Program Verification: Lecture 24

José Meseguer

University of Illinois
at Urbana-Champaign

Proving Properties of IMPL Programs

How can we prove properties about some IMPL program P of interests?

Proving Properties of IMPL Programs

How can we prove properties about some IMPL program P of interests? We should first specify such properties as reachability formulas.

Proving Properties of IMPL Programs

How can we prove properties about some IMPL program P of interests? We should first specify such properties as reachability formulas. We saw in Lecture 23 that reachability formulas for an IMPL program P have the general form:

Proving Properties of IMPL Programs

How can we prove properties about some IMPL program P of interests? We should first specify such properties as reachability formulas. We saw in Lecture 23 that reachability formulas for an IMPL program P have the general form:

$$< P \leadsto K \mid TS \,\&\, \vec{x} \mapsto \vec{X} * VS >\mid \varphi \rightarrow^{\circledast} < K \mid TS \,\&\, \vec{x} \mapsto \vec{X}' * VS >\mid \psi$$

Proving Properties of IMPL Programs

How can we prove properties about some IMPL program P of interests? We should first specify such properties as reachability formulas. We saw in Lecture 23 that reachability formulas for an IMPL program P have the general form:

$$< P \rightsquigarrow K \mid TS \;\&\; \vec{x} \mapsto \vec{X} * VS >\mid \varphi \to^{\circledast} < K \mid TS \;\&\; \vec{x} \mapsto \vec{X'} * VS >\mid \psi$$

where $\vec{x} = x_1, \ldots, x_n$ are the program variables in P,

Proving Properties of IMPL Programs

How can we prove properties about some IMPL program P of interests? We should first specify such properties as reachability formulas. We saw in Lecture 23 that reachability formulas for an IMPL program P have the general form:

$$< P \rightsquigarrow K \mid TS \;\&\; \vec{x} \mapsto \vec{X} * VS >\mid \varphi \rightarrow^{\circledast} < K \mid TS \;\&\; \vec{x} \mapsto \vec{X'} * VS >\mid \psi$$

where $\vec{x} = x_1, \ldots, x_n$ are the program variables in P, and $\vec{x} \mapsto \vec{X}$ (similar for $\vec{x} \mapsto \vec{X'}$) abbreviates the $VStore$ fragment: $x_1 \mapsto X_1 \; * \; \ldots \; * \; x_n \mapsto X_n$.

Proving Properties of IMPL Programs

How can we prove properties about some IMPL program P of interests? We should first specify such properties as reachability formulas. We saw in Lecture 23 that reachability formulas for an IMPL program P have the general form:

$$< P \leadsto K \mid TS \,\&\, \vec{x} \mapsto \vec{X} * VS >\mid \varphi \rightarrow^{\circledast} < K \mid TS \,\&\, \vec{x} \mapsto \vec{X'} * VS >\mid \psi$$

where $\vec{x} = x_1, \ldots, x_n$ are the program variables in P, and $\vec{x} \mapsto \vec{X}$ (similar for $\vec{x} \mapsto \vec{X'}$) abbreviates the $VStore$ fragment: $x_1 \mapsto X_1 \; * \; \ldots \; * \; x_n \mapsto X_n$. We should always ensure that the variables $\vec{X'}$ are fresh, i.e., that $\vec{X} \cap \vec{X'} = \emptyset$.

Proving Properties of IMPL Programs

How can we prove properties about some IMPL program P of interests? We should first specify such properties as reachability formulas. We saw in Lecture 23 that reachability formulas for an IMPL program P have the general form:

$$< P \rightsquigarrow K \mid TS \;\&\; \vec{x} \mapsto \vec{X} * VS >\mid \varphi \rightarrow^{\circledast} < K \mid TS \;\&\; \vec{x} \mapsto \vec{X}' * VS >\mid \psi$$

where $\vec{x} = x_1, \ldots, x_n$ are the program variables in P, and $\vec{x} \mapsto \vec{X}$ (similar for $\vec{x} \mapsto \vec{X}'$) abbreviates the $VStore$ fragment: $x_1 \mapsto X_1 \,*\, \ldots \,*\, x_n \mapsto X_n$. We should always ensure that the variables $\vec{X}'$ are fresh, i.e., that $\vec{X} \cap \vec{X}' = \emptyset$. This is because any semantic relationships between the values of $\vec{X}$ and $\vec{X}'$ can be specified in the midcondition's data constraint ψ.

Proving Properties of IMPL Programs

How can we prove properties about some IMPL program P of interests? We should first specify such properties as reachability formulas. We saw in Lecture 23 that reachability formulas for an IMPL program P have the general form:

$$< P \rightsquigarrow K \mid TS \;\&\; \vec{x} \mapsto \vec{X} * VS >\mid \varphi \rightarrow^{\circledast} < K \mid TS \;\&\; \vec{x} \mapsto \vec{X'} * VS >\mid \psi$$

where $\vec{x} = x_1, \ldots, x_n$ are the program variables in P, and $\vec{x} \mapsto \vec{X}$ (similar for $\vec{x} \mapsto \vec{X'}$) abbreviates the $VStore$ fragment: $x_1 \mapsto X_1 \; * \; \ldots \; * \; x_n \mapsto X_n$. We should always ensure that the variables $\vec{X'}$ are fresh, i.e., that $\vec{X} \cap \vec{X'} = \emptyset$. This is because any semantic relationships between the values of $\vec{X}$ and $\vec{X'}$ can be specified in the midcondition's data constraint ψ. For example, if in reality $X'_2 = X_2$, this will be one of the conjuncts in ψ.

Proving Properties of IMPL Programs

How can we prove properties about some IMPL program P of interests? We should first specify such properties as reachability formulas. We saw in Lecture 23 that reachability formulas for an IMPL program P have the general form:

$$< P \rightsquigarrow K \mid TS \,\&\, \vec{x} \mapsto \vec{X} * VS >\mid \varphi \rightarrow^{\circledast} < K \mid TS \,\&\, \vec{x} \mapsto \vec{X'} * VS >\mid \psi$$

where $\vec{x} = x_1, \ldots, x_n$ are the program variables in P, and $\vec{x} \mapsto \vec{X}$ (similar for $\vec{x} \mapsto \vec{X'}$) abbreviates the $VStore$ fragment: $x_1 \mapsto X_1 \ * \ \ldots \ * \ x_n \mapsto X_n$. We should always ensure that the variables $\vec{X'}$ are fresh, i.e., that $\vec{X} \cap \vec{X'} = \emptyset$. This is because any semantic relationships between the values of $\vec{X}$ and $\vec{X'}$ can be specified in the midcondition's data constraint ψ. For example, if in reality $X'_2 = X_2$, this will be one of the conjuncts in ψ.

Is that all?

Proving Properties of IMPL Programs

How can we prove properties about some IMPL program P of interests? We should first specify such properties as reachability formulas. We saw in Lecture 23 that reachability formulas for an IMPL program P have the general form:

$$< P \rightsquigarrow K \mid TS \,\&\, \vec{x} \mapsto \vec{X} * VS >\mid \varphi \rightarrow^{\circledast} < K \mid TS \,\&\, \vec{x} \mapsto \vec{X}' * VS >\mid \psi$$

where $\vec{x} = x_1, \ldots, x_n$ are the program variables in P, and $\vec{x} \mapsto \vec{X}$ (similar for $\vec{x} \mapsto \vec{X}'$) abbreviates the $VStore$ fragment: $x_1 \mapsto X_1 \; * \; \ldots \; * \; x_n \mapsto X_n$. We should always ensure that the variables $\vec{X}'$ are fresh, i.e., that $\vec{X} \cap \vec{X}' = \emptyset$. This is because any semantic relationships between the values of $\vec{X}$ and $\vec{X}'$ can be specified in the midcondition's data constraint ψ. For example, if in reality $X'_2 = X_2$, this will be one of the conjuncts in ψ.

Is that all? Yes and No.

Proving Properties of IMPL Programs (II)

Yes: In principle, all we need to do specifying some reachability property of an IMPL program P

Proving Properties of IMPL Programs (II)

Yes: In principle, all we need to do specifying some reachability property of an IMPL program P is to follow the general format described in the previous slide, yet

Proving Properties of IMPL Programs (II)

Yes: In principle, all we need to do specifying some reachability property of an IMPL program P is to follow the general format described in the previous slide, yet

No: We will not be very successful in getting the IMPL Prover to help us prove our reachability formulas unless

Proving Properties of IMPL Programs (II)

Yes: In principle, all we need to do specifying some reachability property of an IMPL program P is to follow the general format described in the previous slide, yet

No: We will not be very successful in getting the IMPL Prover to help us prove our reachability formulas unless we have thought carefully about how to specify our desired property

Proving Properties of IMPL Programs (II)

Yes: In principle, all we need to do specifying some reachability property of an IMPL program P is to follow the general format described in the previous slide, yet

No: We will not be very successful in getting the IMPL Prover to help us prove our reachability formulas unless we have thought carefully about how to specify our desired property in a way that will make it easy for the prover to prove it, either automatically or semi-automatically.

Proving Properties of IMPL Programs (II)

Yes: In principle, all we need to do specifying some reachability property of an IMPL program P is to follow the general format described in the previous slide, yet

No: We will not be very successful in getting the IMPL Prover to help us prove our reachability formulas unless we have thought carefully about how to specify our desired property in a way that will make it easy for the prover to prove it, either automatically or semi-automatically.

To arrive at good formulations of IMPL program properties we need two draw from two sources of knowlege:

Proving Properties of IMPL Programs (II)

Yes: In principle, all we need to do specifying some reachability property of an IMPL program P is to follow the general format described in the previous slide, yet

No: We will not be very successful in getting the IMPL Prover to help us prove our reachability formulas unless we have thought carefully about how to specify our desired property in a way that will make it easy for the prover to prove it, either automatically or semi-automatically.

To arrive at good formulations of IMPL program properties we need two draw from two sources of knowlege:

1. **Generic** knowledge about reachability logic, and

Proving Properties of IMPL Programs (II)

Yes: In principle, all we need to do specifying some reachability property of an IMPL program P is to follow the general format described in the previous slide, yet

No: We will not be very successful in getting the IMPL Prover to help us prove our reachability formulas unless we have thought carefully about how to specify our desired property in a way that will make it easy for the prover to prove it, either automatically or semi-automatically.

To arrive at good formulations of IMPL program properties we need two draw from two sources of knowlege:

1. **Generic** knowledge about reachability logic, and
2. **Domain-Specific** knowledge about IMPL semantics.

Proving Properties of IMPL Programs (II)

Yes: In principle, all we need to do specifying some reachability property of an IMPL program P is to follow the general format described in the previous slide, yet

No: We will not be very successful in getting the IMPL Prover to help us prove our reachability formulas unless we have thought carefully about how to specify our desired property in a way that will make it easy for the prover to prove it, either automatically or semi-automatically.

To arrive at good formulations of IMPL program properties we need two draw from two sources of knowlege:

1. **Generic** knowledge about reachability logic, and
2. **Domain-Specific** knowledge about IMPL semantics.

We can condense knowledge from sources (1)-(2) into

Proving Properties of IMPL Programs (II)

Yes: In principle, all we need to do specifying some reachability property of an IMPL program P is to follow the general format described in the previous slide, yet

No: We will not be very successful in getting the IMPL Prover to help us prove our reachability formulas unless we have thought carefully about how to specify our desired property in a way that will make it easy for the prover to prove it, either automatically or semi-automatically.

To arrive at good formulations of IMPL program properties we need two draw from two sources of knowlege:

1. **Generic** knowledge about reachability logic, and
2. **Domain-Specific** knowledge about IMPL semantics.

We can condense knowledge from sources (1)-(2) into General Proof Methods

Proving Properties of IMPL Programs (II)

Yes: In principle, all we need to do specifying some reachability property of an IMPL program P is to follow the general format described in the previous slide, yet

No: We will not be very successful in getting the IMPL Prover to help us prove our reachability formulas unless we have thought carefully about how to specify our desired property in a way that will make it easy for the prover to prove it, either automatically or semi-automatically.

To arrive at good formulations of IMPL program properties we need two draw from two sources of knowlege:

1. **Generic** knowledge about reachability logic, and
2. **Domain-Specific** knowledge about IMPL semantics.

