Program Verification: Lecture 25 (part I)

José Meseguer

University of Illinois
at Urbana-Champaign

Proving Loop Invariants

Recall that to prove a loop invariant of a while loop
while b:$(\vec{x})$ $\{stmt\}$ parametric on K and data parameters $\vec{Y}$,

Proving Loop Invariants

Recall that to prove a loop invariant of a while loop **while** $b{:}(\vec{x})\ \{stmt\}$ parametric on K and data parameters $\vec{Y}$, we need to guess a strong-enough invariant $I_{str}(\vec{Y}, \vec{X})$ such that

Proving Loop Invariants

Recall that to prove a loop invariant of a while loop **while** $b{:}(\vec{x})\ \{stmt\}$ parametric on K and data parameters $\vec{Y}$, we need to guess a strong-enough invariant $I_{str}(\vec{Y}, \vec{X})$ such that it is preserved by $stmt$.

Proving Loop Invariants

Recall that to prove a loop invariant of a while loop **while** $b{:}(\vec{x})\ \{stmt\}$ parametric on K and data parameters $\vec{Y}$, we need to guess a strong-enough invariant $I_{str}(\vec{Y}, \vec{X})$ such that it is preserved by $stmt$. Then, the goal we can send to the IMPL Prover is:

Proving Loop Invariants

Recall that to prove a loop invariant of a while loop **while** $b{:}(\vec{x})\ \{stmt\}$ parametric on K and data parameters $\vec{Y}$, we need to guess a strong-enough invariant $I_{str}(\vec{Y}, \vec{X})$ such that it is preserved by $stmt$. Then, the goal we can send to the IMPL Prover is:

$$< \textbf{while}\ b{:}(\vec{x})\ \{stmt\} \rightsquigarrow K \mid TS\ \&\ \vec{x} \mapsto \vec{X} * VS >|\ I_{str}(\vec{Y}, \vec{X})$$

$$\rightarrow^{\circledast} < K \mid TS\ \&\vec{x} \mapsto \vec{X} * VS >|\ I_{str}(\vec{Y}, \vec{X'})$$

Proving Loop Invariants

Recall that to prove a loop invariant of a while loop **while** $b{:}(\vec{x})\ \{stmt\}$ parametric on K and data parameters $\vec{Y}$, we need to guess a strong-enough invariant $I_{str}(\vec{Y}, \vec{X})$ such that it is preserved by $stmt$. Then, the goal we can send to the IMPL Prover is:

$$< \textbf{while}\ b{:}(\vec{x})\ \{stmt\} \rightsquigarrow K \mid TS\ \&\ \vec{x} \mapsto \vec{X} * VS >|\ I_{str}(\vec{Y}, \vec{X})$$
$$\rightarrow^{\circledast} < K \mid TS\ \&\vec{x} \mapsto \vec{X} * VS >|\ I_{str}(\vec{Y}, \vec{X'})$$

We shall illustrate with two examples how properties of loops in IMPL can be proved this way.

The Migrate Program

Consider the simple program migrate, whose purpose is to copy the list initially stored in x to the (initially empty) variable y:

The Migrate Program

Consider the simple program migrate, whose purpose is to copy the list initially stored in x to the (initially empty) variable y:

migrate := **while** $(!empty(x))$ $\{y =_l y\ \$\colon first(x)\ ;\ x =_l rest(x)\ ;\}$

Intuitively, we would like to prove that **migrate** does copy the list in x to y.

The Migrate Program

Consider the simple program migrate, whose purpose is to copy the list initially stored in x to the (initially empty) variable y:

migrate := **while** $(!empty(x))\ \{y =_l y\ \$:\ first(x)\ ;\ x =_l rest(x)\ ;\}$

Intuitively, we would like to prove that **migrate** does copy the list in x to y. But we can prove a more general property, namely, that if initially x holds X and y holds Y, then, after completion, y will hold $X\$Y$.

The Migrate Program

Consider the simple program migrate, whose purpose is to copy the list initially stored in x to the (initially empty) variable y:

migrate := **while** $(!empty(x))$ $\{y =_l y\ \$\colon first(x)\ ;\ x =_l rest(x)\ ;\}$

Intuitively, we would like to prove that **migrate** does copy the list in x to y. But we can prove a more general property, namely, that if initially x holds X and y holds Y, then, after completion, y will hold $X\$Y$. Our original property is the case $Y = nil$.

The Migrate Program

Consider the simple program migrate, whose purpose is to copy the list initially stored in x to the (initially empty) variable y:

migrate := **while** $(!empty(x))$ $\{y =_l y\ \$: first(x)\ ;\ x =_l rest(x)\ ;\}$

Intuitively, we would like to prove that **migrate** does copy the list in x to y. But we can prove a more general property, namely, that if initially x holds X and y holds Y, then, after completion, y will hold $X\$Y$. Our original property is the case $Y = nil$.

How can we specify this property as a loop invariant?

The Migrate Program

Consider the simple program migrate, whose purpose is to copy the list initially stored in x to the (initially empty) variable y:

migrate := **while** $(!empty(x))$ $\{y =_l y \$: first(x)\ ;\ x =_l rest(x)\ ;\}$

Intuitively, we would like to prove that **migrate** does copy the list in x to y. But we can prove a more general property, namely, that if initially x holds X and y holds Y, then, after completion, y will hold $X\$Y$. Our original property is the case $Y = nil$.

How can we specify this property as a loop invariant? To ease the specification, we will introduce an auxiliary variable z not mentioned in the loop,

The Migrate Program

Consider the simple program migrate, whose purpose is to copy the list initially stored in x to the (initially empty) variable y:

migrate := **while** $(!empty(x))$ $\{y =_l y \$: first(x)\ ;\ x =_l rest(x)\ ;\}$

Intuitively, we would like to prove that **migrate** does copy the list in x to y. But we can prove a more general property, namely, that if initially x holds X and y holds Y, then, after completion, y will hold $X\$Y$. Our original property is the case $Y = nil$.

How can we specify this property as a loop invariant? To ease the specification, we will introduce an auxiliary variable z not mentioned in the loop, initially holding $Z = X\$Y$,

The Migrate Program

Consider the simple program migrate, whose purpose is to copy the list initially stored in x to the (initially empty) variable y:

migrate := **while** $(!empty(x))$ $\{y =_l y \$: first(x)\ ;\ x =_l rest(x)\ ;\}$

Intuitively, we would like to prove that **migrate** does copy the list in x to y. But we can prove a more general property, namely, that if initially x holds X and y holds Y, then, after completion, y will hold $X\$Y$. Our original property is the case $Y = nil$.

