An introduction to static analysis

Eric Goubault

“Modelisation and Analysis of Systems in Interaction” Laboratory

8th of April 2008
Validation, verification, compilation?
- A short demo with INTERPROC/APRON to see what we want to get!

Formalisation for simple “safety properties”:
- collecting semantics
- resolution of the semantic equations (a bit of order theory)

Abstractions for practical calculations
- an example: sign analysis
- simple formalisation by Galois connections

To go further
- interval analysis
- relational analyses… short demo with INTERPROC/APRON
Compilation and verification

[egoubault@is002728 Cours07] > more f.c
int f(int i)
{
    int j,k;
    k = i+j;
    j = 2;
    return k;
}

[egoubault@is002728 Cours07] > gcc -c f.c -O -Wall
f.c: In function ‘f’:
f.c:3: warning: ‘j’ might be used uninitialized in this function
But...

```
[egoubault@is002728 Cours07]>$ more g.c
int g()
{ int j[10];
   return j[12]; }
[egoubault@is002728 Cours07]>$ gcc -c g.c -O -Wall
[egoubault@is002728 Cours07]>

The verifications done by gcc are very basic... → complementary tools for validation!
```
Extensions of these methods for guaranteed correct software: formal methods

- “Formal specifications (B, Z etc.)
- *Proof* (essentially, partial correctness proof): ex. Hoare Logics (interaction with the user)
- *Model-Checking*: Verification of refined “temporal” properties on parallel programs
- *Static Analysis*: fully automatic method to find bugs or synthetise proof terms
Example of “validation”

```c
fft(complex_array_ref a, int n)
{
    complex_array_ref b[n/2], c[n/2];
    if (n > 2)
    {
        for (i=0; i < n; i=i+2)
        {
            b[i/2] = a[i];
            c[i/2] = a[i+1];
        }
        fft(b, n/2);
        fft(c, n/2);
        for (i=0; i < n; i=i+1)
        {
            a[i] = F1(n)*b[i] + F2(n)*c[i];
        }
    }
    else
    {
        a[0] = g*a[0] + d*a[1];
        a[1] = a[0] - 2*d*a[1];
    }
}
```
Validate?

Mathematical proof, not test; but of what?

- No bug at execution time (division by zero, out of bounds array dereference, null dereference, overflows etc.)
- Parseval?

\[
\sum_i |a'[i]|^2 = \sum_i |a[i]|^2
\]

- Equality very hard to achieve... because of floating-point numbers (≠ real numbers). Precision drift?
- “Functional” proof, in general not achievable...
Expected results

Find in an automatic and guaranteed manner:

- **(1)** Out of bounds array dereference (in the else statement) **only** for arrays whose size is not a power of 2! [should crash, or give an incoherent result; Astrée, Polyspace, typically]
- **(2)** Under conditions on the “input signal”, no overflow (Astrée, Polyspace, typically; also FLUCTUAT)
- **(3)** Under conditions, to give bounds for the difference between a real number calculation and the floating-point computation... (see our analyser FLUCTUAT)

→ characterisation of reachable values for all variables (**invariants**) at each control point.
[there are other general properties than these “safety” properties: “liveness” properties]
In order to do this (1)...

```c
fft(a,n)
{
cplx b[n/2], c[n/2];
if (n > 2)
{
    for (i=0; i<n; i=i+2)
    {
        b[i/2]=a[i];
        c[i/2]=a[i+1];
    }
    fft(b,n/2);
    fft(c,n/2);
    for (i=0; i<n; i=i+1)
    {
        a[i]=F1(n)*b[i]+F2(n)*c[i];
    }
} else
{
    a[0]=g*a[0]+d*a[1];
    a[1]=a[0]-2*d*a[1];
}
}
```

 dim a=n, n=2^k, k>0

i+1=2j+1<=n and n=2^k
implies i+1<n

n=2 (n=2^j, 0<=j<=k)
A short demo of INTERPROC/APRON
We construct the control flow graph of a program, as being the transition system \((S, i, E, Tran)\):

- \(S\) is the set of control points of the program (one by syntactic line for us - indicated in the text by \([i]\))
- \(i\) is the initial control point
- \(E = \{x = expr \mid Aexp\} \cup Bexp\)
- We define a certain number of rules for the transitions
Example

void main() { // [0]
    int x=[-100,50]; // [1]
    while [2] (x<100) {
        x=x+1; // [4]
    } // [5]
}

Executions of the program correspond to dipaths on the graph (orbits of this discrete dynamical system), on which the environment ($\sigma : Loc \rightarrow \mathbb{Z}$) is modified
The environments now “collect” the values, for all paths, at each control point.

