

Static Analysis by Abstract Interpretation of Embedded Critical Software

Julien Bertrane

ENS, Julien.Bertrane@ens.fr

Patrick Cousot

ENS & CIMS, Patrick.Cousot@ens.fr

Radhia Cousot

CNRS & ENS, Radhia.Cousot@ens.fr

Jérôme Feret

INRIA & ENS, Jerome.Feret@ens.fr

Laurent Mauborgne

IMDEA Laurent.Mauborgne@imdea.org

Antoine Miné

CNRS & ENS, Antoine.Mine@ens.fr

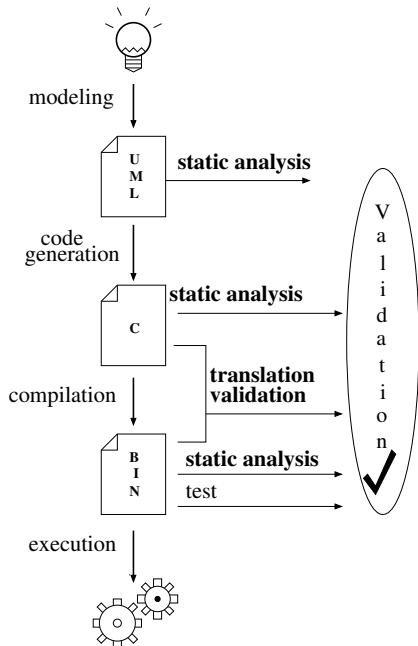
Xavier Rival

INRIA & ENS, Xavier.Rival@ens.fr

Semantics and Abstract Interpretation team

November 16th, 2010





Which level should be statically analyzed ?

- ▶ Static Analysis can be applied at many levels :
 - ▶ formal specification
 - ▶ program source
 - ▶ binary

Which level should be statically analyzed ?

- ▶ Static Analysis can be applied at many levels :
 - ▶ formal specification
 - ▶ program source
 - ▶ binary
- ▶ Static Analysis of high level, **pros** :
 - ▶ purer information
 - ▶ feedback easier
 - ▶ has information on hardware (imperfections)
 - ▶ de-synchronization analysis (made at Modeling level)

Which level should be statically analyzed ?

- ▶ Static Analysis can be applied at many levels :
 - ▶ formal specification
 - ▶ program source
 - ▶ binary
- ▶ Static Analysis of high level, **pros** :
 - ▶ purer information
 - ▶ feedback easier
 - ▶ has information on hardware (imperfections)
 - ▶ de-synchronization analysis (made at Modeling level)
- ▶ Static Analysis of high level, **cons** :
 - ▶ some aspects of computations abstracted (real arithmetics VS actual implementation)

Which level should be statically analyzed ?

- ▶ Static Analysis can be applied at many levels :
 - ▶ formal specification
 - ▶ program source
 - ▶ binary
- ▶ Static Analysis of high level, **pros** :
 - ▶ purer information
 - ▶ feedback easier
 - ▶ has information on hardware (imperfections)
 - ▶ de-synchronization analysis (made at Modeling level)
- ▶ Static Analysis of high level, **cons** :
 - ▶ some aspects of computations abstracted (real arithmetics VS actual implementation)
 - ▶ numeric overflows analysis (made at C level)
 - ▶ precision of floating-point computations analysis (made at C level)
 - ▶ worst case execution time analysis (made at binary level)

Static Analysis and Abstract Interpretation

Static analyzers should extract **automatically** properties.

Difficulties :

- ▶ most interesting properties are **undecidable**

Solutions :

- ▶ an analyzer focuses on a **subset** of properties and programs

Static Analysis and Abstract Interpretation

Static analyzers should extract **automatically** properties.

Difficulties :

- ▶ most interesting properties are **undecidable**
- ▶ the analyzer may consider **spurious** behaviors

Solutions :

- ▶ an analyzer focuses on a **subset** of properties and programs
- ▶ **refine** the analysis, it is always sound

Static Analysis and Abstract Interpretation

Static analyzers should extract **automatically** properties.

Difficulties :

- ▶ most interesting properties are **undecidable**
- ▶ the analyzer may consider **spurious** behaviors
- ▶ what about errors in **interpreting** the specifications or the behavior of the code

Solutions :

- ▶ an analyzer focuses on a **subset** of properties and programs
- ▶ **refine** the analysis, it is always sound
- ▶ work is done **directly on system code** (*i.e.* input of compilers or code generators)

Static Analysis and Abstract Interpretation

Static analyzers should extract **automatically** properties.

