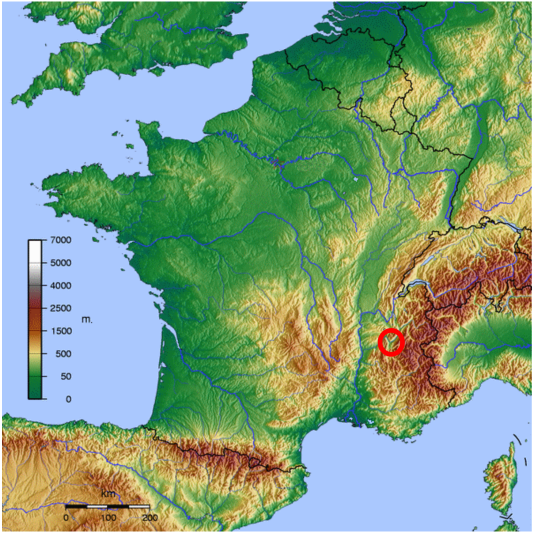
Proving and inferring invariants

David Monniaux

CNRS / VERIMAG
Grenoble, France

December 13, 2013

Grenoble



VERIMAG

VERIMAG is a joint research laboratory of CNRS, Université Joseph Fourier (Grenoble-1) and Grenoble-INP

Plan

Safety properties

Proving properties of programs :

- **safety** : the program never enters an undesirable state (crash, variable too large for specification, assertion violation. . .)
- **liveness** : the program progresses (no entering into deadlocks or neverending loops)

Safety properties

Proving properties of programs :

- **safety** : the program never enters an undesirable state (crash, variable too large for specification, assertion violation. . .)
- **liveness** : the program progresses (no entering into deadlocks or neverending loops)

In this talk, focus on **safety** (liveness often uses safety properties).

Proofs on programs

A program written in a real programmming language
$\Downarrow$
Its **semantics**: its “meaning” in mathematical terms

Proofs on programs

A program written in a real programmming language
$\Downarrow$
Its **semantics**: its "meaning" in mathematical terms

For real languages (C, C++, PHP...), very **difficult** and fraught with errors.
We'll bravely assume the problem solved and suppose a toy language with well-defined mathematical semantics.

Properties to prove

A property in natural language (e.g. "the program sorts the array")
$\Downarrow$
A mathematical property (e.g. definition of the total order on array elements, the output is sorted, it is a permutation of the input...)

Properties to prove

A property in natural language (e.g. "the program sorts the array")
$\Downarrow$
A mathematical property (e.g. definition of the total order on array elements, the output is sorted, it is a permutation of the input...)

Again, fraught with errors.
We'll bravely assume mathematically defined properties.

The setting

A set $\mathcal{C}$ of **control points**:

- instructions
- heads of control blocks
- lines of program

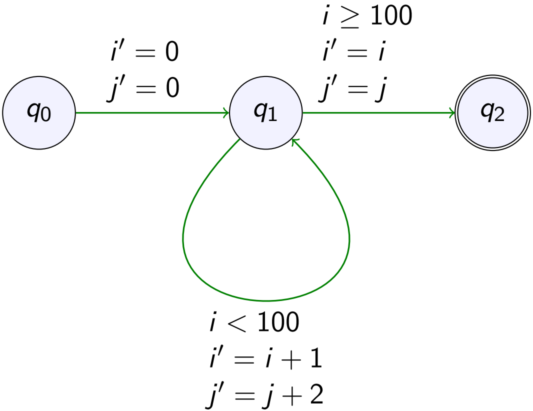
Memory state as a vector of variables in $\mathcal{S}$ (can be $\mathbb{Z}^n$ (or $\mathbb{Q}^n$, or $\mathbb{B}^m \times \mathbb{Q}^n$ where $\mathbb{B} = \{0, 1\}$ Booleans)

For $i, j \in \mathcal{C}$, a transition relation $\tau_{i,j} \subseteq \mathcal{S} \times \mathcal{S}$ (often expressed with $x, y, \ldots$ variables before and $x', y', \ldots$ after)

A starting state $q_0 \in \mathcal{C}$ and a "bad" state $q_B \in \mathcal{C}$.

