The Astrée Analyzer

Xavier Rival

CNRS, ENS and INRIA

KAIST, 7th of Nov., 2013

1

1. The research leading to these results has received funding from the ARTEMIS Joint Undertaking under agreement no 269335 (ARTEMIS Project MBAT)

Certification of embedded softwares

- **Safety critical applications**
  - **avionic** softwares but also automotive, space...
  - **synchronous**
- **Properties** to prove to guarantee **safety** :
  - **absence of runtime errors**
    i.e., no crash, no violation of application specific constraints
  - **synchronous requirement**,
    i.e., time constraint critical sections should take a bounded timeframe
    i.e., the software must be responsive ($\Rightarrow$ **recursion is forbidden**)
  - **resource usage**,
    i.e., **absence of dynamic memory allocation**, **stack usage**

What Astrée is ?

- **A static analyzer**
  - Inputs a **C program** (some restrictions, see later)
  - User-defined **assumptions** about the input values (ranges)
  - Computes an **over-approximation of the reachable states**
  - Produces an **alarm** when an operation **is not proved safe**
    - ⋆ Sound : detects all errors
    - ⋆ Incomplete : false alarms are possible

      Detecting all errors exactly : undecidable !
- **Developped in the École Normale Supérieure (Paris, France)**

  Joint work with B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D Monniaux
- Astrée addresses :
  - **Runtime errors**
  - **Non specified behaviors**

What Astrée is not ?

- **Testing :**
  - Astrée covers all executions, hence is **sound** ; testing is **unsound**
  - Astrée does sound approximation, hence **incompleteness**
  - **Cost** considerations :

    Weeks of heavy test processes vs. a few hours of computation
- **Model checking :**
  - Astrée is designed to implement the **C semantics**

    No separate model extraction phase
  - Astrée uses a **quasi-infinite predicate set**
  - Astrée lagging for **automatic refinement**
- **User-assisted theorem proving :**
  - Astrée is automatic with **little to no user interaction**

    Cost efficient !
  - But Astrée needs to infer indecidable properties ; hence is **incomplete**

Development of Astrée

- **Fall 2001 :** Request for a precise and fast analyzer (Airbus)
  Astrée project started :
  - **Scalability** = main goal
    (data structures, algorithms)
  - **Simple non-relational domains** : intervals + first refinements
  - Analysis of a 10 kLOCs software, few alarms (2002)
- From 2003, **analysis of industrial softwares :**
  - **Inspection of alarms :**
    ⇒ True error ?
    ⇒ Imprecision in the analysis ? if yes, **find origin of imprecisions**
  - Improve **precision**, with **new abstractions**,
    solving imprecisions, but preserving scalability
  - **Successful** analysis of **two families of industrial applications**
- **Commercial diffusion**, since 2009, by **AbsInt**
  Customers in **automotive**, **avionics**, **space**...
- **2012—now, MBAT project** : integration to model based analysis and testing platform

A Specialized Analyzer

- **Specialization with respect to some families of softwares :**

Synchronous, real-time programs

```
declare and initialize state variables;
loop forever
  read volatile input variables,
  compute output and state variables,
  write to volatile output variables;
  wait for next clock tick (10 ms)
end loop
```

- **Properties to establish :** **Absence of runtime errors**, but with respect to a broad definition
  - No **fatal error** as defined by the semantics of the **C language** e.g., **division by** 0, out of segment access
  - No **overflow** in integer or floating point computations
  - **User defined** properties : e.g., no **NaN** !
  - **Architecture dependant** properties (data-type sizes)

Specific Features of Astrée

Simplifications

- **Not all C :** no malloc, no recursion
- Mostly **static data** ; a few local variables

Issues

- **Size of programs to analyze :** > 100 **kLOC,** $> 10\,000$ **variables**
  More typically 1 MLOC, $> 50,000$ variables
- **Floating point computation** should be analyzed precisely
  DSP filetering, non linear control, retroactions, interpolation functions
- Intricate **dependencies between variables :**
  - Stability of computation should be established
  - Relations between numerical and boolean data to infer
  - Long sequences of dependences between inputs and outputs
    e.g., slicing ineffective