How can we specify this property as a loop invariant? To ease the specification, we will introduce an auxiliary variable z not mentioned in the loop, initially holding $Z = X\$Y$, so that Z will be used as a data parameter.

The Migrate Program

Consider the simple program migrate, whose purpose is to copy the list initially stored in x to the (initially empty) variable y:

migrate := **while** $(!empty(x))$ $\{y =_l y \$: first(x)\ ;\ x =_l rest(x)\ ;\}$

Intuitively, we would like to prove that **migrate** does copy the list in x to y. But we can prove a more general property, namely, that if initially x holds X and y holds Y, then, after completion, y will hold $X\$Y$. Our original property is the case $Y = nil$.

How can we specify this property as a loop invariant? To ease the specification, we will introduce an auxiliary variable z not mentioned in the loop, initially holding $Z = X\$Y$, so that Z will be used as a data parameter. Therefore:

The Migrate Program (II)

We will use the initial store VS_0:

The Migrate Program (II)

We will use the initial store VS_0:

$$VS_0 := x \mapsto X \ * \ y \mapsto Y \ * \ z \mapsto Z$$

The Migrate Program (II)

We will use the initial store VS_0:

$$VS_0 := x \mapsto X \ * \ y \mapsto Y \ * \ z \mapsto Z$$

(with the further assumption $Z = X\$Y$),

The Migrate Program (II)

We will use the initial store VS_0:

$$VS_0 := x \mapsto X \ * \ y \mapsto Y \ * \ z \mapsto Z$$

(with the further assumption $Z = X\$Y$), and final store VS:

The Migrate Program (II)

We will use the initial store VS_0:

$$VS_0 := x \mapsto X \ * \ y \mapsto Y \ * \ z \mapsto Z$$

(with the further assumption $Z = X\$Y$), and final store VS:

$$VS := x \mapsto X' \ * \ y \mapsto Y' \ * \ z \mapsto Z'$$

The Migrate Program (II)

We will use the initial store VS_0:

$$VS_0 := x \mapsto X \ * \ y \mapsto Y \ * \ z \mapsto Z$$

(with the further assumption $Z = X\$Y$), and final store VS:

$$VS := x \mapsto X' \ * \ y \mapsto Y' \ * \ z \mapsto Z'$$

All we now need to do is to guess the invariant

The Migrate Program (II)

We will use the initial store VS_0:

$$VS_0 := x \mapsto X \ * \ y \mapsto Y \ * \ z \mapsto Z$$

(with the further assumption $Z = X\$Y$), and final store VS:

$$VS := x \mapsto X' \ * \ y \mapsto Y' \ * \ z \mapsto Z'$$

All we now need to do is to guess the invariant by focusing on the loop's body $\{y =_l y\ \$\colon first(x)\ ;\ x =_l rest(x)\ ;\}$

Finding Migrate's Loop Invariant

Need to understand what changes and how executing the body.

Finding Migrate's Loop Invariant

Need to understand what changes and how executing the body.

Assume $X = X_1\$X_2\$\ldots\$X_n$ is the list stored in x.

Finding Migrate's Loop Invariant

Need to understand what changes and how executing the body.

Assume $X = X_1\$X_2\$\ldots\$X_n$ is the list stored in x. During the first iteration, we

Finding Migrate's Loop Invariant

Need to understand what changes and how executing the body.

Assume $X = X_1\$X_2\$\ldots\$X_n$ is the list stored in x. During the first iteration, we take the first element X_1, of X stored in x, and

Finding Migrate's Loop Invariant

Need to understand what changes and how executing the body.

Assume $X = X_1\$X_2\$\ldots\$X_n$ is the list stored in x. During the first iteration, we take the first element X_1, of X stored in x, and append it to the end of the list stored in y,

Finding Migrate's Loop Invariant

Need to understand what changes and how executing the body.

Assume $X = X_1\$X_2\$\ldots\$X_n$ is the list stored in x. During the first iteration, we take the first element X_1, of X stored in x, and append it to the end of the list stored in y, which becomes $Y^1 = Y\$X_1$.

Finding Migrate's Loop Invariant

Need to understand what changes and how executing the body.

Assume $X = X_1\$X_2\$\ldots\$X_n$ is the list stored in x. During the first iteration, we take the first element X_1, of X stored in x, and append it to the end of the list stored in y, which becomes $Y^1 = Y\$X_1$. x is then assigned to the rest of X, i.e.,

Finding Migrate's Loop Invariant

Need to understand what changes and how executing the body.

Assume $X = X_1\$X_2\$\ldots\$X_n$ is the list stored in x. During the first iteration, we take the first element X_1, of X stored in x, and append it to the end of the list stored in y, which becomes $Y^1 = Y\$X_1$. x is then assigned to the rest of X, i.e., x now holds $X^1 = X_2\$X_3\$\ldots\$X_n$.

Finding Migrate's Loop Invariant

Need to understand what changes and how executing the body.

Assume $X = X_1\$X_2\$\ldots\$X_n$ is the list stored in x. During the first iteration, we take the first element X_1, of X stored in x, and append it to the end of the list stored in y, which becomes $Y^1 = Y\$X_1$. x is then assigned to the rest of X, i.e., x now holds $X^1 = X_2\$X_3\$\ldots\$X_n$. Therefore,

$$Z = Y\$X = Y\$X_1\$X_2\$\ldots\$X_n = Y^1\$X^1$$

Finding Migrate's Loop Invariant

Need to understand what changes and how executing the body.

Assume $X = X_1\$X_2\$\ldots\$X_n$ is the list stored in x. During the first iteration, we take the first element X_1, of X stored in x, and append it to the end of the list stored in y, which becomes $Y^1 = Y\$X_1$. x is then assigned to the rest of X, i.e., x now holds $X^1 = X_2\$X_3\$\ldots\$X_n$. Therefore,

$$Z = Y\$X = Y\$X_1\$X_2\$\ldots\$X_n = Y^1\$X^1$$

Similarly, after k iterations we get:

Finding Migrate's Loop Invariant

Need to understand what changes and how executing the body.

