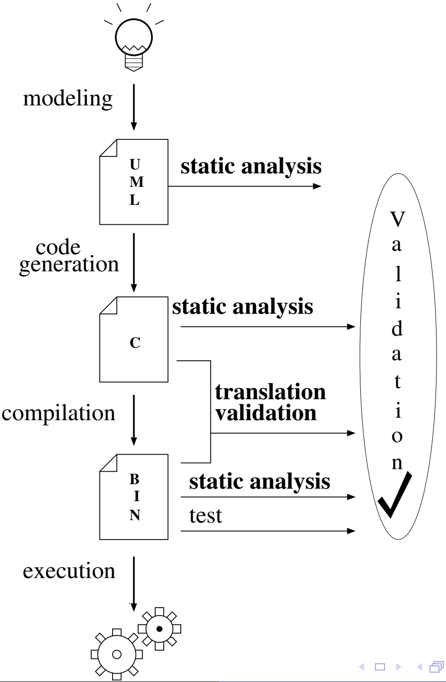
Static Analysis by Abstract Interpretation of Embedded Critical Software

Julien Bertrane
ENS, Julien.bertrane@ens.fr
Patrick Cousot
ENS & CIMS, Patrick.Cousot@ens.fr
Radhia Cousot
CNRS & ENS, Radhia.Cousot@ens.fr
JÃ©rÃ´me Feret
INRIA & ENS, JÃ©rÃ´me 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

SÃ©mantics and Abstract Interpreation team

November 16th, 2010


modeling
UML
static analysis
code generation
C
static analysis
translation validation
compilation
BIN
static analysis
test
execution
Validation

Which level should be statically analyzed ?

- Static Analysis can be applied at many levels :
  - machine-readable specification
  - program source
  - binary

Which level should be statically analyzed ?

- Static Analysis can be applied at many levels :
  - machine-readable 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 :
  - machine-readable 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 :
  - machine-readable 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 :

- the analyzer explores machine-representable supersets of actual behaviors

Static Analysis and Abstract Interpretation

Static analyzers should extract automatically properties.
Difficulties :

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

Solutions :

- the analyzer explores machine-representable supersets of actual behaviors
- 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 :

- the analyzer explores machine-representable supersets of actual behaviors
- refine the analysis, it is always sound
- work is done directly on the concrete system (*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 :

- the analyzer explores machine-representable supersets of actual behaviors
- refine the analysis, it is always sound
- work is done directly on the concrete system (*i.e.* input of compilers or code generators)

Abstract Interpretation framework :

- an analyzer focuses on a subset of properties and programs
- growing library of abstraction domains
- modularity of domains or close cooperation between them

Semantics and Specifications

Semantics :

- Semantics defined for each primitive

Semantics and Specifications

Semantics :

- Semantics defined for each primitive
- We focus on safety properties

Semantics and Specifications

Semantics :

- Semantics defined for each primitive
- We focus on safety properties
- The set of reachable states is therefore enough

Semantics and Specifications

Semantics :

- Semantics defined for each primitive
- We focus on safety properties
- The set of reachable states is therefore enough
- 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
- The set of reachable states is therefore enough
- 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 bad states ε that shouldn't be reached.

Semantics and Specifications

Semantics :

- Semantics defined for each primitive
- We focus on safety properties
- The set of reachable states is therefore enough
- 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 bad states ε that shouldn't be reached.
- So $\mathbf{lfp}^{T} \cap \varepsilon = \emptyset$

Example of Abstract Domain : Intervals

- We abstract a set $V \subseteq \mathbb{Z}$ of integers by the interval $\alpha_i(V) \triangleq [\min V, \max V]$

Example of Abstract Domain : Intervals

- We abstract a set $V \subseteq \mathbb{Z}$ of integers by the interval $\alpha_i(V) \triangleq [\min V, \max V]$
- this is an over-approximation