We can condense knowledge from sources (1)-(2) into General Proof Methods that will be effective in proving IMPL programs.

Generic Properties of Reachability Formulas

Several generic properties about reachability formulas valid in any rewrite theory $\mathcal{R}$ are always very useful:

1. **Constructor Instantiation of a Parameter**. Let $A \rightarrow^{\circledast}_{Y} B$ be a valid reachability formula for a rewrite theory $\mathcal{R}$, where we use the arrow $\rightarrow^{\circledast}_{Y}$ to indicate that the formula is parametric on a set of variables Y.

Generic Properties of Reachability Formulas

Several generic properties about reachability formulas valid in any rewrite theory $\mathcal{R}$ are always very useful:

1. **Constructor Instantiation of a Parameter**. Let $A \rightarrow^{\circledast}_{Y} B$ be a valid reachability formula for a rewrite theory $\mathcal{R}$, where we use the arrow $\rightarrow^{\circledast}_{Y}$ to indicate that the formula is parametric on a set of variables Y. Then it follows directly from the semantics of parametric reachability formulas that if $y \in Y$ and σ is a constructor substitution $\{y \mapsto u\}$ such that $Z = vars(u)$ are fresh variables not appearing in $A \rightarrow^{\circledast}_{Y} B$, then $A\sigma \rightarrow^{\circledast}_{Y \setminus \{y\} \cup Z} B\sigma$ is also a valid reachability formula for $\mathcal{R}$.

Generic Properties of Reachability Formulas

Several generic properties about reachability formulas valid in any rewrite theory $\mathcal{R}$ are always very useful:

1. **Constructor Instantiation of a Parameter**. Let $A \rightarrow^{\circledast}_{Y} B$ be a valid reachability formula for a rewrite theory $\mathcal{R}$, where we use the arrow $\rightarrow^{\circledast}_{Y}$ to indicate that the formula is parametric on a set of variables Y. Then it follows directly from the semantics of parametric reachability formulas that if $y \in Y$ and σ is a constructor substitution $\{y \mapsto u\}$ such that $Z = vars(u)$ are fresh variables not appearing in $A \rightarrow^{\circledast}_{Y} B$, then $A\sigma \rightarrow^{\circledast}_{Y \setminus \{y\} \cup Z} B\sigma$ is also a valid reachability formula for $\mathcal{R}$.

That is, the following is a valid derived rule of Reachability Logic:

Generic Properties of Reachability Formulas

Several generic properties about reachability formulas valid in any rewrite theory $\mathcal{R}$ are always very useful:

1. **Constructor Instantiation of a Parameter**. Let $A \rightarrow^{\circledast}_{Y} B$ be a valid reachability formula for a rewrite theory $\mathcal{R}$, where we use the arrow $\rightarrow^{\circledast}_{Y}$ to indicate that the formula is parametric on a set of variables Y. Then it follows directly from the semantics of parametric reachability formulas that if $y \in Y$ and σ is a constructor substitution $\{y \mapsto u\}$ such that $Z = vars(u)$ are fresh variables not appearing in $A \rightarrow^{\circledast}_{Y} B$, then $A\sigma \rightarrow^{\circledast}_{Y \setminus \{y\} \cup Z} B\sigma$ is also a valid reachability formula for $\mathcal{R}$.

That is, the following is a valid derived rule of Reachability Logic:

$$\frac{A \rightarrow^{\circledast}_{Y} B}{A\sigma \rightarrow^{\circledast}_{Y \setminus \{y\} \cup Z} B\sigma}$$

where $y \in Y$ and σ is a constructor substitution $\{y \mapsto u\}$ such that the variables $Z = vars(u)$ are fresh.

Generic Properties of Reachability Formulas (II)

2. **Chain Rule**. Let $A \to_Y^{\circledast} B$ and $B \to_Y^{\circledast} C$, both parametric on variables Y (the unparametric case is the instance $Y = \emptyset$),

Generic Properties of Reachability Formulas (II)

2. **Chain Rule**. Let $A \rightarrow^{\circledast}_{Y} B$ and $B \rightarrow^{\circledast}_{Y} C$, both parametric on variables Y (the unparametric case is the instance $Y = \emptyset$), be two valid formulas about a given rewrite theory $\mathcal{R}$,

Generic Properties of Reachability Formulas (II)

2. **Chain Rule**. Let $A \to_{Y}^{\circledast} B$ and $B \to_{Y}^{\circledast} C$, both parametric on variables Y (the unparametric case is the instance $Y = \emptyset$), be two valid formulas about a given rewrite theory $\mathcal{R}$, Then, it follows from the semantic definition of parametric reachability formulas that

Generic Properties of Reachability Formulas (II)

2. **Chain Rule**. Let $A \rightarrow^{\circledast}_{Y} B$ and $B \rightarrow^{\circledast}_{Y} C$, both parametric on variables Y (the unparametric case is the instance $Y = \emptyset$), be two valid formulas about a given rewrite theory $\mathcal{R}$, Then, it follows from the semantic definition of parametric reachability formulas that $A \rightarrow^{\circledast}_{Y} C$ is also valid for $\mathcal{R}$.

Generic Properties of Reachability Formulas (II)

2. **Chain Rule**. Let $A \to_Y^{\circledast} B$ and $B \to_Y^{\circledast} C$, both parametric on variables Y (the unparametric case is the instance $Y = \emptyset$), be two valid formulas about a given rewrite theory $\mathcal{R}$, Then, it follows from the semantic definition of parametric reachability formulas that $A \to_Y^{\circledast} C$ is also valid for $\mathcal{R}$.

That is, the following is a valid derived rule of Reachability Logic:

Generic Properties of Reachability Formulas (II)

2. **Chain Rule**. Let $A \to_Y^{\circledast} B$ and $B \to_Y^{\circledast} C$, both parametric on variables Y (the unparametric case is the instance $Y = \emptyset$), be two valid formulas about a given rewrite theory $\mathcal{R}$, Then, it follows from the semantic definition of parametric reachability formulas that $A \to_Y^{\circledast} C$ is also valid for $\mathcal{R}$.

That is, the following is a valid derived rule of Reachability Logic:

$$\frac{A \to_Y^{\circledast} B \qquad B \to_Y^{\circledast} C}{A \to_Y^{\circledast} C}$$

Interlude on Pattern Subsumptions

A pattern formula A is a state predicate defining a set of states $[\![A]\!]$.

Interlude on Pattern Subsumptions

A pattern formula A is a state predicate defining a set of states $[\![A]\!]$. In many cases, given pattern formulas A and B, we would like to check whether $[\![A]\!] \subseteq [\![B]\!]$.

Interlude on Pattern Subsumptions

A pattern formula A is a state predicate defining a set of states $[\![A]\!]$. In many cases, given pattern formulas A and B, we would like to check whether $[\![A]\!] \subseteq [\![B]\!]$. In general this is undecidable.

Interlude on Pattern Subsumptions

A pattern formula A is a state predicate defining a set of states $[\![A]\!]$. In many cases, given pattern formulas A and B, we would like to check whether $[\![A]\!] \subseteq [\![B]\!]$. In general this is undecidable. But is there a symbolic method that can often check $[\![A]\!] \subseteq [\![B]\!]$?

Interlude on Pattern Subsumptions

A pattern formula A is a state predicate defining a set of states $[\![A]\!]$. In many cases, given pattern formulas A and B, we would like to check whether $[\![A]\!] \subseteq [\![B]\!]$. In general this is undecidable. But is there a symbolic method that can often check $[\![A]\!] \subseteq [\![B]\!]$?

Definition. Let $u \mid \varphi$ and $v \mid \psi$ be constructor patterns for a theory $\mathcal{R}$ sharing no variables (resp. sharing variables Y).

Interlude on Pattern Subsumptions

A pattern formula A is a state predicate defining a set of states $[\![A]\!]$. In many cases, given pattern formulas A and B, we would like to check whether $[\![A]\!] \subseteq [\![B]\!]$. In general this is undecidable. But is there a symbolic method that can often check $[\![A]\!] \subseteq [\![B]\!]$?

Definition. Let $u \mid \varphi$ and $v \mid \psi$ be constructor patterns for a theory $\mathcal{R}$ sharing no variables (resp. sharing variables Y). We say that $u \mid \varphi$ is subsumed (resp. Y-subsumed) by $v \mid \psi$, denoted $u \mid \varphi \sqsubseteq v \mid \psi$ (resp. $u \mid \varphi \sqsubseteq_Y v \mid \psi$) iff (by definition)

Interlude on Pattern Subsumptions

A pattern formula A is a state predicate defining a set of states $[\![A]\!]$. In many cases, given pattern formulas A and B, we would like to check whether $[\![A]\!] \subseteq [\![B]\!]$. In general this is undecidable. But is there a symbolic method that can often check $[\![A]\!] \subseteq [\![B]\!]$?

Definition. Let $u \mid \varphi$ and $v \mid \psi$ be constructor patterns for a theory $\mathcal{R}$ sharing no variables (resp. sharing variables Y). We say that $u \mid \varphi$ is subsumed (resp. Y-subsumed) by $v \mid \psi$, denoted $u \mid \varphi \sqsubseteq v \mid \psi$ (resp. $u \mid \varphi \sqsubseteq_Y v \mid \psi$) iff (by definition) there exists a substitution α (resp. α s.t. $(\forall y \in Y)\ \alpha(y) = y$) such that:

Interlude on Pattern Subsumptions

A pattern formula A is a state predicate defining a set of states $[\![A]\!]$. In many cases, given pattern formulas A and B, we would like to check whether $[\![A]\!] \subseteq [\![B]\!]$. In general this is undecidable. But is there a symbolic method that can often check $[\![A]\!] \subseteq [\![B]\!]$?

Definition. Let $u \mid \varphi$ and $v \mid \psi$ be constructor patterns for a theory $\mathcal{R}$ sharing no variables (resp. sharing variables Y). We say that $u \mid \varphi$ is subsumed (resp. Y-subsumed) by $v \mid \psi$, denoted $u \mid \varphi \sqsubseteq v \mid \psi$ (resp. $u \mid \varphi \sqsubseteq_Y v \mid \psi$) iff (by definition) there exists a substitution α (resp. α s.t. $(\forall y \in Y)\ \alpha(y) = y$) such that: (i) $u =_B v\alpha$, and

Interlude on Pattern Subsumptions

A pattern formula A is a state predicate defining a set of states $[\![A]\!]$. In many cases, given pattern formulas A and B, we would like to check whether $[\![A]\!] \subseteq [\![B]\!]$. In general this is undecidable. But is there a symbolic method that can often check $[\![A]\!] \subseteq [\![B]\!]$?

Definition. Let $u \mid \varphi$ and $v \mid \psi$ be constructor patterns for a theory $\mathcal{R}$ sharing no variables (resp. sharing variables Y). We say that $u \mid \varphi$ is subsumed (resp. Y-subsumed) by $v \mid \psi$, denoted $u \mid \varphi \sqsubseteq v \mid \psi$ (resp. $u \mid \varphi \sqsubseteq_Y v \mid \psi$) iff (by definition) there exists a substitution α (resp. α s.t. $(\forall y \in Y)\ \alpha(y) = y$) such that: (i) $u =_B v\alpha$, and (ii) $\mathcal{T}_{\Sigma/E \cup B} \models \varphi \Rightarrow (\psi\alpha)$,

Interlude on Pattern Subsumptions

A pattern formula A is a state predicate defining a set of states $[\![A]\!]$. In many cases, given pattern formulas A and B, we would like to check whether $[\![A]\!] \subseteq [\![B]\!]$. In general this is undecidable. But is there a symbolic method that can often check $[\![A]\!] \subseteq [\![B]\!]$?

Definition. Let $u \mid \varphi$ and $v \mid \psi$ be constructor patterns for a theory $\mathcal{R}$ sharing no variables (resp. sharing variables Y). We say that $u \mid \varphi$ is subsumed (resp. Y-subsumed) by $v \mid \psi$, denoted $u \mid \varphi \sqsubseteq v \mid \psi$ (resp. $u \mid \varphi \sqsubseteq_Y v \mid \psi$) iff (by definition) there exists a substitution α (resp. α s.t. $(\forall y \in Y)\ \alpha(y) = y$) such that: (i) $u =_B v\alpha$, and (ii) $\mathcal{T}_{\Sigma/E\cup B} \models \varphi \Rightarrow (\psi\alpha)$, where $\mathcal{T}_{\Sigma/E\cup B}$ is the initial algebra of the equational part of $\mathcal{R}$.