Difficulties :

- ▶ most interesting properties are **undecidable**
- ▶ the analyzer may consider **spurious** behaviors
- ▶ what about errors in **interpreting** the specifications or the behavior of the code

Solutions :

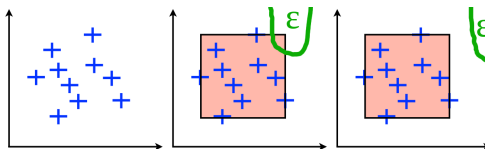
- ▶ an analyzer focuses on a **subset** of properties and programs
- ▶ **refine** the analysis, it is always sound
- ▶ work is done **directly on system code** (*i.e.* input of compilers or code generators)

Abstract Interpretation framework :

- ▶ the analyzer explores **supersets** of actual behaviors
- ▶ growing **library** of abstract domains focusing on a single mathematical theory
- ▶ **modularity** of domains or close cooperation between them

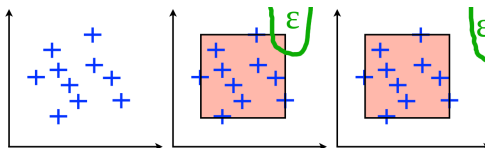
Example of Abstract Domain : Intervals

- ▶ We abstract V set of integers as an interval
 $\alpha_i(V) \triangleq [\min V, \max V]$



Example of Abstract Domain : Intervals

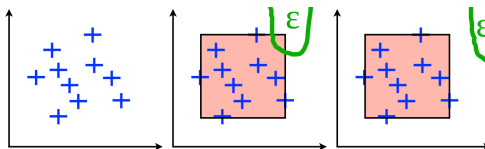
- ▶ We abstract V set of integers as an interval
 $\alpha_i(V) \triangleq [\min V, \max V]$



- ▶ This is an **over**-approximation

Example of Abstract Domain : Intervals

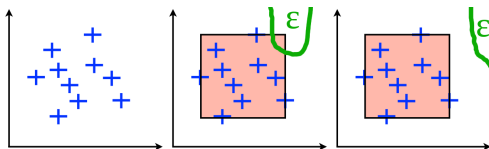
- ▶ We abstract V set of integers as an interval
 $\alpha_i(V) \triangleq [\min V, \max V]$



- ▶ This is an **over**-approximation
 - ▶ $z \notin \alpha_i(V) \Rightarrow z \notin V$

Example of Abstract Domain : Intervals

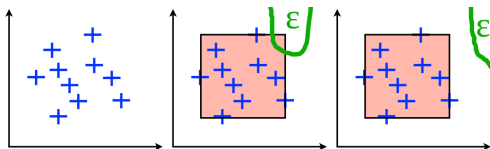
- ▶ We abstract V set of integers as an interval
 $\alpha_i(V) \triangleq [\min V, \max V]$



- ▶ This is an **over**-approximation
 - ▶ $z \notin \alpha_i(V) \Rightarrow z \notin V$
 - ▶ **but** $z \in \alpha_i(V) \not\Rightarrow z \in V$

Example of Abstract Domain : Intervals

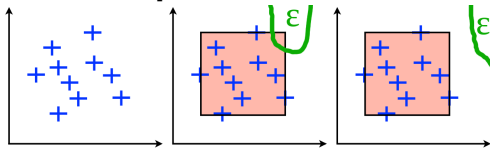
- ▶ We abstract V set of integers as an interval
 $\alpha_i(V) \triangleq [\min V, \max V]$



- ▶ This is an **over**-approximation
 - ▶ $z \notin \alpha_i(V) \Rightarrow z \notin V$
 - ▶ **but** $z \in \alpha_i(V) \not\Rightarrow z \in V$
- ▶ Concretization function $\gamma_i([\ell, h]) \triangleq \{z \in \mathbb{Z} \mid \ell \leq z \leq h\}$.

Example of Abstract Domain : Intervals

- ▶ We abstract V set of integers as an interval
 $\alpha_i(V) \triangleq [\min V, \max V]$



- ▶ This is an **over**-approximation
 - ▶ $z \notin \alpha_i(V) \Rightarrow z \notin V$
 - ▶ **but** $z \in \alpha_i(V) \not\Rightarrow z \in V$
- ▶ Concretization function $\gamma_i([\ell, h]) \triangleq \{z \in \mathbb{Z} \mid \ell \leq z \leq h\}$.
- ▶ $\forall V \in \wp(\mathbb{Z}) : \forall [\ell, h] \in V_i^\# : \alpha_i(V) \subseteq [\ell, h] \iff V \subseteq \gamma_i([\ell, h])$
and so, by definition, the pair $\langle \alpha, \gamma \rangle$ is a Galois connection

$$\langle \wp(\mathbb{Z}), \subseteq \rangle \xrightleftharpoons[\alpha_i]{\gamma_i} \langle V_i^\#, \subseteq \rangle$$