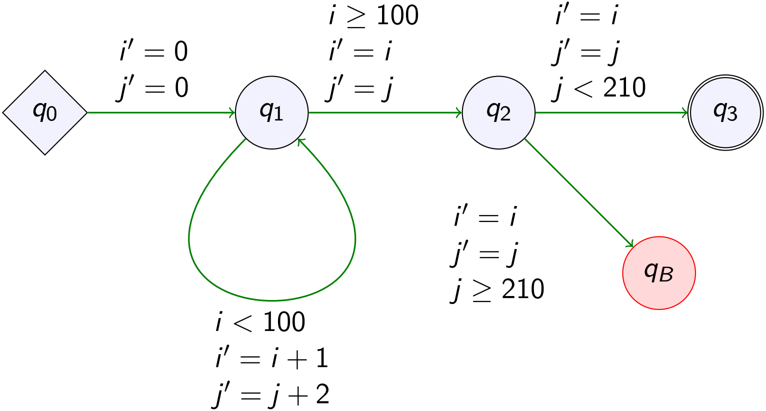
Concrete example

```
j = 0;
for(int i=0; i<100; i++) {
  j = j+2;
}
```



Concrete example with an assertion

```
j = 0;
for(int i=0; i<100; i++) {
  j = j+2;
}
assert(j < 210);
```



Proving safety

Whether q_B is reachable...

Proving safety

Whether q_B is reachable...
Is an undecidable problem (**halting problem**)

Plan

Floyd-Hoare-like proofs

(Ideas dating back to at least Robert Floyd and C.A.R Hoare, late 1960s, and even to Turing):

- Adorn each state q_i in the automaton with a formula ϕ_i
- Show that these formulas are **inductive**: if $\phi_i(\mathbf{x})$ and $\tau_{i,j}(\mathbf{x}, \mathbf{x}')$ then $\phi_j(\mathbf{x})$
- Check that the formula ϕ_0 for q_0 (initial state) is "true"
- Check that the formula ϕ_B for q_B (bad state) is "false"

Floyd-Hoare-like proofs

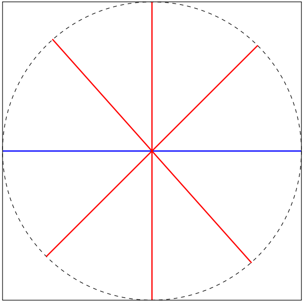
(Ideas dating back to at least Robert Floyd and C.A.R Hoare, late 1960s, and even to Turing):

- Adorn each state q_i in the automaton with a formula ϕ_i
- Show that these formulas are **inductive**: if $\phi_i(\mathbf{x})$ and $\tau_{i,j}(\mathbf{x}, \mathbf{x}')$ then $\phi_j(\mathbf{x})$
- Check that the formula ϕ_0 for q_0 (initial state) is "true"
- Check that the formula ϕ_B for q_B (bad state) is "false"

By induction on the length of the computation, the system state $(c, \mathbf{x}) \in \mathcal{S} \times \mathcal{S}$ can never exit the ϕ_i "invariant":
For any reachable $(c, \mathbf{x})$, $\mathbf{x}$ satifies ϕ_c.

Direct induction does not necessarily work

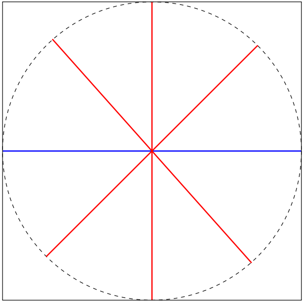
Program initialization: $-1 \leq x \leq 1 \wedge y = 0$
Operation: $(x', y') = \mathit{rotate}((x, y), 45)$



$-1 \leq x \leq 1 \wedge -1 \leq y \leq 1$ is always true...