Interlude on Pattern Subsumptions

A pattern formula A is a state predicate defining a set of states $[\![A]\!]$. In many cases, given pattern formulas A and B, we would like to check whether $[\![A]\!] \subseteq [\![B]\!]$. In general this is undecidable. But is there a symbolic method that can often check $[\![A]\!] \subseteq [\![B]\!]$?

Definition. Let $u \mid \varphi$ and $v \mid \psi$ be constructor patterns for a theory $\mathcal{R}$ sharing no variables (resp. sharing variables Y). We say that $u \mid \varphi$ is subsumed (resp. Y-subsumed) by $v \mid \psi$, denoted $u \mid \varphi \sqsubseteq v \mid \psi$ (resp. $u \mid \varphi \sqsubseteq_Y v \mid \psi$) iff (by definition) there exists a substitution α (resp. α s.t. $(\forall y \in Y)\ \alpha(y) = y$) such that: (i) $u =_B v\alpha$, and (ii) $\mathcal{T}_{\Sigma/E \cup B} \models \varphi \Rightarrow (\psi\alpha)$, where $\mathcal{T}_{\Sigma/E \cup B}$ is the initial algebra of the equational part of $\mathcal{R}$.

Theorem. If $u \mid \varphi \sqsubseteq v \mid \psi$, then

Interlude on Pattern Subsumptions

A pattern formula A is a state predicate defining a set of states $[\![A]\!]$. In many cases, given pattern formulas A and B, we would like to check whether $[\![A]\!] \subseteq [\![B]\!]$. In general this is undecidable. But is there a symbolic method that can often check $[\![A]\!] \subseteq [\![B]\!]$?

Definition. Let $u \mid \varphi$ and $v \mid \psi$ be constructor patterns for a theory $\mathcal{R}$ sharing no variables (resp. sharing variables Y). We say that $u \mid \varphi$ is subsumed (resp. Y-subsumed) by $v \mid \psi$, denoted $u \mid \varphi \sqsubseteq v \mid \psi$ (resp. $u \mid \varphi \sqsubseteq_Y v \mid \psi$) iff (by definition) there exists a substitution α (resp. α s.t. $(\forall y \in Y)\ \alpha(y) = y$) such that: (i) $u =_B v\alpha$, and (ii) $\mathcal{T}_{\Sigma/E \cup B} \models \varphi \Rightarrow (\psi\alpha)$, where $\mathcal{T}_{\Sigma/E \cup B}$ is the initial algebra of the equational part of $\mathcal{R}$.

Theorem. If $u \mid \varphi \sqsubseteq v \mid \psi$, then $[\![u \mid \varphi]\!] \subseteq [\![v \mid \psi]\!]$.

Interlude on Pattern Subsumptions

A pattern formula A is a state predicate defining a set of states $[\![A]\!]$. In many cases, given pattern formulas A and B, we would like to check whether $[\![A]\!] \subseteq [\![B]\!]$. In general this is undecidable. But is there a symbolic method that can often check $[\![A]\!] \subseteq [\![B]\!]$?

Definition. Let $u \mid \varphi$ and $v \mid \psi$ be constructor patterns for a theory $\mathcal{R}$ sharing no variables (resp. sharing variables Y). We say that $u \mid \varphi$ is subsumed (resp. Y-subsumed) by $v \mid \psi$, denoted $u \mid \varphi \sqsubseteq v \mid \psi$ (resp. $u \mid \varphi \sqsubseteq_Y v \mid \psi$) iff (by definition) there exists a substitution α (resp. α s.t. $(\forall y \in Y)\ \alpha(y) = y$) such that: (i) $u =_B v\alpha$, and (ii) $\mathcal{T}_{\Sigma/E \cup B} \models \varphi \Rightarrow (\psi\alpha)$, where $\mathcal{T}_{\Sigma/E \cup B}$ is the initial algebra of the equational part of $\mathcal{R}$.

Theorem. If $u \mid \varphi \sqsubseteq v \mid \psi$, then $[\![u \mid \varphi]\!] \subseteq [\![v \mid \psi]\!]$. If $u \mid \varphi \sqsubseteq_Y v \mid \psi$, then for each ground constructor substitution $\rho \in [Y \to T_\Omega]$

Interlude on Pattern Subsumptions

A pattern formula A is a state predicate defining a set of states $[\![A]\!]$. In many cases, given pattern formulas A and B, we would like to check whether $[\![A]\!] \subseteq [\![B]\!]$. In general this is undecidable. But is there a symbolic method that can often check $[\![A]\!] \subseteq [\![B]\!]$?

Definition. Let $u \mid \varphi$ and $v \mid \psi$ be constructor patterns for a theory $\mathcal{R}$ sharing no variables (resp. sharing variables Y). We say that $u \mid \varphi$ is subsumed (resp. Y-subsumed) by $v \mid \psi$, denoted $u \mid \varphi \sqsubseteq v \mid \psi$ (resp. $u \mid \varphi \sqsubseteq_Y v \mid \psi$) iff (by definition) there exists a substitution α (resp. α s.t. $(\forall y \in Y)\ \alpha(y) = y$) such that: (i) $u =_B v\alpha$, and (ii) $\mathcal{T}_{\Sigma/E \cup B} \models \varphi \Rightarrow (\psi\alpha)$, where $\mathcal{T}_{\Sigma/E \cup B}$ is the initial algebra of the equational part of $\mathcal{R}$.

Theorem. If $u \mid \varphi \sqsubseteq v \mid \psi$, then $[\![u \mid \varphi]\!] \subseteq [\![v \mid \psi]\!]$. If $u \mid \varphi \sqsubseteq_Y v \mid \psi$, then for each ground constructor substitution $\rho \in [Y \to T_\Omega]$ $[\![(u \mid \varphi)\rho]\!] \subseteq [\![(v \mid \psi)\rho]\!]$.

Generic Properties of Reachability Formulas (III)

2. **Expanding Preconditions and Restricting Midconditions**.
Let $A \to^{\circledast}_{Y} D$ and $B \to^{\circledast}_{Y} C$, be both parametric on variables Y for a rewrite theory $\mathcal{R}$. (the unparametric case is the instance $Y = \emptyset$),

Generic Properties of Reachability Formulas (III)

2. **Expanding Preconditions and Restricting Midconditions**. Let $A \rightarrow^{\circledast}_{Y} D$ and $B \rightarrow^{\circledast}_{Y} C$, be both parametric on variables Y for a rewrite theory $\mathcal{R}$. (the unparametric case is the instance $Y = \emptyset$), Then if:

Generic Properties of Reachability Formulas (III)

2. **Expanding Preconditions and Restricting Midconditions**.
Let $A \rightarrow^{\circledast}_{Y} D$ and $B \rightarrow^{\circledast}_{Y} C$, be both parametric on variables Y for a rewrite theory $\mathcal{R}$. (the unparametric case is the instance $Y = \emptyset$),
Then if: (1) $B \rightarrow^{\circledast}_{Y} C$ is valid in $\mathcal{R}$, and

Generic Properties of Reachability Formulas (III)

2. **Expanding Preconditions and Restricting Midconditions**.
Let $A \to^{\circledast}_{Y} D$ and $B \to^{\circledast}_{Y} C$, be both parametric on variables Y for a rewrite theory $\mathcal{R}$. (the unparametric case is the instance $Y = \emptyset$), Then if: (1) $B \to^{\circledast}_{Y} C$ is valid in $\mathcal{R}$, and (2) $A \sqsubseteq_Y B$ and $C \sqsubseteq_Y D$,

Generic Properties of Reachability Formulas (III)

2. **Expanding Preconditions and Restricting Midconditions**. Let $A \to^{\circledast}_{Y} D$ and $B \to^{\circledast}_{Y} C$, be both parametric on variables Y for a rewrite theory $\mathcal{R}$. (the unparametric case is the instance $Y = \emptyset$), Then if: (1) $B \to^{\circledast}_{Y} C$ is valid in $\mathcal{R}$, and (2) $A \sqsubseteq_{Y} B$ and $C \sqsubseteq_{Y} D$, it follows from the semantics of parametric reachability formulas that

Generic Properties of Reachability Formulas (III)

2. **Expanding Preconditions and Restricting Midconditions**.
Let $A \to_Y^{\circledast} D$ and $B \to_Y^{\circledast} C$, be both parametric on variables Y for a rewrite theory $\mathcal{R}$. (the unparametric case is the instance $Y = \emptyset$), Then if: (1) $B \to_Y^{\circledast} C$ is valid in $\mathcal{R}$, and (2) $A \sqsubseteq_Y B$ and $C \sqsubseteq_Y D$, it follows from the semantics of parametric reachability formulas that $A \to_Y^{\circledast} D$ is valid in $\mathcal{R}$.

Generic Properties of Reachability Formulas (III)

2. **Expanding Preconditions and Restricting Midconditions**. Let $A \rightarrow^{\circledast}_{Y} D$ and $B \rightarrow^{\circledast}_{Y} C$, be both parametric on variables Y for a rewrite theory $\mathcal{R}$. (the unparametric case is the instance $Y = \emptyset$), Then if: (1) $B \rightarrow^{\circledast}_{Y} C$ is valid in $\mathcal{R}$, and (2) $A \sqsubseteq_{Y} B$ and $C \sqsubseteq_{Y} D$, it follows from the semantics of parametric reachability formulas that $A \rightarrow^{\circledast}_{Y} D$ is valid in $\mathcal{R}$.

That is, the following is a valid derived rule of Reachability Logic:

Generic Properties of Reachability Formulas (III)

2. **Expanding Preconditions and Restricting Midconditions**. Let $A \rightarrow^{\circledast}_{Y} D$ and $B \rightarrow^{\circledast}_{Y} C$, be both parametric on variables Y for a rewrite theory $\mathcal{R}$. (the unparametric case is the instance $Y = \emptyset$), Then if: (1) $B \rightarrow^{\circledast}_{Y} C$ is valid in $\mathcal{R}$, and (2) $A \sqsubseteq_{Y} B$ and $C \sqsubseteq_{Y} D$, it follows from the semantics of parametric reachability formulas that $A \rightarrow^{\circledast}_{Y} D$ is valid in $\mathcal{R}$.

That is, the following is a valid derived rule of Reachability Logic:

$$\frac{A \sqsubseteq_{Y} B \qquad B \rightarrow^{\circledast}_{Y} C \qquad C \sqsubseteq_{Y} D}{A \rightarrow^{\circledast}_{Y} D}$$

Generic Properties of Reachability Formulas (IV)

The **Split** auxiliary inference rule of Reachability logic ensures that the validity of a reachability formula

$$u \mid \varphi \to^{\circledast} B$$

in a rewrite theory $\mathcal{R}$ is equivalent to the validity of the two reachability formulas:

$$u \mid \varphi \wedge \psi_1 \to^{\circledast} B \qquad \textit{and} \qquad u \mid \varphi \wedge \psi_2 \to^{\circledast} B$$

provided: (1) ψ_1 and ψ_2 do not have any extra variables besides those of $u \mid \varphi$, and

Generic Properties of Reachability Formulas (IV)

The **Split** auxiliary inference rule of Reachability logic ensures that the validity of a reachability formula

$$u \mid \varphi \to^{\circledast} B$$

in a rewrite theory $\mathcal{R}$ is equivalent to the validity of the two reachability formulas:

$$u \mid \varphi \wedge \psi_1 \to^{\circledast} B \qquad \textit{and} \qquad u \mid \varphi \wedge \psi_2 \to^{\circledast} B$$

provided: (1) ψ_1 and ψ_2 do not have any extra variables besides those of $u \mid \varphi$, and (2) $\mathcal{T}_{\Sigma/E \cup B} \models \psi_1 \vee \psi_2$,

Generic Properties of Reachability Formulas (IV)

The **Split** auxiliary inference rule of Reachability logic ensures that the validity of a reachability formula

$$u \mid \varphi \to^{\circledast} B$$

in a rewrite theory $\mathcal{R}$ is equivalent to the validity of the two reachability formulas:

$$u \mid \varphi \wedge \psi_1 \to^{\circledast} B \qquad \textit{and} \qquad u \mid \varphi \wedge \psi_2 \to^{\circledast} B$$

provided: (1) ψ_1 and ψ_2 do not have any extra variables besides those of $u \mid \varphi$, and (2) $\mathcal{T}_{\Sigma/E \cup B} \models \psi_1 \vee \psi_2$, where $\mathcal{T}_{\Sigma/E \cup B}$ is the initial algebra of $\mathcal{R}$'s underlying equational theory.