[ACM Trans. Program. Lang. Syst. 29, 2007] . X. Rival and L. Mauborgne The trace partitioning abstract domain

Semantics and Specifications

Semantics :

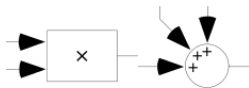
- ▶ Semantics defined for **each primitive**



Semantics and Specifications

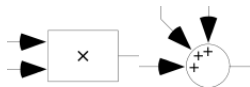
Semantics :

- ▶ Semantics defined for **each primitive**



Semantics and Specifications

Semantics :

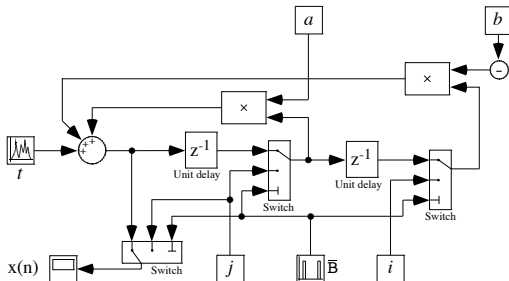


- ▶ Semantics defined for **each primitive**
- ▶ We focus on **safety** properties : we need **reachable states**

Semantics and Specifications

Semantics :

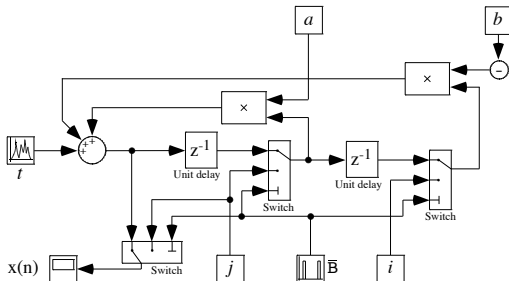
- ▶ Semantics defined for **each primitive**
- ▶ We focus on **safety** properties : we need **reachable states**
- ▶ Theoretically computable as a **fixpoint** of an operator T summarizing all the effects of primitives used in the program



Semantics and Specifications

Semantics :

- ▶ Semantics defined for **each primitive**
- ▶ We focus on **safety** properties : we need **reachable states**
- ▶ Theoretically computable as a **fixpoint** of an operator T summarizing all the effects of primitives used in the program



Specifications :

- ▶ We consider a set of **error states** ϵ that shouldn't be reached.

- ▶ **Intermediate** goal : $\alpha(\text{lfp}^{\gamma} T) = A$ with $\gamma(A) \cap \varepsilon = \emptyset$

[*POPL'77*] P. Cousot and R. Cousot . Abstract interpretation : a unified lattice model for static analysis of programs by construction or approximation of fixpoints. *4th ACM Symp. on Principles of Programming Languages*

Fixpoint Abstraction

- ▶ **Intermediate** goal : $\alpha(\text{lfp}^{\gamma} T) = A$ with $\gamma(A) \cap \varepsilon = \emptyset$
- ▶ $\alpha(\text{lfp}^{\gamma} T)$ is often non-computable

[*POPL'77*] P. Cousot and R. Cousot . Abstract interpretation : a unified lattice model for static analysis of programs by construction or approximation of fixpoints. *4th ACM Symp. on Principles of Programming Languages*

Fixpoint Abstraction

- ▶ **Intermediate** goal : $\alpha(\text{lfp}^{\exists} T) = A$ with $\gamma(A) \cap \varepsilon = \emptyset$
- ▶ $\alpha(\text{lfp}^{\exists} T)$ is often non-computable
- ▶ However, **if** $\alpha \circ T \sqsubseteq T^{\#} \circ \alpha$, **then** $\alpha(\text{lfp}^{\exists} T) \sqsubseteq \text{lfp}^{\exists} T^{\#}$.

[POPL'77] P. Cousot and R. Cousot . Abstract interpretation : a unified lattice model for static analysis of programs by construction or approximation of fixpoints. *4th ACM Symp. on Principles of Programming Languages*

Fixpoint Abstraction

- ▶ **Intermediate** goal : $\alpha(\text{lfp}^{\succ} T) = A$ with $\gamma(A) \cap \varepsilon = \emptyset$
- ▶ $\alpha(\text{lfp}^{\succ} T)$ is often non-computable
- ▶ However, **if** $\alpha \circ T \sqsubseteq T^{\#} \circ \alpha$, **then** $\alpha(\text{lfp}^{\succ} T) \sqsubseteq \text{lfp}^{\sqsubseteq} T^{\#}$.
- ▶ **New** goal : $\gamma(\text{lfp}^{\sqsubseteq} T^{\#}) \cap \varepsilon = \emptyset$.