- An environment is an element of $\wp(Loc \rightarrow \mathbb{Z})$.
- At each state $s \in S$, we want to compute such an environment $\sigma$. 
Collecting semantics - intuitively

- We start from the control flow graph (expressed as a transition system $\mathcal{T} = (S, i, E, Tran)$)

- We associate to each control point $s_i$ of $\mathcal{T}$ an “equation” in associated variables $S_i$, which represent the sets of values that program variables can take, at control points $s_i$:

$$S_i = \bigcup_{s_j \in S \mid (s_j, t, s_i) \in Tran} \llbracket t \rrbracket S_j$$

where $\llbracket t \rrbracket$ is the interpretation of the transition $t$ seen as a function from the set of values of variables to itself

- The solutions $S_i$ to these equations are called the invariants at control points $s_i$ (it is a “property” which is always true for all possible executions)
Example

```c
void main() {
    // [0]
    int x = [-100, 50];
    // [1]
    while (x < 100) { // [2]
        x = x + 1; // [3]
    } // [4]
} // [5]
```

\[
\begin{align*}
x_0 &= T \\
x_1 &= [-100, 50] \\
x_2 &= x_1 \cup x_4 \\
x_3 &= ] - \infty, 99] \cap x_2 \\
x_4 &= x_3 + [1, 1] \\
x_5 &= [100, +\infty[ \cap x_2 \\
\end{align*}
\]
How can we give a meaning to fixed point equations?

The fundamental notion is the one of *successive approximations*, ordered by a *partial order*.
A partial order \((P, \leq)\) is composed of:

- a set \(P\) and a binary relation \(\leq \subseteq P \times P\) such that
- \(\leq\) is reflexive: \(\forall p \in P, p \leq p\)
- \(\leq\) is transitive: \(\forall p, q, r \in P, p \leq q \land q \leq r \implies p \leq r\)
- \(\leq\) is anti-symmetric: \(\forall p, q \in P, p \leq q \land q \leq r \implies p = q\)

Ex.: \((\wp(S), \subseteq)\)

Notice: isomorphic to the boolean lattice of first-order propositional logics (with atoms being predicates on one integer value). \(\subseteq\) is \(\implies\)!
For $X \subseteq P$:

- $p$ is an upper bound of $X$ if $\forall q \in X, q \leq p$
- $p$ is a (the!) least upper bound (lub, sup etc.) if:
  - $p$ is an upper bound of $X$
  - for all upper bounds $q$ of $X$, $p \leq q$
- the least upper bound, if it exists, is denoted $\bigcup X$
- similarly for the notion of lower bound, the greatest lower bound (glb, inf etc.), is denoted $\bigcap X$
- A lattice is a partial order admitting a lub and a glb for all $X$ containing two elements (hence for any non empty finite set $X$)

Ex.: in $(\mathcal{P}(S), \subseteq)$, all $X$ admit a lub (the classical set-theoretic union) and a glb (the classical set-theoretic intersection). It is a particular kind of lattice.
A chain of $P$ is $p_0 \leq p_1 \leq \ldots \leq p_n$

An $\omega$-chain of $P$ if $p_0 \leq p_1 \leq \ldots \leq p_n \leq \ldots$

A partial order is a cpo ("Complete Partial Order") if for all $\omega$-chains $P$, $P$ admits a lub in the partial order

In general, we suppose that a cpo also has a minimal element, denoted $\bot$, i.e. $\forall p \in P, \bot \leq p$

A lattice is a complete lattice if all subsets admit a lub (and hence a glb).