A Methodology for Proving IMPL Programs

We can combine the above generic properties of reachability logic with some domain-specific properties of the semantics of IMPL to

A Methodology for Proving IMPL Programs

We can combine the above generic properties of reachability logic with some domain-specific properties of the semantics of IMPL to arrive at a practical proof methodology to:

A Methodology for Proving IMPL Programs

We can combine the above generic properties of reachability logic with some domain-specific properties of the semantics of IMPL to arrive at a practical proof methodology to:

1. Develop a **Proof Plan** on how to specify the properties of an IMPL program P: (1) in a compositional way,

A Methodology for Proving IMPL Programs

We can combine the above generic properties of reachability logic with some domain-specific properties of the semantics of IMPL to arrive at a practical proof methodology to:

1. Develop a **Proof Plan** on how to specify the properties of an IMPL program P: (1) in a compositional way, that is, in terms of properties of its subprograms, and

A Methodology for Proving IMPL Programs

We can combine the above generic properties of reachability logic with some domain-specific properties of the semantics of IMPL to arrive at a practical proof methodology to:

1. Develop a **Proof Plan** on how to specify the properties of an IMPL program P: (1) in a compositional way, that is, in terms of properties of its subprograms, and (2) so as to maximize the chances of getting an automatic or semi-automatic proof in the IMPL Prover.

A Methodology for Proving IMPL Programs

We can combine the above generic properties of reachability logic with some domain-specific properties of the semantics of IMPL to arrive at a practical proof methodology to:

1. Develop a **Proof Plan** on how to specify the properties of an IMPL program P: (1) in a compositional way, that is, in terms of properties of its subprograms, and (2) so as to maximize the chances of getting an automatic or semi-automatic proof in the IMPL Prover.
2. Then (and only then!) implement the proof plan so obtained in the IMPL Prover.

A Methodology for Proving IMPL Programs

We can combine the above generic properties of reachability logic with some domain-specific properties of the semantics of IMPL to arrive at a practical proof methodology to:

1. Develop a **Proof Plan** on how to specify the properties of an IMPL program P: (1) in a compositional way, that is, in terms of properties of its subprograms, and (2) so as to maximize the chances of getting an automatic or semi-automatic proof in the IMPL Prover.
2. Then (and only then!) implement the proof plan so obtained in the IMPL Prover.

But before, we need some notation.

Interlude on Notation

The same way that t denotes a Σ-term or expression in a given signature Σ of operators, let $t{:}(\vec{x})$ denote an IMPL expression mentioning program variables $\vec{x}$.

Interlude on Notation

The same way that t denotes a Σ-term or expression in a given signature Σ of operators, let $t{:}(\vec{x})$ denote an IMPL expression mentioning program variables $\vec{x}$. The point of this notation is that, given a VS pattern $\vec{x} \mapsto \vec{X}$, we can associate to the IMPL expression $t{:}(\vec{x})$

Interlude on Notation

The same way that t denotes a Σ-term or expression in a given signature Σ of operators, let $t{:}(\vec{x})$ denote an IMPL expression mentioning program variables $\vec{x}$. The point of this notation is that, given a VS pattern $\vec{x} \mapsto \vec{X}$, we can associate to the IMPL expression $t{:}(\vec{x})$ a corresponding Σ_D-expression $t(\vec{X})$,

Interlude on Notation

The same way that t denotes a Σ-term or expression in a given signature Σ of operators, let $t{:}(\vec{x})$ denote an IMPL expression mentioning program variables $\vec{x}$. The point of this notation is that, given a VS pattern $\vec{x} \mapsto \vec{X}$, we can associate to the IMPL expression $t{:}(\vec{x})$ a corresponding Σ_D-expression $t(\vec{X})$, where Σ_D is the signature of the underlying data type (combining natural, list and Boolean operations)

Interlude on Notation

The same way that t denotes a Σ-term or expression in a given signature Σ of operators, let $t{:}(\vec{x})$ denote an IMPL expression mentioning program variables $\vec{x}$. The point of this notation is that, given a VS pattern $\vec{x} \mapsto \vec{X}$, we can associate to the IMPL expression $t{:}(\vec{x})$ a corresponding Σ_D-expression $t(\vec{X})$, where Σ_D is the signature of the underlying data type (combining natural, list and Boolean operations) where the IMPL expressions are evaluated in IMPL's continuation semantics.

Interlude on Notation

The same way that t denotes a Σ-term or expression in a given signature Σ of operators, let $t{:}(\vec{x})$ denote an IMPL expression mentioning program variables $\vec{x}$. The point of this notation is that, given a VS pattern $\vec{x} \mapsto \vec{X}$, we can associate to the IMPL expression $t{:}(\vec{x})$ a corresponding Σ_D-expression $t(\vec{X})$, where Σ_D is the signature of the underlying data type (combining natural, list and Boolean operations) where the IMPL expressions are evaluated in IMPL's continuation semantics.

For example, if $t{:}(\vec{x})$ is the IMPL expression $x *{:}\ (y {+}{:}\ x)$,

Interlude on Notation

The same way that t denotes a Σ-term or expression in a given signature Σ of operators, let $t{:}(\vec{x})$ denote an IMPL expression mentioning program variables $\vec{x}$. The point of this notation is that, given a VS pattern $\vec{x} \mapsto \vec{X}$, we can associate to the IMPL expression $t{:}(\vec{x})$ a corresponding Σ_D-expression $t(\vec{X})$, where Σ_D is the signature of the underlying data type (combining natural, list and Boolean operations) where the IMPL expressions are evaluated in IMPL's continuation semantics.

For example, if $t{:}(\vec{x})$ is the IMPL expression $x *{:}\ (y{+}{:}\ x)$, given the VS pattern $x \mapsto X * y \mapsto Y$,

Interlude on Notation

The same way that t denotes a Σ-term or expression in a given signature Σ of operators, let $t{:}(\vec{x})$ denote an IMPL expression mentioning program variables $\vec{x}$. The point of this notation is that, given a VS pattern $\vec{x} \mapsto \vec{X}$, we can associate to the IMPL expression $t{:}(\vec{x})$ a corresponding Σ_D-expression $t(\vec{X})$, where Σ_D is the signature of the underlying data type (combining natural, list and Boolean operations) where the IMPL expressions are evaluated in IMPL's continuation semantics.

For example, if $t{:}(\vec{x})$ is the IMPL expression $x *{:}\ (y{+}{:}\ x)$, given the VS pattern $x \mapsto X * y \mapsto Y$, $t(\vec{X})$ is the Σ_D-expression $X * (Y + X)$.

Proving Hoare Tiples

Recall from Lecture 23 that, to prove a Hoare triple as a reachability formula:

$$< P \rightsquigarrow done \mid TS \ \& \ \vec{x} \mapsto \vec{X} * VS >\mid \varphi \rightarrow^{\circledast} < done \mid TS \ \& \ \vec{x} \mapsto \vec{X}' * VS >\mid \psi$$

Proving Hoare Tiples

Recall from Lecture 23 that, to prove a Hoare triple as a reachability formula:

$$< P \rightsquigarrow done \mid TS \;\&\; \vec{x} \mapsto \vec{X} * VS >\mid \varphi \rightarrow^{\circledast} < done \mid TS \;\&\; \vec{x} \mapsto \vec{X}' * VS >\mid \psi$$

we can Generalize and Conquer and prove instead the more general formula:

$$< P \rightsquigarrow K \mid TS \;\&\; \vec{x} \mapsto \vec{X} * VS >\mid \varphi \rightarrow^{\circledast} < K \mid TS \;\&\; \vec{x} \mapsto \vec{X}' * VS >\mid \psi$$

Proving Hoare Tiples

Recall from Lecture 23 that, to prove a Hoare triple as a reachability formula:

$$< P \leadsto done \mid TS \ \& \ \vec{x} \mapsto \vec{X} * VS >\mid \varphi \rightarrow^{\circledast} < done \mid TS \ \& \ \vec{x} \mapsto \vec{X}' * VS >\mid \psi$$

we can Generalize and Conquer and prove instead the more general formula:

$$< P \leadsto K \mid TS \ \& \ \vec{x} \mapsto \vec{X} * VS >\mid \varphi \rightarrow^{\circledast} < K \mid TS \ \& \ \vec{x} \mapsto \vec{X}' * VS >\mid \psi$$

But why is this correct?

Proving Hoare Tiples

Recall from Lecture 23 that, to prove a Hoare triple as a reachability formula:

$$< P \rightsquigarrow done \mid TS \;\&\; \vec{x} \mapsto \vec{X} * VS >\mid \varphi \rightarrow^{\circledast} < done \mid TS \;\&\; \vec{x} \mapsto \vec{X}' * VS >\mid \psi$$

we can Generalize and Conquer and prove instead the more general formula:

$$< P \rightsquigarrow K \mid TS \;\&\; \vec{x} \mapsto \vec{X} * VS >\mid \varphi \rightarrow^{\circledast} < K \mid TS \;\&\; \vec{x} \mapsto \vec{X}' * VS >\mid \psi$$

But why is this correct? Because of the **Constructor Instantiation of a Parameter** inference rule, since:

Proving Hoare Tiples

Recall from Lecture 23 that, to prove a Hoare triple as a reachability formula:

$$< P \rightsquigarrow done \mid TS \,\&\, \vec{x} \mapsto \vec{X} * VS >\mid \varphi \rightarrow^{\circledast} < done \mid TS \,\&\, \vec{x} \mapsto \vec{X}' * VS >\mid \psi$$

we can Generalize and Conquer and prove instead the more general formula:

$$< P \rightsquigarrow K \mid TS \,\&\, \vec{x} \mapsto \vec{X} * VS >\mid \varphi \rightarrow^{\circledast} < K \mid TS \,\&\, \vec{x} \mapsto \vec{X}' * VS >\mid \psi$$

But why is this correct? Because of the **Constructor Instantiation of a Parameter** inference rule, since: (1) K is clearly a parameter of the general formula, and

Proving Hoare Tiples

Recall from Lecture 23 that, to prove a Hoare triple as a reachability formula:

$$< P \rightsquigarrow done \mid TS \;\&\; \vec{x} \mapsto \vec{X} {*} VS >\mid \varphi \rightarrow^{\circledast} < done \mid TS \;\&\; \vec{x} \mapsto \vec{X}' {*} VS >\mid \psi$$

we can Generalize and Conquer and prove instead the more general formula:

$$< P \rightsquigarrow K \mid TS \;\&\; \vec{x} \mapsto \vec{X} * VS >\mid \varphi \rightarrow^{\circledast} < K \mid TS \;\&\; \vec{x} \mapsto \vec{X}' * VS >\mid \psi$$

But why is this correct? Because of the **Constructor Instantiation of a Parameter** inference rule, since: (1) K is clearly a parameter of the general formula, and (2) the Hoare triple formula is its instance by the constructor substitution $\{K \mapsto done\}$.

Proving Loops with Loop Invariants

Suppose we want to prove the following while loop property:

$$< \textbf{while}\ b{:}(\vec{x})\ \{stmt\} \rightsquigarrow K \mid TS \;\&\; \vec{x} \mapsto \vec{X} * VS >\mid \varphi$$
$$\rightarrow^{\circledast} < K \mid TS \;\&\; \vec{x} \mapsto \vec{X}' * VS >\mid \psi$$

with $b{:}(\vec{x})$ the loop's guard, $stmt$ its body, and besides K, $\vec{Y}$ its data parameters.