Outline

Principle of the Analyzer

Model of C with an operational semantics

$[\![P]\!]$ = set of executions (aka, traces) of P

- **C-99 standard**
- **IEEE 754-1985 norm** (floating point computations)
- Assumptions about the **target architecture** and **application** :
  - size of integer data-types
  - initialization of static variables
  - ranges for inputs ; duration of an execution

Computation of an over-approximation of reachable states

- **Abstraction** = approximation defined by an abstract domain
- Systematic derivation of a **sound and automatic analyzer**
- **Certification of a piece of code, in two automatic stages :**
  1. **Computation of an approximation** (99.9% of the work...)
  2. **Checking of safety conditions**

Abstraction

- Astrée **computes** an invariant $I \in D^\sharp$ ($D^\sharp$: **our abstract domain**) :
  - For each control point ℓ
  - For each context κ (e.g., calling stack)

  $\Rightarrow$ an approximation $I(\ell, \kappa) \in D^\sharp_{\mathtt{M}}$ of a **set of stores**

  $D^\sharp_{\mathtt{M}}$ expresses a (usually infinite) set of predicates
- **Soundness :**
  - I should account **for all executions in** P
  - Meaning of I : $\gamma(I)$
  - Soundness statement : $\boxed{[\![P]\!] \subseteq \gamma(I)}$

    where $[\![P]\!]$ is a formal **concrete semantics**
- **Next question :** How to compute I ?

  Definition of a generic interpretation scheme

Abstract Interpretation : Computing Invariants

Principle : **run all computations in a unique abstract computation**

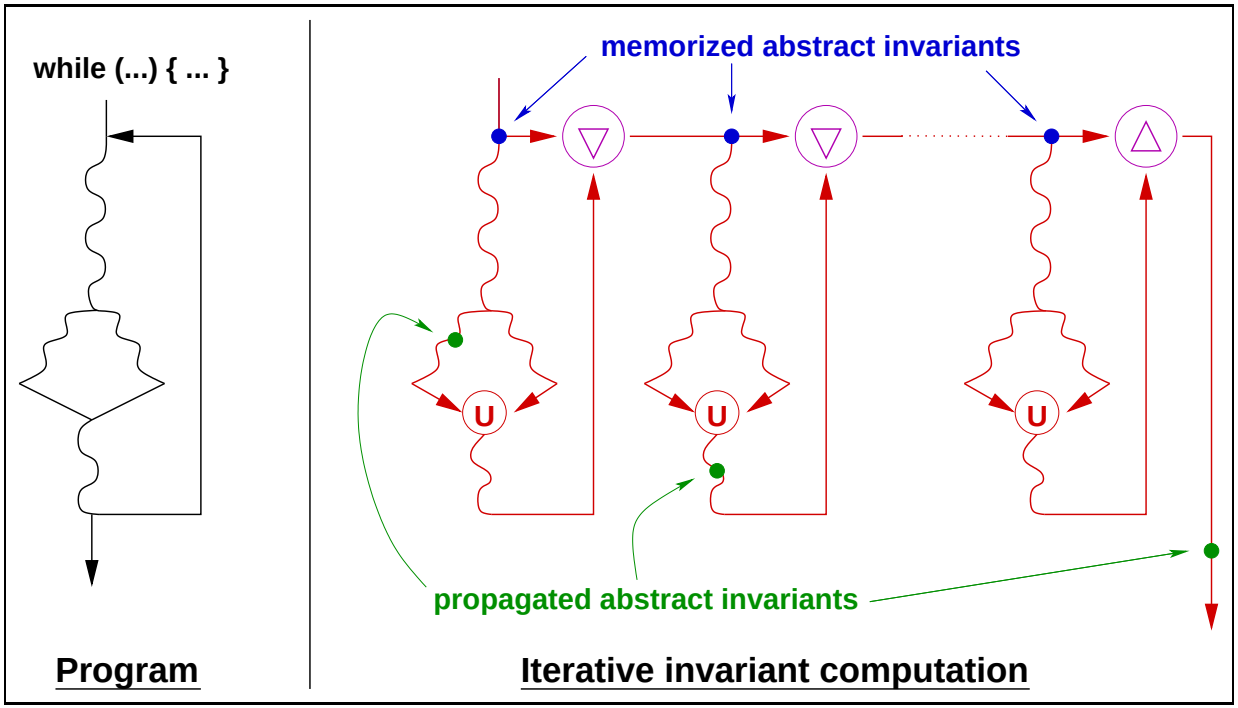
- **Analysis of an atomic statement** $x = e$ **:**
  - Use an abstract transfer function $\mathit{assign}(x = e) : D^\sharp_{\mathbb{M}} \to D^\sharp_{\mathbb{M}}$
  - $D^\sharp_{\mathbb{M}}$: manages addition and removal of constraints
  - **Soundness :** this operation should over-approximate concrete executions
- **Denotational form engine** :
  - For each **concrete, elementary step** F, a sound approximation computed by an abstract store transformer $F^\sharp$:

$$\forall \rho \in \mathbb{M},\ d^\sharp \in D^\sharp_{\mathbb{M}},\ \rho \in \gamma(d^\sharp) \Longrightarrow F(\rho) \subseteq \gamma(F^\sharp(d^\sharp))$$

  - $D^\sharp_{\mathbb{M}}$ provides such sound transfer functions : $\mathit{guard}, \ldots$
- **Control flow joins** (after a conditional) :
  $D^\sharp_{\mathbb{M}}$ provides a sound approximation $\sqcup$ of $\cup$

Analysis of Loops

- **Loops** should require infinitely many iterations
- **Solution :** use a **widening** operator
  - A **sound approximation** ∇ **of** $\sqcup$;
  - **Termination is enforced by the widening properties**



Widening Operator

- **Definition :**
  - **sound approximation of join** : $\forall x, y \in D^\sharp_{\mathtt{M}},\ \gamma(x) \cup \gamma(y) \sqsubseteq x \nabla y$
  - **termination** : for any increasing sequence $(y_n)_{n\in\mathbb{N}}$ of elements of $D^\sharp_{\mathtt{M}}$, the sequence $(x_n)_{n\in\mathbb{N}}$ of elements of $D^\sharp_{\mathtt{M}}$ defined by $x_0 = y_0$ and $\forall n \in \mathbb{N},\ x_{n+1} = x_n \nabla y_n$ is **not strictly increasing**
- **Widening :**
  - **Example, with intervals :**

    $I_0 : 0 \leq x \leq 10; \qquad I_1 : 1 \leq x \leq 11; \qquad I_0 \nabla I_1 : 0 \leq x$
  - **In practice : remove unstable constraints**

    Convergence : ensured by the finiteness of the number of constraints at the first iteration

Though : the **analysis** may still involve an **unbounded number of predicates**
domain may still be **infinite**
widening chains are still **unbounded**

Widening Improvements

- **"Unrolling"** of the first iterations (better precision)
  - **Idea :** postpone widening to iteration 2 or 3
  - **More precision** in the first abstract join operations
- **Thresholds :**
  - **Principle :** when $x < 4$ is not stable, $x < 8$ may be stable
  - **Threshold widening :** ordered families of constraints T
    ∇ gives a chance to each constraint $c \in T$ to stabilize
  - **Implementation** based on strategies such as :
    if $x == 4$ appears in the code, automatically add step 4 for x in ∇
- **Note :**
  Better precision $\Rightarrow$ smaller state space
  $\Rightarrow$ shorter widening chains

A precise analysis is NOT incompatible with efficiency

Practical experience : imprecise analyses with many alarms are very slow

Outline

Abstract Domain

- **Abstractions of sets of states** (e.g., stores)
- **Usual transfer functions :**
  - **Guard :** for conditions (**if**, **while**, **assert**,...)
  - **Assign**
  - Variable **creation**, **disposal**...
- A **widening operator**, a **lower upper bound**
- **Ordering :** usually a sound approximation of the concrete ordering
  - If $\inf^\sharp(x^\sharp, y^\sharp)$ returns `TRUE`, then $\gamma(x^\sharp) \subseteq \gamma(y^\sharp)$
  - But the test may fail ! (decidability !)
- **Support for communication** with other abstract domains :
  - **Dozens of domains** implemented in Astrée...
  - Need for **information communication** across domains