Ex.: $(\wp(S), \subseteq)$ is a complete lattice ($\bot = \emptyset$, $\top = S$).
A function \( f : D \rightarrow E \) between two cpos \( D \) and \( E \) is increasing iff
\[
\forall d, d' \in D, \ d \sqsubseteq d' \Rightarrow f(d) \subseteq f(d')
\]
\( f \) increasing is continuous iff for all chains \( d_0 \sqsubseteq d_1 \sqsubseteq \ldots \sqsubseteq d_n \sqsubseteq \ldots \) of \( D \), we have:
\[
\bigsqcup_{n \in \mathbb{N}} f(d_n) = f \left( \bigcup_{n \in \mathbb{N}} d_n \right)
\]
Let $f : D \rightarrow D$ increasing on a partial order $D$. A fixed point of $f$ is an element $d$ of $D$ such that $f(d) = d$.

A post-fixed point of $f$ is an element $d$ of $D$ such that $f(d) \sqsubseteq d$

A pre-fixed point of $f$ is an element $d$ of $D$ such that $d \sqsubseteq f(d)$
(1) Let $f : D \to D$ be an increasing function on a complete lattice $D$. Then $f$ admits at least a fixed point. Furthermore, the set of fixed points of $f$ is a complete lattice, thus there is a least fixed point (noted $\text{lfp}(f)$) and a greatest fixed point (noted $\text{gfp}(f)$).

(2) Let $f : D \to D$ be a continuous function on a cpo $D$ (with a $\bot$ element). Then, 

$$\text{fix}(f) = \bigsqcup_{n \in \mathbb{N}} f^n(\bot)$$

is the smallest fixed point of $f$, $\text{lfp}(f)$ [Kleene iteration].
We compute the Kleene ($\text{fix}$) iteration for the functional:

\[
F \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} \top \\ [-100, 50] \\ x_1 \cup x_4 \\ ] - \infty, 99[ \cap x_2 \\ x_3 + [1, 1] \\ [100, +\infty[ \cap x_2 \end{pmatrix}
\]

where we wrote $x_i$ as denoting the set of possible values at control point $i$. 
We begin with $x_i = \bot \ (1 \leq i \leq 5)$

We compute the Kleene iteration... slightly ameliorated
(chaotic - similar to Gauss-Seidl versus Jacobi)
Iteration 1

\[
\begin{align*}
    x_0 &= \top \\
    x_1 &= [-100, 50] \\
    x_2 &= x_1 \cup x_4 \\
    x_3 &= ]-\infty, 99[ \cap x_2 \\
    x_4 &= x_3 + [1, 1] \\
    x_5 &= [100, +\infty[ \cap x_2 \\
    x_0^1 &= \top \\
    x_1^1 &= [-100, 50] \\
    x_2^1 &= [-100, 50] \\
    x_3^1 &= ]-\infty, 99[ \cap [-100, 50] \\
    x_4^1 &= [-100, 50] + [1, 1] \\
    x_5^1 &= [100, +\infty[ \cap [-100, 50] \\
    &= \bot
\end{align*}
\]
Iteration $i + 1$ ($i < 50$) [Kleene]

$x_0 = \top$
$x_1 = [-100, 50]$  
$x_2 = x_1 \cup x_4$
$x_3 = ]-\infty, 99] \cap x_2$
$x_4 = x_3 + [1, 1]$
$x_5 = [100, +\infty[ \cap x_2$

$x_0^{i+1} = \top$
$x_1^{i+1} = [-100, 50]$  
$x_2^{i+1} = [-100, 50 + i]$  
$x_3^{i+1} = ]-\infty, 99] \cap [-100, 50 + i]$  
$x_4^{i+1} = [-100 + i, 50 + i] + [1, 1]$  
$x_5^{i+1} = [100, +\infty[ \cap [-99, 50 + i + 1]$  
$= \bot$
Iteration 100

\[ x_0 = \top \]
\[ x_1 = [-100, 50] \]
\[ x_2 = x_1 \cup x_4 \]
\[ x_3 = ] - \infty, 99] \cap x_2 \]
\[ x_4 = x_3 + [1, 1] \]
\[ x_5 = [100, +\infty[ \cap x_2 \]

\[ x_0^{100} = \top \]
\[ x_1^{100} = [-100, 50] \]
\[ x_2^{100} = [-100, 100] \]
\[ x_3^{100} = ] - \infty, 99] \cap ([-100, 100]) \]
\[ x_4^{100} = [100, +\infty[ \cap ([-99, 100]) \]
\[ x_5^{100} = [100, 100] \]
Partial conclusion

- We were lucky, as there is no reason that we can solve these semantic equations in general in finite time (undecidability results)
- $\rightarrow$ abstract the semantics!