Assume $X = X_1\$X_2\$\ldots\$X_n$ is the list stored in x. During the first iteration, we take the first element X_1, of X stored in x, and append it to the end of the list stored in y, which becomes $Y^1 = Y\$X_1$. x is then assigned to the rest of X, i.e., x now holds $X^1 = X_2\$X_3\$\ldots\$X_n$. Therefore,

$$Z = Y\$X = Y\$X_1\$X_2\$\ldots\$X_n = Y^1\$X^1$$

Similarly, after k iterations we get: $Y^k = Y\$X_1\$\ldots\$X_k$ and $X^k = X_{k+1}\$\ldots\$X_n$,

Finding Migrate's Loop Invariant

Need to understand what changes and how executing the body.

Assume $X = X_1\$X_2\$\ldots\$X_n$ is the list stored in x. During the first iteration, we take the first element X_1, of X stored in x, and append it to the end of the list stored in y, which becomes $Y^1 = Y\$X_1$. x is then assigned to the rest of X, i.e., x now holds $X^1 = X_2\$X_3\$\ldots\$X_n$. Therefore,

$$Z = Y\$X = Y\$X_1\$X_2\$\ldots\$X_n = Y^1\$X^1$$

Similarly, after k iterations we get: $Y^k = Y\$X_1\$\ldots\$X_k$ and $X^k = X_{k+1}\$\ldots\$X_n$, so that $Z = Y\$X = Y^k\$X^k$.

Finding Migrate's Loop Invariant

Need to understand what changes and how executing the body.

Assume $X = X_1\$X_2\$\ldots\$X_n$ is the list stored in x. During the first iteration, we take the first element X_1, of X stored in x, and append it to the end of the list stored in y, which becomes $Y^1 = Y\$X_1$. x is then assigned to the rest of X, i.e., x now holds $X^1 = X_2\$X_3\$\ldots\$X_n$. Therefore,

$$Z = Y\$X = Y\$X_1\$X_2\$\ldots\$X_n = Y^1\$X^1$$

Similarly, after k iterations we get: $Y^k = Y\$X_1\$\ldots\$X_k$ and $X^k = X_{k+1}\$\ldots\$X_n$, so that $Z = Y\$X = Y^k\$X^k$.

This clearly suggests the loop invariant $I(Z, X, Y) \equiv (Z = Y\$X)$

Finding Migrate's Loop Invariant

Need to understand what changes and how executing the body.

Assume $X = X_1\$X_2\$\ldots\$X_n$ is the list stored in x. During the first iteration, we take the first element X_1, of X stored in x, and append it to the end of the list stored in y, which becomes $Y^1 = Y\$X_1$. x is then assigned to the rest of X, i.e., x now holds $X^1 = X_2\$X_3\$\ldots\$X_n$. Therefore,

$$Z = Y\$X = Y\$X_1\$X_2\$\ldots\$X_n = Y^1\$X^1$$

Similarly, after k iterations we get: $Y^k = Y\$X_1\$\ldots\$X_k$ and $X^k = X_{k+1}\$\ldots\$X_n$, so that $Z = Y\$X = Y^k\$X^k$.

This clearly suggests the loop invariant $I(Z, X, Y) \equiv (Z = Y\$X)$ parametic on Z,

Finding Migrate's Loop Invariant

Need to understand what changes and how executing the body.

Assume $X = X_1\$X_2\$\ldots\$X_n$ is the list stored in x. During the first iteration, we take the first element X_1, of X stored in x, and append it to the end of the list stored in y, which becomes $Y^1 = Y\$X_1$. x is then assigned to the rest of X, i.e., x now holds $X^1 = X_2\$X_3\$\ldots\$X_n$. Therefore,

$$Z = Y\$X = Y\$X_1\$X_2\$\ldots\$X_n = Y^1\$X^1$$

Similarly, after k iterations we get: $Y^k = Y\$X_1\$\ldots\$X_k$ and $X^k = X_{k+1}\$\ldots\$X_n$, so that $Z = Y\$X = Y^k\$X^k$.

This clearly suggests the loop invariant $I(Z, X, Y) \equiv (Z = Y\$X)$ parametic on Z, and the goal:

Finding Migrate's Loop Invariant (II)

$$< \textbf{while}\ (!empty(x))\ \{y =_l y\ \$:\ first(x)\ ;\ x =_l rest(x)\ ;\} \rightsquigarrow K$$
$$\quad \mid TS\ \&\ x \mapsto X\ *\ y \mapsto Y\ *\ z \mapsto Z * VS > \mid Z = Y\$X\ \rightarrow^{\circledast}$$
$$< K \mid TS\ \&\ x \mapsto X'\ *\ y \mapsto Y'\ *\ z \mapsto Z' * VS > \mid Z = Y'\$X'$$

Finding Migrate's Loop Invariant (II)

$$< \textbf{while}\ (!empty(x))\ \{y =_l y\ \$:\ first(x)\ ;\ x =_l rest(x)\ ;\} \rightsquigarrow K$$
$$\mid TS\ \&\ x \mapsto X\ *\ y \mapsto Y\ *\ z \mapsto Z * VS >\ \mid Z = Y\$X\ \rightarrow^{\circledast}$$
$$< K \mid TS\ \&\ x \mapsto X'\ *\ y \mapsto Y'\ *\ z \mapsto Z' * VS >\ \mid Z = Y'\$X'$$

But does this loop invariant capture the properties we had in mind?

Finding Migrate's Loop Invariant (II)

$$< \textbf{while}\ (!empty(x))\ \{y =_l y\ \$:\ first(x)\ ;\ x =_l rest(x)\ ;\} \rightsquigarrow K$$
$$\mid TS\ \&\ x \mapsto X\ *\ y \mapsto Y\ *\ z \mapsto Z * VS >\ \mid Z = Y\$X\ \rightarrow^{\circledast}$$
$$< K \mid TS\ \&\ x \mapsto X'\ *\ y \mapsto Y'\ *\ z \mapsto Z' * VS >\ \mid Z = Y'\$X'$$

But does this loop invariant capture the properties we had in mind? Yes.