Direct induction does not necessarily work

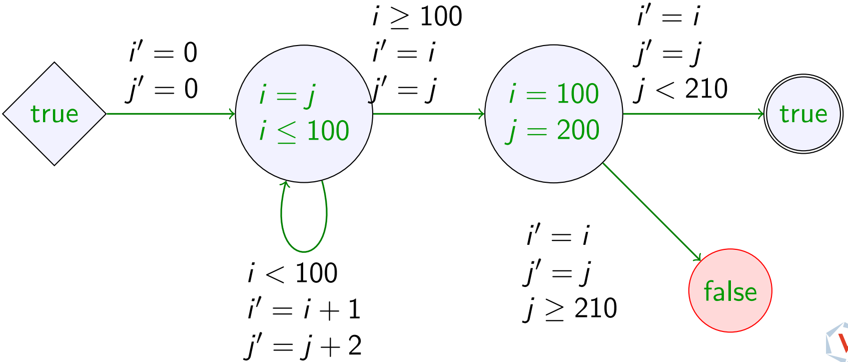
Program initialization: $-1 \leq x \leq 1 \wedge y = 0$
Operation: $(x', y') = \mathit{rotate}((x, y), 45)$



$-1 \leq x \leq 1 \wedge -1 \leq y \leq 1$ is always true...
But not by induction! Need some **stronger** inductive property e.g. $x^2 + y^2 \leq 1$.

With invariants

```
j = 0;
for(int i=0; i<100; i++) {
  j = j+2;
}
assert(j < 210);
```



Checking inductive invariants

A tool requires the user to provide invariants, and **checks** that they are inductive.

Possible if the invariants ϕ_i and the transition relations $\tau_{i,j}$ are within a **decidable theory**:

Check that $\phi_i \wedge \tau_{i,j} \wedge \neg\phi_j$ is **unsatisfiable** for all i, j.

Various degrees of automation

Tools : Frama-C, Why, B-Method, Frama-C...

Inferring inductive invariants

More ambitious: complete automation!

The problem:

- exhibit ϕ_c at all control state $c \in \mathcal{C}$
- so that the ϕ_c are inductive
- and ϕ_0 is "true" and ϕ_B is "false"

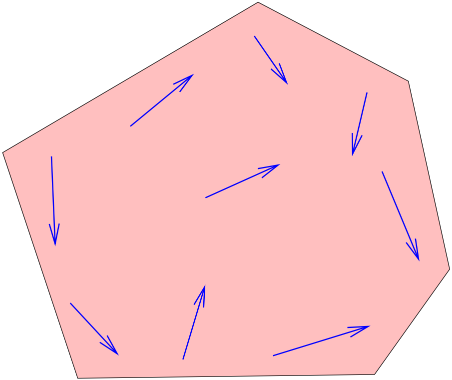
But what is ϕ_c? An arbitrary first-order formula?

Abstract domains

So as to automatize the task: look for ϕ_c in a particular class (or **domain**) of properties: e.g.

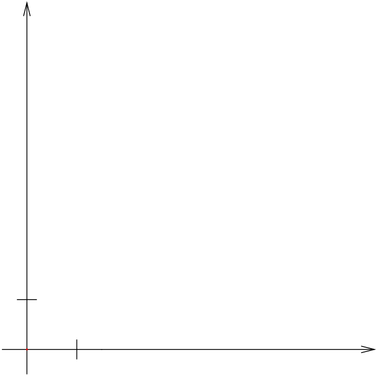
- propositional formulas over the Boolean variables
- conjunctions of linear inequalities over rational/integer variables (**convex polyhedra**)
- **intervals** over rational/integer variables

Example of an inductive polyhedron



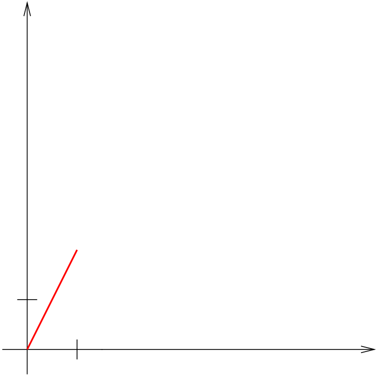
Abstract interpretation in convex polyhedra

```
j = 0;
for(int i=0; i<100; i++) {
  j = j+2;
}
```