[POPL'77] P. Cousot and R. Cousot . Abstract interpretation : a unified lattice model for static analysis of programs by construction or approximation of fixpoints. *4th ACM Symp. on Principles of Programming Languages*

Abstract Fixpoint Approximation

The iterates of the $T^\#$ operation converge to the fixpoint :

- ▶ but maybe in infinitely many iterations

Abstract Fixpoint Approximation

The iterates of the $T^\#$ operation converge to the fixpoint :

- ▶ but maybe in infinitely many iterations
- ▶ and at a combinatorial time and memory cost

Abstract Fixpoint Approximation

The iterates of the T^\sharp operation converge to the fixpoint :

- ▶ but maybe in infinitely many iterations
- ▶ and at a combinatorial time and memory cost
- ▶ convergence has to be accelerated using a widening ∇

Abstract Fixpoint Approximation

The iterates of the T^\sharp operation converge to the fixpoint :

- ▶ but maybe in infinitely many iterations
- ▶ and at a combinatorial time and memory cost
- ▶ convergence has to be accelerated using a widening ∇
- ▶ naïve example of widening for intervals is

$$[\ell^i, h^i] \nabla [\ell^{i+1}, h^{i+1}]$$

$$\triangleq [\text{if } \ell^{i+1} < \ell^i \text{ then } -\infty \text{ else } \ell^i, \text{if } h^{i+1} > h^i \text{ then } +\infty \text{ else } h^i]$$

Semantics :

- ▶ *ASTRÉE* input written in large subset of C, (*no* dynamic memory allocation, recursivity, and parallelism)

Semantics :

- ▶ *ASTRÉE* input written in large subset of C, (*no* dynamic memory allocation, recursivity, and parallelism)
- ▶ syntax and semantics based on the C99 norm, supplemented with the IEEE 754-1985 norm

Semantics :

- ▶ *ASTRÉE* input written in large subset of C, (*no* dynamic memory allocation, recursivity, and parallelism)
- ▶ syntax and semantics based on the C99 norm, supplemented with the IEEE 754-1985 norm

Properties proved : [run-time errors](#)

- ▶ overflows in unsigned and signed integer and float arithmetics and casts

Semantics :

- ▶ *ASTRÉE* input written in large subset of C, (*no* dynamic memory allocation, recursivity, and parallelism)
- ▶ syntax and semantics based on the C99 norm, supplemented with the IEEE 754-1985 norm

Properties proved : [run-time errors](#)

- ▶ overflows in unsigned and signed integer and float arithmetics and casts
- ▶ divisions by zero

Semantics :

- ▶ ASTRÉE input written in large subset of C, (*no* dynamic memory allocation, recursivity, and parallelism)
- ▶ syntax and semantics based on the C99 norm, supplemented with the IEEE 754-1985 norm

Properties proved : [run-time errors](#)

- ▶ overflows in unsigned and signed integer and float arithmetics and casts
- ▶ divisions by zero
- ▶ out-of-bound array accesses

Semantics :

- ▶ ASTRÉE input written in large subset of C, (*no* dynamic memory allocation, recursivity, and parallelism)
- ▶ syntax and semantics based on the C99 norm, supplemented with the IEEE 754-1985 norm

Properties proved : [run-time errors](#)

- ▶ overflows in unsigned and signed integer and float arithmetics and casts
- ▶ divisions by zero
- ▶ out-of-bound array accesses
- ▶ NULL, dangling, out-of-bound and misaligned pointer dereferences,

Semantics :

- ▶ ASTRÉE input written in large subset of C, (*no* dynamic memory allocation, recursivity, and parallelism)
- ▶ syntax and semantics based on the C99 norm, supplemented with the IEEE 754-1985 norm

Properties proved : [run-time errors](#)

- ▶ overflows in unsigned and signed integer and float arithmetics and casts
- ▶ divisions by zero
- ▶ out-of-bound array accesses
- ▶ NULL, dangling, out-of-bound and misaligned pointer dereferences,
- ▶ assertion failures (in calls to the `assert` C function).

Synchronous programming

- ▶ ▶ Initialize(S)
- ▶ while true do
 - ▶ (O, S) := Compute (S, I)
 - ▶ wait for clock
- ▶ od

with I : Inputs, S : (internal) State variables, O : Outputs

Analyzed Codes

- ▶ **ASTRÉE** focuses on analyzing **control/command synchronous** programs automatically generated from **Modeling Languages**.

