BOXES: Abstract Domain of Boxes

Arie Gurfinkel and Sagar Chaki

Software Engineering Institute
Carnegie Mellon University

January 28, 2011

This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

Software Engineering Institute (SEI)

Department of Defense R&D Laboratory (FFRDC)

Created in 1984

Under contract to Carnegie Mellon University

Offices in Pittsburgh, PA; Washington, DC; and Frankfurt, Germany

SEI Mission: advance software engineering and related disciplines to ensure the development and operation of systems with predictable and improved cost, schedule, and quality.

SEI Technical Programs

Networked Systems Survivability (CERT)

- Secure Software and Systems
- Cyberthreat and Vulnerability Analysis
- Enterprise Workforce Development
- Forensics

Software Engineering Process Management (SEPM)

- Capability Maturity Model Integration (CMMI)
- Team Software Process (TSP)
- Software Engineering Measurement and Analysis (SEMA)

Acquisition Support (ASP)

Research, Technology, and System Solutions (RTSS)

- Architecture-Centric Engineering
- Product Line Practice
- System of Systems Practice
- System of Systems Software Assurance
- Ultra-Large-Scale (ULS) System Perspective

Independent Research and Development (IR&D)

Research, Technology, and System Solutions (RTSS) Program

Mission

Discover the mutual influences of **structure and behavior for software-reliant systems at all scales** to assure key quality attributes for the achievement of business and mission goals.

Vision

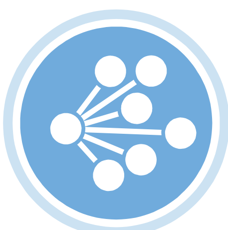
Assured and flexible
system capabilities at all scales

Problems Faced by the DoD

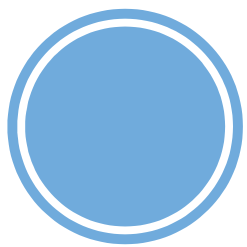
DoD’s ability to rapidly develop and field software-reliant capability across a variety of systems is deficient. Part of the reason rests with DoD’s acquisition process; part of the reason is technical.

Software systems science and engineering are inadequate to

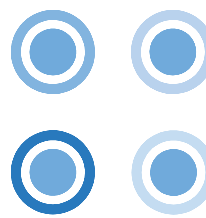
- determine how to structure and adapt systems at all scales
- manage interactions among these types of systems
- assure software-reliant capabilities that are sufficiently reliable, secure, responsive, and adaptable to change



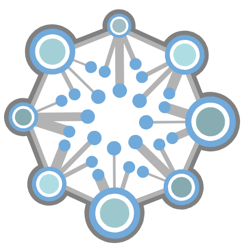
Embedded systems
software embedded in hardware devices



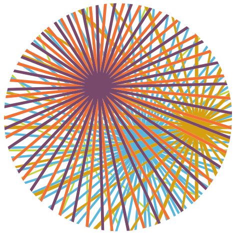
Stand-alone systems
software applications



Software product lines
families of similar systems



Systems of systems
federations of independent systems



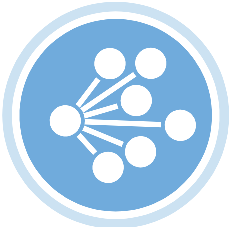
Ultra-large-scale systems
webs of software-reliant systems, people, economies, and cultures

Predict and control behavior

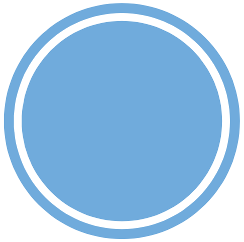
Assure and bound behavior

Coupling to organizational structure and practices increases

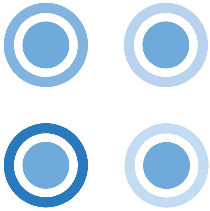
RTSS: Research Mapping



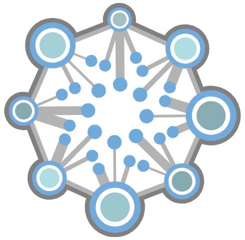
Embedded systems



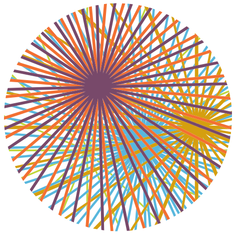
Stand-alone systems



Software product lines



Systems of systems



Ultra-large-scale systems

Product Line Practice focus

SoS Practice focus

- Engineering and technology for SoS
- Integrated practices for SoS

SoS Software Assurance focus

- Failure patterns and mitigations in SoS
- Theories, Principles, and Methods
- Barriers/incentives to assurance technology adoption