Proving Loops with Loop Invariants

Suppose we want to prove the following while loop property:

$$< \textbf{while}\ b{:}(\vec{x})\ \{stmt\} \rightsquigarrow K \mid TS \;\&\; \vec{x} \mapsto \vec{X} * VS >\mid \varphi$$
$$\rightarrow^{\circledast} < K \mid TS \;\&\; \vec{x} \mapsto \vec{X}' * VS >\mid \psi$$

with $b{:}(\vec{x})$ the loop's guard, $stmt$ its body, and besides K, $\vec{Y}$ its data parameters. Sending this formula to the IMPL prover without any forethought may be quite ineffective.

Proving Loops with Loop Invariants

Suppose we want to prove the following while loop property:

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

$$\rightarrow^{\circledast} < K \mid TS \ \&\ \vec{x} \mapsto \vec{X}' * VS >\mid \psi$$

with $b{:}(\vec{x})$ the loop's guard, $stmt$ its body, and besides K, $\vec{Y}$ its data parameters. Sending this formula to the IMPL prover without any forethought may be quite ineffective.

What can we do?

Proving Loops with Loop Invariants

Suppose we want to prove the following while loop property:

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

with $b{:}(\vec{x})$ the loop's guard, $stmt$ its body, and besides K, $\vec{Y}$ its data parameters. Sending this formula to the IMPL prover without any forethought may be quite ineffective.

What can we do? Use two ideas:

Proving Loops with Loop Invariants

Suppose we want to prove the following while loop property:

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

with $b{:}(\vec{x})$ the loop's guard, $stmt$ its body, and besides K, $\vec{Y}$ its data parameters. Sending this formula to the IMPL prover without any forethought may be quite ineffective.

What can we do? Use two ideas: (1) We can apply the **Expanding Preconditions and Restricting Midconditions** rule to replace this formula by a much easier to prove one.

Proving Loops with Loop Invariants

Suppose we want to prove the following while loop property:

$$< \textbf{while}\ b{:}(\vec{x})\ \{stmt\} \leadsto K \mid TS\ \&\ \vec{x} \mapsto \vec{X} * VS >\mid \varphi$$

$$\to^{\circledast} < K \mid TS\ \&\ \vec{x} \mapsto \vec{X}' * VS >\mid \psi$$

with $b{:}(\vec{x})$ the loop's guard, $stmt$ its body, and besides K, $\vec{Y}$ its data parameters. Sending this formula to the IMPL prover without any forethought may be quite ineffective.

What can we do? Use two ideas: (1) We can apply the **Expanding Preconditions and Restricting Midconditions** rule to replace this formula by a much easier to prove one. By calling A and D the above pre- and postcondition, the above formula is summarized as:

Proving Loops with Loop Invariants

Suppose we want to prove the following while loop property:

$$< \textbf{while}\ b{:}(\vec{x})\ \{stmt\} \rightsquigarrow K \mid TS\ \&\ \vec{x} \mapsto \vec{X} * VS >\mid \varphi$$
$$\to^{\circledast} < K \mid TS\ \&\ \vec{x} \mapsto \vec{X'} * VS >\mid \psi$$

with $b{:}(\vec{x})$ the loop's guard, $stmt$ its body, and besides K, $\vec{Y}$ its data parameters. Sending this formula to the IMPL prover without any forethought may be quite ineffective.

What can we do? Use two ideas: (1) We can apply the **Expanding Preconditions and Restricting Midconditions** rule to replace this formula by a much easier to prove one. By calling A and D the above pre- and postcondition, the above formula is summarized as:

$$A[\textbf{while}\ b{:}(\vec{x})\ \{stmt\} \rightsquigarrow K] \to^{\circledast} D[K]$$

Proving Loops with Loop Invariants

Suppose we want to prove the following while loop property:

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

with $b{:}(\vec{x})$ the loop's guard, $stmt$ its body, and besides K, $\vec{Y}$ its data parameters. Sending this formula to the IMPL prover without any forethought may be quite ineffective.

What can we do? Use two ideas: (1) We can apply the **Expanding Preconditions and Restricting Midconditions** rule to replace this formula by a much easier to prove one. By calling A and D the above pre- and postcondition, the above formula is summarized as:

$$A[\textbf{while } b{:}(\vec{x})\ \{stmt\} \rightsquigarrow K] \rightarrow^{\circledast} D[K]$$

We need to find a much easier to prove

$$B[\textbf{while } b{:}(\vec{x})\ \{stmt\} \rightsquigarrow K] \rightarrow^{\circledast} C[K]$$

such that

Proving Loops with Loop Invariants

Suppose we want to prove the following while loop property:

$$< \textbf{while}\ b{:}(\vec{x})\ \{stmt\} \rightsquigarrow K \mid TS \ \&\ \vec{x} \mapsto \vec{X} * VS >\mid \varphi$$
$$\to^{\circledast} < K \mid TS \ \&\ \vec{x} \mapsto \vec{X}' * VS >\mid \psi$$

with $b{:}(\vec{x})$ the loop's guard, $stmt$ its body, and besides K, $\vec{Y}$ its data parameters. Sending this formula to the IMPL prover without any forethought may be quite ineffective.

What can we do? Use two ideas: (1) We can apply the **Expanding Preconditions and Restricting Midconditions** rule to replace this formula by a much easier to prove one. By calling A and D the above pre- and postcondition, the above formula is summarized as:

$$A[\textbf{while}\ b{:}(\vec{x})\ \{stmt\} \rightsquigarrow K] \to^{\circledast} D[K]$$

We need to find a much easier to prove

$$B[\textbf{while}\ b{:}(\vec{x})\ \{stmt\} \rightsquigarrow K] \to^{\circledast} C[K]$$

such that $A \sqsubseteq_{K,\vec{Y}} B$ and $C \sqsubseteq_{K,\vec{Y}} D$.

Proving Loops with Loop Invariants

Suppose we want to prove the following while loop property:

$$< \textbf{while}\ b\!:\!(\vec{x})\ \{stmt\} \leadsto K \mid TS\ \&\ \vec{x} \mapsto \vec{X} * VS >\mid \varphi$$
$$\to^{\circledast} < K \mid TS\ \&\ \vec{x} \mapsto \vec{X}' * VS >\mid \psi$$

with $b\!:\!(\vec{x})$ the loop's guard, $stmt$ its body, and besides K, $\vec{Y}$ its data parameters. Sending this formula to the IMPL prover without any forethought may be quite ineffective.

What can we do? Use two ideas: (1) We can apply the **Expanding Preconditions and Restricting Midconditions** rule to replace this formula by a much easier to prove one. By calling A and D the above pre- and postcondition, the above formula is summarized as:

$$A[\textbf{while}\ b\!:\!(\vec{x})\ \{stmt\} \leadsto K] \to^{\circledast} D[K]$$

We need to find a much easier to prove

$$B[\textbf{while}\ b\!:\!(\vec{x})\ \{stmt\} \leadsto K] \to^{\circledast} C[K]$$

such that $A \sqsubseteq_{K,\vec{Y}} B$ and $C \sqsubseteq_{K,\vec{Y}} D$.

Proving Loops with Loop Invariants

Suppose we want to prove the following while loop property:

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

with $b{:}(\vec{x})$ the loop's guard, $stmt$ its body, and besides K, $\vec{Y}$ its data parameters. Sending this formula to the IMPL prover without any forethought may be quite ineffective.

What can we do? Use two ideas: (1) We can apply the **Expanding Preconditions and Restricting Midconditions** rule to replace this formula by a much easier to prove one. By calling A and D the above pre- and postcondition, the above formula is summarized as:

$$A[\textbf{while}\ b{:}(\vec{x})\ \{stmt\} \rightsquigarrow K] \rightarrow^{\circledast} D[K]$$

We need to find a much easier to prove

$$B[\textbf{while}\ b{:}(\vec{x})\ \{stmt\} \rightsquigarrow K] \rightarrow^{\circledast} C[K]$$

such that $A \sqsubseteq_{K,\vec{Y}} B$ and $C \sqsubseteq_{K,\vec{Y}} D$. How?

Proving Loops with Loop Invariants

Suppose we want to prove the following while loop property:

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

with $b{:}(\vec{x})$ the loop's guard, $stmt$ its body, and besides K, $\vec{Y}$ its data parameters. Sending this formula to the IMPL prover without any forethought may be quite ineffective.

What can we do? Use two ideas: (1) We can apply the **Expanding Preconditions and Restricting Midconditions** rule to replace this formula by a much easier to prove one. By calling A and D the above pre- and postcondition, the above formula is summarized as:

$$A[\textbf{while}\ b{:}(\vec{x})\ \{stmt\} \rightsquigarrow K] \rightarrow^{\circledast} D[K]$$

We need to find a much easier to prove

$$B[\textbf{while}\ b{:}(\vec{x})\ \{stmt\} \rightsquigarrow K] \rightarrow^{\circledast} C[K]$$

such that $A \sqsubseteq_{K,\vec{Y}} B$ and $C \sqsubseteq_{K,\vec{Y}} D$. How? Using Idea (2).

Proving Loops with Loop Invariants (II)

Idea (2): The key remark is that loops are a repetitive computation.

Proving Loops with Loop Invariants (II)

Idea (2): The key remark is that loops are a repetitive computation. If we can find a property I that is preserved by each iteration of the loop's body, so that it remains invariant,

Proving Loops with Loop Invariants (II)

Idea (2): The key remark is that loops are a repetitive computation. If we can find a property I that is preserved by each iteration of the loop's body, so that it remains invariant,then we can choose $B = I$ and $C = I\sigma$ (σ remames $\vec{X}$ to $\vec{X}'$)

Proving Loops with Loop Invariants (II)

Idea (2): The key remark is that loops are a repetitive computation. If we can find a property I that is preserved by each iteration of the loop's body, so that it remains invariant,then we can choose $B = I$ and $C = I\sigma$ (σ remames $\vec{X}$ to $\vec{X}'$) to get a much easier to prove formula:

Proving Loops with Loop Invariants (II)

Idea (2): The key remark is that loops are a repetitive computation. If we can find a property I that is preserved by each iteration of the loop's body, so that it remains invariant,then we can choose $B = I$ and $C = I\sigma$ (σ remames $\vec{X}$ to $\vec{X}'$) to get a much easier to prove formula:

$$I[\textbf{while}\ b{:}(\vec{x})\ \{stmt\} \rightsquigarrow K] \rightarrow^{\circledast} I\sigma[K]$$

Proving Loops with Loop Invariants (II)

Idea (2): The key remark is that loops are a repetitive computation. If we can find a property I that is preserved by each iteration of the loop's body, so that it remains invariant,then we can choose $B = I$ and $C = I\sigma$ (σ remames $\vec{X}$ to $\vec{X}'$) to get a much easier to prove formula:

$$I[\textbf{while}\ b{:}(\vec{x})\ \{stmt\} \rightsquigarrow K] \rightarrow^{\circledast} I\sigma[K]$$

Why much easier to prove?

Proving Loops with Loop Invariants (II)

Idea (2): The key remark is that loops are a repetitive computation. If we can find a property I that is preserved by each iteration of the loop's body, so that it remains invariant,then we can choose $B = I$ and $C = I\sigma$ (σ remames $\vec{X}$ to $\vec{X}'$) to get a much easier to prove formula:

$$I[\textbf{while}\ b{:}(\vec{x})\ \{stmt\} \rightsquigarrow K] \rightarrow^{\circledast} I\sigma[K]$$

Why much easier to prove? Because it greatly increases the chances that the **Axiom** rule, which is a seven league boots rule designed to detect repetitive behavior will kick in and allow

Proving Loops with Loop Invariants (II)

Idea (2): The key remark is that loops are a repetitive computation. If we can find a property I that is preserved by each iteration of the loop's body, so that it remains invariant,then we can choose $B = I$ and $C = I\sigma$ (σ remames $\vec{X}$ to $\vec{X}'$) to get a much easier to prove formula:

$$I[\textbf{while}\ b{:}(\vec{x})\ \{stmt\} \rightsquigarrow K] \rightarrow^{\circledast} I\sigma[K]$$

Why much easier to prove? Because it greatly increases the chances that the **Axiom** rule, which is a seven league boots rule designed to detect repetitive behavior will kick in and allow the **Subsumption** rule to prove the goal.