Example of Abstract Domain : Intervals

- We abstract a set $V \subseteq \mathbb{Z}$ of integers by the 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 a set $V \subseteq \mathbb{Z}$ of integers by the interval $\alpha_i(V) \triangleq [\min V, \max V]$
- this is an over-approximation
- $z \notin \alpha_i(V) \Rightarrow z \notin V$
- $z \in \alpha_i(V) \not\Longrightarrow z \in V$

Example of Abstract Domain : Intervals

- We abstract a set $V \subseteq \mathbb{Z}$ of integers by the interval $\alpha_i(V) \triangleq [\min V, \max V]$
- this is an over-approximation
- $z \notin \alpha_i(V) \Rightarrow z \notin V$
- $z \in \alpha_i(V) \not\Rightarrow z \in V$
- Concretization function $\gamma_i([\ell, h]) \triangleq \{z \in \mathbb{Z} \mid \ell \leqslant z \leqslant h\}$.

Example of Abstract Domain : Intervals

- We abstract a set $V \subseteq \mathbb{Z}$ of integers by the interval $\alpha_i(V) \triangleq [\min V, \max V]$
- this is an over-approximation
- $z \notin \alpha_i(V) \Rightarrow z \notin V$
- $z \in \alpha_i(V) \not\Longrightarrow z \in V$
- Concretization function $\gamma_i([\ell, h]) \triangleq \{z \in \mathbb{Z} \mid \ell \leqslant z \leqslant h\}$.
- $\forall V \in \wp(\mathbb{Z}) : \forall [\ell, h] \in V_i^\sharp : \alpha_i(V) \subseteq [\ell, h] \Longleftrightarrow V \subseteq \gamma_i([\ell, h])$ and so, by definition, the pair $\langle \alpha, \gamma \rangle$ is a Galois connection

$$\langle \wp(\mathbb{Z}), \subseteq \rangle \underset{\alpha_i}{\overset{\gamma_i}{\leftrightarrows}} \langle V_i^\sharp, \subseteq \rangle$$

Fixpoint Abstraction

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

Fixpoint Abstraction

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

Fixpoint Abstraction

- Intermediate goal : $\alpha(\mathbf{lfp}^{\preceq}\ T) = A$ with $\gamma(A) \cap \varepsilon = \emptyset$
- $\alpha(\mathbf{lfp}^{\preceq}\ T)$ is often non-computable
- However, if $\alpha \circ T \mathrel{\dot{\sqsubseteq}} T^\sharp \circ \alpha$, then $\alpha(\mathbf{lfp}^{\preceq}\ T) \sqsubseteq \mathbf{lfp}^{\sqsubseteq}\ T^\sharp$.

Fixpoint Abstraction

- Intermediate goal : $\alpha(\mathbf{lfp}^{\preceq}\ T) = A$ with $\gamma(A) \cap \varepsilon = \emptyset$
- $\alpha(\mathbf{lfp}^{\preceq}\ T)$ is often non-computable
- However, if $\alpha \circ T \mathrel{\dot{\sqsubseteq}} T^{\sharp} \circ \alpha$, then $\alpha(\mathbf{lfp}^{\preceq}\ T) \sqsubseteq \mathbf{lfp}^{\sqsubseteq}\ T^{\sharp}$.
- New goal : $\gamma(\alpha(\mathbf{lfp}^{\preceq}\ T)) \cap \varepsilon = \emptyset$.

Abstract Fixpoint Approximation

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

- but maybe in infinitely many iterations

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

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 $\triangledown$
- A naïve example of widening for intervals is

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

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

ASTREE

Semantics :

- ASTRÉE input written in large subset of C, (*not* dynamic memory allocation, recursivity, and parallelism)

ASTREE

Semantics :

- ASTRÉE input written in large subset of C, (*not* dynamic memory allocation, recursivity, and parallelism)
- syntax and semantics based on the C99 norm, supplemented with the IEEE 754-1985 norm

ASTREE

Semantics :

- ASTRÉE input written in large subset of C, (*not* 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

ASTREE

Semantics :

- ASTRÉE input written in large subset of C, (*not* 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

ASTREE

Semantics :

- ASTRÉE input written in large subset of C, (*not* 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

ASTREE

Semantics :

- ASTRÉE input written in large subset of C, (*not* 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,

ASTREE

Semantics :

- ASTRÉE input written in large subset of C, (*not* 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).