Architecture-Centric Engineering addresses all scales of systems

- Quality attribute foundations and analysis
- Architecture-centric practices
- Architecture principles for ULS systems

Outline

Software Engineering Institute (SEI)

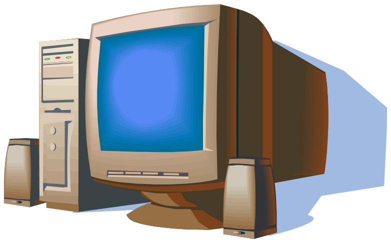
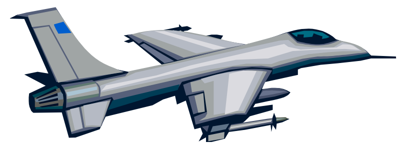
Introduction

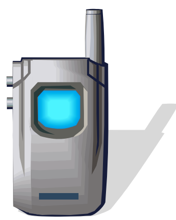
Basic Abstract Interpretation

Boxes Abstract Domain

Conclusion

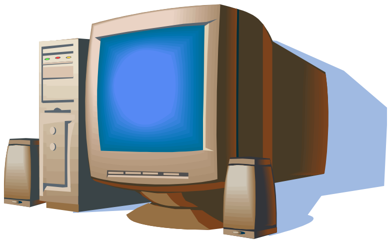
Software is Everywhere

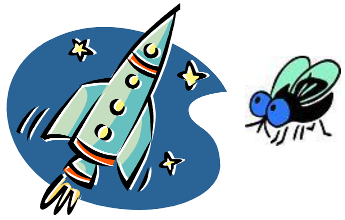







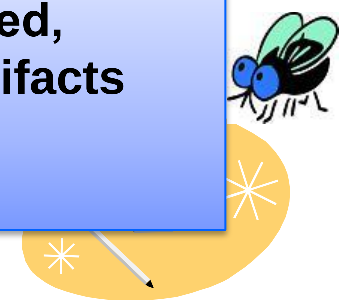

Software is Full of Bugs!





“Software easily rates as the most poorly constructed, unreliable, and least maintainable technological artifacts invented by man”

Paul Strassman, former CIO of Xerox



Software Bugs are Expensive!

Intel Pentium FDIV Bug

- Estimated cost: $500 Million

Y2K bug

- Estimated cost: >$500 Billion

Northeast Blackout of 2003

- “a programming error identified as the cause of alarm failure”
- Estimated cost: $6-$10 Billion

“The cost of software bugs to the U.S. economy is estimated at $60 B/year”

NIST, 2002

Some Examples of Software Disasters

Between 1985 and 1987, **Therac-25** gave patients massive overdoses of radiation, approximately 100 times the intended dose. Three patients died as a direct consequence.

On February 25, 1991, during the Gulf War, an American **Patriot Missile** battery in Dharan, Saudi Arabia, failed to track and intercept an incoming Iraqi Scud missile. The Scud struck an American Army barracks, killing 28 soldiers and injuring around 100 other people.

On June 4, 1996 an unmanned **Ariane 5** rocket launched by the European Space Agency forty seconds after lift-off. The rocket was on its first voyage, after a decade of development costing $7 billion. The destroyed rocket and its cargo were valued at $500 million.

Details at http://www5.in.tum.de/~huckle/bugse.html

Recent Examples

In July 2010, The Food and Drug Administration ordered Baxter International to recall all of its Colleague infusion pumps in use and provide a refund or no-cost replacement to United States customers. It has been working with Baxter since 1999 to correct numerous device flaws. Some of the issues were caused by simple buffer overflow.

In December 2010, the Skype network went down for 3 days. The source of the outage was traced to a software bug in Skype version 5.

In January 2011, two German researchers have shown that most “feature” mobile phones can be “killed” by sending a simple SMS message (**SMS of Death**). The attack exploits many bugs in the implementation of SMS protocol in the phones. It can potentially bring down all mobile communication…

Why so many bugs?

