Program Static Analysis: A Brief Introduction with Applications to the Internet

Patrick COUSOT

École Normale Supérieure, Paris, France

Patrick.Cousot@ens.fr www.di.ens.fr/~cousot

Introduction

Software Costs

- The cost of software is:
  - huge (e.g. 5 to 15 % of the cost of a plane),
  - increasing rapidly with the size of software (frequently 1 up to 40 000 000 lines!);

Software Costs

- The cost of software is:
  - huge (e.g. 5 to 15 % of the cost of a plane),
  - increasing rapidly with the size of software (frequently 1 up to 40 000 000 lines!);
- How to cut down costs and enhance software quality?
  - ...
  - Automate the reasonings about software (the early idea of using computers to reason about computers);
  - ...

Reasoning About Programs

We must be able to reason about programs:

- to design programs;
  - manually: e.g. coding,
  - automatically: e.g. program generation;

Reasoning About Programs

We must be able to reason about programs:

- to design programs;
  - manually: e.g. coding,
  - automatically: e.g. program generation;
- to manipulate programs:
  - manually: e.g. modification of a reused program,
  - automatically: e.g. compilation;

Reasoning About Programs

We must be able to reason about programs:

- to design programs;
  - manually: e.g. coding,
  - automatically: e.g. program generation;
- to manipulate programs:
  - manually: e.g. modification of a reused program,
  - automatically: e.g. compilation;
- to check program correctness:
  - manually: e.g. debuggers,
  - automatically: e.g. analyzers, provers.

Basis for Reasoning about Programs: Semantics

- The semantics of a computer system is the formal description of the behavior of this computer system when running in interaction with its environment.

Undecidability

- All questions about the semantics of a program are undecidable

Undecidability

- All (interesting) questions about the semantics of a program (written in a non trivial computer language) are undecidable (i.e. cannot be always and fully automatically answered with a computer in finite time);

Undecidability

- All (interesting) questions about the semantics of a program (written in a non trivial computer language) are undecidable (i.e. cannot be always and fully automatically answered with a computer in finite time);
- Examples of undecidable questions:
  - Is my program bug-free? (i.e. correct with respect to a given specification);
  - Can a program variable take two different values during execution?

Coping With Undecidable Questions on the Semantics

- Consider simple specifications or programs (hopeless);

Coping With Undecidable Questions on the Semantics

- Consider simple specifications or programs (hopeless);
- Consider decidable questions only or semi-algorithms (e.g. model-checking);

Coping With Undecidable Questions on the Semantics

- Consider simple specifications or programs (hopeless);
- Consider decidable questions only or semi-algorithms (e.g. model-checking);
- Ask the programmer to help (e.g. theorem proving);

Coping With Undecidable Questions on the Semantics

- Consider simple specifications or programs (hopeless);
- Consider decidable questions only or semi-algorithms (e.g. model-checking);
- Ask the programmer to help (e.g. theorem proving);
- Consider approximations to handle practical complexity limitations (the whole purpose of abstract interpretation).

Abstract Interpretation

The Theory of Abstract Interpretation

- **Abstract interpretation** is a theory of conservative approximation of the semantics of computer systems.

The Theory of Abstract Interpretation

- **Abstract interpretation** is a theory of conservative approximation of the semantics of computer systems.

  Approximation: observation of the behavior of a computer system at some level of abstraction, ignoring irrelevant details;

The Theory of Abstract Interpretation

- **Abstract interpretation** is a theory of conservative approximation of the semantics of computer systems.

  Approximation: observation of the behavior of a computer system at some level of abstraction, ignoring irrelevant details;

  Conservative: the approximation cannot lead to any erroneous conclusion.

Usefulness of Abstract Interpretation

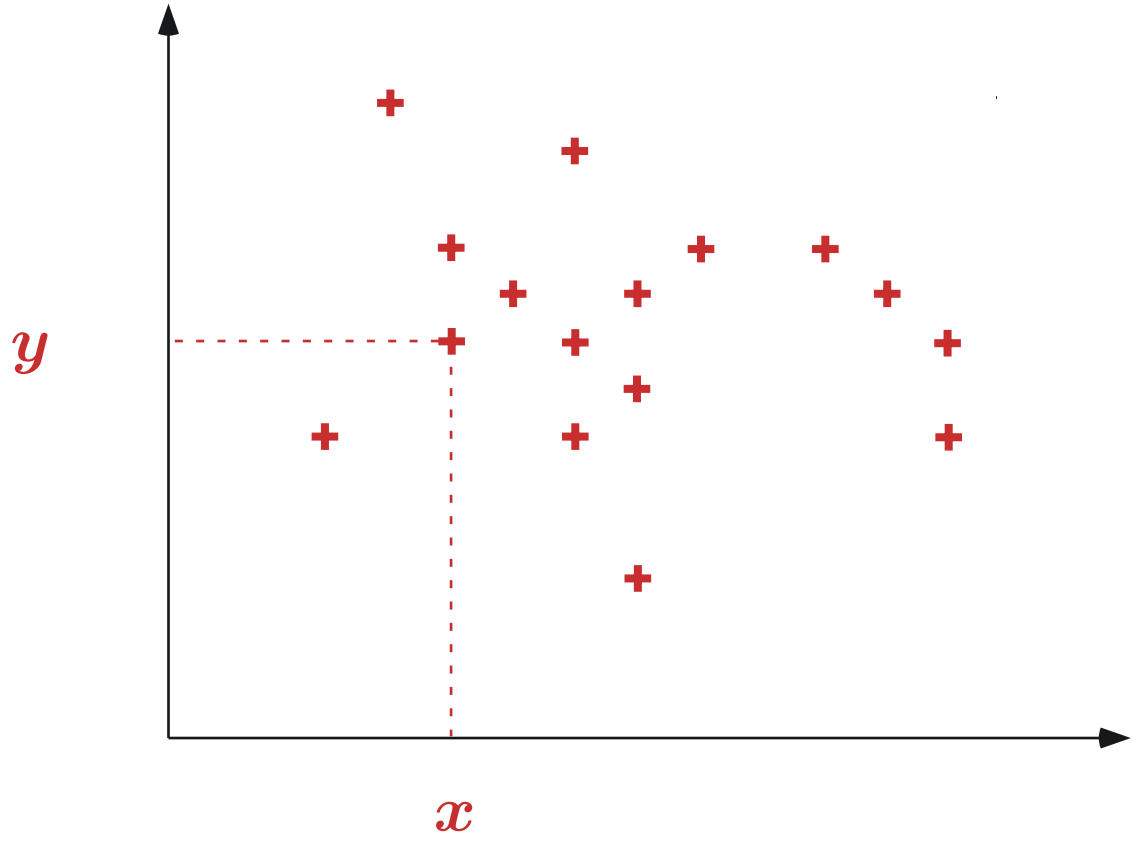
- **Thinking tools**: the idea of abstraction is central to reasoning (in particular on computer systems);

Usefulness of Abstract Interpretation

- **Thinking tools**: the idea of abstraction is central to reasoning (in particular on computer systems);
- **Mechanical tools**: the idea of effective approximation leads to automatic semantics-based program manipulation tools.

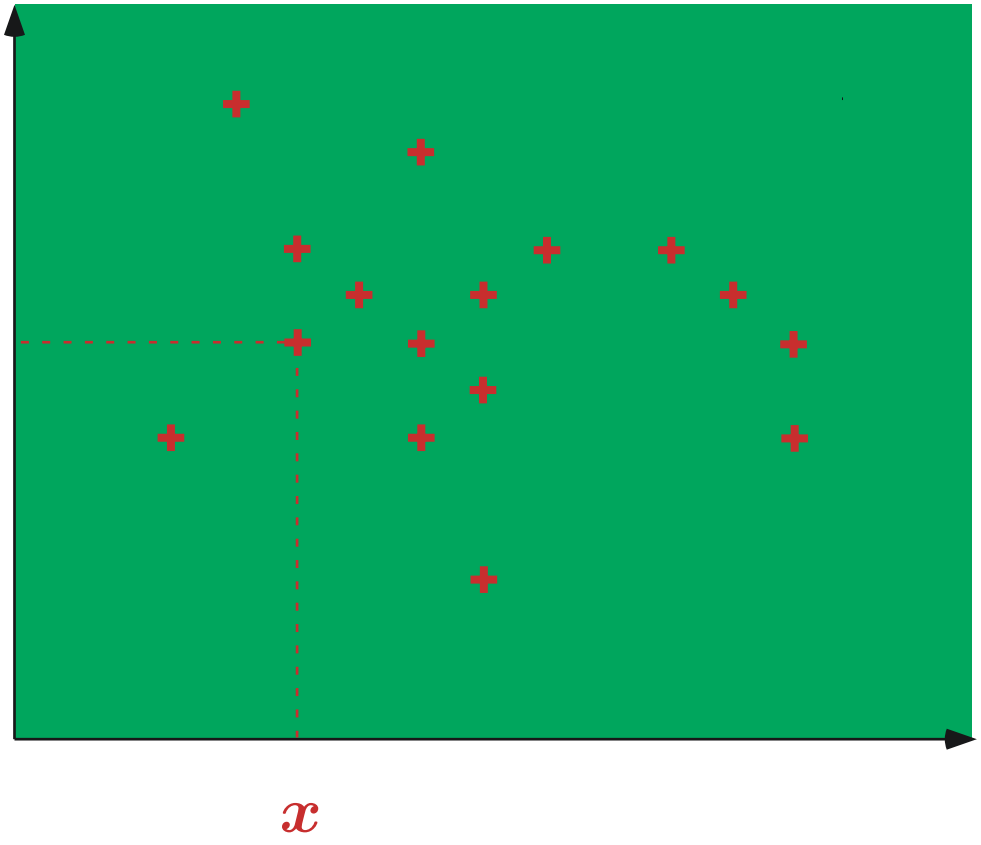
Example 1 of Abstraction (Sets of Points)

