inv(s) \wedge halting(s) \Rightarrow post(s)$

4. $\forall s : inv(s) \wedge \neg halting(s) \Rightarrow m(step(s)) \prec m(s)$

How do we define a $clock$?

Inductive Invariants to Clock Functions

$$clock(s) \triangleq \begin{cases} 0 & \text{if } halting(s) \vee \neg inv(s) \\ clock(step(s)) + 1 & \text{otherwise} \end{cases}$$

Is this definition sound? Yes. The recursion terminates:

- $\forall s : inv(s) \wedge \neg halting(s) \Rightarrow m(step(s)) \prec m(s)$

Inductive Invariants to Clock Functions

$$clock(s) \triangleq \begin{array}{ll} 0 & \text{if } halting(s) \vee \neg inv(s) \\ clock(step(s)) + 1 & \text{otherwise} \end{array}$$

Is this definition sound? Yes. The recursion terminates:

- $\forall s : inv(s) \wedge \neg halting(s) \Rightarrow m(step(s)) \prec m(s)$

By induction we can prove:

- $\forall s : inv(s) \Rightarrow halting(run(s, clock(s)))$

Partial Correctness

For partial correctness, we do not have the mapping m.

We define the following predicate:

$$haltexists(s) \triangleq (\exists n : halting(run(s, n)))$$

Partial Correctness

For partial correctness, we do not have the mapping m.

We define the following predicate:

$$haltexists(s) \triangleq (\exists n : halting(run(s, n)))$$

Let $wit(s)$ be the Skolem witness for this predicate.

Define $clock(s) \triangleq wit(s)$.

Partial Correctness

For partial correctness, we do not have the mapping m.

We define the following predicate:

$$haltexists(s) \triangleq (\exists n : halting(run(s, n)))$$

Let $wit(s)$ be the Skolem witness for this predicate.

Define $clock(s) \triangleq wit(s)$.

Easy to prove:

$$\forall s, n : pre(s) \wedge halting(run(s, n)) \Rightarrow halting(run(s, clock(s))).$$

Clock Functions to Inductive Invariants

Given the clock function theorems:

1. $\forall s : pre(s) \Rightarrow halting(run(s, clock(s)))$

2. $\forall s : pre(s) \Rightarrow post(run(s, clock(s)))$

Can we define inv?

Clock Functions to Inductive Invariants

Given the clock function theorems:

1. $\forall s : pre(s) \Rightarrow halting(run(s, clock(s)))$

2. $\forall s : pre(s) \Rightarrow post(run(s, clock(s)))$

Can we define inv? Yes.

$$inv(s) \triangleq (\exists init, i : pre(init) \wedge (s = run(init, i)))$$

$inv(s)$ is an inductive invariant.

Outline

- Operational Semantics
- Proof Styles
  - Inductive Invariants
  - Clock Functions
- Equivalence Theorems
- Proof Composition
- Discussions

Proof Composition

How do we verify sub-routines (or any other program component)?

- A return from a sub-routine is not a halt.

Proof Composition

How do we verify sub-routines (or any other program components?

- A return from a sub-routine is not a halt.
- We replace the predicate $halting$ in the two proof styles with an arbitary predicate $external$.
  - The predicate $external$ indicates the exit from a program component.

Proof Composition

How do we verify sub-routines (or any other program components?

- A return from a sub-routine is not a halt.
- We replace the predicate $halting$ in the two proof styles with an arbitary predicate $external$
  - The predicate $external$ indicates the exit from a program component.
- We adjust the two styles so that the postcondition is required to hold on the first exit from a component.

Proof Composition

How do we verify sub-routines (or any other program components?

- A return from a sub-routine is not a halt.
- We replace the predicate $halting$ in the two proof styles with an arbitary predicate $external$
  - The predicate $external$ indicates the exit from a program component.
- We adjust the two styles so that the postcondition is required to hold on the first exit from a component.
- The adjusted proof styles are also proved to be equivalent.

Proof Composition

Sequential composition of programs is achieved by a trivial observation about run.

$$run(s, m + n) = run(run(s, m), n)$$

Proof Composition

Sequential composition of programs is achieved by a trivial observation about run.

$$run(s, m + n) = run(run(s, m), n)$$

If $clock_1$ and $clock_2$ are clock functions for two sequential blocks, the clock function of the composition is given by:

$$clock(s) \triangleq clock_1(s) + clock_2(run(s, clock_1(s)))$$

Proof Composition

Sequential composition of programs is achieved by a trivial observation about run.

$$run(s, m + n) = run(run(s, m), n)$$

If $clock_1$ and $clock_2$ are clock functions for two sequential blocks, the clock function of the composition is given by:

$$clock(s) \triangleq clock_1(s) + clock_2(run(s, clock_1(s)))$$

Other compositions, like conditional, loop, etc. can be built out of sequential compositions.

Outline

- Operational Semantics
- Proof Styles
  - Inductive Invariants
  - Clock Functions
- Equivalence Theorems
- Proof Composition
- Discussions

Discussions

The equivalence theorems have been mechanically verified with the ACL2 theorem prover.

- We have used ACL2's encapsulation feature to verify the equivalence theorems for generic functions $step$, inv, m, and $clock$.
- We have implemented macros that automatically convert proofs from one style to the other by instantiation of the generic equivalence theorems.

Discussions

- We do not advocate one proof style over the other.
  - One proof style might be easier to apply in a certain context than the other.
- Since the two styles are shown to be equivalent, we can now go back and forth between them.
  - Each program component can be verified using the style best suited for that component.
- Our work also shows the importance of using first-order quantification effectively in mechanical theorem proving.