Program verification involves 3 tasks

1) Specification
   1) for each function that constitutes the interface
   2) for each helper function
   3) for each loop

- Specification is hard

- 2 and 3 are considered onerous requirements while 1 is generally thought to be good practice

- In some cases the specification is available implicitly
  - No integer computations should overflow
  - No array access should be out-of-bounds
  - No uninitialized var. are used
  - No NIL pointer is dereferenced
  - No dangling pointers are created
  - No garbage memory cells are created
Task b) verification condition generation
Task c) theorem proving
b) is easy if all loops are annotated with invariants
c) we have looked at.

Question today: how to create loop invariants automatically?
- can only hope to have some heuristics

Strategy A
- use only sound heuristics
  all invariants discovered are indeed invariants

Strategy B
- use potentially unsound heuristics as well
- use VCGen + ThProver to check the candidate invariants

Problem: ThProver is incomplete so we don't know if the invariant is not good or the prover is unable to prove it. Nevertheless this can lead to some good invariants.
One formulation of the invariant issue:

- a cutpoint is a point in the program
- there is a cutpoint right before each return
- there is at least a cutpoint on each circular path through the program
- we associate assertions $I_k$ with cutpoint $P_k$

**Conventions**

- let $\bar{x}$ be the set of variables
- for a path $\alpha$ we define
  - $r_\alpha(\bar{x})$ a function that gives the values of variables at end of path as a function of values at beginning
  - $P_\alpha(\bar{x})$ a predicate that says whether the path $\alpha$ is taken given some values of variables before the path

A set of invariants is correct if

1) for each path $\alpha$ from START to cutpoint $k$ (without crossing other cutpoints)

$$\forall \bar{x}. \ q(\bar{x}) \land P_\alpha(\bar{x}) \Rightarrow I_k(\ r_\alpha(\bar{x}))$$

$q(\bar{x})$ is the precondition
and

b) for each path \( \alpha \) from cutpoint \( i \) to \( j \) we have

\[ \forall x \ I_i(x) \land P_\alpha(x) \Rightarrow I_j(r(x)) \]

a) says that invariants are established initially

b) says that they are maintained.

A simple example

\[ \begin{array}{c}
\text{Need} \\
1. \forall x. \varphi(x) \Rightarrow I_0(r_1(x)) \\
2. \forall x. I_0(x) \land \neg t(x) \Rightarrow I_0(r_2(x)) \\
3. \forall x. I_0(x) \land t(x) \Rightarrow I_1(x) \\
\end{array} \]

These are equations that we have to solve for \( I_0 \) and \( I_1 \).

\( I_1 \) might be given (the postcondition)
Pick a candidate for $I_0$.

- If it is not satisfied then $I_0$ is too strong.
- If 3 is not satisfied then $I_0$ is too weak.
  - It might be an invariant, but it is not a useful one because it is not strong enough to allow us to prove the postcondition.
- If 2 is not satisfied then $I_0$ is just not good.

**Heuristics**

- Start from 3 and set $I_0(x) = t(x) \Rightarrow I_1(x)$
  - This is the weakest solution that satisfies 3.
- Use heuristics to strengthen $I_0$.
  - We do not need to check 3 anymore.
  - We can use 1 as a sanity check.
  - We use 2 to drive the heuristics.

Define a family of predicates.
\[ P^0(x) = t(x) \Rightarrow I_1(x) \]
\[ P^k(x) = P^0(x) \land \neg t(x) \Rightarrow P^{k-1}(\bigcirc r_2(x)) \]

One can prove that

\[ P^k(x) \] is the weakest predicate on \( x \) that ensures that if the execution starts at cutpoint 0 with \( \bar{x} \) and if cutpoint 1 is reached after at most \( k \) iterations then \( I_1(\bar{x}') \) is satisfied.

We can show that \( P^k \) is a chain

\[ P^0_0 \subseteq P^1_1 \subseteq P^2_2 \subseteq \ldots \]

Proof by induction on \( k \).

\[ P^1 \Rightarrow P^0 \]

since \( P^{k-1} \Rightarrow P^{k-2} \) \( k \geq 2 \)

\[ (\neg t(x) \Rightarrow P^{k-1}(x)) \Rightarrow (\neg t(x) \Rightarrow P^{k-2}(x)) \]

and then

\[ P^k(x) \Rightarrow P^{k-1}(x) \]

\( k \geq 2 \)
The weakest "useful" invariant is
\[ \forall k. k \geq 0 \Rightarrow P^k(x) \]
(but this is a strange form of quantification
since \( P^k(x) \) is a family of predicates
more appropriately is
\[ \bigwedge_{k \geq 0} P^k(x) \]
This \( \infty \) conjunction is impossible to calculate
in general.