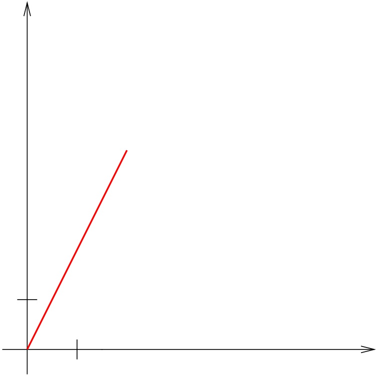
Abstract interpretation in convex polyhedra

```
j = 0;
for(int i=0; i<100; i++) {
  j = j+2;
}
```



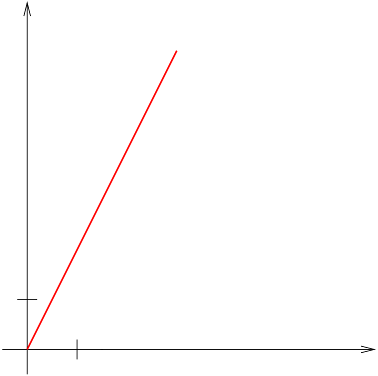
Abstract interpretation in convex polyhedra

```
j = 0;
for(int i=0; i<100; i++) {
  j = j+2;
}
```



Abstract interpretation in convex polyhedra

```
j = 0;
for(int i=0; i<100; i++) {
  j = j+2;
}
```



Idea

(Cousot / Halbwachs, 1978)

- All ϕ_c are (possibly empty) convex polyhedra (conjunctions of linear inequalities)
- "Push" these polyhedra through control edges: compute the image (or over-approximation of image) of the polyhedron by the edge, add (convex hull) to target polyhedron
- Stop when inductive (saturation: no edge modifies the target polyhedron)
- Check that ϕ_B is an empty polyhedron

Is termination guaranteed?

Slow termination

```
j = 0;
for(int i=0; i<100; i++) {
  j = j+2;
}
```

With the above method, needs 100 iterations. Still tolerable... but what if it had been 10^9?

Widenings

Iteration 0: $i = 0 \wedge j = 0$
Iteration 1: $0 \leq i \leq 1 \wedge j = 2i$
Iteration 2: $0 \leq i \leq 2 \wedge j = 2i$
Iteration 3: $0 \leq i \leq 3 \wedge j = 2i$
$\vdots$

Widenings

Iteration 0: $i = 0 \wedge j = 0$
Iteration 1: $0 \leq i \leq 1 \wedge j = 2i$
Iteration 2: $0 \leq i \leq 2 \wedge j = 2i$
Iteration 3: $0 \leq i \leq 3 \wedge j = 2i$
$\vdots$
Widen (**extrapolate**) to $0 \leq i \wedge j = 2i$
Is it inductive?

Widenings

Iteration 0: $i = 0 \wedge j = 0$
Iteration 1: $0 \leq i \leq 1 \wedge j = 2i$
Iteration 2: $0 \leq i \leq 2 \wedge j = 2i$
Iteration 3: $0 \leq i \leq 3 \wedge j = 2i$
$\vdots$
Widen (**extrapolate**) to $0 \leq i \wedge j = 2i$
Is it inductive?
YES! WE WON!

Widenings

Iteration 0: $i = 0 \wedge j = 0$
Iteration 1: $0 \leq i \leq 1 \wedge j = 2i$
Iteration 2: $0 \leq i \leq 2 \wedge j = 2i$
Iteration 3: $0 \leq i \leq 3 \wedge j = 2i$
$\vdots$
Widen (**extrapolate**) to $0 \leq i \wedge j = 2i$
Is it inductive?
YES! WE WON!
One can even narrow down (**refine**) to $0 \leq i \leq 100 \wedge j = 2i$.