(cf. theory of abstract interpretation, P. et R. Cousot)
Example, the rule of signs:

- "Intuitive" notion of a rule of signs (where ? means "indeterminate sign"):
  
<table>
<thead>
<tr>
<th>add</th>
<th>+</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>+</td>
<td>?</td>
</tr>
<tr>
<td>-</td>
<td>?</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>mult</th>
<th>+</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
</tbody>
</table>

- How can we formalize this? [does correspond to restricting proof principles to a smaller set of predicates and consequences]
+ “denotes” the set of natural numbers
− “denotes” the set of opposites of natural numbers

We are making approximations: what can we say about the addition of + with −?
→ we should be able to return a value “I do not know”

We write this value: ⊤ (value which will be the maximal element of the lattice of signs)

⊤ “denotes” the set of relative integers (\( \mathbb{Z} \)
In order to have a complete lattice and a Galois connection...

The lattice of signs is written: \((S, \sqsubseteq)\)
Let $C$ be a complete lattice of **concrete properties**: (e.g. $(\wp(\mathbb{Z}), \subseteq)$)

Let $A$ be a complete lattice of **abstract properties**: (e.g. $(S, \subseteq)$)

$\alpha : C \to A$ (**abstraction**) and $\gamma : A \to C$ (**concretisation**) two monotonic functions (we could forget this hypothesis) such that

$$\alpha(x) \leq_A y \iff x \leq_C \gamma(y)$$
Galois connection in the case of signs

Here:

- $\gamma(\bot) = \emptyset$
- $\gamma(0) = \{0\}$
- $\gamma(+) = \mathbb{N}$
- $\gamma(\neg) = -\mathbb{N}$
- $\gamma(\top) = \mathbb{Z}$
Here:

- \( \alpha(\emptyset) = \bot \)
- \( \alpha(\{0\}) = 0 \)
- \( \alpha(S) = \)
  \[
  + \text{ if } S \text{ contains only positive values (and at least a strictly positive one)} \\
  - \text{ if } S \text{ only contains negative values (and at least a strictly negative one)} \\
  \top \text{ if } S \text{ contains at least a negative (or zero) value and a distinct positive (or zero) value}
  \]
We can check that we have the following properties:

(1) $\alpha \circ \gamma(x) \leq_{A} x$
(2) $y \leq_{C} \gamma \circ \alpha(y)$

In fact, we even have in the case of signs: $\alpha \circ \gamma(x) = x$ ("Galois insertion").

(1) or (2) are equivalent conditions to the fact that $(\alpha, \gamma)$ is a Galois connection.
• \( \alpha \) and \( \gamma \) are said to be “quasi-inverses”: they are almost bijections

• as for inverses, one determines the other:

\[
\begin{align*}
\alpha(x) &= \bigcap \{y \mid x \leq c \gamma(y)\} \\
\gamma(x) &= \bigcup \{y \mid \alpha(y) \leq A x\}
\end{align*}
\]

• In other terms:
  • \( \alpha \) give the most precise abstract value representing a given concrete property
  • \( \gamma \) gives the semantics of abstract values, in terms of concrete properties
The abstract functions corresponding to +, -, etc. can in fact be determined by calculus, given a Galois connection \((\alpha, \gamma)\) between a concrete domain \(C\) and an abstract domain \(A\):

For all concrete functionals \(F : C \rightarrow C\) (for example, the collecting semantics), we define an abstract functional \(F^\# : A \rightarrow A\) by

\[
F^\#(y) = \alpha \circ F \circ \gamma(y)
\]

It is the best possible abstraction of \(F\).