Approximations of an [in]finite set of points;



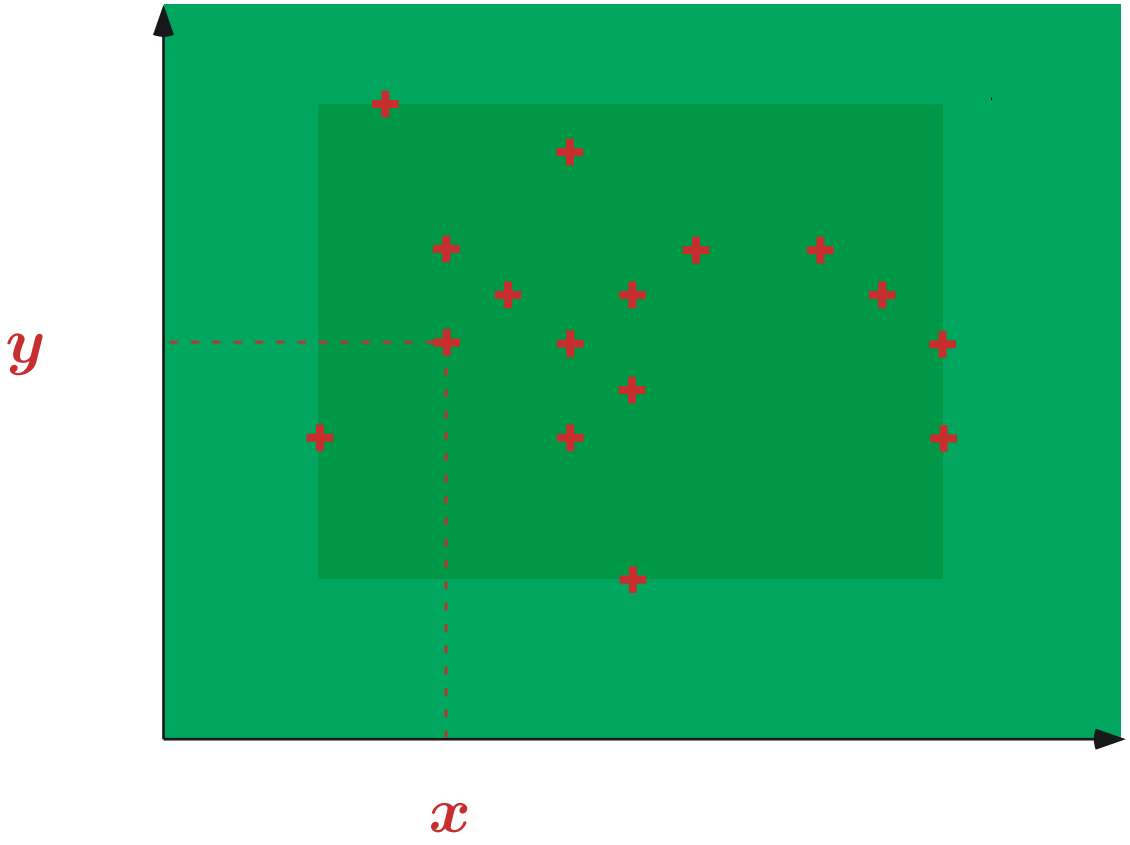
$$\{\ldots, \langle 19,\ 78\rangle, \ldots, \langle 20,\ 01\rangle, \ldots\}$$

Effective computable approximations of an [in]finite set of points; Signs



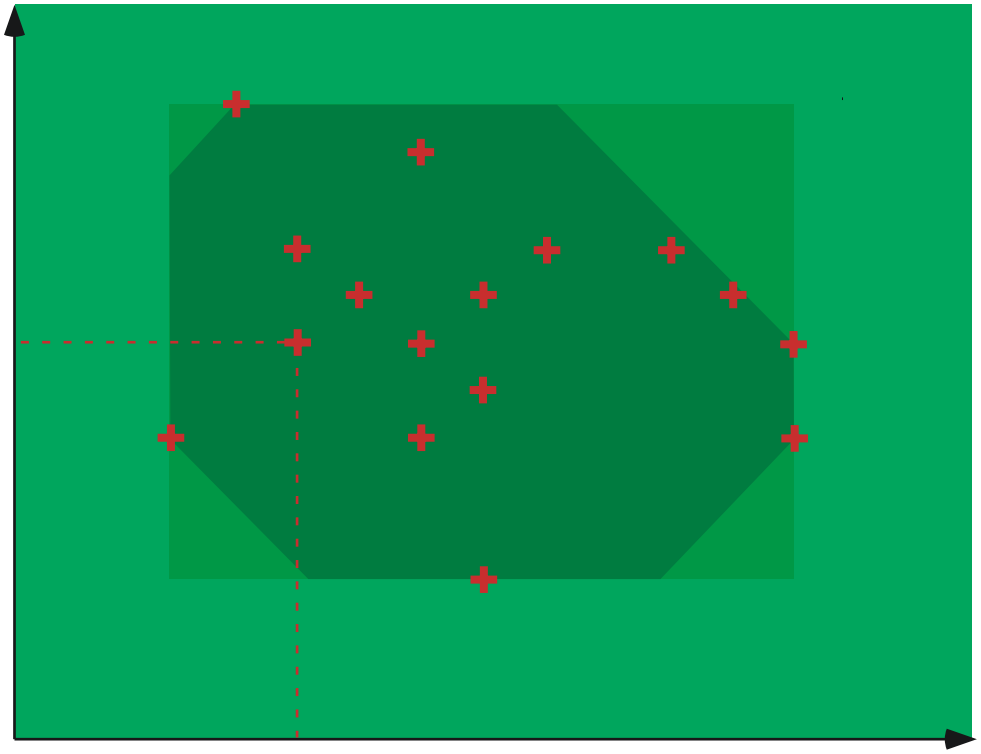
$$\begin{cases} x \geq 0 \\ y \geq 0 \end{cases}$$

Effective computable approximations of an [in]finite set of points; Intervals



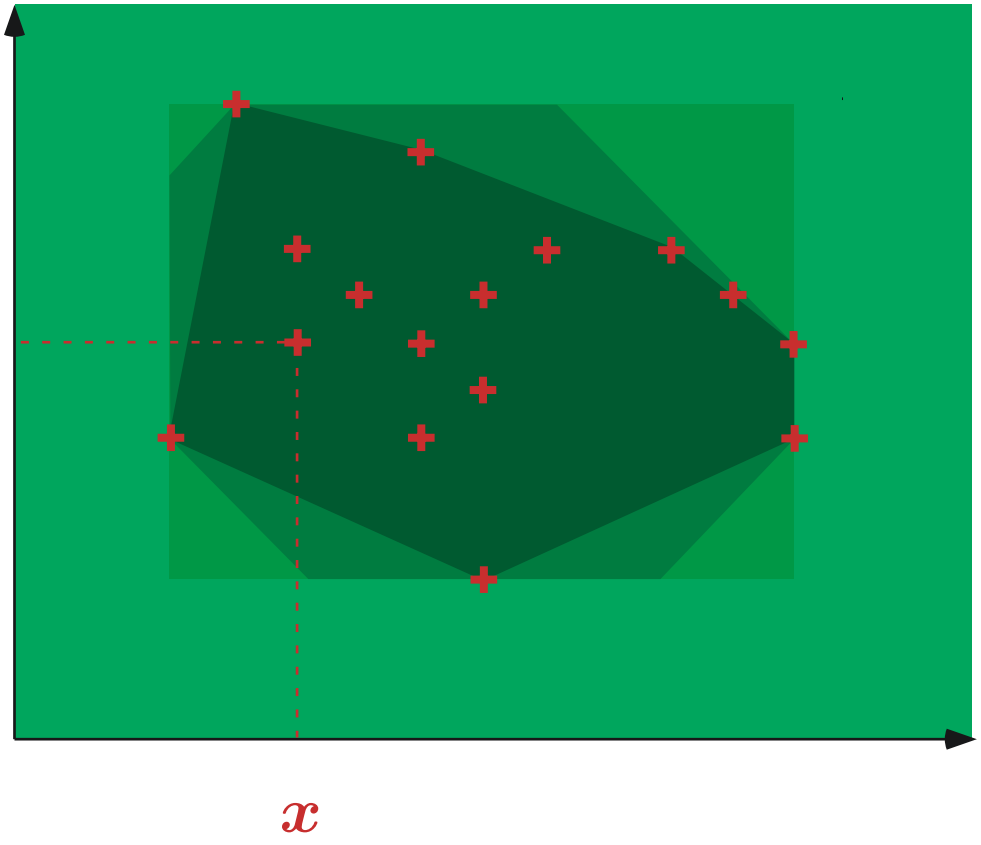
$$\begin{cases} x \in [19,\ 78] \\ y \in [20,\ 33] \end{cases}$$