    Different domains typically establish complimentary properties

A Non-relational Abstraction : Intervals

- Simplification : **contrived memory model :**
  - 1 **abstract cell** $\equiv$ 1 or several concrete cells (*smashed* arrays)
  - Information about **pointers** : points-to
- **Interval-based approximation :**
  - Constraints $a \leq x \leq b$ (x : abstract cell)
  - **Not an expensive** analysis
  - Implementation : **sound** approximation of **floating point computations**
    Should be ensured for *all* numerical abstractions
- Astrée : started with an interval analysis
  - **Enough to express the absence of runtime errors**
    Array bounds, overflows, division by 0
  - **Not expressive enough to *infer/prove* precise invariants**
- Next frames : imprecisions + new abstract domains

Outline

Octagons

```
assume(x ∈ [−10, 10])
if(x < 0){y = −x; }
else{y = x; }
①if(y ≤ 5)
  {②assert(−5 ≤ x ≤ 5); }
```

With an interval analysis :

- At point ① : $x \in [-10, 10];\ y \in [0, 10]$
- At point ② : $x \in [-10, 10];\ y \in [0, 5]$
- **Analyzer alarm** (assert **not proved**)

- **We need a relation between x and y :**
  $\Rightarrow$ i.e., a **relational abstraction :** *polyhedra ?*
- **Octagons :**
  - Express constraints of the form $\pm x \pm y \leq c$.
    In the example :
    ① $0 \leq y - x \leq 20,\ 0 \leq y + x \leq 20$
    ② $y \in [0, 5];\ 0 \leq y - x \leq 20;\ 0 \leq y + x \leq 20,$
    hence $x \in [-5, 5]$
  - **More reasonable cost :** $\mathcal{O}(n^2)$ space ; $\mathcal{O}(n^3)$ time (still high)
- **Several issues** to solve to integrate octagons : cost, floating points...

Preserving Scalability with Octagons

- **Still too costly** :
  - $\mathcal{O}(n^3)$ **time complexity per operation**, if n variables
  - So if $n \equiv 10\,000$: **will not work**
- **A remark :** we will **not** need (or get) a relation between all pair (x, y)

Therefore, we should use several smaller octagons

- **Pack : small group** of variables **to relate**
- **Strategy and heuristics** used to choose packs
  - syntactic **pre-analysis** : variables "used together"
  - possibility to add **user-defined** packs (rarely needed)
- **Cost :** linear in the number of packs
  - Size of packs : bounded by a fixed constant
  - Number of packs : linear in the size of the code

  $\Rightarrow$ **linear cost**

Soundness and Floating Point Computations

- **Rounding errors in floating point** concrete computations
  But the domain is defined with real numbers
  (same for all relational abstractions, such as polyhedra, linear equalities...)
- **Approximation of expressions :**
  bounded with linear combinations with ranges as coefficients
  Example :

$$\left.\begin{array}{l} y \in [-10.5.] \\ x := y \star z + c \end{array}\right\} \Longrightarrow x := [-10. - \epsilon_0, 5. + \epsilon_0] \star z + [c - \epsilon_2, c + \epsilon_2]$$

  - **Linearized forms** can be handled by an octagon transfer function
  - **Interval constraints** used to make the transformation
- **Relational abstraction :**
  - **Semantics of octagons in terms of real numbers**
  - Linearization bridges the gap with the floating point values
    Takes into account **all** possible **rounding errors**