Finding Migrate's Loop Invariant (II)

$$< \textbf{while}\ (!empty(x))\ \{y =_l y\ \$:\ first(x)\ ;\ x =_l rest(x)\ ;\} \rightsquigarrow K$$
$$\mid TS\ \&\ x \mapsto X\ *\ y \mapsto Y\ *\ z \mapsto Z * VS >\ \mid Z = Y\$X\ \rightarrow^{\circledast}$$
$$< K \mid TS\ \&\ x \mapsto X'\ *\ y \mapsto Y'\ *\ z \mapsto Z' * VS >\ \mid Z = Y'\$X'$$

But does this loop invariant capture the properties we had in mind? Yes. Because:

Finding Migrate's Loop Invariant (II)

$$< \textbf{while}\ (!empty(x))\ \{y =_l y\ \$:\ first(x)\ ;\ x =_l rest(x)\ ;\} \rightsquigarrow K$$
$$\mid TS\ \&\ x \mapsto X\ *\ y \mapsto Y\ *\ z \mapsto Z * VS >\ \mid Z = Y\$X\ \rightarrow^{\circledast}$$
$$< K \mid TS\ \&\ x \mapsto X'\ *\ y \mapsto Y'\ *\ z \mapsto Z' * VS >\ \mid Z = Y'\$X'$$

But does this loop invariant capture the properties we had in mind? Yes. Because:

1. Since $(!empty(x))$ is the loop's guard, we know that $X' = nil$,

Finding Migrate's Loop Invariant (II)

$$< \textbf{while}\ (!empty(x))\ \{y =_l y\ \$:\ first(x)\ ;\ x =_l rest(x)\ ;\} \rightsquigarrow K$$
$$\mid TS\ \&\ x \mapsto X\ *\ y \mapsto Y\ *\ z \mapsto Z * VS >\ \mid Z = Y\$X\ \rightarrow^{\circledast}$$
$$< K \mid TS\ \&\ x \mapsto X'\ *\ y \mapsto Y'\ *\ z \mapsto Z' * VS >\ \mid Z = Y'\$X'$$

But does this loop invariant capture the properties we had in mind? Yes. Because:

1. Since $(!empty(x))$ is the loop's guard, we know that $X' = nil$, and therefore $Y' = Y'\&X' = Z = Y\$X$.

Finding Migrate's Loop Invariant (II)

$$< \textbf{while}\ (!empty(x))\ \{y =_l y\ \$:\ first(x)\ ;\ x =_l rest(x)\ ;\} \rightsquigarrow K$$
$$\mid TS\ \&\ x \mapsto X\ *\ y \mapsto Y\ *\ z \mapsto Z * VS >\ \mid Z = Y\$X\ \rightarrow^{\circledast}$$
$$< K \mid TS\ \&\ x \mapsto X'\ *\ y \mapsto Y'\ *\ z \mapsto Z' * VS >\ \mid Z = Y'\$X'$$

But does this loop invariant capture the properties we had in mind? Yes. Because:

1. Since $(!empty(x))$ is the loop's guard, we know that $X' = nil$, and therefore $Y' = Y'\&X' = Z = Y\$X$.
2. When, furthermore, $Y = nil$,

Finding Migrate's Loop Invariant (II)

$$< \textbf{while}\ (!empty(x))\ \{y =_l y\ \$:\ first(x)\ ;\ x =_l rest(x)\ ;\} \rightsquigarrow K$$
$$\mid TS\ \&\ x \mapsto X\ *\ y \mapsto Y\ *\ z \mapsto Z * VS >\ \mid Z = Y\$X\ \rightarrow^{\circledast}$$
$$< K \mid TS\ \&\ x \mapsto X'\ *\ y \mapsto Y'\ *\ z \mapsto Z' * VS >\ \mid Z = Y'\$X'$$

But does this loop invariant capture the properties we had in mind? Yes. Because:

1. Since $(!empty(x))$ is the loop's guard, we know that $X' = nil$, and therefore $Y' = Y'\&X' = Z = Y\$X$.
2. When, furthermore, $Y = nil$, we get $Y' = Y'\&X' = Z = Y\$X = X$,

Finding Migrate's Loop Invariant (II)

$$< \textbf{while}\ (!empty(x))\ \{y =_l y\ \$:\ first(x)\ ;\ x =_l rest(x)\ ;\} \rightsquigarrow K$$
$$\mid TS\ \&\ x \mapsto X\ *\ y \mapsto Y\ *\ z \mapsto Z * VS >\ \mid Z = Y\$X\ \rightarrow^{\circledast}$$
$$< K \mid TS\ \&\ x \mapsto X'\ *\ y \mapsto Y'\ *\ z \mapsto Z' * VS >\ \mid Z = Y'\$X'$$

But does this loop invariant capture the properties we had in mind? Yes. Because:

1. Since $(!empty(x))$ is the loop's guard, we know that $X' = nil$, and therefore $Y' = Y'\&X' = Z = Y\$X$.
2. When, furthermore, $Y = nil$, we get $Y' = Y'\&X' = Z = Y\$X = X$, as desired.

Finding Migrate's Loop Invariant (II)

$$< \textbf{while}\ (!empty(x))\ \{y =_l y\ \$:\ first(x)\ ;\ x =_l rest(x)\ ;\} \rightsquigarrow K$$
$$\mid TS\ \&\ x \mapsto X\ *\ y \mapsto Y\ *\ z \mapsto Z * VS >\ \mid Z = Y\$X\ \rightarrow^{\circledast}$$
$$< K \mid TS\ \&\ x \mapsto X'\ *\ y \mapsto Y'\ *\ z \mapsto Z' * VS >\ \mid Z = Y'\$X'$$

But does this loop invariant capture the properties we had in mind? Yes. Because:

1. Since $(!empty(x))$ is the loop's guard, we know that $X' = nil$, and therefore $Y' = Y'\&X' = Z = Y\$X$.
2. When, furthermore, $Y = nil$, we get $Y' = Y'\&X' = Z = Y\$X = X$, as desired.

A proof script to verify this loop invariant can be found in:

Finding Migrate's Loop Invariant (II)

$$< \textbf{while}\ (!empty(x))\ \{y =_l y\ \$:\ first(x)\ ;\ x =_l rest(x)\ ;\} \rightsquigarrow K$$
$$\mid TS\ \&\ x \mapsto X\ *\ y \mapsto Y\ *\ z \mapsto Z * VS >\ \mid Z = Y\$X\ \rightarrow^{\circledast}$$
$$< K \mid TS\ \&\ x \mapsto X'\ *\ y \mapsto Y'\ *\ z \mapsto Z' * VS >\ \mid Z = Y'\$X'$$

But does this loop invariant capture the properties we had in mind? Yes. Because:

1. Since $(!empty(x))$ is the loop's guard, we know that $X' = nil$, and therefore $Y' = Y'\&X' = Z = Y\$X$.
2. When, furthermore, $Y = nil$, we get $Y' = Y'\&X' = Z = Y\$X = X$, as desired.