Effective computable approximations of an [in]finite set of points; Octagons



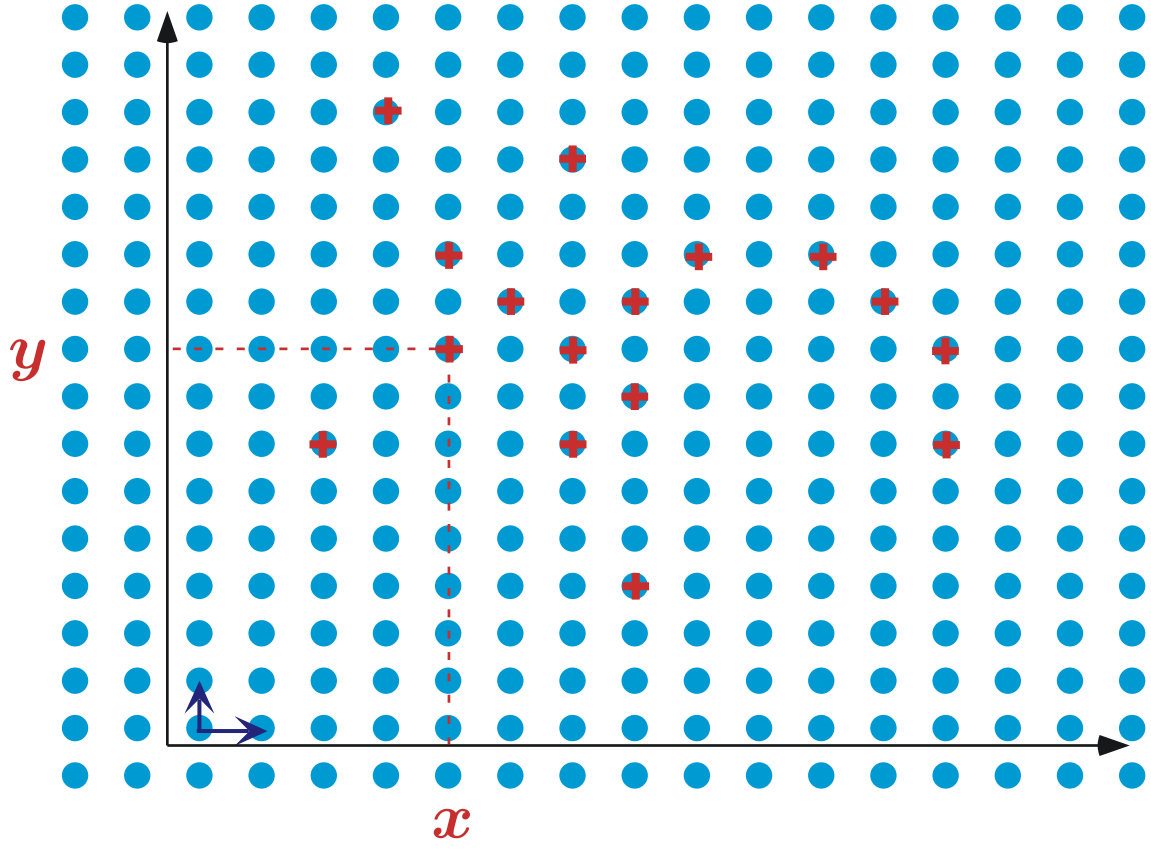
$$\begin{cases} 1 \leq x \leq 9 \\ x + y \leq 78 \\ 1 \leq y \leq 9 \\ x - y \leq 99 \end{cases}$$

Effective computable approximations of an [in]finite set of points; Polyhedra



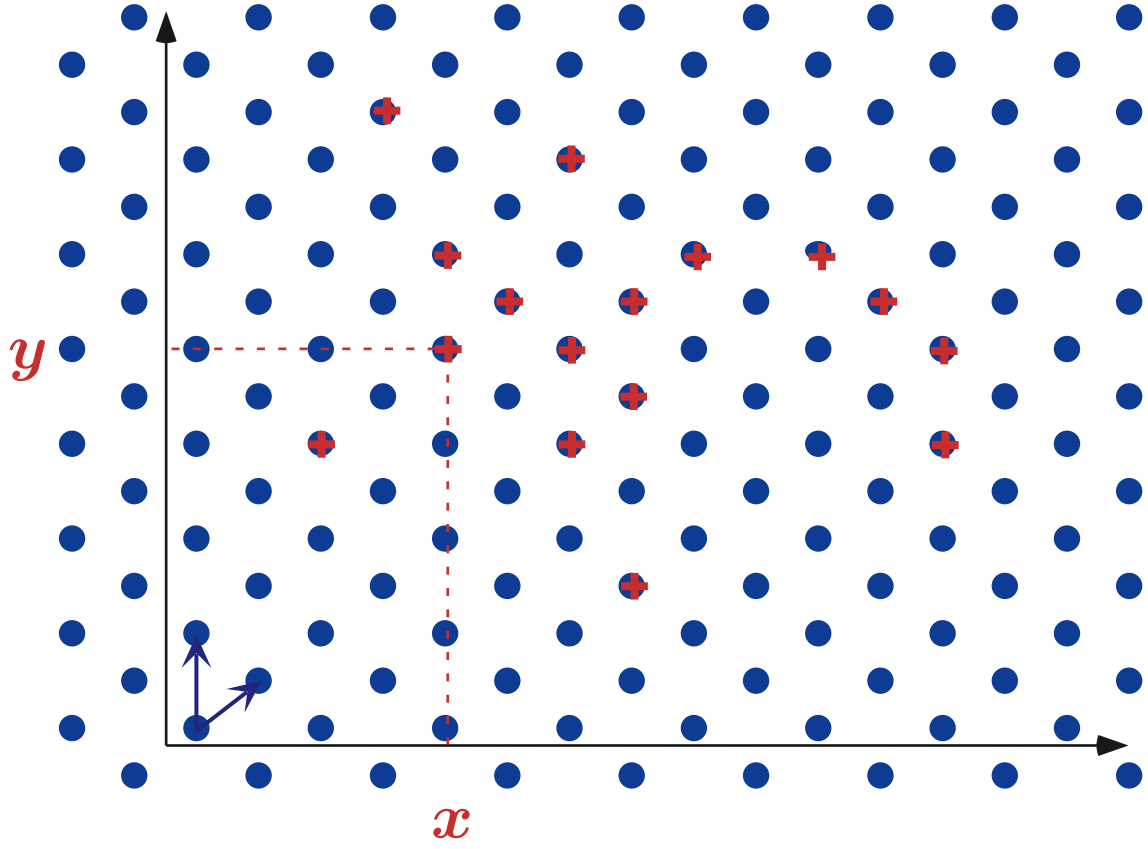
$$\begin{cases} 19x + 78y \leq 2000 \\ 20x + 01y \geq 0 \end{cases}$$

Effective computable approximations of an [in]finite set of points; Simple congruences



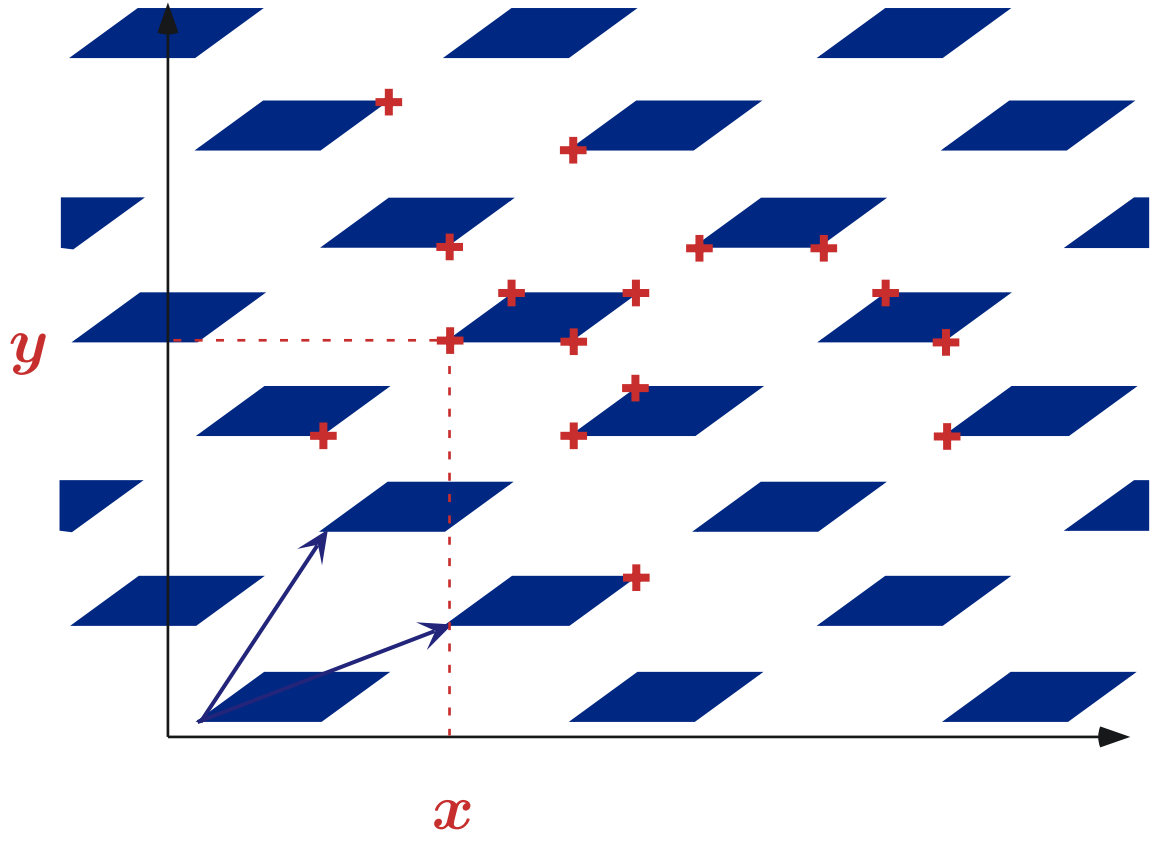
$$\begin{cases} x = 19 \bmod 78 \\ y = 20 \bmod 99 \end{cases}$$

Effective computable approximations of an [in]finite set of points; Linear congruences



$$\begin{cases} 1x + 9y = 7 \bmod 8 \\ 2x - 1y = 9 \bmod 9 \end{cases}$$

Effective computable approximations of an [in]finite set of points; Trapezoidal linear congruences



$$\begin{cases} 1x + 9y \in [0, 78] \bmod 10 \\ 2x - 1y \in [0, 99] \bmod 11 \end{cases}$$