Problems with widenings and CEGAR

Widenings are **brittle**

- Sometimes (as in this example) they work well
- Sometimes they give very bad invariants (e.g. “true”)
- Sometimes knowing more information on the system leads to worse invariants (non-monotonicity)
- Sometimes they work well on a program and not well on a similar program. . .

Similar problems hold for predicate abstraction with CEGAR (counterexample-guided abstraction refinement) using Craig interpolants.

Gratuitous advertisement: Astrée

Intervals + widenings + "octahedra" + many domain-specific analyses (linear filters, quaternions. . .) =

Astrée static analysis tool used in avionic industry.

Proves the absence of runtime errors and assertion violations.

Capable of analyzing full fly-by-wire control-code, hundreds of kLOC, thousands of variables

with few or none **false alarms** (unproved true properties)

http://www.astree.ens.fr

http://www.absint.com/astree/

An ideal case

What if we could find the **strongest** inductive invariant in the domain? E.g.

- The **smallest** inductive polyhedra (definition problem: does not necessarily exist)
- The **smallest** inductive intervals
- ...

Recall: denoting by the P property to prove, and by I the invariant, we must have $I \Rightarrow P$, so stronger I is better.

An ideal case

What if we could find the **strongest** inductive invariant in the domain? E.g.

- The **smallest** inductive polyhedra (definition problem: does not necessarily exist)
- The **smallest** inductive intervals
- ...

Recall: denoting by the P property to prove, and by I the invariant, we must have $I \Rightarrow P$, so stronger I is better.

Also leads to a **decision problem**: is there an inductive invariant in the chosen domain capable of proving the unreachability of the bad state?

- computability
- complexity

Plan

A very simple loop

```
i=0;
while (i < 100) {
  i=i+1;
}
```

Find an inductive loop invariant as an interval $[-l, h]$:

- $[-l, h]$ must contain the initial state: $l \geq 0$, $h \geq 0$
- $[-l, h]$ must be stable by "pushing the interval through the loop"
  - test maps $[-l, h]$ to $[-l, \min(h, 99)]$
  - then $i = i + 1$ maps $[-l, \min(h, 99)]$ to $[-(l-1), \min(h, 99) + 1]$

  Thus inclusion: $l \geq l - 1$ and $h \geq \min(h, 99) + 1$

Thus the least solution satisfies

- $l = \max(0, l - 1)$
- $h = \max(0, \min(h, 99) + 1)$

How to solve min-max equations

We end with equations with "min", "max", and monotone affine-linear expressions

$$h = \max(0, \min(h, 99) + 1)$$

How to solve them?

How to solve min-max equations

We end with equations with "min", "max", and monotone affine-linear expressions

$$h = \max(0, \min(h, 99) + 1)$$

How to solve them?

Naive approach:

- Enumerate all argument choices for "min" and "max"
- For each choice, compute solution of linear equation system
- Discard if not a solution of the original problem (wrong choices of arguments of "min" and "max")
- Take the least one

Solving the naive way

$$h = \max(0, \min(h, 99) + 1) \tag{1}$$

Turned into 3 different equations:

- $h = \max(\underline{0}, \min(h, 99) + 1) \rightsquigarrow h = 0$ (left-arg to "max"), solution $h = 0$, but not solution of (1): $\max(0, \min(0, 99) + 1)$, the right argument of "max" is greater $\Rightarrow$ discarded
- $h = \max(0, \underline{\min(\underline{h}, 99) + 1}) \rightsquigarrow h = h + 1$ (right-arg to "max", left-arg to "min"), solution $h = +\infty$, but not solution of (1): $\min(+\infty, 99)$, the argument of "min" is smaller $\Rightarrow$ discarded
- $h = \max(0, \underline{\min(h, \underline{99}) + 1}) \rightsquigarrow h = 99 + 1 = 100$ (right-arg to "max", right-arg to "min"), **solution** of the original problem.

But **exponential blowup**.