A proof script to verify this loop invariant can be found in: M. Abir and J. Meseguer, *Continuation Semantics and Program Verification for the IMPL Language* (CSPV.IMPL).

The Reverse Program

Consider the program

The Reverse Program

Consider the program

reverse $:=$ **while** $(!empty(x))\ \{y =_l first(x)\ \$:\ y\ ;\ x =_l rest(x)\ ;\}$

The Reverse Program

Consider the program

reverse $:=$ **while** $(!empty(x))\ \{y =_l first(x)\ \$:\ y\ ;\ x =_l rest(x)\ ;\}$

whose purpose is to compute and store in an initially empty variable y the reverse of the list X initially stored in x.

The Reverse Program

Consider the program

reverse $:=$ **while** $(!empty(x))\ \{y =_l first(x)\ \$:\ y\ ;\ x =_l rest(x)\ ;\}$

whose purpose is to compute and store in an initially empty variable y the reverse of the list X initially stored in x. We would like to prove that **reverse** is true to its name and does indeed achieve this purpose.

The Reverse Program

Consider the program

reverse := **while** $(!empty(x))\ \{y =_l first(x)\ \$:\ y\ ;\ x =_l rest(x)\ ;\}$

whose purpose is to compute and store in an initially empty variable y the reverse of the list X initially stored in x. We would like to prove that **reverse** is true to its name and does indeed achieve this purpose. We can generalize and conquer by not assuming anything about the initial contents Y of y and proving that, after program completion, y will hold $rev(X)\$Y$,

The Reverse Program

Consider the program

reverse $:=$ **while** $(!empty(x))\ \{y =_l first(x)\ \$:\ y\ ;\ x =_l rest(x)\ ;\}$

whose purpose is to compute and store in an initially empty variable y the reverse of the list X initially stored in x. We would like to prove that **reverse** is true to its name and does indeed achieve this purpose. We can generalize and conquer by not assuming anything about the initial contents Y of y and proving that, after program completion, y will hold $rev(X)\$Y$, where rev denotes the list-reversing function.

The Reverse Program

Consider the program

reverse := **while** $(!empty(x))$ $\{y =_l first(x)$ \$: y ; $x =_l rest(x)$; $\}$

whose purpose is to compute and store in an initially empty variable y the reverse of the list X initially stored in x. We would like to prove that **reverse** is true to its name and does indeed achieve this purpose. We can generalize and conquer by not assuming anything about the initial contents Y of y and proving that, after program completion, y will hold $rev(X)\$Y$, where rev denotes the list-reversing function. We will then get our original result as the special case $Y = nil$.

The Reverse Program

Consider the program

reverse := **while** $(!empty(x))\ \{y =_l first(x)\ \$:\ y\ ;\ x =_l rest(x)\ ;\}$

whose purpose is to compute and store in an initially empty variable y the reverse of the list X initially stored in x. We would like to prove that **reverse** is true to its name and does indeed achieve this purpose. We can generalize and conquer by not assuming anything about the initial contents Y of y and proving that, after program completion, y will hold $rev(X)\$Y$, where rev denotes the list-reversing function. We will then get our original result as the special case $Y = nil$.

As for **migrate**, to ease the specification, we will introduce an auxiliary variable z not mentioned in the loop,

The Reverse Program

Consider the program

reverse := **while** $(!empty(x))\ \{y =_l first(x)\ \$:\ y\ ;\ x =_l rest(x)\ ;\}$

whose purpose is to compute and store in an initially empty variable y the reverse of the list X initially stored in x. We would like to prove that **reverse** is true to its name and does indeed achieve this purpose. We can generalize and conquer by not assuming anything about the initial contents Y of y and proving that, after program completion, y will hold $rev(X)\$Y$, where rev denotes the list-reversing function. We will then get our original result as the special case $Y = nil$.

As for **migrate**, to ease the specification, we will introduce an auxiliary variable z not mentioned in the loop, initially holding $Z = rev(Y)\$X$,

The Reverse Program

Consider the program

reverse := **while** $(!empty(x))\ \{y =_l first(x)\ \$:\ y\ ;\ x =_l rest(x)\ ;\}$

whose purpose is to compute and store in an initially empty variable y the reverse of the list X initially stored in x. We would like to prove that **reverse** is true to its name and does indeed achieve this purpose. We can generalize and conquer by not assuming anything about the initial contents Y of y and proving that, after program completion, y will hold $rev(X)\$Y$, where rev denotes the list-reversing function. We will then get our original result as the special case $Y = nil$.

As for **migrate**, to ease the specification, we will introduce an auxiliary variable z not mentioned in the loop, initially holding $Z = rev(Y)\$X$, so that Z will be used as a data parameter.

Finding Reverse's Loop Invariant

We should look at the body!

$$\{y =_l first(x)\ \$\!:\ y\ ;\ x =_l rest(x)\ ;\}$$

Finding Reverse's Loop Invariant

We should look at the body!

$$\{y =_l first(x)\ \$:\ y\ ;\ x =_l rest(x)\ ;\}$$

Assuming X is the list $X = X_1\$X_2\$\ldots\$X_n$,

Finding Reverse's Loop Invariant

We should look at the body!

$$\{y =_l first(x)\ \$:\ y\ ;\ x =_l rest(x)\ ;\}$$

Assuming X is the list $X = X_1\$X_2\$\ldots\$X_n$, after one iteration we will have: $Y^1 = X_1\$Y$

Finding Reverse's Loop Invariant

We should look at the body!

$$\{y =_l first(x)\ \$:\ y\ ;\ x =_l rest(x)\ ;\}$$

Assuming X is the list $X = X_1\$X_2\$\ldots\$X_n$, after one iteration we will have: $Y^1 = X_1\$Y$ and $X^1 = X_2\$\ldots\$X_n$.