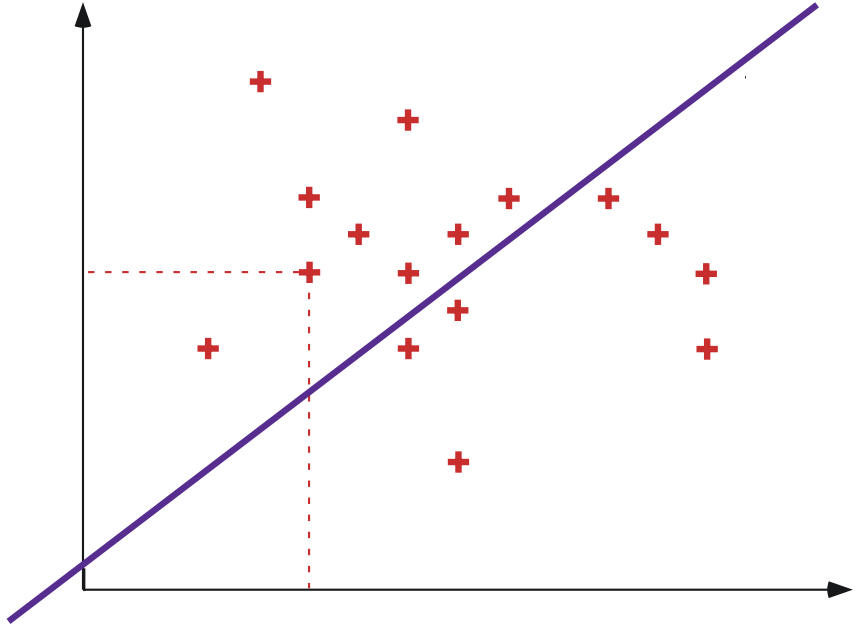
Intuition Behind Sound/Conservative Approximation for Example 1 (Sets of Points)

Conservative Approximation

- Is the operation `1/(x+1-y)` well defined at run-time?

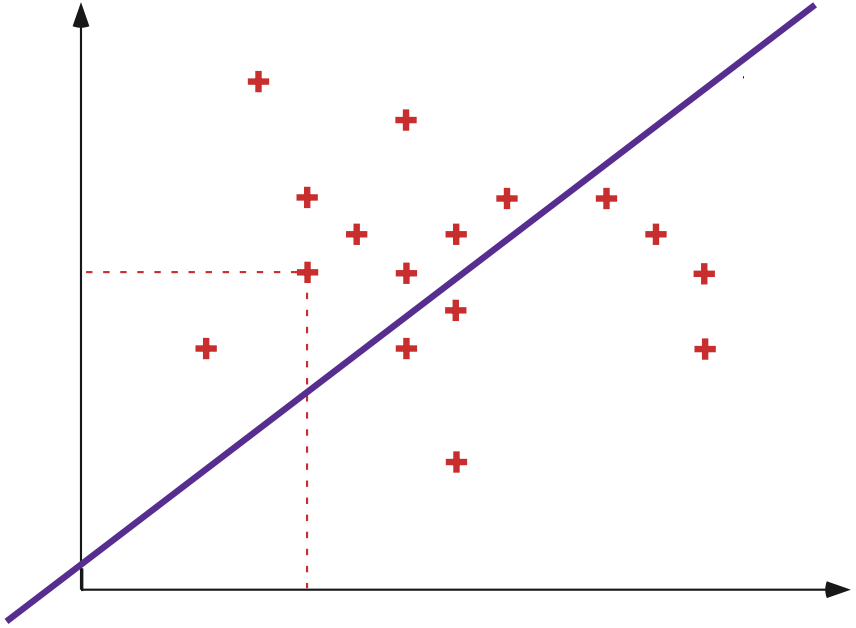
Conservative Approximation

- Is the operation `1/(x+1-y)` well defined at run-time?
- Concrete semantics:



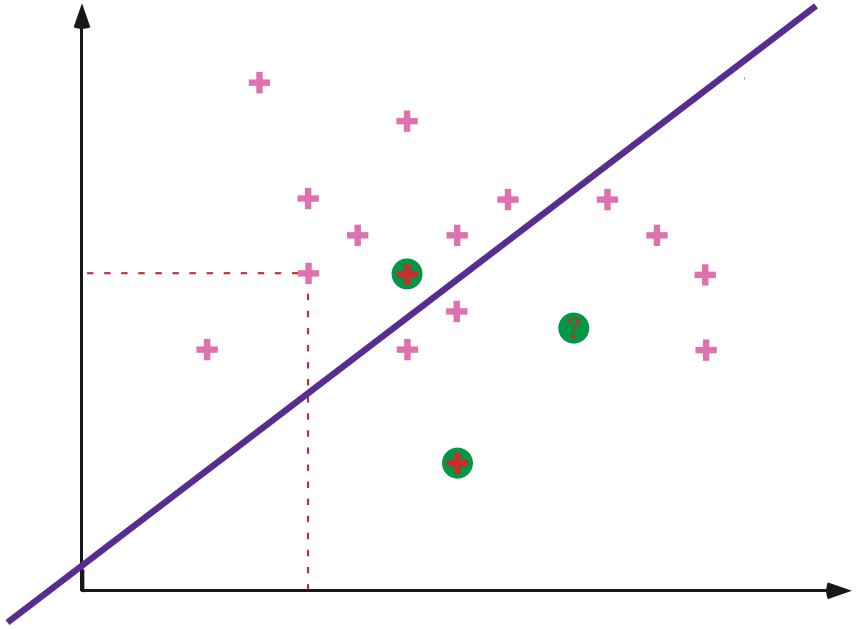
Conservative Approximation

- Is the operation `1/(x+1-y)` well defined at run-time?
- Concrete semantics: **yes**



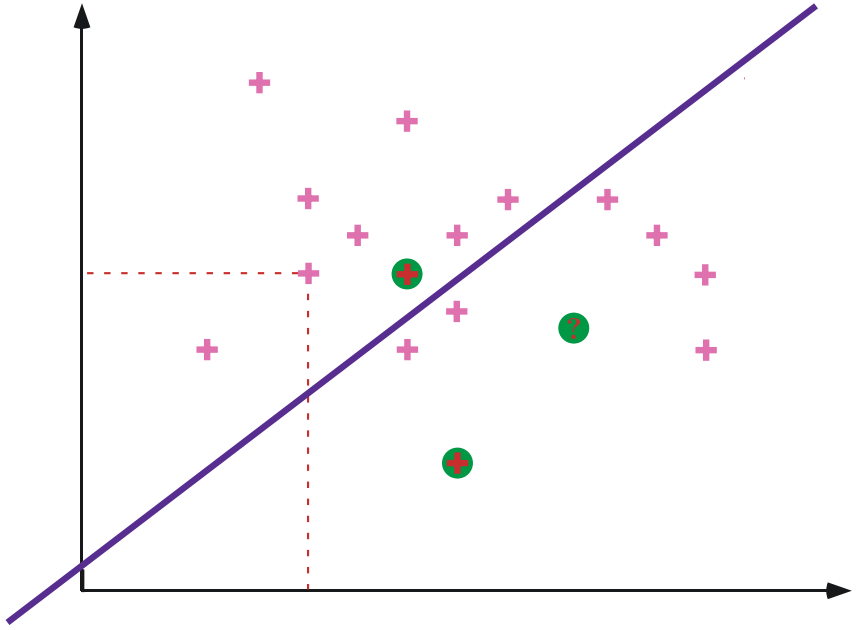
Conservative Approximation

- Is the operation `1/(x+1-y)` well defined at run-time?
- Testing :



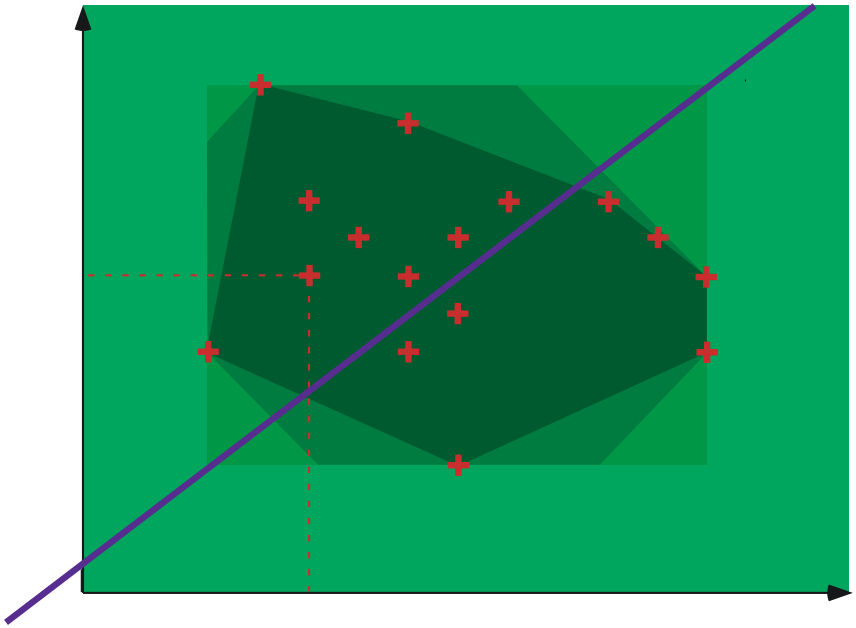
Conservative approximation

- Is the operation `1/(x+1-y)` well defined at run-time?
- Testing : **You <u>never</u> know!**