Min-policy iteration

Only choose for "min":

- $h = \max(0, \min(\underline{h}, 99) + 1) \rightsquigarrow h = \max(0, h + 1) \rightsquigarrow$ find least solution of $h \geq 0 \land h \geq h + 1$ (linear programming) $\rightsquigarrow h = +\infty$
  $\min(+\infty, 99) = 99$, so flip to right argument of "min"
- $h = \max(0, \min(h, \underline{99}) + 1) \rightsquigarrow h = \max(0, 100) \rightsquigarrow$ find least solution of $h \geq 0 \land h \geq 100$ (linear programming) $\rightsquigarrow h = 100$

Solution: $h = 100$
Always the least one?

Min-policy iteration

Only choose for "min":

- $h = \max(0, \min(\underline{h}, 99) + 1) \rightsquigarrow h = \max(0, h + 1) \rightsquigarrow$ find least solution of $h \geq 0 \wedge h \geq h + 1$ (linear programming) $\rightsquigarrow h = +\infty$
  $\min(+\infty, 99) = 99$, so flip to right argument of "min"
- $h = \max(0, \min(h, \underline{99}) + 1) \rightsquigarrow h = \max(0, 100) \rightsquigarrow$ find least solution of $h \geq 0 \wedge h \geq 100$ (linear programming) $\rightsquigarrow h = 100$

Solution: $h = 100$
Always the least one?

In general, the min-policy iteration process may stop on a solution of the system of min-max equation that is not the least one.

Min-policy iteration: explainer

Was introduced into program verification by Éric Goubault's group.

Why "policy"? Because of a similar problem and resolution method in game theory, where the "policy" or "strategy" is how the "min player" plays.

Produces a sequence of systems of max-equations whose solutions form a descending sequence **upper bounds** on the least solution of the original system.
These solutions give **inductive invariants**.
Can stop the descending sequence at any point and still get an inductive invariant!

Min-policy iteration: generalization

Let A_c be a family of constant matrices, find invariants ϕ_c of the form $A_c X \leq B_c$ where X the program variables.

Programs with linear affine assignments, linear affine inequalities in tests. Restrict $\tau_{i,j}$ to $\bigvee \exists \mathbf{y} \bigwedge \mathit{linear} - \mathit{inequality}(\mathbf{x}, \mathbf{x}', \mathbf{y})$

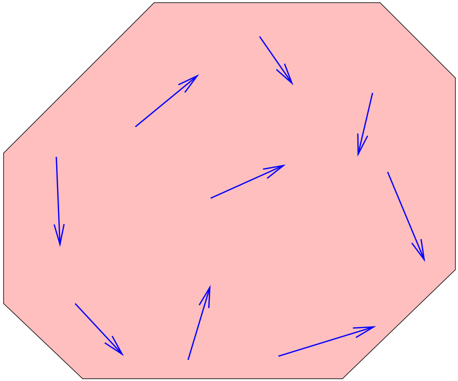
Includes, with appropriate choice for A_c:

- intervals
- "difference bounds": intervals and $x - y \leq b_{x,y}$

How to compute least $B_c = (b_{c,1}, \dots, b_{c,m})$ (coordinate-wise)?

Example of an inductive “octahedron”

Some specific choice for A:



Min-max equations with linear programming

Obtain a system of equations

$$b_{c,i} = \max(LP(\mathbf{b}), \dots, LP(\mathbf{b}))$$

with LP some linear programming problems of the form

$$\sup\{\mathbf{l} \cdot \mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\}$$

Why **min**-max equations?

Min-max equations with linear programming

Obtain a system of equations

$$b_{c,i} = \max(LP(\mathbf{b}), \ldots, LP(\mathbf{b}))$$

with LP some linear programming problems of the form

$$\sup\{\mathbf{l} \cdot \mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\}$$

Why **min**-max equations?