Reduction

- **Intuition : Use distinct predicates to improve precision**
  - $D^\sharp_{\mathtt{M}}, \gamma$ is **reduced** iff $\gamma(x^\sharp) = \gamma(y^\sharp) \Rightarrow x^\sharp = y^\sharp$
    - ⋆ most domains are **not** reduced ;
    - ⋆ this is source of imprecision
  - **Reduction :** should map $x^\sharp$ into a more preicse $y^\sharp$
  - **Non reduction may cause incompleteness of the ordering**
- **Intra-domain reduction :**
  - **Principle :**

$$x - y \leq a \wedge y - z \leq b \Longrightarrow x - z \leq a + b$$

  - **Cost considerations : cubic cost**, hence should be used sparsely
    Equivalent to a graph shortest path problem (Floyd Warshall algorithm)
- **Extra-domain reduction :**
  - Use the more precise bounds found above, refine **interval constraints**
  - Get more **precise constraints** from other domains (some shown later)

Outline

Analysis Imprecision : a Simple Example

```
     int x, sgn;
ℓ0   if(x < 0){
ℓ1     sgn = −1;
ℓ2   }else{
ℓ3     sgn = 1;
ℓ4   }
ℓ5   y = x/sgn;
ℓ6   ...
```

at ℓ_2, $x < 0, sgn = -1$
at ℓ_4, $x \geq 0, sgn = 1$
at ℓ_5, $sgn \in \{-1, 1\}$

- **Interval abstraction :**
  - At ℓ_5, **approximation :** $sgn \in [-1, 1]$
  - **Consequence :** the division is **not** proved safe **(alarm at ℓ_5)**
  - Clearly, $sgn \neq 0$ for any real execution **(false alarm)**
- If $\mathbb{D}^\sharp$ is a domain such that $\forall x \in \mathbb{D}^\sharp$, x stands for a **convex** set of concrete values : **same result**
  Hence, octagons will **not** fix this !

Disjunction-based Refinement

ℓ_0 **if**$(x < 0)\{$
ℓ_1 $\quad sgn = -1;$
ℓ_2 $\}$**else**$\{$
ℓ_3 $\quad sgn = 1;$
ℓ_4 $\}$
ℓ_5 $y = x/sgn;$
ℓ_6 $\ldots$

- **Solution : perform a case analysis on x**
  in order to avoid the fictitious case $sgn = 0$
- **Refined analysis :**
  - **At ℓ_5 :** $(x < 0 \wedge sgn = -1) \vee (x \geq 0 \wedge sgn = 1)$
  - **At ℓ_6 :** $y \geq 0$
  - The division is **safe**

Definition of the domain :

- We **rely on the control history** to distinguish states prior abstraction
- Invariant at ℓ_5 :

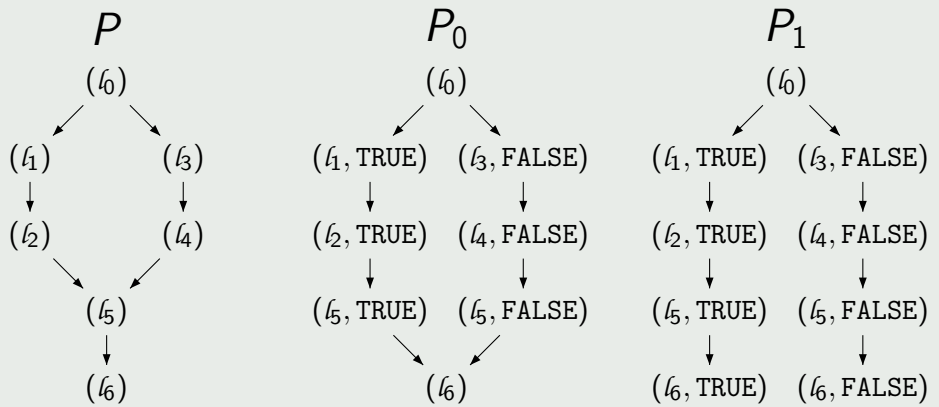
$$\begin{cases} \texttt{TRUE branch} & \Longrightarrow \quad x < 0 \wedge sgn = -1 \\ \texttt{FALSE branch} & \Longrightarrow \quad x \geq 0 \wedge sgn = 1 \end{cases}$$