Analyzed Codes

- ▶ ASTRÉE focuses on analyzing **control/command synchronous** programs automatically generated from **Modeling Languages**.
- ▶ communication by “volatile” memory locations allowed

Analyzed Codes

- ▶ ASTRÉE focuses on analyzing **control/command synchronous** programs automatically generated from **Modeling Languages**.
- ▶ communication by “volatile” memory locations allowed
- ▶ **proof of absence of run-time errors** in 2 families of industrial control/command software :
 - ▶ aeronautics
 - ▶ 100 K code lines
 - ▶ 10 K global variables (5 K are floats)
 - ▶ analysis takes approx. 2 h
 - ▶ up to 1 M code lines analyzed in 50 h

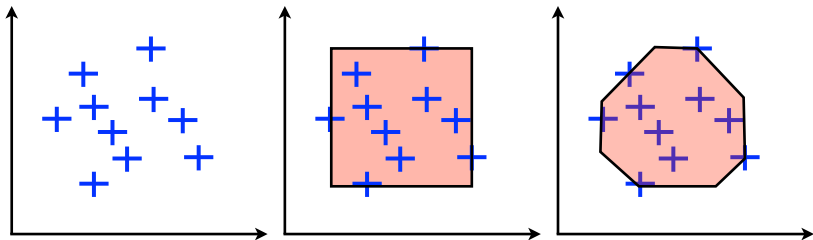
Analyzed Codes

- ▶ ASTRÉE focuses on analyzing **control/command synchronous** programs automatically generated from **Modeling Languages**.
- ▶ communication by “volatile” memory locations allowed
- ▶ **proof of absence of run-time errors** in 2 families of industrial control/command software :
 - ▶ aeronautics
 - ▶ 100 K code lines
 - ▶ 10 K global variables (5 K are floats)
 - ▶ analysis takes approx. 2 h
 - ▶ up to 1 M code lines analyzed in 50 h
 - ▶ space software
 - ▶ 14 K lines C code generated from SCADE
 - ▶ analysis takes approx. 1 h

Analyzed Codes

- ▶ **ASTRÉE** focuses on analyzing **control/command synchronous** programs automatically generated from **Modeling Languages**.
- ▶ communication by “volatile” memory locations allowed
- ▶ **proof of absence of run-time errors** in 2 families of industrial control/command software :
 - ▶ aeronautics
 - ▶ 100 K code lines
 - ▶ 10 K global variables (5 K are floats)
 - ▶ analysis takes approx. 2 h
 - ▶ up to 1 M code lines analyzed in 50 h
 - ▶ space software
 - ▶ 14 K lines C code generated from SCADE
 - ▶ analysis takes approx. 1 h
- ▶ **ASTRÉE** now analyzes code generated by dSPACE TargetLink (code generator for MATLAB, Simulink and Stateflow, added by AbsInt).

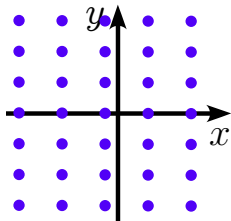
Toward Relational Domains



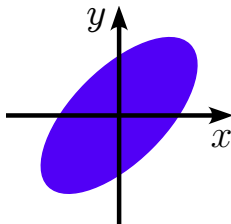
- ▶ relational domains bring **fine-tuned preciseness** (more precise than intervals)
- ▶ at a **bounded computational cost**
- ▶ and take **rounding errors** of float variables in considerations