The LP can be rewritten by linear duality into

$$\min(\mathbf{h}_1 \cdot \mathbf{b}, \ldots, \mathbf{h}_N \cdot \mathbf{b})$$

(where N may be exponential in the size of the original problem)

Min-policy iteration: executive summary

1. Start with a problem with explicit or implicit “min” operators in the right-hand side
2. For each $\min(a_1, \ldots, a_n)$, pick an a_i and replace $\min(a_1, \ldots, a_n)$ by a_i in the equation
3. Solve the resulting system (perhaps with overapproximation)
4. For each $\min(a_1, \ldots, a_n)$, check that the value of picked a_i from the solution is really the minimum; if not, change to a_j minimal and go back to point 3
5. Otherwise, terminate (not necessarily with best inductive invariant in domain)

If everything affine linear, each intermediate problem is just **linear programming**.
Each intermediate result is an **inductive invariant**.

Max-policy iteration

(Developed by H. Seidl, T. Gawlitza)

$$h = \max(-\infty, 0, \min(h, 99) + 1)$$

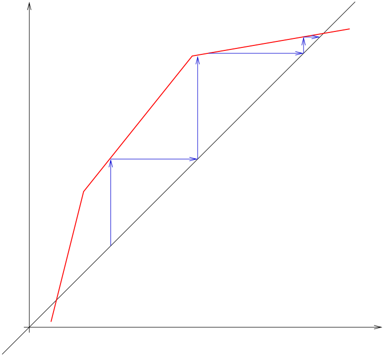
Pick an argument for "max":

- Initial value for $h = -\infty$
- $h = \max(\underline{-\infty}, 0, \min(h, 99) + 1)$; $h = -\infty$; replace: $\max(-\infty, \underline{0}, -\infty)$, found higher argument $h = 0$
- $h = \max(-\infty, \underline{0}, \min(h, 99) + 1)$; $h = 0$; replace: $\max(-\infty, 0, \underline{1})$, found higher argument $h = 1$
- $h = \max(-\infty, 0, \underline{\min(h, 99) + 1})$; solve $h = \min(h, 99) + 1$ for solution $h \geq 1$:
  Solve $h \leq h + 1 \land h \leq 99 + 1$ for maximal finite h: $h = 100$.

High level view

Transforms the original problem (with "max") into a sequence of problems (without "max") with increasing "value".

Intuition: solution is maximum of "order-concave" functions



It's like solving $h = F(h)$ by infinite ascending sequence
$-\infty, F(-\infty), F \circ F(-\infty), F \circ F \circ F(-\infty) \dots$
but taking "big strides"!

Executive summary

- Produces a sequence of problems without “max”
- Continue iterating until an inductive invariant is found
- If everything affine linear, each intermediate problem is just **linear programming**
- Terminates on least (strongest) inductive invariant in domain

Scaling issues

- Currently, does not scale to the kind of large-scale application targeted by e.g. Astrée.
- Complexity upper bound on policy iteration algorithms is **exponential** (two choices per binary "max" or "min", consider all combinations).
- Complexity as a decision problem is unclear (in NP; seems to be in PPAD and PLS?).

Nonlinear stuff

Policy iteration can be adapted to nonlinear problems

- By linearization
- Using semidefinite programming instead of linear programming

(I won't talk about this here.)

Motivation

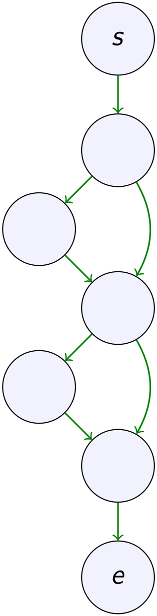
```
void rate_limiter() {
  int x_old = 0;
  while (1) {
    int x = input(-100000, 100000);
    if (x > x_old+10) x = x_old+10;
    if (x < x_old-10) x = x_old-10;
    x_old = x;
} }
```

To analyze this program and get good results

- Consider a single inductive invariant at loop head
- . . . but not at intermediate points inside the loop
- Consider separately paths inside the loop