System Refinement

- **Refining the control structure :**

  We enrich control states ℓ_i with **tokens** t_j (e.g., $\texttt{TRUE}, \texttt{FALSE}$)

ℓ_0 **if**$(x < 0)\{$
ℓ_1 $\quad sgn = -1;$
ℓ_2 $\}$**else**$\{$
ℓ_3 $\quad sgn = 1;$
ℓ_4 $\}$
ℓ_5 $y = x/sgn;$
ℓ_6 $\ldots$



- At the **semantics level**, we have a **partition** :

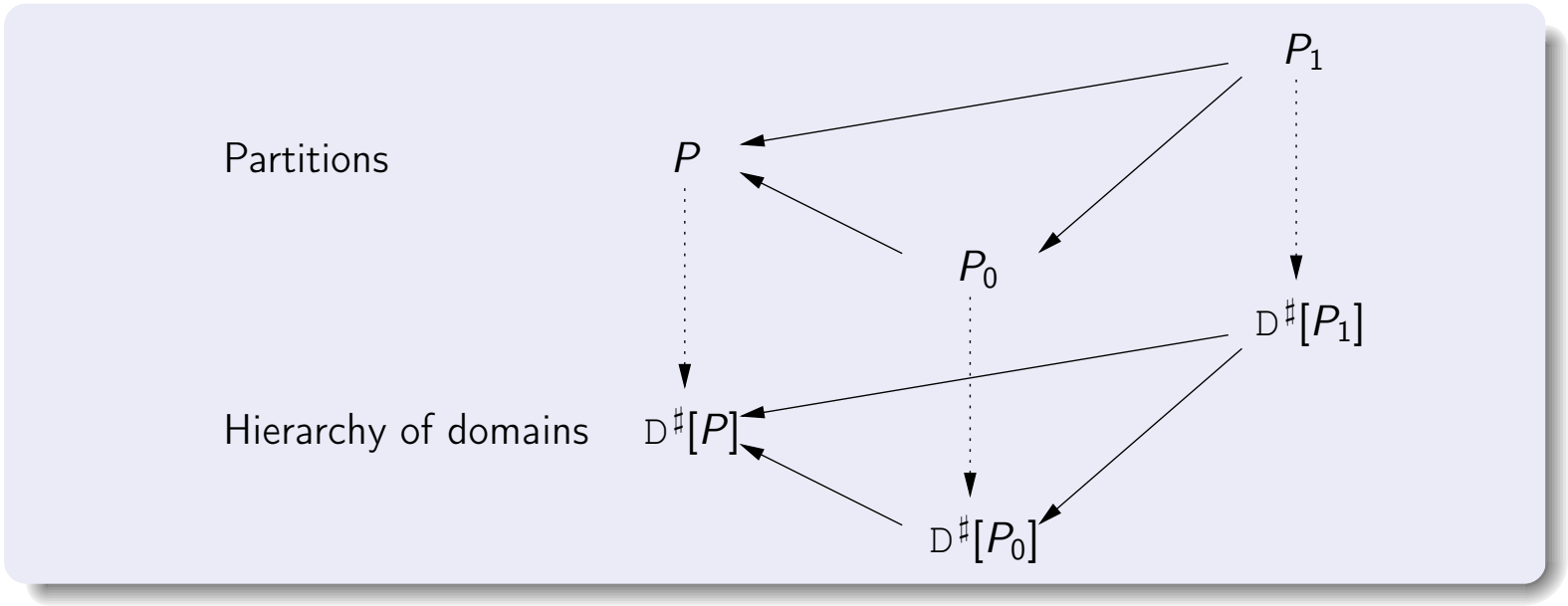
$$[\![P]\!](\ell_5) = [\![P_0]\!](\ell_5, \texttt{TRUE}) \uplus [\![P_0]\!](\ell_5, \texttt{FALSE})$$

- A **hierarchy** of refining control structures

  P_0 refines P $\qquad$ P_1 refines P_0

Construction of the Partitioning Domain

- For each refined control structure P_i : **a domain**
  $\mathbb{D}^\sharp[P_i] = \mathbb{L} \times T_i \to \mathbb{D}^\sharp$
  Concrete level function mapping **partitions** into **sets of traces**
  Abstract level function mapping **partitions** into **abstract invariants**
- **Overall structure**, for a given $\mathbb{D}^\sharp$:



- A **trace partitioning domain** value :
  A partition P_i + a semantic value $V \in \mathbb{D}^\sharp[P_i]$

Instantiation in Astrée

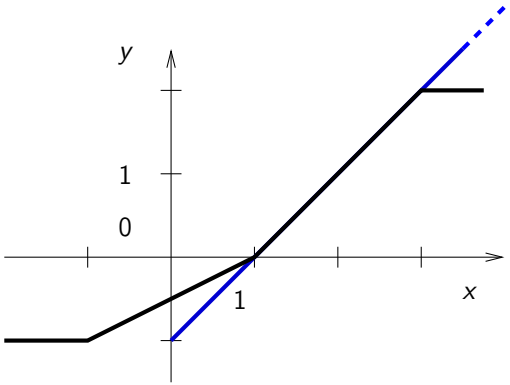
Trace partitioning criteria

- **If statements :** delayed abstract join
- **Loop unrolling :** distinguish the n first iterations ; delay abstract join
- **Variable value :** distinguish all possible values of a variable
  - do **not** partition if too many values
  - possible values are determined by the analysis
    Partitioning is **dynamic**, i.e. known only at analysis time

Partitioning strategies

- **Exhaustive application of the criteria** would **not scale up**
  - huge number of partitions available
  - most partitions are of no interest or may play against widening
- **Strategies :** determine which partitions to consider
  - where to distinguish flow paths
  - where to merge distinguished flow paths

A Realistic Example : Linear Interpolation



$$y = \begin{cases} -1 & \text{if } x \leq -1 \\ -\frac{1}{2} + \frac{x}{2} & \text{if } x \in [-1, 1] \\ -1 + x & \text{if } x \in [1, 3] \\ 2 & \text{if } 3 \leq x \end{cases}$$

Implementation

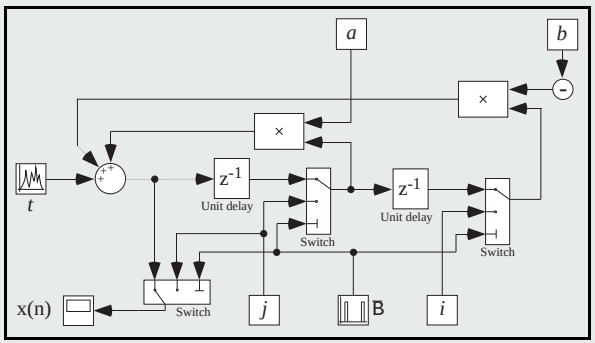
$$\begin{array}{ll} \ell_0: & \text{int } i = 0; \\ \ell_1: & \textbf{while}(i < 4 \;\&\&\; x > tx[i+1]) \\ \ell_2: & \quad i{+}{+}; \\ \ell_3: & y = tc[i] \times (x - tx[i]) + ty[i] \\ \ell_4: & \ldots \end{array}$$

- **Without partitioning :**
  - No relation between x and the slope (corresponding range i)
  - Analysis, with input $x > 0$, **output possibly unbounded** (slope in blue)
- **With partitioning** of the loop : above issues fixed, output in $[-1, 2]$
- **Strategy :** partition **loops** computing variables used as **array index** after the loop exit

Outline

Analyzing Digital Filters

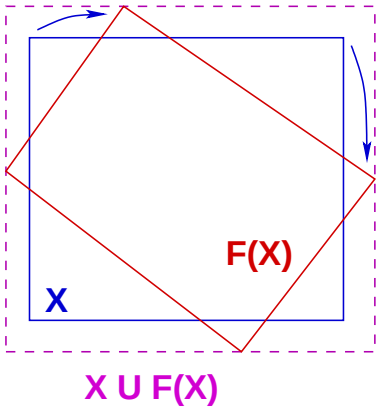
Simplified 2^{nd} order filter



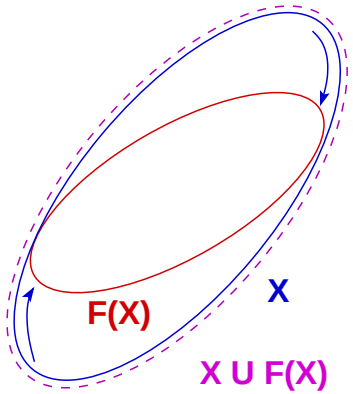
- Computed sequence :