[HOSC'06] A. Miné . The octagon abstract domain

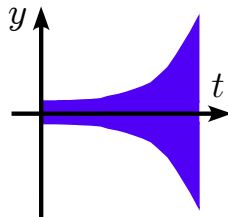
Specialized domains



Linear Congruences :
 $x \equiv a[b]$



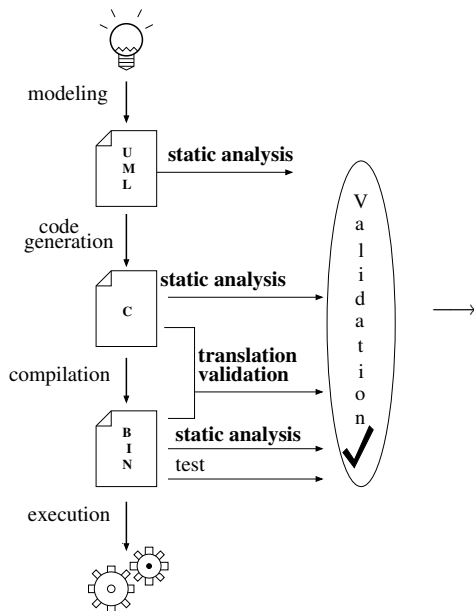
Ellipses :
 $x^2 + by^2 - axy \leq d$



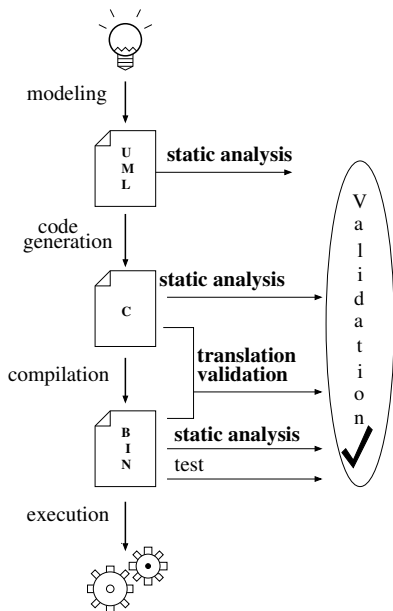
Exponentials :
 $-a^{bt} \leq y(t) \leq a^{bt}$

[ESOP'04] J. Feret . Static analysis of digital filters

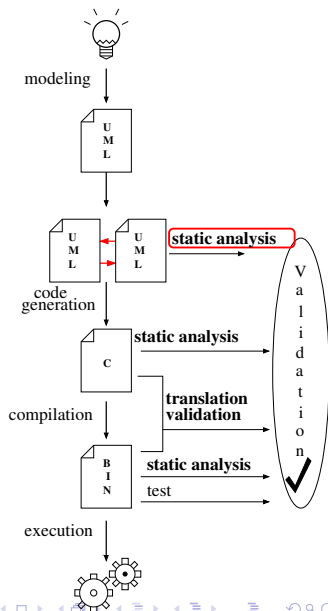
Imperfectly-Clocked Synchronous Systems



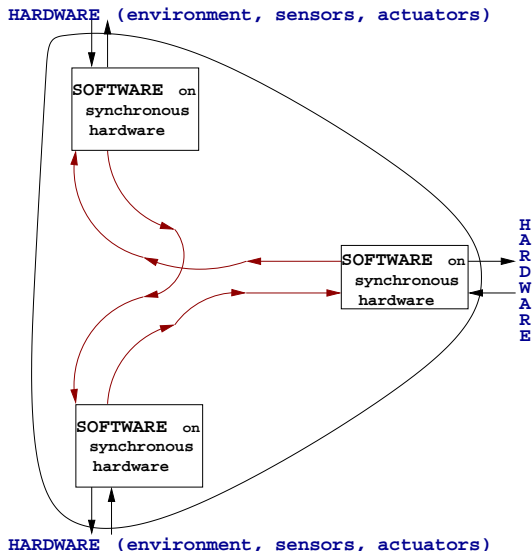
Imperfectly-Clocked Synchronous Systems



→



Typical system to verify : embedded systems



Imperfectly-Clocked Synchronous Systems

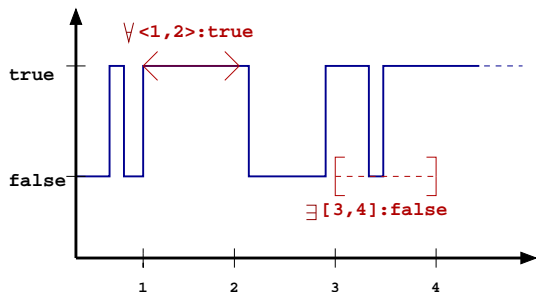
A new semantics

This non-standard semantics : **continuous-time**

- ▶ allows a **more precise modeling** of reality
 - ▶ imperfect clocks
 - ▶ communication channels with unknown latency
- ▶ reuses **continuous theories**
 - ▶ integral theory
 - ▶ directed homology
- ▶ allows a **precise and efficient** static analysis

Imperfectly-Clocked Synchronous Systems

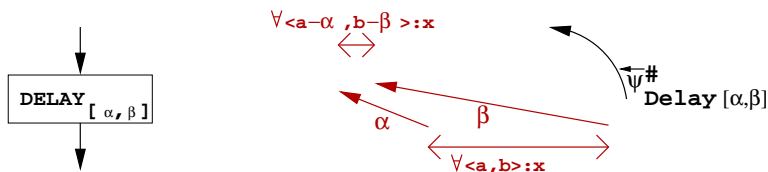
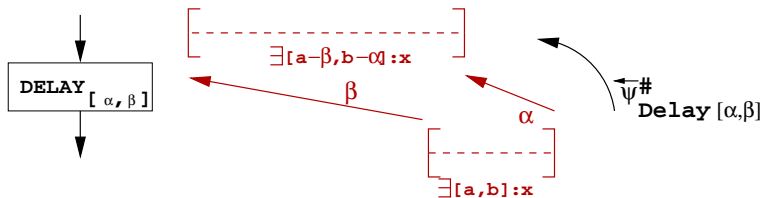
1st temporal abstract domain : constraints