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 embedded avionic control/command software generated from high level specifications :
  - aeronautics 100 K code lines and 10 K global variables — half of which are floats in around 2 h up to 1 M code lines analyzed in 50 h

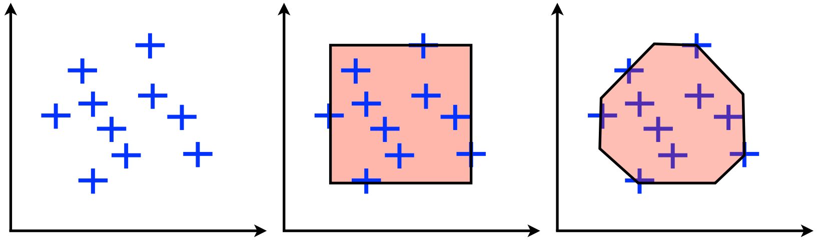
Analyzed Codes

- 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 embedded avionic control/command software generated from high level specifications :
  - aeronautics 100 K code lines and 10 K global variables — half of which are floats in around 2 h up to 1 M code lines analyzed in 50 h
  - space software 14 K lines C code generated from a SCADE model designed by Astrium ST.

Analyzed Codes

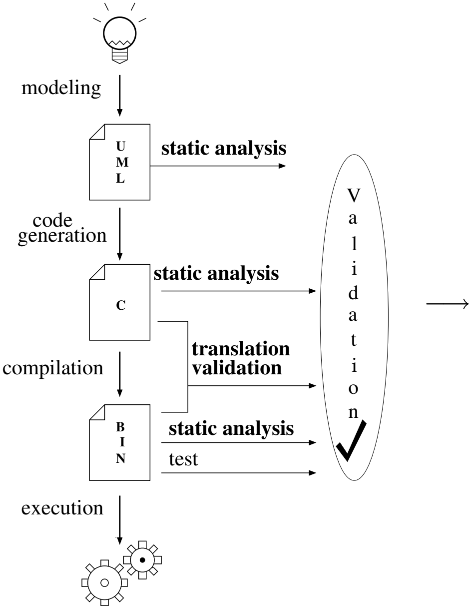
- ASTRÉE focuses on analyzing control/command synchronous programs automatically generated from Modeling Languages.
- communication by “volatile” memory locations allowed
- proof of absence of run-time errors in 2 families of industrial embedded avionic control/command software generated from high level specifications :
  - aeronautics 100 K code lines and 10 K global variables — half of which are floats in around 2 h up to 1 M code lines analyzed in 50 h
  - space software 14 K lines C code generated from a SCADE model designed by Astrium ST.
- ASTRÉE now handle code generated by dSPACE TargetLink (code generator for MATLAB, Simulink and Stateflow) (added by AbsInt).