In practice, \(\alpha\) and/or \(\gamma\) are not computable (algorithmically speaking). We use in general an over-approximation \(F'\) such that \(F^\#(y) \leq_A F'(y)\).
When we have a Galois connection $(\alpha, \gamma)$ between a concrete domain $C$ and an abstract domain $A$, we have the following property, for all concrete functionals $F : C \to C$:

$$\alpha(lfp(F)) \leq A lfp(F^\#)$$

or, equivalently:

$$lfp(F) \leq_C \gamma \circ lfp(F^\#)$$

Hence, the abstract calculus gives an *over-approximation* of the concrete invariants!
Example of a computation

For the same old program... we then have the following semantic equations:

\[
\begin{align*}
x_0 &= \top \\
x_1 &= 0 \\
x_2 &= x_1 \cup x_4 \\
x_3 &= x_2 \\
x_4 &= "x_3 + [1, 1]" \\
x_5 &= + \cap x_2
\end{align*}
\]

where we must define the abstract transfer function corresponding to “\(\text{incr}(x_3) = x_3 + 1\)”
Transfer function of $\textit{incr} : x \rightarrow x + 1$

We can easily check that:

$$\textit{incr}^\# \begin{array}{cccc}
+ & - & 0 & \top & \bot \\
\top & \top & \top & \top & \bot \\
\end{array}$$
We begin with $x_i^0 = \bot$.

The first iteration gives:

\[
\begin{align*}
x_0 &= \top \\
x_1 &= 0 \\
x_2 &= x_1 \cup x_4 \\
x_3 &= x_2 \\
x_4 &= "x_3 + [1, 1]" \\
x_5 &= + \cap x_2
\end{align*}
\]
The second iteration gives:

\[
\begin{align*}
x_0 &= \top \\
x_1 &= 0 \\
x_2 &= x_1 \cup x_4 \\
x_3 &= x_2 \\
x_4 &= "x_3 + [1, 1]"
\end{align*}
\]

This is the fixed point!

In fact, we can easily show that in the lattice of signs, we have at most three iterations before getting to the least fixed point.
Use of abstract invariants for validation

We have over-approximations of possible values that variables can take, by solving systems of abstract equations. We can conclude in general for safety properties (i.e., no run time error [RTE]) of programs:

Invariants are wrong (inside)
But we are still far from real static analysis based validation!

→ Finer analysis of the ranges of variables to detect at least these RTEs:

- overflows (for given types of variables)
- division by zero
- square root of a negative number
- (not in this course really... but not too far) access of an element in an array, out of bounds

Simple analysis: interval analysis
We define the poset of intervals \( I \) with bounds in \( \overline{\mathbb{R}} \) as follows:

- an element of \( I \) is \([a, b]\) where \( a, b \in \overline{\mathbb{R}} \) and \( a \leq b \); or is \( \bot \) (identified with all \([a, b]\) with \( a > b \)); or is \( \top \), identified with \([−\infty, \infty]\)
- \( \bot \leq [a, b] \leq \top \)
- \([a, b] \leq [c, d]\) iff \( a \geq c \) and \( b \leq d \)
Lattice of intervals

\[ [a, b] \cup [c, d] = [\min(a, b), \max(c, d)] \]

\[ [a, b] \cap [c, d] = [\max(a, b), \min(c, d)] \]

Complete!

\[ \bigcup_{i \in I} [a_i, b_i] = [\inf_{i \in I} a_i, \sup_{i \in I} b_i] \]
Galois connection with $(\mathcal{P}(\mathbb{Z}), \subseteq)$

Define:

- $\gamma([a, b]) = \{ x \in \mathbb{Z} \mid a \leq x \leq b \}$
- $\gamma(\bot) = \emptyset$
- $\gamma(\top) = \mathbb{Z}$
Applying the formula:

- if $S$ is empty: $\alpha(S) = \perp$
- $\alpha(S) = [\inf S, \sup S]$
Transfer functions for expressions

As usual, they are strict i.e. map $\bot$ to $\bot$, and:

- $[a, b] +^\# [c, d] = [a + c, b + d]$
- $[a, b] -^\# [c, d] = [a - d, b - c]$
- $[a, b] \ast^\# [c, d] = \min(ac, ad, bc, bd), \max(ac, ad, bc, bd)]$

Tests are interpreted by constraint solving within intervals

etc.