Software Engineering is very complex

- Complicated algorithms
- Many interconnected components
- Legacy systems
- Huge programming APIs
- …

Software Engineers need better tools to deal with this complexity!

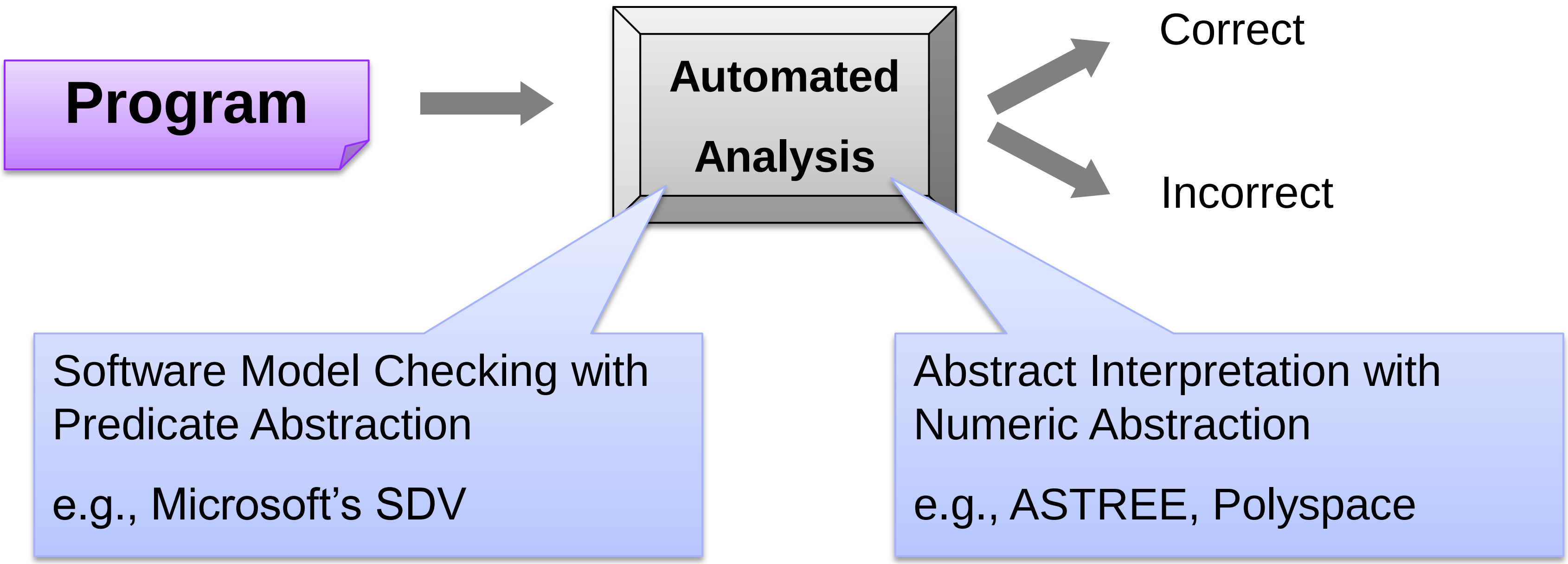
What Software Engineers Need Are …

Tools that give better confidence than testing while remaining easy to use

And at the same time, are

- … fully automatic
- … (reasonably) easy to use
- … provide (measurable) guarantees
- … come with guidelines and methodologies to apply effectively
- … apply to real software systems

Automated Software Analysis



Numeric Abstract Interpretation

Analysis is restricted to a fixed Abstract Domain

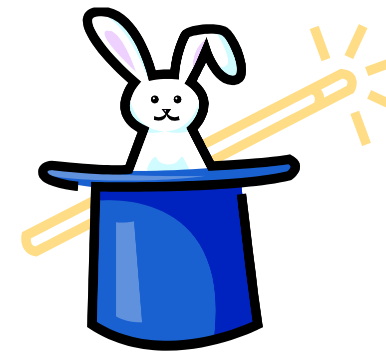
Abstract Domain ≡ "a (possibly infinite) set of predicates from a fixed theory" + efficient (*abstract*) operations