Finding Reverse's Loop Invariant

We should look at the body!

$$\{y =_l first(x)\ \$:\ y\ ;\ x =_l rest(x)\ ;\}$$

Assuming X is the list $X = X_1\$X_2\$\ldots\$X_n$, after one iteration we will have: $Y^1 = X_1\$Y$ and $X^1 = X_2\$\ldots\$X_n$. After $k \leq n$ iterations we will have: $Y^k = X_k\$X_{k-1}\$\ldots\$X_1\$Y$

Finding Reverse's Loop Invariant

We should look at the body!

$$\{y =_l first(x)\ \$:\ y\ ;\ x =_l rest(x)\ ;\}$$

Assuming X is the list $X = X_1\$X_2\$\ldots\$X_n$, after one iteration we will have: $Y^1 = X_1\$Y$ and $X^1 = X_2\$\ldots\$X_n$. After $k \leq n$ iterations we will have: $Y^k = X_k\$X_{k-1}\$\ldots\$X_1\$Y$ and $X^k = X_{k+1}\$\ldots\$X_n$.

Finding Reverse's Loop Invariant

We should look at the body!

$$\{y =_l first(x)\ \$:\ y\ ;\ x =_l rest(x)\ ;\}$$

Assuming X is the list $X = X_1\$X_2\$\ldots\$X_n$, after one iteration we will have: $Y^1 = X_1\$Y$ and $X^1 = X_2\$\ldots\$X_n$. After $k \leq n$ iterations we will have: $Y^k = X_k\$X_{k-1}\$\ldots\$X_1\$Y$ and $X^k = X_{k+1}\$\ldots\$X_n$.

Aha! but this means that: $rev(Y^k) = rev(Y)\$X_1\$X_2\$\ldots\$X_k$,

Finding Reverse's Loop Invariant

We should look at the body!

$$\{y =_l first(x)\ \$:\ y\ ;\ x =_l rest(x)\ ;\}$$

Assuming X is the list $X = X_1\$X_2\$\ldots\$X_n$, after one iteration we will have: $Y^1 = X_1\$Y$ and $X^1 = X_2\$\ldots\$X_n$. After $k \leq n$ iterations we will have: $Y^k = X_k\$X_{k-1}\$\ldots\$X_1\$Y$ and $X^k = X_{k+1}\$\ldots\$X_n$.

Aha! but this means that: $rev(Y^k) = rev(Y)\$X_1\$X_2\$\ldots\$X_k$, and therefore that,

Finding Reverse's Loop Invariant

We should look at the body!

$$\{y =_l first(x)\ \$:\ y\ ;\ x =_l rest(x)\ ;\}$$

Assuming X is the list $X = X_1\$X_2\$\ldots\$X_n$, after one iteration we will have: $Y^1 = X_1\$Y$ and $X^1 = X_2\$\ldots\$X_n$. After $k \leq n$ iterations we will have: $Y^k = X_k\$X_{k-1}\$\ldots\$X_1\$Y$ and $X^k = X_{k+1}\$\ldots\$X_n$.

Aha! but this means that: $rev(Y^k) = rev(Y)\$X_1\$X_2\$\ldots\$X_k$, and therefore that, $rev(Y^k)\$X^k = rev(Y)\$X = Z$ holds for any $0 \leq k \leq n$.

Finding Reverse's Loop Invariant

We should look at the body!

$$\{y =_l first(x)\ \$:\ y\ ;\ x =_l rest(x)\ ;\}$$

Assuming X is the list $X = X_1\$X_2\$\ldots\$X_n$, after one iteration we will have: $Y^1 = X_1\$Y$ and $X^1 = X_2\$\ldots\$X_n$. After $k \leq n$ iterations we will have: $Y^k = X_k\$X_{k-1}\$\ldots\$X_1\$Y$ and $X^k = X_{k+1}\$\ldots\$X_n$.

Aha! but this means that: $rev(Y^k) = rev(Y)\$X_1\$X_2\$\ldots\$X_k$, and therefore that, $rev(Y^k)\$X^k = rev(Y)\$X = Z$ holds for any $0 \leq k \leq n$.

So we have found our desired loop invariant, $I(Z, X, Y)$, parametric on Z,

Finding Reverse's Loop Invariant

We should look at the body!

$$\{y =_l first(x)\ \$:\ y\ ;\ x =_l rest(x)\ ;\}$$

Assuming X is the list $X = X_1\$X_2\$\ldots\$X_n$, after one iteration we will have: $Y^1 = X_1\$Y$ and $X^1 = X_2\$\ldots\$X_n$. After $k \leq n$ iterations we will have: $Y^k = X_k\$X_{k-1}\$\ldots\$X_1\$Y$ and $X^k = X_{k+1}\$\ldots\$X_n$.

Aha! but this means that: $rev(Y^k) = rev(Y)\$X_1\$X_2\$\ldots\$X_k$, and therefore that, $rev(Y^k)\$X^k = rev(Y)\$X = Z$ holds for any $0 \leq k \leq n$.

So we have found our desired loop invariant, $I(Z, X, Y)$, parametric on Z, namely, $I(Z, X, Y) \equiv (Z = rev(Y)\$X)$.

Finding Reverse's Loop Invariant

We should look at the body!

$$\{y =_l first(x)\ \$:\ y\ ;\ x =_l rest(x)\ ;\}$$

Assuming X is the list $X = X_1\$X_2\$\ldots\$X_n$, after one iteration we will have: $Y^1 = X_1\$Y$ and $X^1 = X_2\$\ldots\$X_n$. After $k \leq n$ iterations we will have: $Y^k = X_k\$X_{k-1}\$\ldots\$X_1\$Y$ and $X^k = X_{k+1}\$\ldots\$X_n$.

Aha! but this means that: $rev(Y^k) = rev(Y)\$X_1\$X_2\$\ldots\$X_k$, and therefore that, $rev(Y^k)\$X^k = rev(Y)\$X = Z$ holds for any $0 \leq k \leq n$.