$$X_n = \begin{cases} \alpha X_{n-1} + \beta X_{n-2} + Y_n \\ I_n \end{cases}$$

- Concrete computations are **bounded**
  Issue : how to **infer** an abstract bound ?
- **No stable octagon or interval**
- A **polyhedra** with **many faces** ?
- Most simple stable surface : **ellipsoid**



Interval instable



Stable ellipsoide

(Un)Stability of Floating Point Computations

```
x = 1.0;
while(TRUE){①
  x = x/3.0;
  x = x ⋆ 3.0;
}
```

- **Real numbers :** $x = 1.0$ at ①
- **Floating point numbers**
  $\Rightarrow$ **Rounding errors** (concrete semantics)
- Accumulation of rounding errors :
  may cause a (slow) divergence

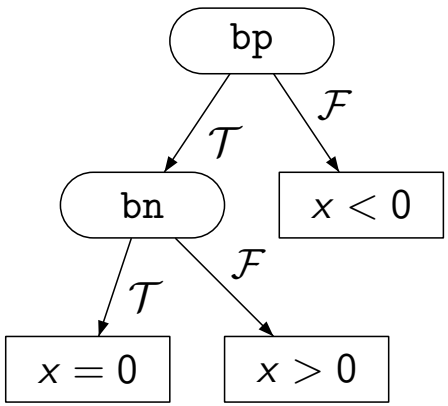
- **Solution : use arithmetico-geometric progressions** to bound rounding errors with a function of the **number of concrete iterates :**
  - **Constraint** $| x | \leq A \cdot B^n + C$,
    where A, B, C are constants and n is the iteration number
  - **Number of iterations : bounded by** N **:** $\Rightarrow | x | \leq A \cdot B^N + C$
- **Ellipsoids, progressions :**
  - Domains using **mathematical theorems** proved once (design time)
  - **Beyond what automatic refinement can do !**

Binary Decision Trees

```
bp = x ≤ 0.;
bn = x ≥ 0.;
①if(bp && bn)
   ②y = 0.0;
else y = 1.0/x;
```

- **Non-relational analysis :**
  Alarm at ②, possible **division by 0**
- **Relations needed at ① :**
  - $\texttt{bp} = \texttt{FALSE} \Rightarrow \texttt{x} \neq 0$
  - $\texttt{bn} = \texttt{FALSE} \Rightarrow \texttt{x} \neq 0$

An abstract domain inspired from BDDs



- **Nodes labeled with boolean variables**
- **Leaves :** values in a numerical domain
  e.g., intervals in this example
- **Scalability issues :**
  - Same as in the case of *octagons*;
  - We also use a **variable packing strategy**

Outline

Benchmarks

- 2 families of **synchronous embedded programs**
  A340 and A380 Airbus Aircraft fly-by-wire systems
- 2.2 GHz Bi-opteron, 1 processor used (64-bits arch)

LOC	70 000	226 000	400 000
Iterations	32	51	88
Memory used (Gb)	0.6	1.3	2.2
Time	46mn	3h57mn	11h48mn
False alarms	0	0	0

Conclusion :

- **Few or no false alarms :** can be used to **certify critical code**
- **Memory and time requirements are reasonable**
- Due to the **specialization** of the analyzer

Recent and Future Extensions

- **Extensions** :
  - **Asynchronous softwares** (ongoing)
  - **Automatize alarm diagnostics**...
- **For more information :**
  - **Accademic version website** : `http://www.astree.ens.fr/`
  - **AbsInt website** : `http://www.absint.com/astree/`