Common Numeric Abstract Domains

Abstract Domain	Abstract Elements
Sign	$0 < x, \quad x = 0, \quad x > 0$
Box (or Interval)	$c_1 \le x \le c_2$
Octagon	$\pm x \pm y \le c$
Polyhedra	$a_1x_1 + a_2x_2 + a_3x_3 + a_4 \le 0$

Legend	
x,y	program variables
c, c_i, a_i	numeric constants

Abstract Interpretation w/ Box Domain (1)



widening

Program

```
x := 0

while (x < 1000) {

  x := x + 1;

}

assert (x == 1000);
```

0<= x <=2

x = 0

0<= x <=2

x = 0

1<= x <=3

0<= x <=1000

0<= x < 1000

1<= x <= 1000

x = 1000

Steps: 1 2 3 4 5 6 7 8 9 10 11 12 13 14

Abstract Domain as an Interface

interface AbstractDomain(V) :

- V – set of variables
- A – abstract elements
- E – expressions
- S – statements

abstract

concretize

$\boldsymbol{\alpha} : E \rightarrow A$	$\boldsymbol{\gamma} : A \rightarrow E$	**meet** : $A \times A \rightarrow A$
isTop : $A \rightarrow$ bool	**isBot** : $A \rightarrow$ bool	**join** : $A \times A \rightarrow A$
leq : $A \times A \rightarrow$ bool	$\boldsymbol{\alpha}$**Post** : $S \rightarrow (A \rightarrow A)$	**widen** : $A \times A \rightarrow A$

order

abstract transformer

All operations are over-approximations, e.g.,

$$\boldsymbol{\gamma}(a) \,||\, \boldsymbol{\gamma}(b) \Rightarrow \boldsymbol{\gamma}(\textbf{join}(a, b))$$

$$\boldsymbol{\gamma}(a) \;\&\&\; \boldsymbol{\gamma}(b) \Rightarrow \boldsymbol{\gamma}(\textbf{meet}(a,b))$$

Example: Box Abstract Domain

abstract

concretize

$$1 \le x \le 10 \xrightarrow{\alpha} (1, 10) \xrightarrow{\gamma} 1 \le x \le 10$$

Definition of Operations	**Examples**
(a, b) **meet** (c, d) = (max(a,c), min(b,d))	(1, 10) **meet** (2, 12) = (2,10)
(a, b) **join** (c, d) = (min(a,c),max(b,d))	(1, 3) **join** (7, 12) = (1,12)
αPost (x := x + 1) ((a, b)) = (a+1, b+1)	(1, 10) + 1 = (2, 11)

over-approximation

Abstract Interpretation w/ Box Domain (2)

Program

```
assume (i=1 || i=2)
if (i = 1)
   x1 := i;
else if (i = 2)
   x2 := -4;

if (i = 1)
   assert (x1 > 0);
else if (i = 2)
   assert (x2 < 0);
```

1 <= i <= 2

i=1

i=1 && x1=1

i=2

i=2 && x2=-4

1 <= i <= 2

i=1

i=2

Loss of precision due to join

False Positive

Steps: 1 2 3 4 5 6 7 8

Disjunctive Refinement of an Abstract Domain

Bounded disjunctions

- extend base domain with disjunctions of size at most k
- *all* operations are done by lifting corresponding base domain operations
- easy to implement by modifying program control flow graph

Finite Powerset Domain [Bagnara et al.]

- extend base domain with all *finite* disjunctions
- *most* operations are done by lifting corresponding base domain opertions
- finding a good widening is complex (and often tricky)

Predicate Abstraction

- extend *finite* base domain with *all* disjunctions
- domain elements are represented by BDDs
- no widening required

OUR WORK

Outline

Software Engineering Institute (SEI)