Proving Loops with Loop Invariants (II)

Idea (2): The key remark is that loops are a repetitive computation. If we can find a property I that is preserved by each iteration of the loop's body, so that it remains invariant,then we can choose $B = I$ and $C = I\sigma$ (σ remames $\vec{X}$ to $\vec{X}'$) to get a much easier to prove formula:

$$I[\textbf{while}\ b\text{:}(\vec{x})\ \{stmt\} \rightsquigarrow K] \to^{\circledast} I\sigma[K]$$

Why much easier to prove? Because it greatly increases the chances that the **Axiom** rule, which is a seven league boots rule designed to detect repetitive behavior will kick in and allow the **Subsumption** rule to prove the goal.

Ideas (1) and (2) are combined in a proof method of loop invariants based on the following steps:

Loop Invariants

Step 1: We guess an invariant I that:

Loop Invariants

Step 1: We guess an invariant I that: (i) will be true when entering the loop, and

Loop Invariants

Step 1: We guess an invariant I that: (i) will be true when entering the loop, and (ii) will be preserved by the loop's body $stmt$.

Loop Invariants

Step 1: We guess an invariant I that: (i) will be true when entering the loop, and (ii) will be preserved by the loop's body $stmt$. That is, I should satisfy:

Loop Invariants

Step 1: We guess an invariant I that: (i) will be true when entering the loop, and (ii) will be preserved by the loop's body $stmt$. That is, I should satisfy:

$$< stmt \rightsquigarrow K \mid TS \;\&\; \vec{x} \mapsto \vec{X} * VS >\mid I(\vec{Y}, \vec{X}) \wedge b(\vec{X}) = true$$
$$\rightarrow^{\circledast} < K \mid TS \;\&\; \vec{x} \mapsto \vec{X}' * VS >\mid I(\vec{Y}, \vec{X}')$$

Loop Invariants

Step 1: We guess an invariant I that: (i) will be true when entering the loop, and (ii) will be preserved by the loop's body $stmt$. That is, I should satisfy:

$$< stmt \rightsquigarrow K \mid TS \,\&\, \vec{x} \mapsto \vec{X} * VS >\mid I(\vec{Y}, \vec{X}) \wedge b(\vec{X}) = true$$
$$\rightarrow^{\circledast} < K \mid TS \,\&\, \vec{x} \mapsto \vec{X}' * VS >\mid I(\vec{Y}, \vec{X}')$$

Intuitively, $I(\vec{Y}, \vec{X})$ is a property that:

Loop Invariants

Step 1: We guess an invariant I that: (i) will be true when entering the loop, and (ii) will be preserved by the loop's body $stmt$. That is, I should satisfy:

$$< stmt \leadsto K \mid TS \,\&\, \vec{x} \mapsto \vec{X} * VS >\mid I(\vec{Y}, \vec{X}) \wedge b(\vec{X}) = true$$
$$\rightarrow^{\circledast} < K \mid TS \,\&\, \vec{x} \mapsto \vec{X}' * VS >\mid I(\vec{Y}, \vec{X}')$$

Intuitively, $I(\vec{Y}, \vec{X})$ is a property that: holds before $stmt$ is executed (assuming $b(\vec{X}) = true$), and

Loop Invariants

Step 1: We guess an invariant I that: (i) will be true when entering the loop, and (ii) will be preserved by the loop's body $stmt$. That is, I should satisfy:

$$< stmt \rightsquigarrow K \mid TS \ \& \ \vec{x} \mapsto \vec{X} * VS >\mid I(\vec{Y}, \vec{X}) \wedge b(\vec{X}) = true$$
$$\rightarrow^{\circledast} < K \mid TS \ \& \ \vec{x} \mapsto \vec{X}' * VS >\mid I(\vec{Y}, \vec{X}')$$

Intuitively, $I(\vec{Y}, \vec{X})$ is a property that: holds before $stmt$ is executed (assuming $b(\vec{X}) = true$), and after $stmt$ is executed.

Loop Invariants (II)

Step 2: Since a terminating execution of the loop

Loop Invariants (II)

Step 2: Since a terminating execution of the loop will just execute $stmt$ a finite number of times,

Loop Invariants (II)

Step 2: Since a terminating execution of the loop will just execute $stmt$ a finite number of times, until $b{:}(\vec{x})$ becomes false,

Loop Invariants (II)

Step 2: Since a terminating execution of the loop will just execute $stmt$ a finite number of times, until $b{:}(\vec{x})$ becomes false, the invariant I will be true not only before entering the loop, but also immediately after exiting it.

Loop Invariants (II)

Step 2: Since a terminating execution of the loop will just execute $stmt$ a finite number of times, until b:$(\vec{x})$ becomes false, the invariant I will be true not only before entering the loop, but also immediately after exiting it.

Therefore, a good initial guess for the loop invariant is:

Loop Invariants (II)

Step 2: Since a terminating execution of the loop will just execute $stmt$ a finite number of times, until $b{:}(\vec{x})$ becomes false, the invariant I will be true not only before entering the loop, but also immediately after exiting it.

Therefore, a good initial guess for the loop invariant is:

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

Loop Invariants (III)

Step 3: Our initial guess for the loop invariant may be insufficient.

Loop Invariants (III)

Step 3: Our initial guess for the loop invariant may be insufficient.
For two reasons:

Loop Invariants (III)

Step 3: Our initial guess for the loop invariant may be insufficient. For two reasons: (1) The invariant $I(\vec{Y}, \vec{X})$ may be too general,

Loop Invariants (III)

Step 3: Our initial guess for the loop invariant may be insufficient. For two reasons: (1) The invariant $I(\vec{Y}, \vec{X})$ may be too general, called a weak invariant,

Loop Invariants (III)

Step 3: Our initial guess for the loop invariant may be insufficient. For two reasons: (1) The invariant $I(\vec{Y}, \vec{X})$ may be too general, called a weak invariant, making it hard for the IMPL prover to prove it.

Loop Invariants (III)

Step 3: Our initial guess for the loop invariant may be insufficient. For two reasons: (1) The invariant $I(\vec{Y}, \vec{X})$ may be too general, called a weak invariant, making it hard for the IMPL prover to prove it. (2) $I(\vec{Y}, \vec{X})$ may not fit well with our original formula:

Loop Invariants (III)

Step 3: Our initial guess for the loop invariant may be insufficient. For two reasons: (1) The invariant $I(\vec{Y}, \vec{X})$ may be too general, called a weak invariant, making it hard for the IMPL prover to prove it. (2) $I(\vec{Y}, \vec{X})$ may not fit well with our original formula:

$$A[\textbf{while } b{:}(\vec{x})\ \{stmt\} \rightsquigarrow K] \rightarrow^{\circledast} D[K]$$

Loop Invariants (III)

Step 3: Our initial guess for the loop invariant may be insufficient. For two reasons: (1) The invariant $I(\vec{Y}, \vec{X})$ may be too general, called a weak invariant, making it hard for the IMPL prover to prove it. (2) $I(\vec{Y}, \vec{X})$ may not fit well with our original formula:

$$A[\textbf{while}\ b{:}(\vec{x})\ \{stmt\} \rightsquigarrow K] \rightarrow^{\circledast} D[K]$$

For example, $I(\vec{Y}, \vec{X}')$ may be too general,

Loop Invariants (III)

Step 3: Our initial guess for the loop invariant may be insufficient. For two reasons: (1) The invariant $I(\vec{Y}, \vec{X})$ may be too general, called a weak invariant, making it hard for the IMPL prover to prove it. (2) $I(\vec{Y}, \vec{X})$ may not fit well with our original formula:

$$A[\textbf{while}\ b{:}(\vec{x})\ \{stmt\} \rightsquigarrow K] \to^{\circledast} D[K]$$

For example, $I(\vec{Y}, \vec{X}')$ may be too general, and therefore may not allow us to prove the original midcondition D.

Reasons (1) and (2) both move us in the same direction.

Loop Invariants (III)

Step 3: Our initial guess for the loop invariant may be insufficient. For two reasons: (1) The invariant $I(\vec{Y}, \vec{X})$ may be too general, called a weak invariant, making it hard for the IMPL prover to prove it. (2) $I(\vec{Y}, \vec{X})$ may not fit well with our original formula:

$$A[\textbf{while } b{:}(\vec{x})\ \{stmt\} \rightsquigarrow K] \to^{\circledast} D[K]$$

For example, $I(\vec{Y}, \vec{X}')$ may be too general, and therefore may not allow us to prove the original midcondition D.

Reasons (1) and (2) both move us in the same direction. We should strengthen the invariant I into a stronger:

$$I_{str}(\vec{Y}, \vec{X}) = I(\vec{Y}, \vec{X}) \wedge \phi.$$

Loop Invariants (IV)

Step 4: Abusing notation and calling I_{str} not just the data constraint but the entire pattern predicate, if we can show that:

$$A[\textbf{while } b{:}(\vec{x})\ \{stmt\} \rightsquigarrow K] \sqsubseteq_{K,\vec{Y}} I_{str}[\textbf{while } b{:}(\vec{x})\ \{stmt\} \rightsquigarrow K]$$

and

$$I_{str}\sigma[K] \sqsubseteq_{K,\vec{Y}} D[K]$$

then, by the **Expanding Preconditions and Restricting Midconditions** rule, if the IMPL Prover can prove:

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

we have also proved our original goal:

$$A[\textbf{while } b{:}(\vec{x})\ \{stmt\} \rightsquigarrow K] \rightarrow^{\circledast} D[K].$$

Proving Properties of Assignments

Suppose that we have a pattern formula for the precondition of an assignment statement,

Proving Properties of Assignments

Suppose that we have a pattern formula for the precondition of an assignment statement, and we would like to guess for the pattern formula of its midcondition.

Proving Properties of Assignments

Suppose that we have a pattern formula for the precondition of an assignment statement, and we would like to guess for the pattern formula of its midcondition. That is, we would like to resolve the guess: (the case for list assignments is similar):

Proving Properties of Assignments

Suppose that we have a pattern formula for the precondition of an assignment statement, and we would like to guess for the pattern formula of its midcondition. That is, we would like to resolve the guess: (the case for list assignments is similar):

$$< x_0 = t{:}(x_0, \vec{x}); \rightsquigarrow K \mid TS \;\&\; x_0 \mapsto X_0 * \vec{x} \mapsto \vec{X} * VS >\mid \varphi \rightarrow^{\circledast} \;\; ??$$

Proving Properties of Assignments

Suppose that we have a pattern formula for the precondition of an assignment statement, and we would like to guess for the pattern formula of its midcondition. That is, we would like to resolve the guess: (the case for list assignments is similar):

$$< x_0 = t{:}(x_0, \vec{x}); \rightsquigarrow K \mid TS \;\&\; x_0 \mapsto X_0 * \vec{x} \mapsto \vec{X} * VS >\mid \varphi \rightarrow^{\circledast} \;??$$

(x_0 might or might not appear in $t{:}(x_0, \vec{x})$)

Proving Properties of Assignments

Suppose that we have a pattern formula for the precondition of an assignment statement, and we would like to guess for the pattern formula of its midcondition. That is, we would like to resolve the guess: (the case for list assignments is similar):

$$< x_0 = t{:}(x_0, \vec{x}); \rightsquigarrow K \mid TS \;\&\; x_0 \mapsto X_0 * \vec{x} \mapsto \vec{X} * VS >\mid \varphi \rightarrow^{\circledast} \;??$$

(x_0 might or might not appear in $t{:}(x_0, \vec{x})$) What can we do?