- ▶ express many **local temporal properties**
- ▶ and **prove** some of these properties
 - ▶ Boolean variable v satisfies $\forall \langle a; b \rangle : x$, then $\neg v : \forall \langle a; b \rangle : \neg x$

[VMCAI'05] J. Bertrane. Static analysis by abstract of the quasi-synchronous composition of synchronous programs. *Paris*

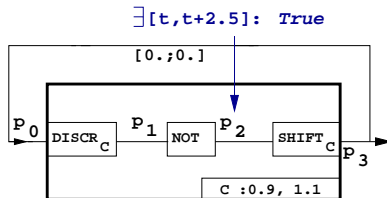
Abstract Operators and Constraints : an example



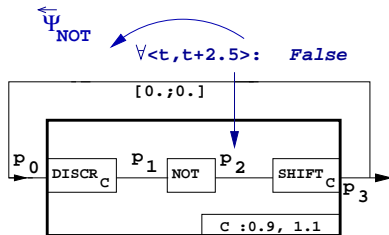
- ▶ $\overleftarrow{\Psi}^{\#}_{\text{DELAY}[\alpha, \beta]}(\exists [a; b] : x) \triangleq \exists [a - \beta; b - \alpha] : x$
- ▶ $\overleftarrow{\Psi}^{\#}_{\text{DELAY}[\alpha, \beta]}(\forall \langle a; b \rangle : x) \triangleq \forall \langle a - \alpha; b - \beta \rangle : x$

Analysis inside the abstract domain of **Constraints**

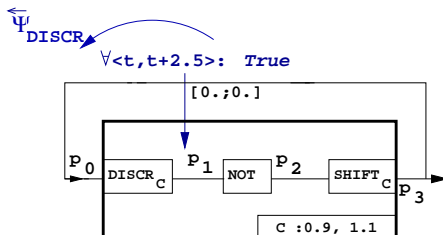
- ▶ Proving the following abstract property.



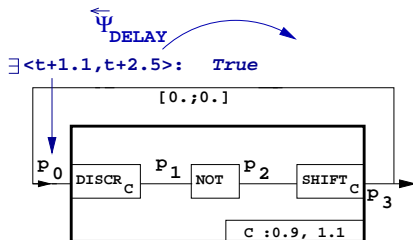
Analysis inside the abstract domain of **Constraints**



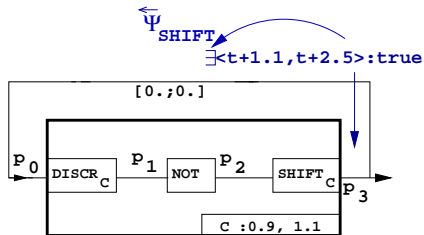
Analysis inside the abstract domain of **Constraints**



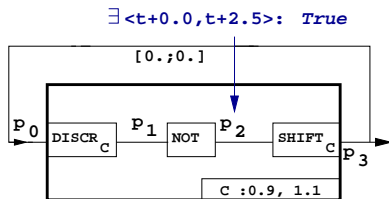
Analysis inside the abstract domain of **Constraints**



Analysis inside the abstract domain of **Constraints**



Analysis inside the abstract domain of **Constraints**



At point p_2 : 2 constraints have to be satisfied :

- ▶ $\forall \langle t; t + 2.5 \rangle : \text{False}$
- ▶ $\exists [t; t + 2.5] : \text{True}$

which is impossible and thus invalidates the initial hypothesis

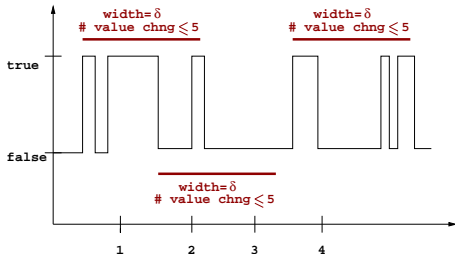
$\forall \langle t; t + 2.5 \rangle : \text{False}$.

- ▶ Thus $\exists [t; t + 2.5] : \text{True}$ is certified.

Imperfectly-Clocked Synchronous Systems

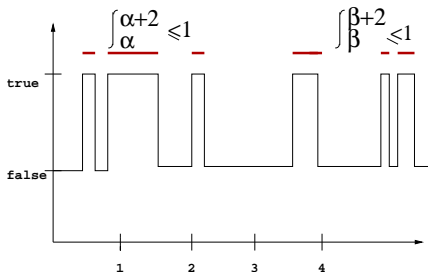
More **temporels** abstract domains

value changes counting



- ▶ express **stability specifications**.