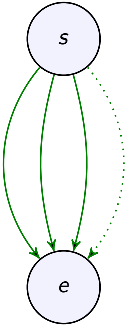
Distinguishing paths

```
void rate_limiter() {
  int x_old = 0;
  while (1) {
    int x = input(-100000, 100000);
    if (x > x_old+10) x = x_old+10;
    if (x < x_old-10) x = x_old-10;
    x_old = x;
} }
```



Distinguishing paths

```
void rate_limiter() {
  int x_old = 0;
  while (1) {
    int x = input(-100000, 100000);
    if (x > x_old+10) x_old = x_old+10;
    else if (x < x_old-10) x_old = x_old-10;
    else x_old = x;
} }
```



Edge-implicit graphs

Instead of considering all program points $\mathcal{C}$ (or heads of blocks), consider a **cut-set** $\mathcal{H}$: set of nodes such that removing them breaks all cycles (like heads of loops).

Edges between nodes in $\mathcal{H}$ are the paths between these nodes in the original graph.

There may be an exponential number of them.

Algorithm for max-policy iteration on edge-implicit graphs

(Gawlitza & Monniaux)

- Invariants of the form $A\mathbf{x} \leq \mathbf{B}$, A fixed matrix, unknown B
- **No exponential expansion**
- Enumerates paths "as needed" using a SMT-solver
- Exponential worst-case complexity
- Decision problem ("is there an invariant in the domain proving the unreachability of the bad state") is Σ_2^p**-complete** (NP-complete with a co-NP-complete oracle)

(Fully implicit graphs, with compact representation of an exponential number of control nodes, in forthcoming Monniaux & Schrammel)

Plan

Linear template

So far we have supposed A fixed, looked for inductive invariants $A\mathbf{x} \leq \mathbf{b}$ such that $b_B = -\infty$ ("bad state is unreachable") and $b_0 = +\infty$ ("starting point" has any value)

What if A is left unknown? (Generic convex polyhedron with fixed number of constraints.)

The unknown template problem

Find A_c and $\mathbf{b}_c$ such that for all c, c':

$$\forall \mathbf{x} \forall x' \forall y \; A_c \mathbf{x} \leq \mathbf{b_c} \wedge D\mathbf{x} + E\mathbf{x}' + F\mathbf{y} \leq \mathbf{g} \Rightarrow A_{c'}\mathbf{x} \leq \mathbf{B_{c'}}$$

(and $b_B = -\infty$ and $b_0 = +\infty$)

If everything is linear, Farkas' lemma enables us to turn the **universal** $\forall \mathbf{x} \forall x' \ldots$ into an **existential** with unknowns Λ, M, $\mathbf{s}$:

$$\begin{aligned} A_{c'} &= \Lambda E \\ M A_c + \Lambda D &= 0 \\ B_{c'} &= \Lambda \mathbf{g} + M \mathbf{b}_c + \mathbf{s} \end{aligned}$$

(and still $b_B = -\infty$ and $b_0 = +\infty$)

Unfortunately the terms in **red** are **nonlinear**.

Executive summary

Looking for a convex polyhedron $A\mathbf{x} \leq \mathbf{b}$ with unknown A and $\mathbf{b}$, stable by linear transitions...

is reduced to solving a big system of nonlinear equations!

Does not scale...
Current methods (Barcelogic group) involve e.g. looking for "small integer coefficients" in A.

Extensions

Nonlinear constraints?
Nonlinear transitions?

Even more costly!

See work by e.g. Deepak Kapur

Plan

Finding inductive invariants

- Is the major method for proving safety properties on programs (and circuits etc.)
- Is hard
- If restricted to certain geometrical classes, can be reduced to solving systems of numerical equations
- In certain cases, systems solvable (in exponential time) by combinations of linear programming and iterations
- Systems can be implicitly represented (for implicit control-flow graphs)
- In other cases, nonlinear equations ensue
- In practice, most tools do not use these "precise" methods and use widening (extrapolation) and/or predicate abstraction with Craig interpolation

Gratuitous advertisement

The ERC (European Research Council) project **STATOR**
is looking for

- PhD students
- interns
- **post-docs**

`http://stator.imag.fr/`