Toward Relational Domains

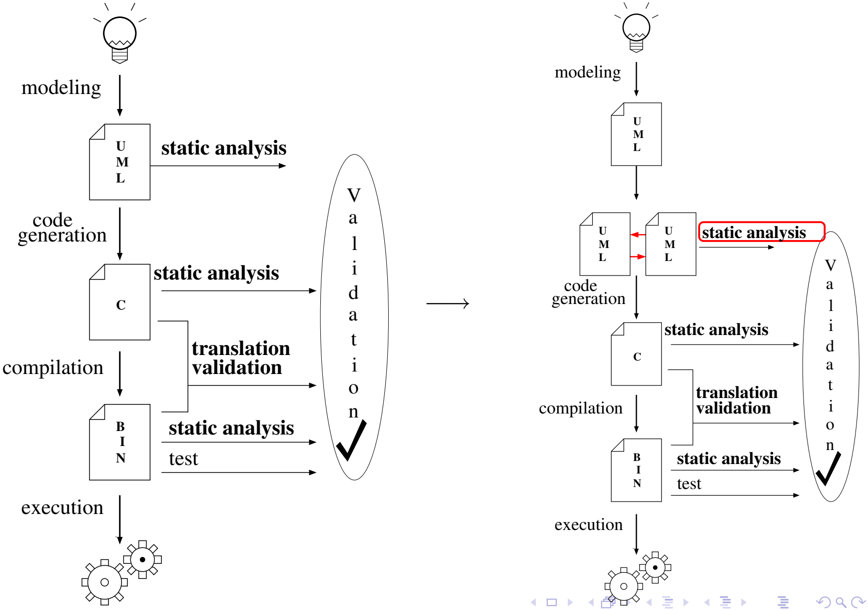


- relational domains bring fine-tuned preciseness (more precise than intervals)
- at a bounded computational cost.

Imperfectly-Clocked Synchronous Systems

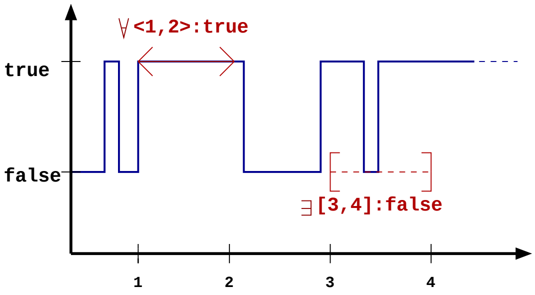


Imperfectly-Clocked Synchronous Systems



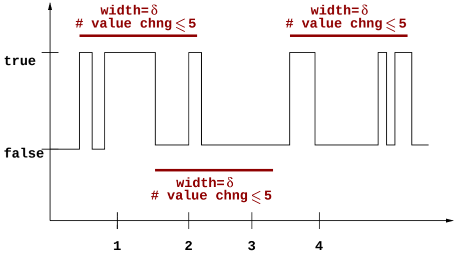
This non-standard semantics : **continuous-time**

- 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



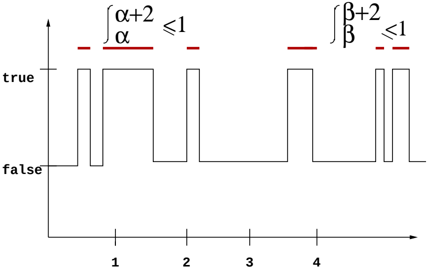
- express many **local temporal properties**
- and **prove** some of these properties

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



- express **stability specifications**.

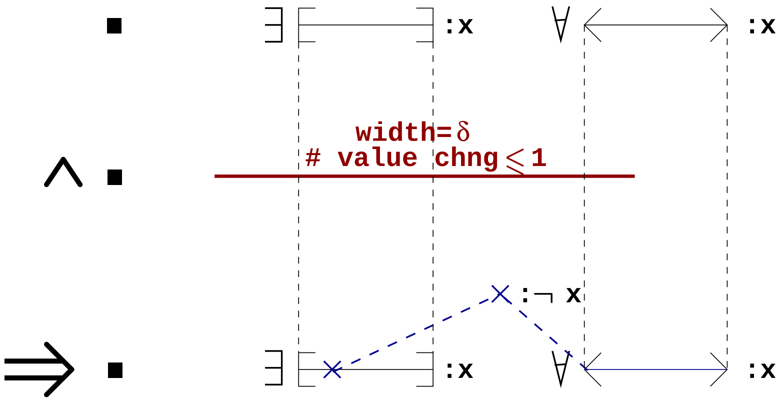
integral boundings



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

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

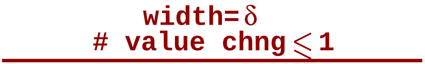
Reduce product Constraints - Value changes counting



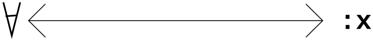
Reduce product Constraints - Value changes counting

- 

$\wedge$

- 

$\Rightarrow$

- 

Conclusion

- Abstract Interpretation is able to define a static analysis at several levels of the development of embedded systems
- It may help designers from early stages to product shipping
- It may even check that the translation from one level to another is correct
- Static analysis community can only benefit from a better formalization of different layers, as proposed by UML