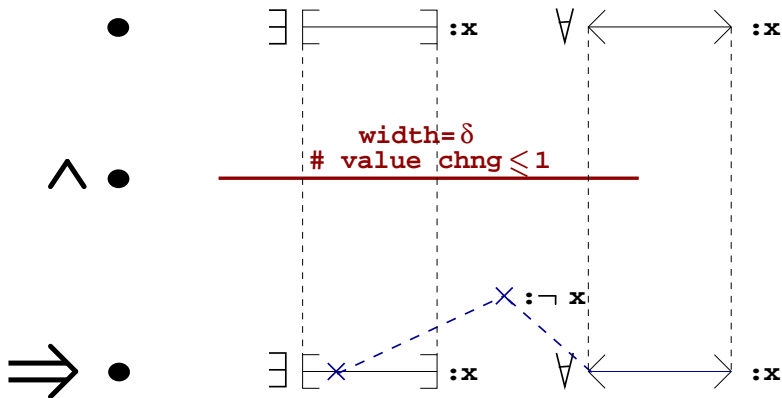
integral boundings



- ▶ express **quantitative properties (average value, ...)**

[SAS'06] J. Bertrane. Proving the properties of communicating imperfectly-clocked synchronous systems. *Seoul*

Reduce product Constraints - Value changes counting



Reduce product Constraints - Value changes counting

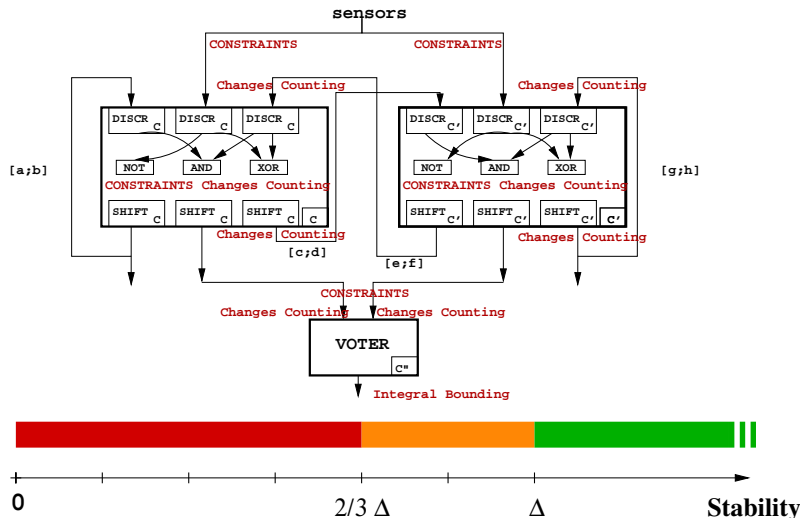
• $\exists [\text{---}] : \mathbf{x}$ $\forall \langle \text{---} \rangle : \mathbf{x}$

• \wedge $\text{width} = \delta$
 $\# \text{ value chng} \leq 1$

• \Rightarrow $\forall \langle \text{---} \rangle : \mathbf{x}$

Analyzed codes

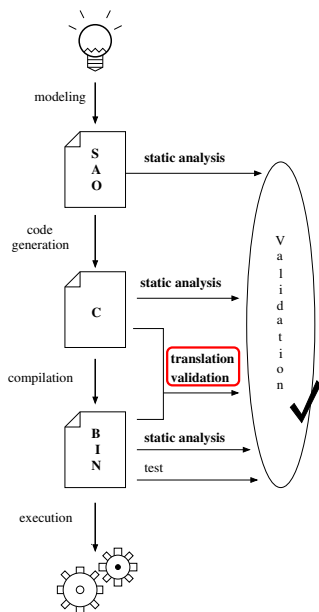
Static Analysis of communicating redundant units with imperfect clocks



analysis points a counter-example | ? | analysis proves the specification



Translation Validation



[POPL'04] X. Rival . Symbolic transfer function-based approaches to certified compilation

Conclusion

- ▶ Abstract Interpretation enables static analysis at **several levels** of the development of embedded systems.
- ▶ Only requirement : **good formalization** of different layers, as proposed by UML.
- ▶ It may check the **correctness of the translation** from one level to another.
- ▶ It may take a **hardware hypothesis** in consideration.
- ▶ **Growing collection** of abstract domains, analysis covers always more bigger areas.
- ▶ Abstract domains should not be too expressive : **trade-of** between precision and speed.
- ▶ Abstract domains should **not cover too wide** areas : rather build two domains with precise **reduction**.