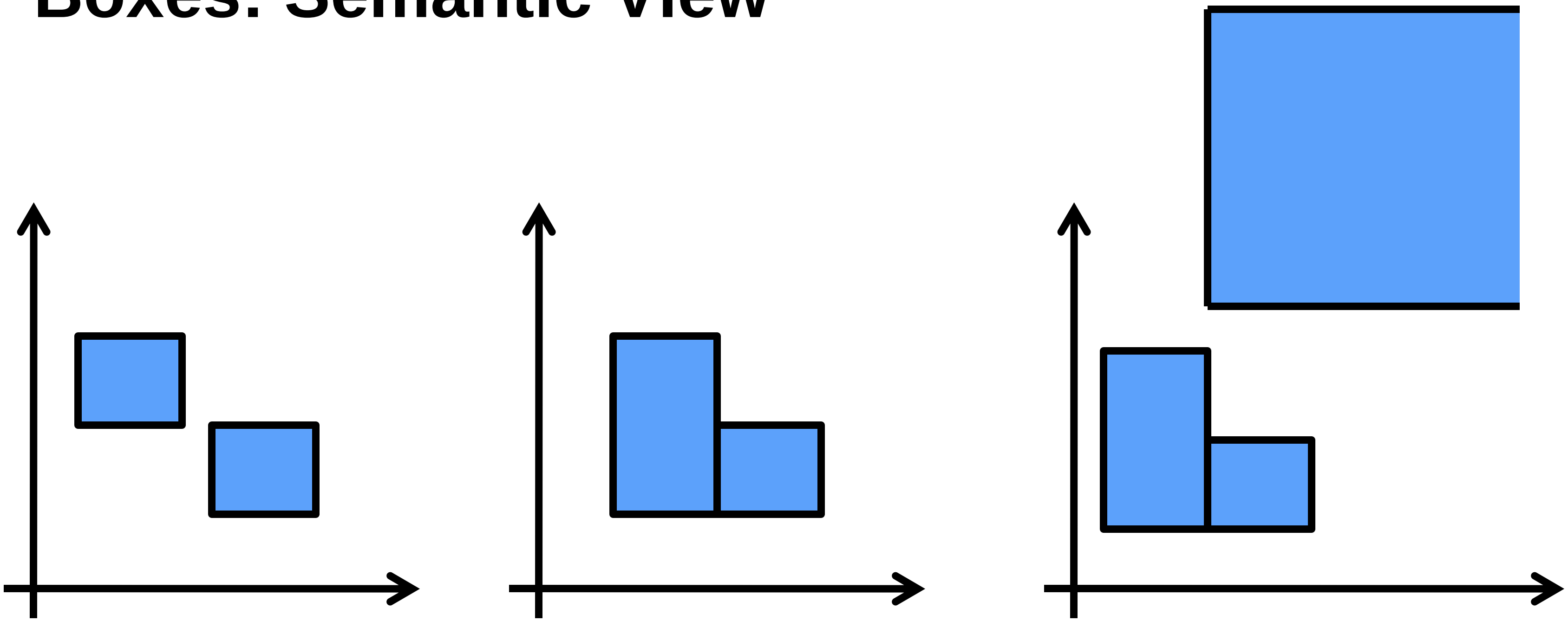
Introduction

Basic Abstract Interpretation

Boxes Abstract Domain

Conclusion

Boxes: Semantic View



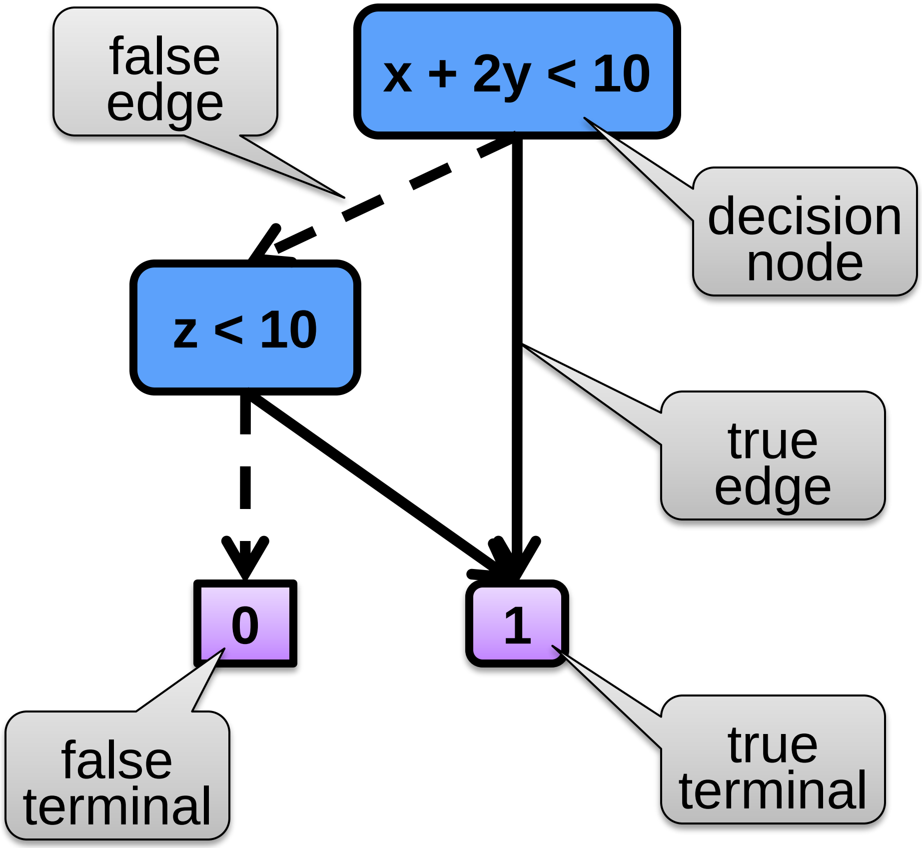
Boxes are “finite union of box values”