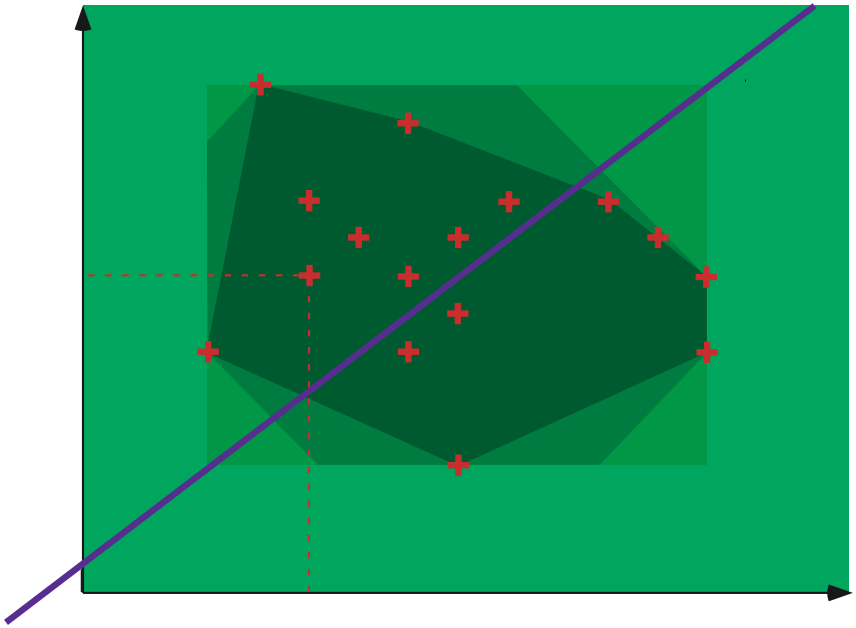
Conservative Approximation

- Is the operation `1/(x+1-y)` well defined at run-time?
- Abstract semantics 1:



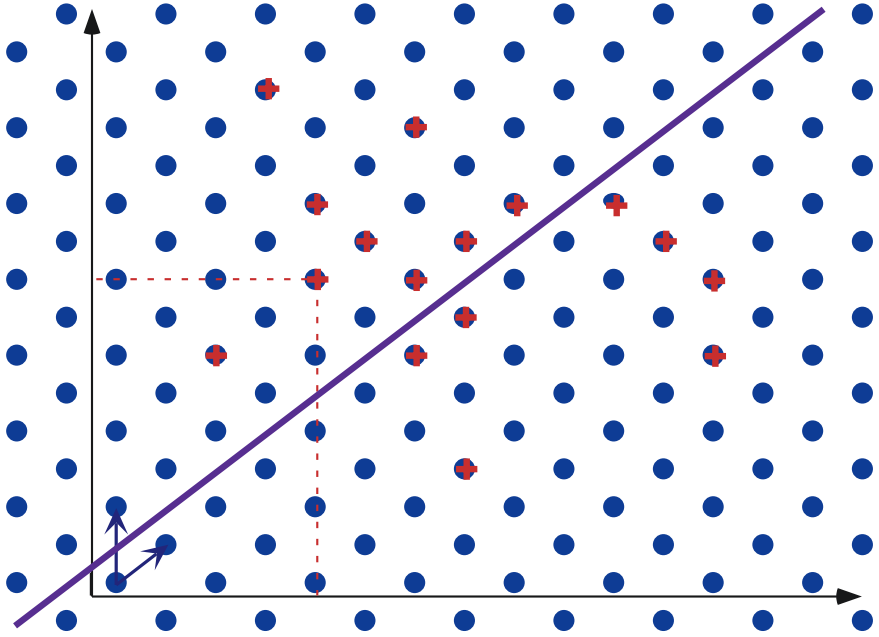
Conservative Approximation

- Is the operation `1/(x+1-y)` well defined at run-time?
- Abstract semantics 1: **I don't know**



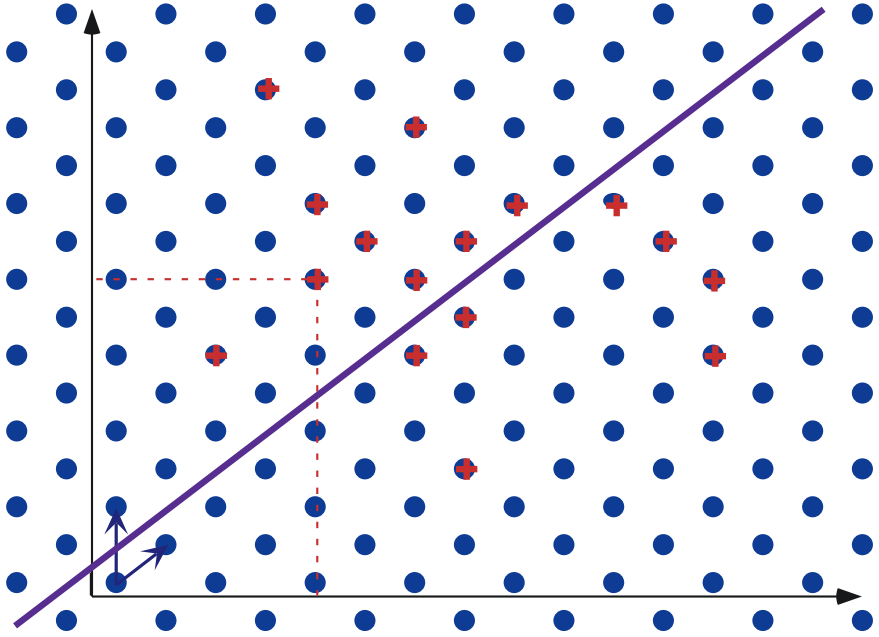
Conservative Approximation

- Is the operation `1/(x+1-y)` well defined at run-time?
- Abstract semantics 2:



Conservative Approximation

- Is the operation `1/(x+1-y)` well defined at run-time?
- Abstract semantics 2: **yes**



Example 2 of Abstraction (Texts)

Approximations of a Text (Set of Words)

- Choose a $\mathbf{thesaurus}$ (set of representative keywords);
- $\alpha(\mathbf{text}) = \mathbf{text} \cap \mathbf{thesaurus}$.

Intuition Behind Sound/Conservative Approximation for Example 2 (Texts)

Conservative Approximation

- Concrete question: Is some **word** in the **text**?

Conservative Approximation

- Concrete question: Is some $\mathbf{word}$ in the $\mathbf{text}$?
- Abstract answer (knowing only $\alpha(\mathbf{text})$):

Conservative Approximation

- Concrete question: Is some $\mathbf{word}$ in the $\mathbf{text}$?
- Abstract answer (knowing only $\alpha(\mathbf{text})$):
  - If $\mathbf{word} \in \alpha(\mathbf{text})$: **Yes**

Conservative Approximation

- Concrete question: Is some $\mathbf{word}$ in the $\mathbf{text}$?
- Abstract answer (knowing only $\alpha(\mathbf{text})$):
  - If $\mathbf{word} \in \alpha(\mathbf{text})$: **Yes**
  - If $\mathbf{word} \notin \alpha(\mathbf{text})$:

Conservative Approximation

- Concrete question: Is some $\mathbf{word}$ in the $\mathbf{text}$?
- Abstract answer (knowing only $\alpha(\mathbf{text})$):
  - If $\mathbf{word} \in \alpha(\mathbf{text})$: **Yes**
  - If $\mathbf{word} \notin \alpha(\mathbf{text})$: **I don't know**

Program Static Analysis by Abstract Interpretation

Program Static Analysis

- Static program analysis is the automatic compile-time determination of run-time properties of programs;
- Used in many applications from optimizing compilers, to abstract debuggers and semantics based program manipulation tools (such as partial evaluators, error detection and program understanding tools).

Abstract Interpretation

- Supporting theory;
- General idea: a program static analyzer computes an effective approximation of the program semantics (semantics = formal specification of all possible run-time behaviors).

Principle of Program Static Analysis

In order to determine runtime properties of a program P, a static analyzer:

- inputs the program P;
- builts a system of equations/constraints $X \sqsupseteq F[\![P]\!]X$;
- solves it $A \sqsupseteq \mathit{lfp}\, F$;
- outputs the solution A (in some user understandable form).

Example: Interval Analysis [1]