You know them, this is interval arithmetics.
Example

```c
void main() {
    int x=[-100,50]; [1]
    while [2] (x<100) {
        [3] x=x+1; [4]
    } [5]
}
```

\[
\begin{align*}
x_0 & = \top \\
x_1 & = [-100, 50] \\
x_2 & = x_1 \cup x_4 \\
x_3 & = ] - \infty, 99] \cap x_2 \\
x_4 & = x_3 + [1, 1] \\
x_5 & = [100, +\infty[ \cap x_2
\end{align*}
\]
Iteration 1

\[ x_0 = \top \]
\[ x_1 = [-100, 50] \]
\[ x_2 = x_1 \cup x_4 \]
\[ x_3 = ]-\infty, 99] \cap x_2 \]
\[ x_4 = x_3 + [1, 1] \]
\[ x_5 = [100, +\infty] \cap x_2 \]

\[ x_0^1 = \top \]
\[ x_1^1 = [-100, 50] \]
\[ x_2^1 = [-100, 50] \]
\[ x_3^1 = ]-\infty, 99] \cap [-100, 50] \]
\[ x_4^1 = [-100, 50] + [1, 1] \]
\[ x_5^1 = [100, +\infty] \cap [-100, 50] \]
\[ = \bot \]
Iteration $i + 1$ ($i < 50$) [Kleene]

\[ x_0 = \top \]
\[ x_1 = [-100, 50] \]
\[ x_2 = x_1 \cup x_4 \]
\[ x_3 = ] - \infty, 99] \cap x_2 \]
\[ x_4 = x_3 + [1, 1] \]
\[ x_5 = [100, +\infty] \cap x_2 \]

\[ x_0^{i+1} = \top \]
\[ x_1^{i+1} = [-100, 50] \]
\[ x_2^{i+1} = [-100, 50 + i] \]
\[ x_3^{i+1} = ] - \infty, 99] \cap [-100, 50 + i] \]
\[ x_4^{i+1} = [-100 + i, 50 + i] + [1, 1] \]
\[ x_5^{i+1} = [100, +\infty] \cap [-99, 50 + i + 1] \]
\[ = \perp \]
Iteration 50

\[
\begin{align*}
x_0 &= \top \\
x_1 &= [-100, 50] \\
x_2 &= x_1 \cup x_4 \\
x_3 &= ] - \infty, 99] \cap x_2 \\
x_4 &= x_3 + [1, 1] \\
x_5 &= [100, +\infty[ \cap x_2 \\
x_0^{50} &= \top \\
x_1^{50} &= [-100, 50] \\
x_2^{50} &= [-100, 100] \\
x_3^{50} &= ] - \infty, 99] \cap [-100, 100]) \\
x_4^{50} &= [-100, 99] + [1, 1] \\
x_5^{50} &= [100, +\infty[ \cap [-99, 100] \\
\end{align*}
\]
Acceleration of convergence

Classically:

- Use of extrapolation operators ("widening") to quickly cut down Kleene iteration sequence, to get to a postfixpoint
- Use of a "narrowing" operator to get to a fixpoint (not necessarily the least one though)
We take here (for the widening at point 2):

\[ [a, b] \lor [c, d] = [e, f] \text{ with } \]

\[
e = \begin{cases}
a & \text{if } a \leq c \\
-\infty & \text{otherwise}
\end{cases}
\quad \text{and} \quad
\]

\[
f = \begin{cases}
b & \text{if } d \leq b \\
\infty & \text{otherwise}
\end{cases}
\]
\[ \begin{aligned}
    x_{0}^{10} &= T \\
    x_{1}^{10} &= [-100, 50] \\
    x_{2}^{10} &= [-100, 58] \bigtriangledown [-100, 59] \\
    &= [-100, \infty] \\
    x_{3}^{10} &= ] - \infty, 99] \cap [-100, \infty] \\
    &= [-100, 99] \\
    x_{4}^{10} &= [-100, 99] + [1, 1] \\
    &= [-99, 100] \\
    x_{5}^{10} &= [100, +\infty] \cap [-100, \infty] \\
    &= [100, +\infty]
\end{aligned} \]
Here we take:

\[ [a, b] \Delta [c, d] = [e, f] \text{ with } e = \begin{cases} 
  c & \text{if } a = -\infty \\
  a & \text{otherwise}
\end{cases} \quad \text{and} \quad f = \begin{cases} 
  d & \text{if } b = \infty \\
  b & \text{otherwise}
\end{cases} \]
\[\begin{align*}
x_0^{11} &= \top \\
x_1^{11} &= [-100, 50] \\
x_2^{11} &= [-100, \infty) \cap [-100, 100] \\
&= [-100, 100] \\
x_3^{11} &= ] - \infty, 99] \cap [-100, 100] \\
&= [-100, 99] \\
x_4^{11} &= [-100, 99] + [1, 1] \\
&= [-99, 100] \\
x_5^{11} &= [100, +\infty] \cap [-100, 100] \\
&= [100, 100]
\end{align*}\]

This is the least fixed point!
To go further

- Weaker frameworks of abstractions
- Interprocedural analysis
- Backward analysis and combinations forward/backward
- Aliases
- Relational domains
- Better solvers for fixpoint equations, iteration strategies but also optimisation methods, or policy iteration
- Underapproximations, test case generation (see Sylvie’s talk)
- Modular analysis, concurrency, objects, functional programs
- etc.
We have to interpret *aliases*, even for just finding numerical invariants. Example:

```c
int *x, *z, y;
y = 1;
x = &y;
z = &y;
y = y+1;
```

(should find that `x` and `z` point to the same location, with value 2)

More generally, any variable can point through a series of * (a dereference path), to similar locations.

Impossible to find exactly the set of aliases in general (here, at last control point, this set is `{(x, z), (*x, y)}`).
Relational analyses: interest

\[ X = [0,10]; \] //2
\[ Y = [0,10]; \] //3
\[ S = X-Y; \] //4
if (S>=2) //5
\[ Y = Y+2; \] // 6

In intervals:
Y in [0,12] at 6
Using relations:
At 5: \( X-Y\geq 2 \) hence \( Y\leq 8 \)
So at 6, Y in [0,10]
A partial zoo of domains

Constant Propagation
\[ X_i = c_i \]
[Kil73]

Signs
\[ X_i \geq 0, \ X_i \leq 0 \]
[CC76]

Intervals
\[ X_i \in [a_i, b_i] \]
[CC76]

Simple Congruences
\[ X_i \equiv a_i \ [b_i] \]
[Gra89, Gra97]

Interval Congruences
\[ X_i \in \alpha_i [a_i, b_i] \]
[Mas93]

Power Analysis
\[ X_i \in \alpha_i^{a_i[z+b_i]}, \ \alpha_i^{[a_i, b_i]}, \ etc. \]
[Mas01]
A partial zoo of domains

- **Linear Equalities**
  \[ \sum_i \alpha_{ij} X_i = \beta_j \]  
  [Kar76]

- **Linear Congruences**
  \[ \sum_i \alpha_{ij} X_i \equiv \beta_j [\gamma_j] \]  
  [Gra91]

- **Trapezoidal Congruences**
  \[ X_i = \sum_j \lambda_j \alpha_{ij} + \beta_j \]  
  [Mas92]

- **Polyhedra**
  \[ \sum_i \alpha_{ij} X_i \leq \beta_j \]  
  [CH78]

- **Ellipsoids**
  \[ \alpha X^2 + \beta Y^2 + \gamma XY \leq \delta \]  
  [Fer04b]

- **Varieties**
  \[ P_i(\vec{X}) = 0, \quad P_i \in \mathbb{R}[V] \]  
  [RCK04a]

Also: our affine arithmetic based domains (FLUCTUAT)  
(see http://pop-art.inrialpes.fr/interproc/interprocweb.cgi to play with some of these domains!)
A last short demo of INTERPROC/APRON

...