(alternatively)

Boxes are “Boolean formulas over interval constraints”

Linear Decision Diagrams in a Nutshell*

Linear Decision Diagram



Linear Arithmetic Formula

(x + 2y < 10) **OR**
(x + 2y ≥ 10 **AND** z < 10)

Compact Representation

- Sharing sub-expressions
- Local numeric reductions
- Dynamic node reordering

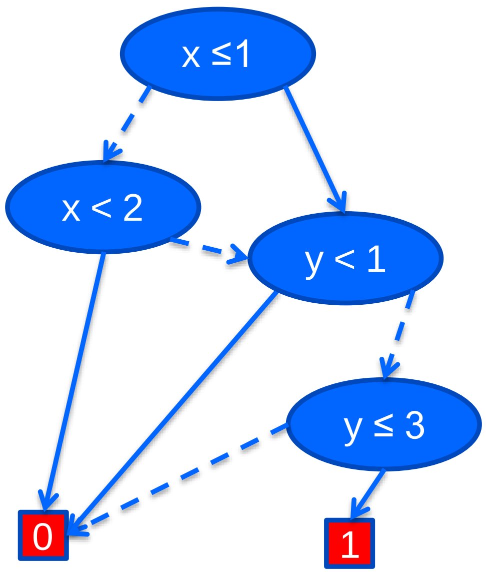
Operations

- Propositional (AND, OR, NOT)
- Existential Quantification

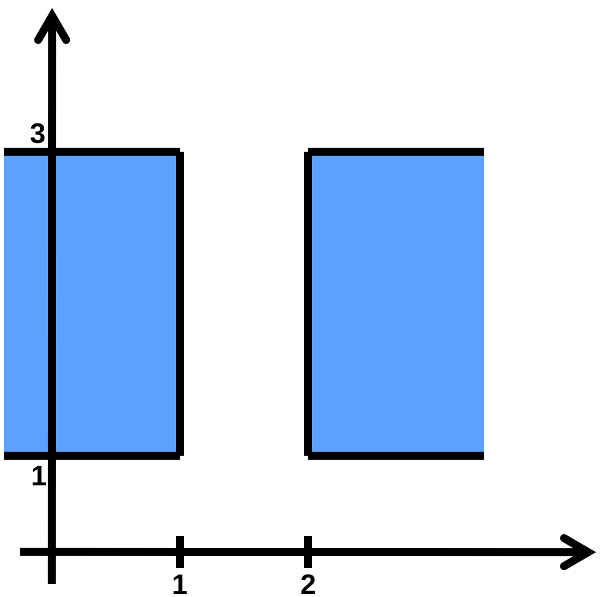
*joint work w/ Ofer Strichman

Boxes: Representation

LDD | **Syntax** | **Semantics**



$(x \leq 1 \;||\; x \geq 2)$
$\&\&$
$1 \leq y \leq 3$



Represented by (Interval) Linear Decision Diagrams (LDD)