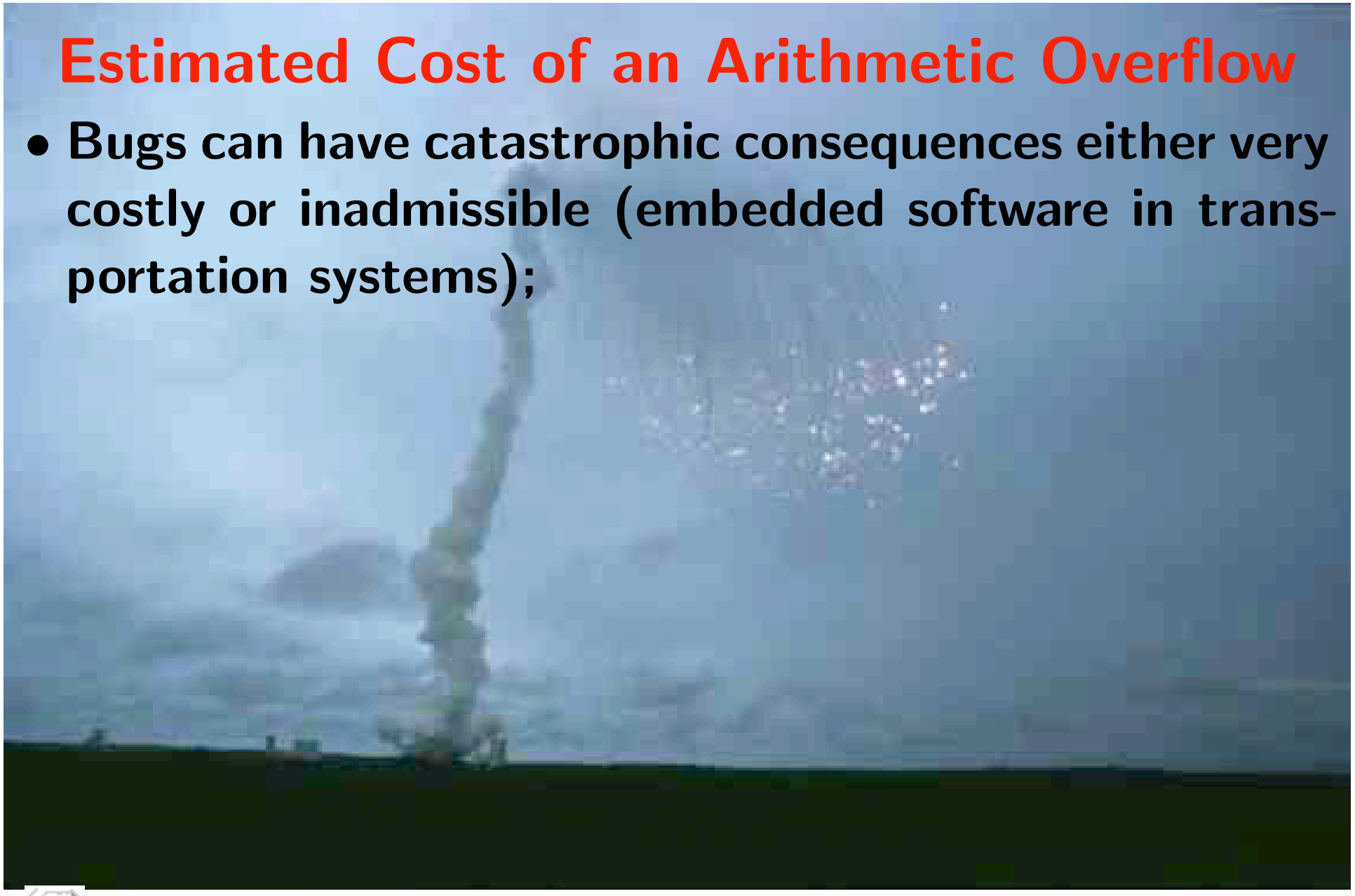
program	equations	solution
x := 1;		
1:		
while x < 10000 do	$X_1 = [1,1]$	$A_1 = [1,1]$
2:	$X_2 = (X_1 \cup X_3) \cap [-\infty, 9999]$	$A_2 = [1,9999]$
x := x + 1	$X_3 = X_2 \oplus [1,1]$	$A_3 = [2,10000]$
3:	$X_4 = (X_1 \cup X_3) \cap [10000, +\infty]$	$A_4 = [10000,10000]$
od;		
4:		

[1] P. Cousot & R. Cousot, ISOP'1976, POPL'77.

Examples of Applications to Embedded Systems

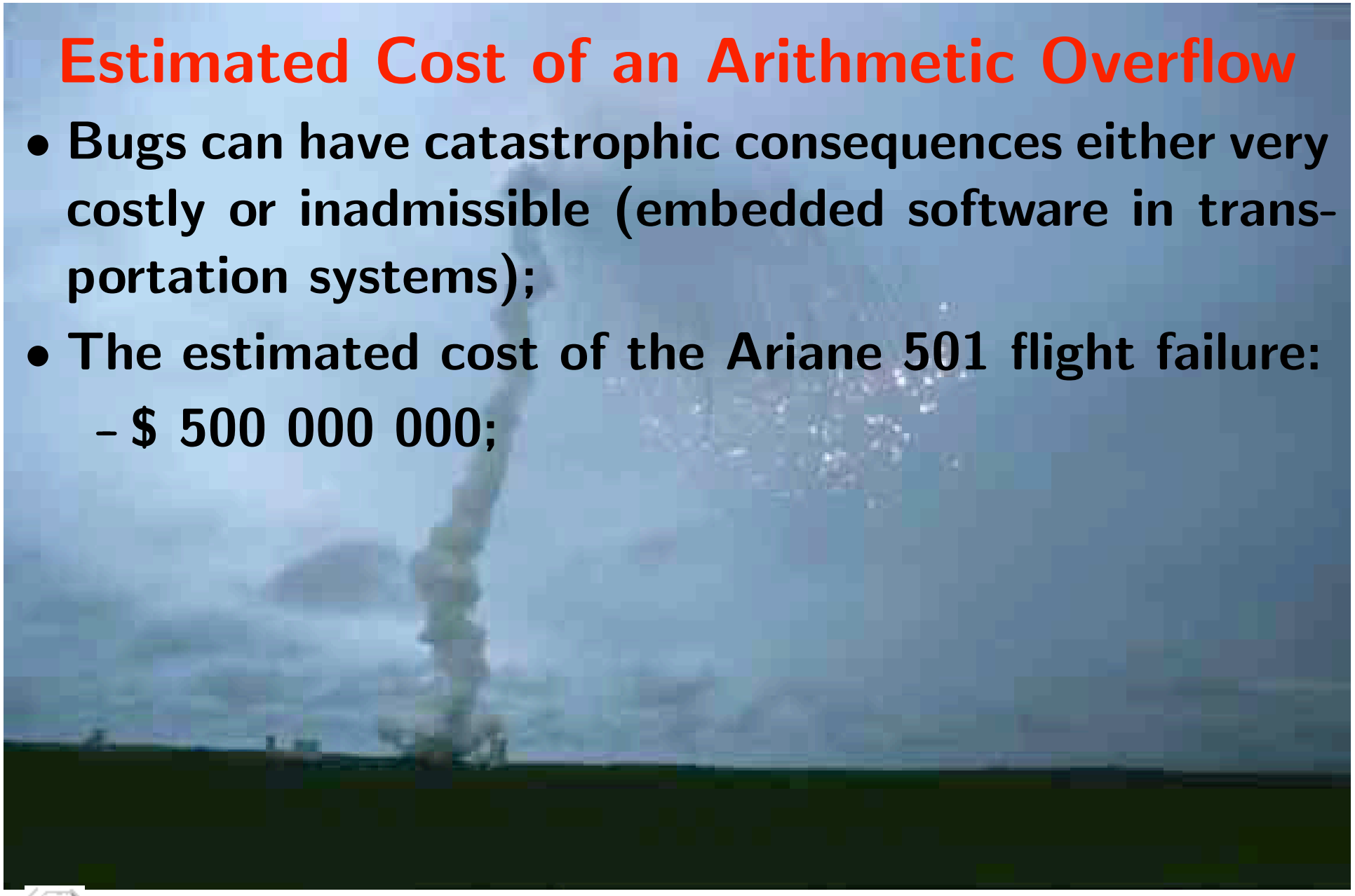
Estimated Cost of an Arithmetic Overflow

- **Bugs can have catastrophic consequences either very costly or inadmissible (embedded software in transportation systems);**



Estimated Cost of an Arithmetic Overflow

- Bugs can have catastrophic consequences either very costly or inadmissible (embedded software in transportation systems);
- The estimated cost of the Ariane 501 flight failure:
  - $ 500 000 000;



Estimated Cost of an Arithmetic Overflow

- Bugs can have catastrophic consequences either very costly or inadmissible (embedded software in transportation systems);
- The estimated cost of the Ariane 501 flight failure:
  - $ 500 000 000;
  - Including indirect costs (delays, lost markets, etc): $ 2 000 000 000.

An Impressive Application of Abstract Interpretation (1996/97)

- Abstract interpretation is used for the static analysis of the embedded ADA software of the Ariane 5 launcher [2];

[1] Flight software (60,000 lines of Ada code) and Inertial Measurement Unit (30,000 lines of Ada code).

An Impressive Application of Abstract Interpretation (1996/97)

- Abstract interpretation is used for the static analysis of the embedded ADA software of the Ariane 5 launcher [2];
- Automatic detection of the definiteness, potentiality, impossibility or inaccessibility of run-time errors [3];

[1] Flight software (60,000 lines of Ada code) and Inertial Measurement Unit (30,000 lines of Ada code).

[2] such as scalar and floating-point overflows, array index errors, divisions by zero and related arithmetic exceptions, uninitialized variables, data races on shared data structures, etc.

An Impressive Application of Abstract Interpretation (1996/97)

- Abstract interpretation is used for the static analysis of the embedded ADA software of the Ariane 5 launcher [2];
- Automatic detection of the definiteness, potentiality, impossibility or inaccessibility of run-time errors [3];
- Success for the following Ariane V flights and the ARD [4].

---

[1] Flight software (60,000 lines of Ada code) and Inertial Measurement Unit (30,000 lines of Ada code).

[2] such as scalar and floating-point overflows, array index errors, divisions by zero and related arithmetic exceptions, uninitialized variables, data races on shared data structures, etc.

[3] Atmospheric Reentry Demonstrator: module coming back to earth.

Examples of Applications to the Internet

Software Attacks

Principle of Data Overflow Based Attacks

- Pass data to a system procedure larger than the allocated memory size;

Principle of Data Overflow Based Attacks

- Pass data to a system procedure larger than the allocated memory size;
- If the system procedure does not check for possible overflow , that will overwrite some system memory area;

Principle of Data Overflow Based Attacks

- Pass data to a system procedure larger than the allocated memory size;
- If the system procedure does not check for possible overflow , that will overwrite some system memory area;
- If the system code is publicly available, the effect of this overwriting is perfectly understandable;

Principle of Data Overflow Based Attacks

- Pass data to a system procedure larger than the allocated memory size;
- If the system procedure does not check for possible overflow , that will overwrite some system memory area;
- If the system code is publicly available, the effect of this overwriting is perfectly understandable;
- This vulnerability can be used to overwrite the system code at some appropriate address with the attacking code.