Proving Properties of Assignments

Suppose that we have a pattern formula for the precondition of an assignment statement, and we would like to guess for the pattern formula of its midcondition. That is, we would like to resolve the guess: (the case for list assignments is similar):

$$< x_0 = t{:}(x_0, \vec{x}); \rightsquigarrow K \mid TS \,\&\, x_0 \mapsto X_0 * \vec{x} \mapsto \vec{X} * VS >\mid \varphi \to^{\circledast} \; ??$$

(x_0 might or might not appear in $t{:}(x_0, \vec{x})$) What can we do? Using our knowledge of the continuation semantics of IMPL, we know that the following is always a correct guess:

Proving Properties of Assignments

Suppose that we have a pattern formula for the precondition of an assignment statement, and we would like to guess for the pattern formula of its midcondition. That is, we would like to resolve the guess: (the case for list assignments is similar):

$$< x_0 = t{:}(x_0, \vec{x}); \rightsquigarrow K \mid TS \ \& \ x_0 \mapsto X_0 * \vec{x} \mapsto \vec{X} * VS >\mid \varphi \rightarrow^{\circledast} \ ??$$

(x_0 might or might not appear in $t{:}(x_0, \vec{x})$) What can we do? Using our knowledge of the continuation semantics of IMPL, we know that the following is always a correct guess:

$$< x_0 = t{:}(x_0, \vec{x}); \rightsquigarrow K \mid TS \ \& \ x_0 \mapsto X_0 * \vec{x} \mapsto \vec{X} * VS >\mid \varphi$$
$$\rightarrow^{\circledast} < K \mid TS \ \& \ x_0 \mapsto X_0' * \vec{x} \mapsto \vec{X} * VS >\mid \varphi \wedge X_0' = t(X_0, \vec{X}).$$

Proving Properties about Conditional Statements

Suppose we want to prove a reachability formula of the form:

Proving Properties about Conditional Statements

Suppose we want to prove a reachability formula of the form:

$$< \textbf{if}\ (b{:}(\vec{x}))\ stmt\ \textbf{else}\ stmt' \rightsquigarrow K \mid TS\ \&\ \vec{x} \mapsto \vec{X} * VS >\mid \varphi$$

$$\rightarrow^{\circledast} < K \mid TS\ \&\ \vec{x} \mapsto \vec{X}' * VS >\mid \psi$$

Proving Properties about Conditional Statements

Suppose we want to prove a reachability formula of the form:

$$< \textbf{if } (b{:}(\vec{x}))\ stmt \textbf{ else } stmt' \rightsquigarrow K \mid TS \,\&\, \vec{x} \mapsto \vec{X} * VS >\mid \varphi$$

$$\to^{\circledast} < K \mid TS \,\&\, \vec{x} \mapsto \vec{X}' * VS >\mid \psi$$

What can we do?

Proving Properties about Conditional Statements

Suppose we want to prove a reachability formula of the form:

$$< \textbf{if}\ (b{:}(\vec{x}))\ stmt\ \textbf{else}\ stmt' \rightsquigarrow K \mid TS\ \&\ \vec{x} \mapsto \vec{X} * VS >\mid \varphi$$

$$\rightarrow^{\circledast} < K \mid TS\ \&\ \vec{x} \mapsto \vec{X}' * VS >\mid \psi$$

What can we do? We can apply the **Split** rule to reduce our original goal to the two simpler goals semantically equivalent to it:

Proving Properties about Conditional Statements

Suppose we want to prove a reachability formula of the form:

$$< \textbf{if}\ (b{:}(\vec{x}))\ stmt\ \textbf{else}\ stmt' \rightsquigarrow K \mid TS\ \&\ \vec{x} \mapsto \vec{X} * VS >\mid \varphi$$
$$\rightarrow^{\circledast} < K \mid TS\ \&\ \vec{x} \mapsto \vec{X}' * VS >\mid \psi$$

What can we do? We can apply the **Split** rule to reduce our original goal to the two simpler goals semantically equivalent to it:

$$< stmt\ \rightsquigarrow K \mid TS\ \&\ \vec{x} \mapsto \vec{X} * VS >\mid \varphi \wedge b(\vec{X}) = true$$
$$\rightarrow^{\circledast} < K \mid TS\ \&\ \vec{x} \mapsto \vec{X}' * VS >\mid \psi$$
$$< stmt'\ \rightsquigarrow K \mid TS\ \&\ \vec{x} \mapsto \vec{X} * VS >\mid \varphi \wedge b(\vec{X}) = false$$
$$\rightarrow^{\circledast} < K \mid TS\ \&\ \vec{x} \mapsto \vec{X}' * VS >\mid \psi.$$

Proving Properties about Conditional Statements

Suppose we want to prove a reachability formula of the form:

$$< \textbf{if}\ (b{:}(\vec{x}))\ stmt\ \textbf{else}\ stmt' \rightsquigarrow K \mid TS\ \&\ \vec{x} \mapsto \vec{X} * VS >\mid \varphi$$
$$\rightarrow^{\circledast} < K \mid TS\ \&\ \vec{x} \mapsto \vec{X}' * VS >\mid \psi$$

What can we do? We can apply the **Split** rule to reduce our original goal to the two simpler goals semantically equivalent to it:

$$< stmt\ \rightsquigarrow K \mid TS\ \&\ \vec{x} \mapsto \vec{X} * VS >\mid \varphi \wedge b(\vec{X}) = true$$
$$\rightarrow^{\circledast} < K \mid TS\ \&\ \vec{x} \mapsto \vec{X}' * VS >\mid \psi$$
$$< stmt'\ \rightsquigarrow K \mid TS\ \&\ \vec{x} \mapsto \vec{X} * VS >\mid \varphi \wedge b(\vec{X}) = false$$
$$\rightarrow^{\circledast} < K \mid TS\ \&\ \vec{x} \mapsto \vec{X}' * VS >\mid \psi.$$

This application is correct because the equational theory $(\Sigma_D, E_D \cup B_D)$ of the data type where expressions are evaluated in the IMPL semantics protects the Booleans.

Proving Properties about Conditional Statements

Suppose we want to prove a reachability formula of the form:

$$< \textbf{if}\ (b{:}(\vec{x}))\ stmt\ \textbf{else}\ stmt' \rightsquigarrow K \mid TS\ \&\ \vec{x} \mapsto \vec{X} * VS >\mid \varphi$$
$$\rightarrow^{\circledast} < K \mid TS\ \&\ \vec{x} \mapsto \vec{X}' * VS >\mid \psi$$

What can we do? We can apply the **Split** rule to reduce our original goal to the two simpler goals semantically equivalent to it:

$$< stmt\ \rightsquigarrow K \mid TS\ \&\ \vec{x} \mapsto \vec{X} * VS >\mid \varphi \wedge b(\vec{X}) = true$$
$$\rightarrow^{\circledast} < K \mid TS\ \&\ \vec{x} \mapsto \vec{X}' * VS >\mid \psi$$
$$< stmt'\ \rightsquigarrow K \mid TS\ \&\ \vec{x} \mapsto \vec{X} * VS >\mid \varphi \wedge b(\vec{X}) = false$$
$$\rightarrow^{\circledast} < K \mid TS\ \&\ \vec{x} \mapsto \vec{X}' * VS >\mid \psi.$$

This application is correct because the equational theory $(\Sigma_D, E_D \cup B_D)$ of the data type where expressions are evaluated in the IMPL semantics protects the Booleans. Therefore we have: $\mathcal{T}_{\Sigma_D/E_D \cup B_D} \models b(\vec{X}) = true \vee b(\vec{X}) = false$,

Proving Properties about Conditional Statements

Suppose we want to prove a reachability formula of the form:

$$< \textbf{if } (b\colon(\vec{x}))\ stmt \textbf{ else } stmt' \rightsquigarrow K \mid TS \,\&\, \vec{x} \mapsto \vec{X} * VS >\mid \varphi$$

$$\rightarrow^{\circledast} < K \mid TS \,\&\, \vec{x} \mapsto \vec{X}' * VS >\mid \psi$$

What can we do? We can apply the **Split** rule to reduce our original goal to the two simpler goals semantically equivalent to it:

$$< stmt \rightsquigarrow K \mid TS \,\&\, \vec{x} \mapsto \vec{X} * VS >\mid \varphi \wedge b(\vec{X}) = true$$

$$\rightarrow^{\circledast} < K \mid TS \,\&\, \vec{x} \mapsto \vec{X}' * VS >\mid \psi$$

$$< stmt' \rightsquigarrow K \mid TS \,\&\, \vec{x} \mapsto \vec{X} * VS >\mid \varphi \wedge b(\vec{X}) = false$$

$$\rightarrow^{\circledast} < K \mid TS \,\&\, \vec{x} \mapsto \vec{X}' * VS >\mid \psi.$$

This application is correct because the equational theory $(\Sigma_D, E_D \cup B_D)$ of the data type where expressions are evaluated in the IMPL semantics protects the Booleans. Therefore we have: $\mathcal{T}_{\Sigma_D/E_D \cup B_D} \models b(\vec{X}) = true \vee b(\vec{X}) = false$, as well as the isomorphism: $\mathcal{T}_{\Sigma_{IMPL}/E_{IMPL} \cup B}|_{\Sigma_D} \cong \mathcal{T}_{\Sigma_D/E_D \cup B_D}$.

Proving Properties of Sequential Compositions

Suppose we want to prove a reachability formula $A[stmt\ stmt' \rightsquigarrow K] \rightarrow^{\circledast} C[K]$ parametric on K and $\vec{Y}$ about a sequential composition. What can we do?

Proving Properties of Sequential Compositions

Suppose we want to prove a reachability formula $A[stmt\ stmt' \rightsquigarrow K] \rightarrow^{\circledast} C[K]$ parametric on K and $\vec{Y}$ about a sequential composition. What can we do? Assuming $\vec{x}$ are the program variables of $stmt\ stmt'$,

Proving Properties of Sequential Compositions

Suppose we want to prove a reachability formula $A[stmt\ stmt' \rightsquigarrow K] \rightarrow^{\circledast} C[K]$ parametric on K and $\vec{Y}$ about a sequential composition. What can we do? Assuming $\vec{x}$ are the program variables of $stmt\ stmt'$, If we can prove:

Proving Properties of Sequential Compositions

Suppose we want to prove a reachability formula $A[stmt\ stmt' \rightsquigarrow K] \rightarrow^{\circledast} C[K]$ parametric on K and $\vec{Y}$ about a sequential composition. What can we do? Assuming $\vec{x}$ are the program variables of $stmt\ stmt'$, If we can prove:

$$< stmt \rightsquigarrow K' \mid TS \ \&\ \vec{x} \mapsto \vec{X} * VS > \varphi_1 \rightarrow^{\circledast} < K' \mid TS \ \&\ \vec{x} \mapsto \vec{X'} * VS > \varphi_2$$

Proving Properties of Sequential Compositions

Suppose we want to prove a reachability formula $A[stmt\ stmt' \rightsquigarrow K] \rightarrow^{\circledast} C[K]$ parametric on K and $\vec{Y}$ about a sequential composition. What can we do? Assuming $\vec{x}$ are the program variables of $stmt\ stmt'$, If we can prove:

$$< stmt \rightsquigarrow K' \mid TS \ \& \ \vec{x} \mapsto \vec{X} * VS > \varphi_1 \rightarrow^{\circledast} < K' \mid TS \ \& \ \vec{x} \mapsto \vec{X}' * VS > \varphi_2$$

parametric on K' and $\vec{Y}$,

Proving Properties of Sequential Compositions

Suppose we want to prove a reachability formula $A[stmt\ stmt' \rightsquigarrow K] \rightarrow^{\circledast} C[K]$ parametric on K and $\vec{Y}$ about a sequential composition. What can we do? Assuming $\vec{x}$ are the program variables of $stmt\ stmt'$, If we can prove:

$$< stmt \rightsquigarrow K' \mid TS \ \& \ \vec{x} \mapsto \vec{X} * VS > \varphi_1 \rightarrow^{\circledast} < K' \mid TS \ \& \ \vec{x} \mapsto \vec{X}' * VS > \varphi_2$$

parametric on K' and $\vec{Y}$, whose precondition is $A[stmt \ \rightsquigarrow K]$,

Proving Properties of Sequential Compositions

Suppose we want to prove a reachability formula $A[stmt\ stmt' \leadsto K] \rightarrow^{\circledast} C[K]$ parametric on K and $\vec{Y}$ about a sequential composition. What can we do? Assuming $\vec{x}$ are the program variables of $stmt\ stmt'$, If we can prove:

$$< stmt \leadsto K' \mid TS \ \& \ \vec{x} \mapsto \vec{X} * VS > \varphi_1 \rightarrow^{\circledast} < K' \mid TS \ \& \ \vec{x} \mapsto \vec{X'} * VS > \varphi_2$$

parametric on K' and $\vec{Y}$, whose precondition is $A[stmt \ \leadsto K]$, and we then prove:

Proving Properties of Sequential Compositions

Suppose we want to prove a reachability formula $A[stmt\ stmt' \leadsto K] \to^{\circledast} C[K]$ parametric on K and $\vec{Y}$ about a sequential composition. What can we do? Assuming $\vec{x}$ are the program variables of $stmt\ stmt'$, If we can prove:

$$< stmt \leadsto K' \mid TS \ \&\ \vec{x} \mapsto \vec{X} * VS > \varphi_1 \to^{\circledast} < K' \mid TS \ \&\ \vec{x} \mapsto \vec{X'} * VS > \varphi_2$$

parametric on K' and $\vec{Y}$, whose precondition is $A[stmt\ \leadsto K]$, and we then prove:

$$< stmt' \leadsto K \mid TS \ \&\ \vec{x} \mapsto \vec{X'} * VS > \varphi_2 \to^{\circledast} < K \mid TS \ \&\ \vec{x} \mapsto \vec{X''} * VS > \varphi_3$$

Proving Properties of Sequential Compositions

Suppose we want to prove a reachability formula $A[stmt\ stmt' \rightsquigarrow K] \rightarrow^{\circledast} C[K]$ parametric on K and $\vec{Y}$ about a sequential composition. What can we do? Assuming $\vec{x}$ are the program variables of $stmt\ stmt'$, If we can prove:

$$< stmt \rightsquigarrow K' \mid TS\ \&\ \vec{x} \mapsto \vec{X} * VS > \varphi_1 \rightarrow^{\circledast} < K' \mid TS\ \&\ \vec{x} \mapsto \vec{X'} * VS > \varphi_2$$

parametric on K' and $\vec{Y}$, whose precondition is $A[stmt\ \rightsquigarrow K]$, and we then prove:

$$< stmt' \rightsquigarrow K \mid TS\ \&\ \vec{x} \mapsto \vec{X'} * VS > \varphi_2 \rightarrow^{\circledast} < K \mid TS\ \&\ \vec{x} \mapsto \vec{X''} * VS > \varphi_3$$

parametric on K and $\vec{Y}$,

Proving Properties of Sequential Compositions

Suppose we want to prove a reachability formula $A[stmt\ stmt' \rightsquigarrow K] \rightarrow^{\circledast} C[K]$ parametric on K and $\vec{Y}$ about a sequential composition. What can we do? Assuming $\vec{x}$ are the program variables of $stmt\ stmt'$, If we can prove:

$$< stmt \rightsquigarrow K' \mid TS \ \& \ \vec{x} \mapsto \vec{X} * VS > \varphi_1 \rightarrow^{\circledast} < K' \mid TS \ \& \ \vec{x} \mapsto \vec{X}' * VS > \varphi_2$$

parametric on K' and $\vec{Y}$, whose precondition is $A[stmt \ \rightsquigarrow K]$, and we then prove:

$$< stmt' \rightsquigarrow K \mid TS \ \& \ \vec{x} \mapsto \vec{X}' * VS > \varphi_2 \rightarrow^{\circledast} < K \mid TS \ \& \ \vec{x} \mapsto \vec{X}'' * VS > \varphi_3$$

parametric on K and $\vec{Y}$, whose midcondition is $C[K]$,

Proving Properties of Sequential Compositions

Suppose we want to prove a reachability formula $A[stmt\ stmt' \leadsto K] \rightarrow^{\circledast} C[K]$ parametric on K and $\vec{Y}$ about a sequential composition. What can we do? Assuming $\vec{x}$ are the program variables of $stmt\ stmt'$, If we can prove:

$$< stmt \leadsto K' \mid TS \ \&\ \vec{x} \mapsto \vec{X} * VS > \varphi_1 \rightarrow^{\circledast} < K' \mid TS \ \&\ \vec{x} \mapsto \vec{X'} * VS > \varphi_2$$

parametric on K' and $\vec{Y}$, whose precondition is $A[stmt\ \leadsto K]$, and we then prove:

$$< stmt' \leadsto K \mid TS \ \&\ \vec{x} \mapsto \vec{X'} * VS > \varphi_2 \rightarrow^{\circledast} < K \mid TS \ \&\ \vec{x} \mapsto \vec{X''} * VS > \varphi_3$$

parametric on K and $\vec{Y}$, whose midcondition is $C[K]$, then, thanks to the **Chain Rule**, we have proved our original goal, i.e.,

Proving Properties of Sequential Compositions

Suppose we want to prove a reachability formula $A[stmt\ stmt' \rightsquigarrow K] \rightarrow^{\circledast} C[K]$ parametric on K and $\vec{Y}$ about a sequential composition. What can we do? Assuming $\vec{x}$ are the program variables of $stmt\ stmt'$, If we can prove:

$$< stmt \rightsquigarrow K' \mid TS \ \&\ \vec{x} \mapsto \vec{X}{*}VS > \varphi_1 \rightarrow^{\circledast} < K' \mid TS \ \&\ \vec{x} \mapsto \vec{X'}{*}VS > \varphi_2$$

parametric on K' and $\vec{Y}$, whose precondition is $A[stmt \ \rightsquigarrow K]$, and we then prove:

$$< stmt' \rightsquigarrow K \mid TS \ \&\ \vec{x} \mapsto \vec{X'}{*}VS > \varphi_2 \rightarrow^{\circledast} < K \mid TS \ \&\ \vec{x} \mapsto \vec{X''}{*}VS > \varphi_3$$

parametric on K and $\vec{Y}$, whose midcondition is $C[K]$, then, thanks to the **Chain Rule**, we have proved our original goal, i.e.,

$$< stmt\ stmt' \rightsquigarrow K \mid TS \ \&\ \vec{x} \mapsto \vec{X}{*}VS > \varphi_1 \rightarrow^{\circledast} < K \mid TS \ \&\ \vec{x} \mapsto \vec{X''}{*}VS > \varphi_3$$

Proving Properties of Sequential Compositions

Suppose we want to prove a reachability formula $A[stmt\ stmt' \rightsquigarrow K] \rightarrow^{\circledast} C[K]$ parametric on K and $\vec{Y}$ about a sequential composition. What can we do? Assuming $\vec{x}$ are the program variables of $stmt\ stmt'$, If we can prove:

$$< stmt \rightsquigarrow K' \mid TS \ \&\ \vec{x} \mapsto \vec{X} * VS > \varphi_1 \rightarrow^{\circledast} < K' \mid TS \ \&\ \vec{x} \mapsto \vec{X'} * VS > \varphi_2$$

parametric on K' and $\vec{Y}$, whose precondition is $A[stmt \rightsquigarrow K]$, and we then prove:

$$< stmt' \rightsquigarrow K \mid TS \ \&\ \vec{x} \mapsto \vec{X'} * VS > \varphi_2 \rightarrow^{\circledast} < K \mid TS \ \&\ \vec{x} \mapsto \vec{X''} * VS > \varphi_3$$

parametric on K and $\vec{Y}$, whose midcondition is $C[K]$, then, thanks to the **Chain Rule**, we have proved our original goal, i.e.,

$$< stmt\ stmt' \rightsquigarrow K \mid TS \ \&\ \vec{x} \mapsto \vec{X} * VS > \varphi_1 \rightarrow^{\circledast} < K \mid TS \ \&\ \vec{x} \mapsto \vec{X''} * VS > \varphi_3$$

parametric on K and $\vec{Y}$.

Proving Properties of Sequential Compositions

Suppose we want to prove a reachability formula $A[stmt\ stmt' \rightsquigarrow K] \rightarrow^{\circledast} C[K]$ parametric on K and $\vec{Y}$ about a sequential composition. What can we do? Assuming $\vec{x}$ are the program variables of $stmt\ stmt'$, If we can prove:

$$< stmt \rightsquigarrow K' \mid TS \ \& \ \vec{x} \mapsto \vec{X} * VS > \varphi_1 \rightarrow^{\circledast} < K' \mid TS \ \& \ \vec{x} \mapsto \vec{X'} * VS > \varphi_2$$

parametric on K' and $\vec{Y}$, whose precondition is $A[stmt \ \rightsquigarrow K]$, and we then prove:

$$< stmt' \rightsquigarrow K \mid TS \ \& \ \vec{x} \mapsto \vec{X'} * VS > \varphi_2 \rightarrow^{\circledast} < K \mid TS \ \& \ \vec{x} \mapsto \vec{X''} * VS > \varphi_3$$

parametric on K and $\vec{Y}$, whose midcondition is $C[K]$, then, thanks to the **Chain Rule**, we have proved our original goal, i.e.,

$$< stmt\ stmt' \rightsquigarrow K \mid TS \ \& \ \vec{x} \mapsto \vec{X} * VS > \varphi_1 \rightarrow^{\circledast} < K \mid TS \ \& \ \vec{x} \mapsto \vec{X''} * VS > \varphi_3$$

parametric on K and $\vec{Y}$. How so?

Proving Properties of Sequential Compositions

Suppose we want to prove a reachability formula $A[stmt\ stmt' \rightsquigarrow K] \rightarrow^{\circledast} C[K]$ parametric on K and $\vec{Y}$ about a sequential composition. What can we do? Assuming $\vec{x}$ are the program variables of $stmt\ stmt'$, If we can prove:

$$< stmt \rightsquigarrow K' \mid TS \ \& \ \vec{x} \mapsto \vec{X}{*}VS > \varphi_1 \rightarrow^{\circledast} < K' \mid TS \ \& \ \vec{x} \mapsto \vec{X'}{*}VS > \varphi_2$$

parametric on K' and $\vec{Y}$, whose precondition is $A[stmt \ \rightsquigarrow K]$, and we then prove:

$$< stmt' \rightsquigarrow K \mid TS \ \& \ \vec{x} \mapsto \vec{X'}{*}VS > \varphi_2 \rightarrow^{\circledast} < K \mid TS \ \& \ \vec{x} \mapsto \vec{X''}{*}VS > \varphi_3$$

parametric on K and $\vec{Y}$, whose midcondition is $C[K]$, then, thanks to the **Chain Rule**, we have proved our original goal, i.e.,

$$< stmt\ stmt' \rightsquigarrow K \mid TS \ \& \ \vec{x} \mapsto \vec{X}{*}VS > \varphi_1 \rightarrow^{\circledast} < K \mid TS \ \& \ \vec{x} \mapsto \vec{X''}{*}VS > \varphi_3$$

parametric on K and $\vec{Y}$. How so? Just by first applying to the first formula the constructor substitution $\{K' \mapsto stmt' \rightsquigarrow K\}$,

Proving Properties of Sequential Compositions

Suppose we want to prove a reachability formula $A[stmt\ stmt' \rightsquigarrow K] \rightarrow^{\circledast} C[K]$ parametric on K and $\vec{Y}$ about a sequential composition. What can we do? Assuming $\vec{x}$ are the program variables of $stmt\ stmt'$, If we can prove:

$$< stmt \rightsquigarrow K' \mid TS \ \&\ \vec{x} \mapsto \vec{X} * VS > \varphi_1 \rightarrow^{\circledast} < K' \mid TS \ \&\ \vec{x} \mapsto \vec{X}' * VS > \varphi_2$$

parametric on K' and $\vec{Y}$, whose precondition is $A[stmt\ \rightsquigarrow K]$, and we then prove:

$$< stmt' \rightsquigarrow K \mid TS \ \&\ \vec{x} \mapsto \vec{X}' * VS > \varphi_2 \rightarrow^{\circledast} < K \mid TS \ \&\ \vec{x} \mapsto \vec{X}'' * VS > \varphi_3$$

parametric on K and $\vec{Y}$, whose midcondition is $C[K]$, then, thanks to the **Chain Rule**, we have proved our original goal, i.e.,

$$< stmt\ stmt' \rightsquigarrow K \mid TS \ \&\ \vec{x} \mapsto \vec{X} * VS > \varphi_1 \rightarrow^{\circledast} < K \mid TS \ \&\ \vec{x} \mapsto \vec{X}'' * VS > \varphi_3$$

parametric on K and $\vec{Y}$. How so? Just by first applying to the first formula the constructor substitution $\{K' \mapsto stmt' \rightsquigarrow K\}$,and then applying the **Chain Rule**.

Acknowledgements

The program proving methodology presented in this lecture has been developed in joint work with Michael Abir. A more detailed document containing all the details of this proof methodology is in preparation.