- BDDs + non-terminal nodes are labeled by interval constraints + extra rules
- retain complexity of BDD operations
- canonical representation for Boxes Abstract Domain
- available at http://lindd.sf.net

Abstract Domain Operations

Basic domain operations are ... ed by LDD operations

meet

order (semantic)

join

Operation	Complexity
$f \wedge g$	O(|f||g|)
ITE(h, f, g)	O(|h||f||g|)
$\neg f$	O(1)

Operation	Complexity
$f \vee g$	O(|f||g|)
$f \Rightarrow g$	O(|f||g|)
$\exists U. f$	$O(\|f\| 2^{\|U\|})$

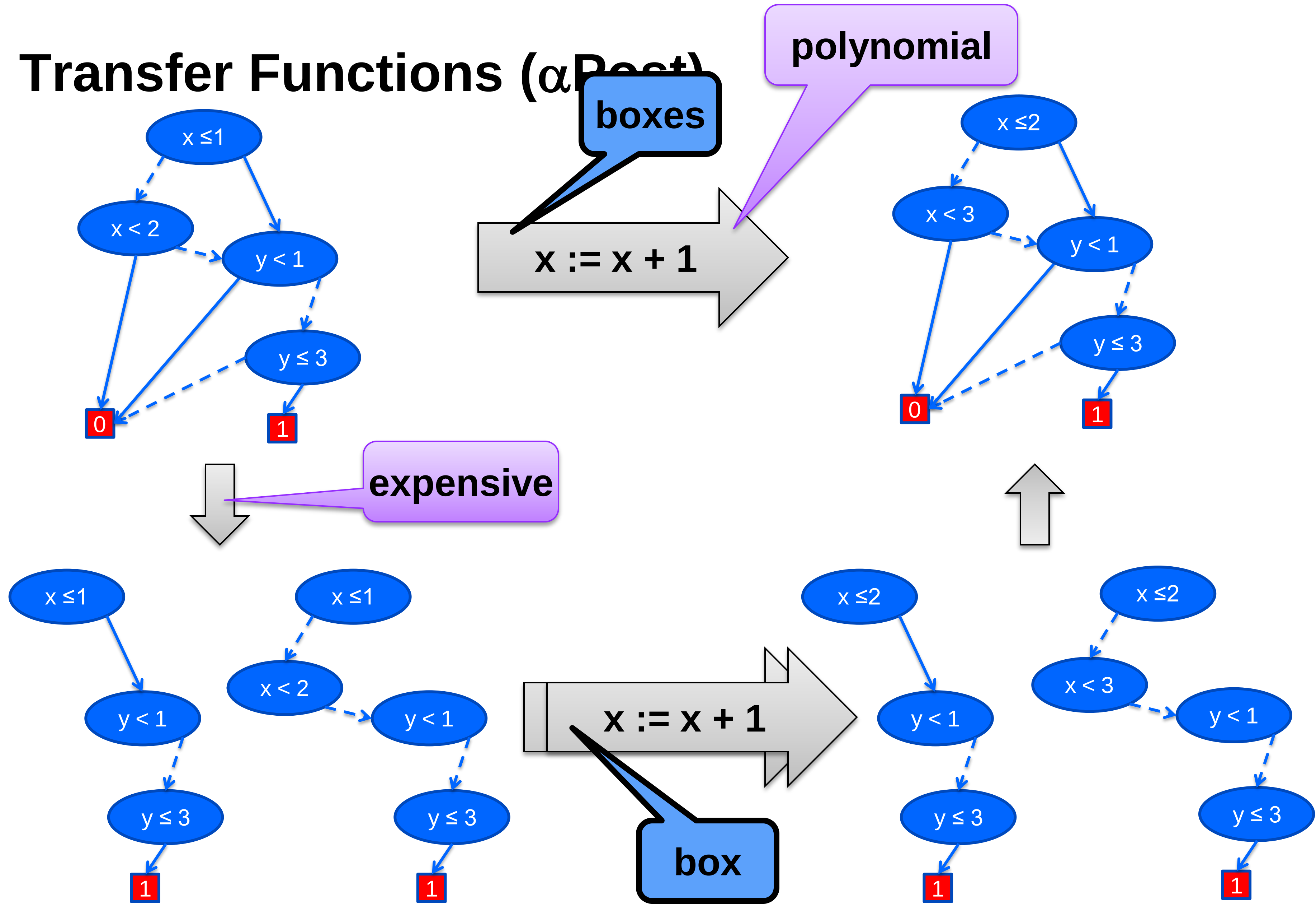
projection

Additional operations

- set difference $f \setminus g$ implemented by $f \wedge \neg g$
- BoxHull (f) – smallest Box containing f
- BoxJoin (f, g) – smallest Box containing the union of Box f and Box g