Frequency of Data Overflow Based Attacks

- 60% of the UNIX™ failures reported in the 1995 FUZZ study are string-manipulation cleanness violations [5];

[4] http://www.cs.wisc.edu/~bart/fuzz/fuzz.html

Frequency of Data Overflow Based Attacks

- 60% of the UNIX™ failures reported in the 1995 FUZZ study are string-manipulation cleanness violations [5];
- In Nov. 1988, the Internet worm incident used a buffer overflow in `fingerid` to attack 60,000 computers [6];

[4] http://www.cs.wisc.edu/~bart/fuzz/fuzz.html

[5] E. Spafford. *The Internet worm: Crisis and aftermath. CACM 165, June 1989, pp. 678–687.*

Frequency of Data Overflow Based Attacks

- 60% of the UNIX™ failures reported in the 1995 FUZZ study are string-manipulation cleanness violations [5];
- In Nov. 1988, the Internet worm incident used a buffer overflow in `fingerid` to attack 60,000 computers [6];
- CERT advisories indicate that buffer overflows account for up to 50% of today's software vulnerabilities [7].

---

4 http://www.cs.wisc.edu/~bart/fuzz/fuzz.html

5 E. Spafford. *The Internet worm: Crisis and aftermath. CACM 165, June 1989, pp. 678–687.*

6 D. Wagner et al. A first step towards automated detection of buffer vulnerabilities, NDSS'00, San Diego, Feb. 2000.

Static Analysis of Data Overflow

- Present-day program static analysis technology can cope with data overflow;
- Abstraction:
  - $\alpha(\text{Data}) = \text{Data size}$,
  - $\alpha(\text{Set of vectors of integers}) = \text{Convex hull}$;
- Quite effective: very few false alarms (I don't know) on Unix code[8].

[7] N. Dor, M. Rodeh and M. Sagiv. In SAS'01, LNCS 2126, Springer, Jul. 2001, pp. 194–212.

Cryptographic Protocols

Cryptographic Protocols

- Cryptographic protocole: specifications for sequences of messages to be exchanged by machines on an insecure network to establish private or authenticated communication;

Cryptographic Protocols

- Cryptographic protocole: specifications for sequences of messages to be exchanged by machines on an insecure network to establish private or authenticated communication;
- Used to distribute sensitive information (classified material, credit card numbers or trade secrets) or to create digital signatures;

Cryptographic Protocols

- Cryptographic protocole: specifications for sequences of messages to be exchanged by machines on an insecure network to establish private or authenticated communication;
- Used to distribute sensitive information (classified material, credit card numbers or trade secrets) or to create digital signatures;
- Flaws in cryptographic protocols: an intruder who is able to read, suppress and forge messages can get sensitive information or take someone else identity;

Cryptographic Protocols

- Cryptographic protocole: specifications for sequences of messages to be exchanged by machines on an insecure network to establish private or authenticated communication;
- Used to distribute sensitive information (classified material, credit card numbers or trade secrets) or to create digital signatures;
- Flaws in cryptographic protocols: an intruder who is able to read, suppress and forge messages can get sensitive information or take someone else identity;
- Even when assuming perfect cryptography.

Abstraction of Cryptographic Protocols

- Impossible to check all possible sequences of messages that can be exchanged between principals and their possible corruptions by an intruder;

Abstraction of Cryptographic Protocols

- Impossible to check all possible sequences of messages that can be exchanged between principals and their possible corruptions by an intruder;
- Abstract into a superset (using tree automata) and compute an overestimation of all concrete possibilities;

Abstraction of Cryptographic Protocols

- Impossible to check all possible sequences of messages that can be exchanged between principals and their possible corruptions by an intruder;
- Abstract into a superset (using tree automata) and compute an overestimation of all concrete possibilities;
- The abstract model is infinite so use convergence acceleration to enforce the abstract computations to be finite;

Example of Possible Intrusion

- Otway-Rees protocol [9]:
  - $\texttt{encrypt(pair(N}_\texttt{a}\texttt{; pair(M;pair(A;B)));K}_\texttt{as}\texttt{)}$: an intruder can discover the secret message;
  - $\texttt{encrypt(pair(pair(N}_\texttt{a}\texttt{;M); pair(A;B));K}_\texttt{as}\texttt{)}$: intrusion impossible;

[8] D. Monniaux, *Abstracting Cryptographic Protocols with Tree Automata*, SAS'99, LNCS 1694, Springer, pp. 149–163.

Example of Possible Intrusion

- Otway-Rees protocol [10]:
  - $\texttt{encrypt(pair(N}_\texttt{a}\texttt{; pair(M;pair(A;B)));K}_\texttt{as}\texttt{)}$: an intruder can discover the secret message;
  - $\texttt{encrypt(pair(pair(N}_\texttt{a}\texttt{;M); pair(A;B));K}_\texttt{as}\texttt{)}$: intrusion impossible;
- A very subtle error.

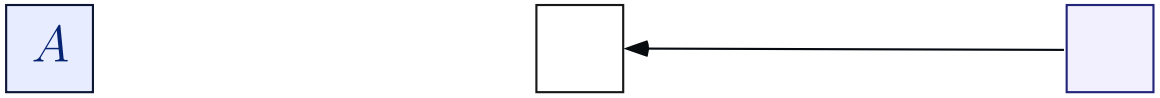
[9] D. Monniaux, *Abstracting Cryptographic Protocols with Tree Automata*, SAS'99, LNCS 1694, Springer, pp. 149–163.

Mobility

On the Evolution of the Internet

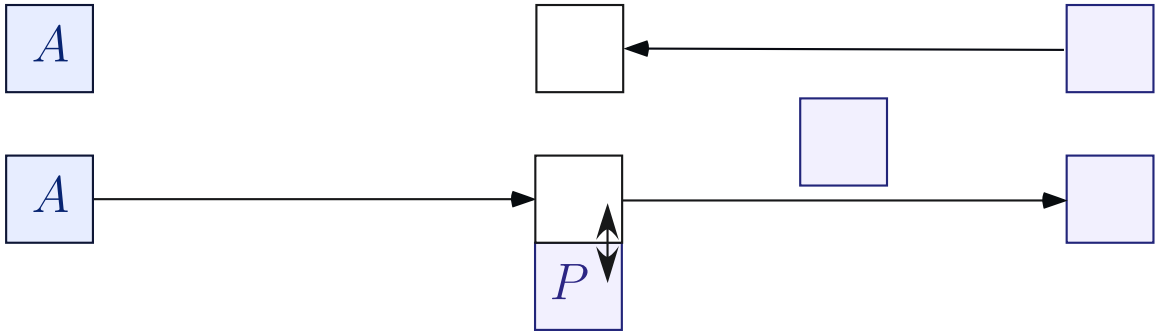
- The Internet protocols have evolved from:
  - Static: e.g. email path routing;
  - Dynamic:
    - Data driven: e.g. fixed software + routing tables,
    - Software driven: (future) mobile software routing.

Data Confidentiality in Mobile Processes [5]



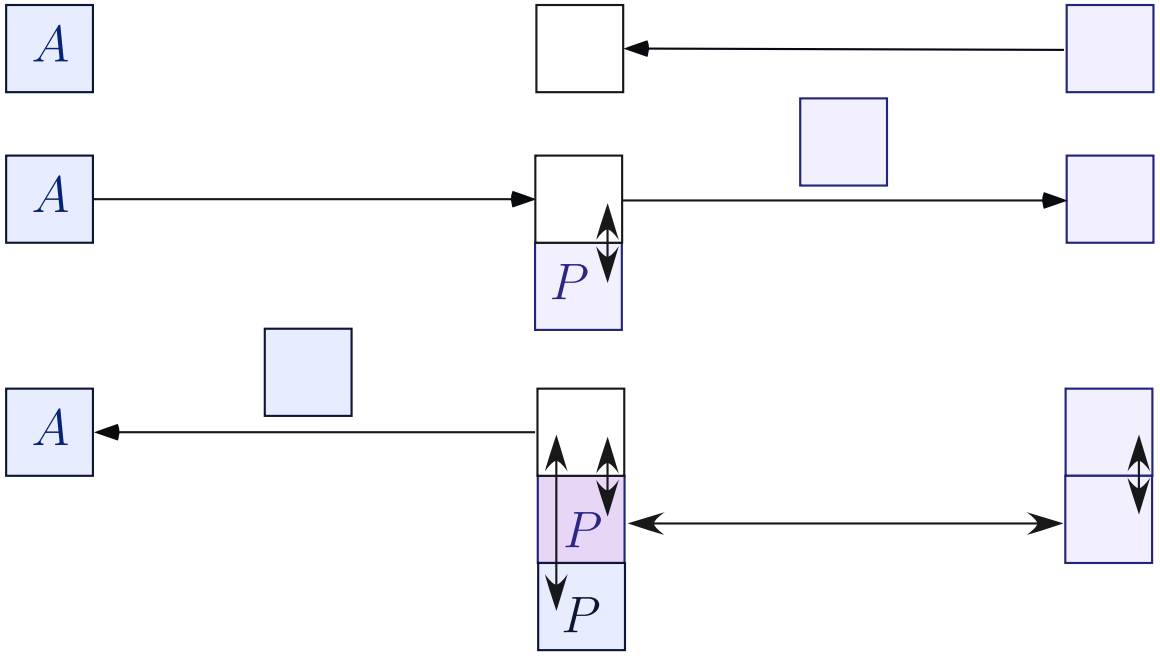
[4] J. Feret, SAS'00, ENTCS Vol. 39

Data Confidentiality in Mobile Processes [5]