So we have found our desired loop invariant, $I(Z, X, Y)$, parametric on Z, namely, $I(Z, X, Y) \equiv (Z = rev(Y)\$X)$. Therefore we get the goal:

Finding Reverse's Loop Invariant (II)

$$< \textbf{while}\ (!empty(x))\ \{y =_l first(x)\ \$:\ y\ ;\ x =_l rest(x)\ ;\} \rightsquigarrow K$$
$$\mid TS\ \&\ x \mapsto X\ *\ y \mapsto Y\ *\ z \mapsto Z * VS > \mid Z = rev(Y)\$X\ \rightarrow^{\circledast}$$
$$< K \mid TS\ \&\ x \mapsto X'\ *\ y \mapsto Y'\ *\ z \mapsto Z' * VS > \mid Z = rev(Y')\$X'$$

Finding Reverse's Loop Invariant (II)

$$< \textbf{while}\ (!empty(x))\ \{y =_l first(x)\ \$:\ y\ ;\ x =_l rest(x)\ ;\} \rightsquigarrow K$$
$$\mid TS\ \&\ x \mapsto X\ *\ y \mapsto Y\ *\ z \mapsto Z * VS > \mid Z = rev(Y)\$X\ \rightarrow^{\circledast}$$
$$< K \mid TS\ \&\ x \mapsto X'\ *\ y \mapsto Y'\ *\ z \mapsto Z' * VS > \mid Z = rev(Y')\$X'$$

But does this loop invariant capture the properties we had in mind?

Finding Reverse's Loop Invariant (II)

$$< \textbf{while}\ (!empty(x))\ \{y =_l first(x)\ \$:\ y\ ;\ x =_l rest(x)\ ;\} \rightsquigarrow K$$
$$\mid TS\ \&\ x \mapsto X\ *\ y \mapsto Y\ *\ z \mapsto Z * VS > \mid Z = rev(Y)\$X\ \rightarrow^{\circledast}$$
$$< K \mid TS\ \&\ x \mapsto X'\ *\ y \mapsto Y'\ *\ z \mapsto Z' * VS > \mid Z = rev(Y')\$X'$$

But does this loop invariant capture the properties we had in mind? Yes.

Finding Reverse's Loop Invariant (II)

$$< \textbf{while}\ (!empty(x))\ \{y =_l first(x)\ \$:\ y\ ;\ x =_l rest(x)\ ;\} \rightsquigarrow K$$
$$\mid TS\ \&\ x \mapsto X\ *\ y \mapsto Y\ *\ z \mapsto Z * VS > \mid Z = rev(Y)\$X\ \rightarrow^{\circledast}$$
$$< K \mid TS\ \&\ x \mapsto X'\ *\ y \mapsto Y'\ *\ z \mapsto Z' * VS > \mid Z = rev(Y')\$X'$$

But does this loop invariant capture the properties we had in mind? Yes. Because:

Finding Reverse's Loop Invariant (II)

$$< \textbf{while}\ (!empty(x))\ \{y =_l first(x)\ \$:\ y\ ;\ x =_l rest(x)\ ;\} \rightsquigarrow K$$
$$\mid TS\ \&\ x \mapsto X\ *\ y \mapsto Y\ *\ z \mapsto Z * VS > \mid Z = rev(Y)\$X\ \rightarrow^{\circledast}$$
$$< K \mid TS\ \&\ x \mapsto X'\ *\ y \mapsto Y'\ *\ z \mapsto Z' * VS > \mid Z = rev(Y')\$X'$$

But does this loop invariant capture the properties we had in mind? Yes. Because:

1. Since $(!empty(x))$ is the loop's guard, we know that $X' = nil$,

Finding Reverse's Loop Invariant (II)

$$< \textbf{while}\ (!empty(x))\ \{y =_l first(x)\ \$:\ y\ ;\ x =_l rest(x)\ ;\} \rightsquigarrow K$$
$$\mid TS\ \&\ x \mapsto X\ *\ y \mapsto Y\ *\ z \mapsto Z * VS > \mid Z = rev(Y)\$X\ \rightarrow^{\circledast}$$
$$< K \mid TS\ \&\ x \mapsto X'\ *\ y \mapsto Y'\ *\ z \mapsto Z' * VS > \mid Z = rev(Y')\$X'$$

But does this loop invariant capture the properties we had in mind? Yes. Because:

1. Since $(!empty(x))$ is the loop's guard, we know that $X' = nil$, and therefore $rev(Z) = rev(X')\$Y' = Y' = rev(X)\$Y$.

Finding Reverse's Loop Invariant (II)

$$< \textbf{while}\ (!empty(x))\ \{y =_l first(x)\ \$:\ y\ ;\ x =_l rest(x)\ ;\} \rightsquigarrow K$$
$$\mid TS\ \&\ x \mapsto X\ *\ y \mapsto Y\ *\ z \mapsto Z * VS > \mid Z = rev(Y)\$X\ \rightarrow^{\circledast}$$
$$< K \mid TS\ \&\ x \mapsto X'\ *\ y \mapsto Y'\ *\ z \mapsto Z' * VS > \mid Z = rev(Y')\$X'$$

But does this loop invariant capture the properties we had in mind? Yes. Because:

1. Since $(!empty(x))$ is the loop's guard, we know that $X' = nil$, and therefore $rev(Z) = rev(X')\$Y' = Y' = rev(X)\$Y$.
2. When, furthermore, $Y = nil$,

Finding Reverse's Loop Invariant (II)

$$< \textbf{while}\ (!empty(x))\ \{y =_l first(x)\ \$:\ y\ ;\ x =_l rest(x)\ ;\} \rightsquigarrow K$$
$$\mid TS\ \&\ x \mapsto X\ *\ y \mapsto Y\ *\ z \mapsto Z * VS > \mid Z = rev(Y)\$X\ \rightarrow^{\circledast}$$
$$< K \mid TS\ \&\ x \mapsto X'\ *\ y \mapsto Y'\ *\ z \mapsto Z' * VS > \mid Z = rev(Y')\$X'$$

But does this loop invariant capture the properties we had in mind? Yes. Because:

1. Since $(!empty(x))$ is the loop's guard, we know that $X' = nil$, and therefore $rev(Z) = rev(X')\$Y' = Y' = rev(X)\$Y$.
2. When, furthermore, $Y = nil$, we get

Finding Reverse's Loop Invariant (II)

$$< \textbf{while}\ (!empty(x))\ \{y =_l first(x)\ \$:\ y\ ;\ x =_l rest(x)\ ;\} \rightsquigarrow K$$
$$\mid TS\ \&\ x \mapsto X\ *\ y \mapsto Y\ *\ z \mapsto Z * VS > \mid Z = rev(Y)\$X\ \rightarrow^{\circledast}$$
$$< K \mid TS\ \&\ x \mapsto X'\ *\ y \mapsto Y'\ *\ z \mapsto Z' * VS > \mid Z = rev(Y')\$X'$$

But does this loop invariant capture the properties we had in mind? Yes. Because:

1. Since $(!empty(x))$ is the loop's guard, we know that $X' = nil$, and therefore $rev(Z) = rev(X')\$Y' = Y' = rev(X)\$Y$.
2. When, furthermore, $Y = nil$, we get $Y' = rev(X)$,

Finding Reverse's Loop Invariant (II)

$$< \textbf{while}\ (!empty(x))\ \{y =_l first(x)\ \$:\ y\ ;\ x =_l rest(x)\ ;\} \rightsquigarrow K$$
$$\mid TS\ \&\ x \mapsto X\ *\ y \mapsto Y\ *\ z \mapsto Z * VS > \mid Z = rev(Y)\$X\ \rightarrow^{\circledast}$$
$$< K \mid TS\ \&\ x \mapsto X'\ *\ y \mapsto Y'\ *\ z \mapsto Z' * VS > \mid Z = rev(Y')\$X'$$

But does this loop invariant capture the properties we had in mind? Yes. Because:

1. Since $(!empty(x))$ is the loop's guard, we know that $X' = nil$, and therefore $rev(Z) = rev(X')\$Y' = Y' = rev(X)\$Y$.
2. When, furthermore, $Y = nil$, we get $Y' = rev(X)$, as desired.

Finding Reverse's Loop Invariant (II)

$$< \textbf{while}\ (!empty(x))\ \{y =_l first(x)\ \$:\ y\ ;\ x =_l rest(x)\ ;\} \rightsquigarrow K$$
$$\mid TS\ \&\ x \mapsto X\ *\ y \mapsto Y\ *\ z \mapsto Z * VS > \mid Z = rev(Y)\$X\ \rightarrow^{\circledast}$$
$$< K \mid TS\ \&\ x \mapsto X'\ *\ y \mapsto Y'\ *\ z \mapsto Z' * VS > \mid Z = rev(Y')\$X'$$

But does this loop invariant capture the properties we had in mind? Yes. Because:

1. Since $(!empty(x))$ is the loop's guard, we know that $X' = nil$, and therefore $rev(Z) = rev(X')\$Y' = Y' = rev(X)\$Y$.
2. When, furthermore, $Y = nil$, we get $Y' = rev(X)$, as desired.

But how about the rev function?

Finding Reverse's Loop Invariant (II)

$$< \textbf{while}\ (!empty(x))\ \{y =_l first(x)\ \$:\ y\ ;\ x =_l rest(x)\ ;\} \rightsquigarrow K$$
$$\mid TS\ \&\ x \mapsto X\ *\ y \mapsto Y\ *\ z \mapsto Z * VS > \mid Z = rev(Y)\$X\ \rightarrow^{\circledast}$$
$$< K \mid TS\ \&\ x \mapsto X'\ *\ y \mapsto Y'\ *\ z \mapsto Z' * VS > \mid Z = rev(Y')\$X'$$

But does this loop invariant capture the properties we had in mind? Yes. Because:

1. Since $(!empty(x))$ is the loop's guard, we know that $X' = nil$, and therefore $rev(Z) = rev(X')\$Y' = Y' = rev(X)\$Y$.
2. When, furthermore, $Y = nil$, we get $Y' = rev(X)$, as desired.

But how about the rev function? This is part of the property specification, we have to define it in a funcional module

Finding Reverse's Loop Invariant (II)

$$< \textbf{while}\ (!empty(x))\ \{y =_l first(x)\ \$:\ y\ ;\ x =_l rest(x)\ ;\} \rightsquigarrow K \mid TS\ \&\ x \mapsto X\ *\ y \mapsto Y\ *\ z \mapsto Z * VS > \mid Z = rev(Y)\$X\ \rightarrow^{\circledast}$$
$$< K \mid TS\ \&\ x \mapsto X'\ *\ y \mapsto Y'\ *\ z \mapsto Z' * VS > \mid Z = rev(Y')\$X'$$

But does this loop invariant capture the properties we had in mind? Yes. Because:

1. Since $(!empty(x))$ is the loop's guard, we know that $X' = nil$, and therefore $rev(Z) = rev(X')\$Y' = Y' = rev(X)\$Y$.
2. When, furthermore, $Y = nil$, we get $Y' = rev(X)$, as desired.

But how about the rev function? This is part of the property specification, we have to define it in a funcional module protecting the data type module `IMP-LIST`.

Finding Reverse's Loop Invariant (II)

$$< \textbf{while}\ (!empty(x))\ \{y =_l first(x)\ \$:\ y\ ;\ x =_l rest(x)\ ;\} \rightsquigarrow K \mid TS\ \&\ x \mapsto X\ *\ y \mapsto Y\ *\ z \mapsto Z * VS > \mid Z = rev(Y)\$X\ \rightarrow^{\circledast}$$
$$< K \mid TS\ \&\ x \mapsto X'\ *\ y \mapsto Y'\ *\ z \mapsto Z' * VS > \mid Z = rev(Y')\$X'$$

But does this loop invariant capture the properties we had in mind? Yes. Because:

1. Since $(!empty(x))$ is the loop's guard, we know that $X' = nil$, and therefore $rev(Z) = rev(X')\$Y' = Y' = rev(X)\$Y$.
2. When, furthermore, $Y = nil$, we get $Y' = rev(X)$, as desired.

But how about the rev function? This is part of the property specification, we have to define it in a funcional module protecting the data type module `IMP-LIST`. For the `REV` module and the proof script see (CSPV.IMPL).

Acknowledgements

Acknowledgements

The program proving methodology presented in this lecture has been developed in joint work with Michael Abir.

Acknowledgements

The program proving methodology presented in this lecture has been developed in joint work with Michael Abir.

Full details about the methodology and these examples can be found in:

Acknowledgements

The program proving methodology presented in this lecture has been developed in joint work with Michael Abir.

Full details about the methodology and these examples can be found in:

M. Abir and J. Meseguer, *Continuation Semantics and Program Verification for the IMPL Language*

Acknowledgements

The program proving methodology presented in this lecture has been developed in joint work with Michael Abir.

Full details about the methodology and these examples can be found in:

M. Abir and J. Meseguer, *Continuation Semantics and Program Verification for the IMPL Language*

Acknowledgements

The program proving methodology presented in this lecture has been developed in joint work with Michael Abir.

Full details about the methodology and these examples can be found in:

M. Abir and J. Meseguer, *Continuation Semantics and Program Verification for the IMPL Language*

Available in the course's web page.