[Note: all this material was basically
weakest preconditions seen again]

What if we can find a \( k \) such that
\[ \forall k. P^k(x) \Rightarrow P^{k+1}(x) \]
then
\[ \bigwedge_{k \geq 0} P^k(x) = P^k(x) \]

yes, since \( P^k(x) \leftrightarrow P^{k+1}(x) \)

This method is called induction iteration
Norihisa Fujikie, Kiyoshi Ishihata
"Implementation of an Array Bound
Checker", 4th POPL (1977)
A slight improvement

- The predicates $P_k$ becomes more complex
- It becomes more and more complicated to verify that $P_k \Rightarrow P_{k+1}$
- So we simplify $P_k$ before using it to compute the $P_{k+1}$
- We can also strengthen $P_k$

$$R^0(x) = t(x) \Rightarrow I_1(x)$$

$$R^k(x) = R^0(x) \land \neg t(x) \Rightarrow \text{strengthen } (R^{k-1}(x_2(x)))$$

If $\text{strengthen } (R) \Rightarrow R$ for all $R$
then we still have

$$R^k(x) \Rightarrow P^k(x)$$

- We are looking the "weakest" property
- And the completeness

Typical strengthening heuristics

- Drop disjuncts

$$\text{str } (t(x) \Rightarrow I_1(x)) = I_1(x)$$
Integer heuristics

- Try to eliminate from $P^k$ variables that are modified in the loop (by $r_2(x)$)
- As an extreme case if $P^k$ refers only to variables that are not modified by $r_2(x)$
then $P^k(r_2(x)) = P^k$

also $P^k \Rightarrow (\neg t(x) \Rightarrow P^k)$

$P^k \Rightarrow P^0$

thus $P^k \Rightarrow P^{k+1}$ bingo!

- One way to eliminate vars is using Fourier-Motzkin elimination if
  the predicates are linear inequalities and/or equalities
Example. Consider the program

\[ \begin{align*}
& l \leftarrow \text{len}(a) \\
& i \leftarrow 0 \\
& s \leftarrow 0 \\
& * \text{endpoint} \\
& s \leftarrow s + a[i] \\
& i \leftarrow i + 1 \\
& \text{if } i < l \\
& \text{then } s \leftarrow s + a[i] \\
& \text{else return } s
\end{align*} \]

- take the implicit assertion that at endpoint
  the invariant must imply:
  \[ \begin{align*}
  & i \geq 0 \\
  & i < \text{len}(a)
\end{align*} \]

- so we start with

\[ P^0 = i < \text{len}(a) \rightarrow \text{not a tautology} \]

\[ P' = P^0 \land (i+1 < l \Rightarrow i+1 < \text{len}(a)) \]

check \[ P^0 \Rightarrow (P^0 \land (i+1 < l \Rightarrow i+1 < \text{len}(a)) \]

need to show

\[ i < \text{len}(a) \land i+1 < l \Rightarrow i+1 < \text{len}(a) \]

cannot show this.
we can strengthen
\[ i + 1 < l \Rightarrow i + 1 < \text{len}(a) \]

...we negate again \[ l \leq \text{len}(a) \]
we have \[ l \leq \text{len}(a) \Rightarrow (i + 1 < l \Rightarrow i + 1 < \text{len}(a)) \]
\[ l \leq \text{len}(a) \] is stronger

\[ P^2 = P^0 \land (i + 1 < l \Rightarrow l \leq \text{len}(a)) \]

Now we can show that \[ P^1 \Rightarrow P^2 \], thus \( P^1 \) is an invariant

and is useful

We now check condition 1. by backpropagating
\[ P^1 = i < \text{len}(a) \land l \leq \text{len}(a) \] to the START
check \( 0 < \text{len}(a) \land \text{len}(a) \leq \text{len}(a) \)

Need precondition \( 0 < \text{len}(a) \)