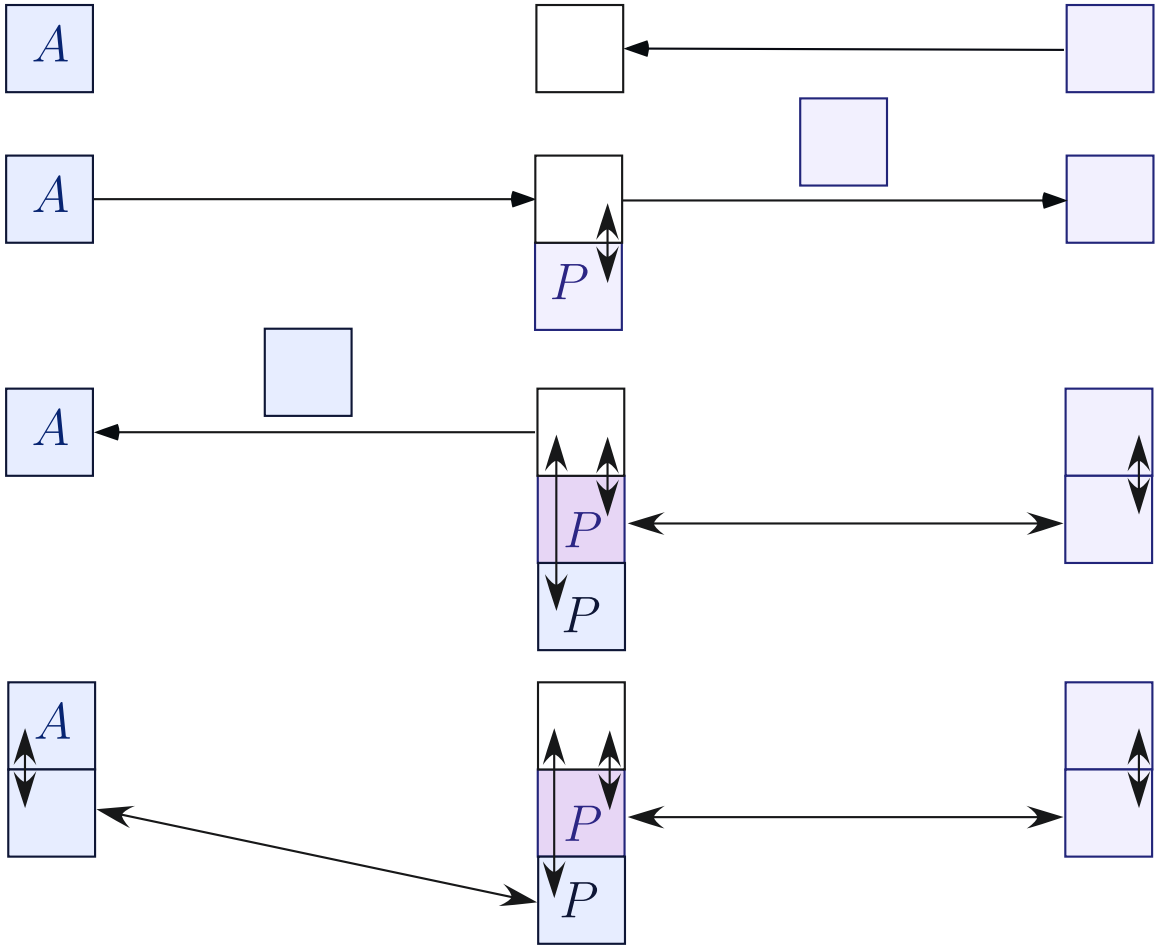
[4] J. Feret, SAS'00, ENTCS Vol. 39

Data Confidentiality in Mobile Processes [5]



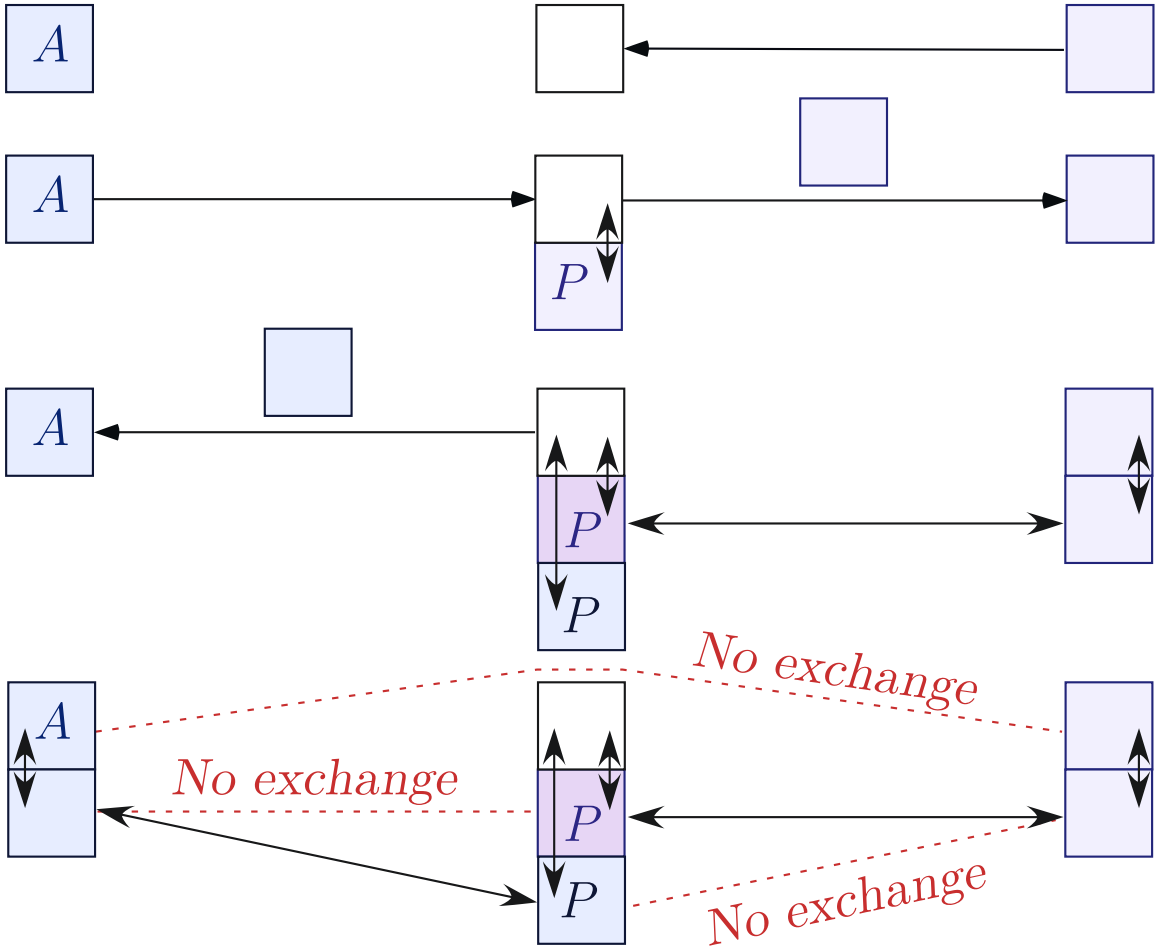
[4] J. Feret, SAS'00, ENTCS Vol. 39

Data Confidentiality in Mobile Processes [5]



[4] J. Feret, SAS'00, ENTCS Vol. 39

Data Confidentiality in Mobile Processes [5]



[4] J. Feret, SAS'00, ENTCS Vol. 39

Cookies

Present Criteria for Accepting Cookies

- Possible choices:
  1. Refuse all cookies;
  2. Accept all cookies;
  3. Accept all cookies of a site;
  4. Decide online when a cookie is loaded.

Present Criteria for Accepting Cookies

- Possible choices:
  1. Refuse all cookies;
  2. Accept all cookies;
  3. Accept all cookies of a site;
  4. Decide online when a cookie is loaded.
- 4. seems a good choice, but:
  - Cookies can arrive at a high rate (several per minute);
  - The information to decide about acceptable cookies is poor.

Which Information is Available on Cookies?

- Example of cookie properties:

Name : B
Server : yahoo.com
Path : /
Value : 7fe7d2otmjhbg&b=2
Expires on : Tue 15 Apr 2010 20:00 GMT
Secured : No
State : Activated

Which Information is Available on Cookies?

- Example of cookie properties:

  Name : B
  Server : yahoo.com
  Path : /
  Value : 7fe7d2otmjhbg&b=2
  Expires on : Tue 15 Apr 2010 20:00 GMT
  Secured : No
  State : Activated

- How can I decide anything about this cookie with such poor information?

Choosing Cookies According to their Behavior

- The ideal situation would be an acceptance/refusal decision that would be:
  - online (since some sites only work well with cookies);
  - automatic (to cope with high incoming rates);
  - based on the cookie behavior.

Choosing Cookies According to their Behavior

- The ideal situation would be an acceptance/refusal decision that would be:
  - online (since some sites only work well with cookies);
  - automatic (to cope with high incoming rates);
  - based on the cookie behavior.
- Automatic checking of acceptable cookies is possible with present-day static program analysis technology.

Example of Behavioral Acceptance/Refusal Decision Criteria

- Possible user-defined acceptance criterion:

  Always in the future:

  if Read private data then afterwards

  Never in the future:

  Send message on the Internet

Example of Behavioral Acceptance/Refusal Decision Criteria

- Possible user-defined acceptance criterion:

  Always in the future:

  if Read private data then afterwards

  Never in the future:

  Send message on the Internet

- Such security policies might be predefined for casual users.

Conclusion

Future of Internet

- From data to software (services);
- Towards mobility and global computing;
- From naive to sophisticated internauts;

Future of Static Analysis by Abstract Interpretation on the Internet

- Precaution principle:

  Never run any software which potential disastrous effects cannot be anticipated.
- Software static analysis is a good candidate for such online verification (simple analyzes are already present in the JVM);
- Abstract Interpretation is the supporting theory of a corroborative practice.

THE END

THE END, THANK YOU.