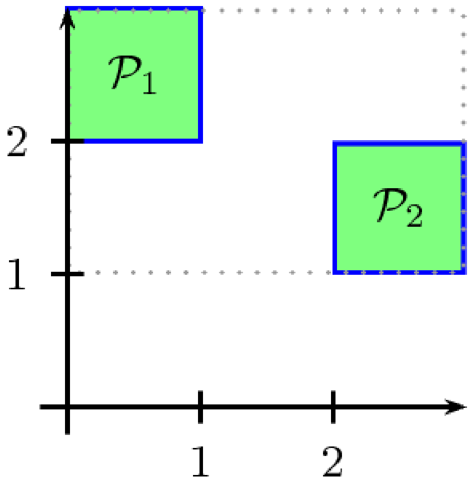
All operations are polynomial in the size of the representation

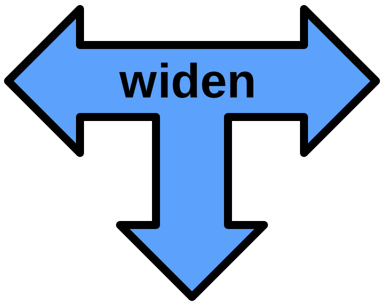
Transfer Functions (αPost)



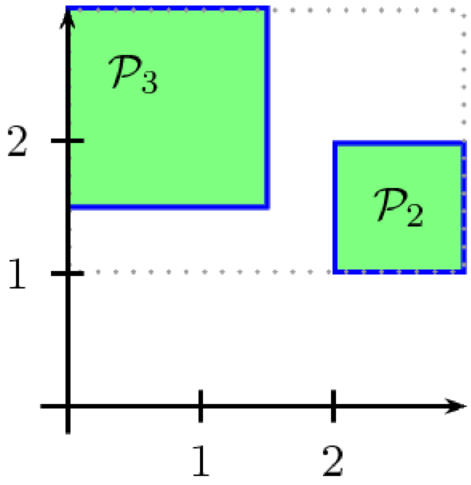
Widening: The Problem

$(\quad x \leq 1 \wedge 2 \leq y \leq 3) \vee$
$(2 \leq x \leq 3 \wedge 1 \leq y \leq 2)$

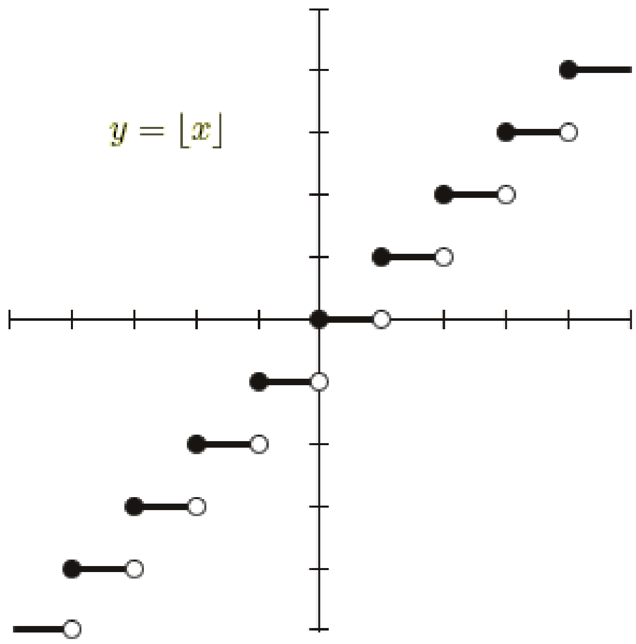




$(\quad x \leq 1.5 \wedge 1.5 \leq y \leq 3) \vee$
$(2 \leq x \leq 3 \quad \wedge \quad 1 \leq y \leq 2)$



Step Function



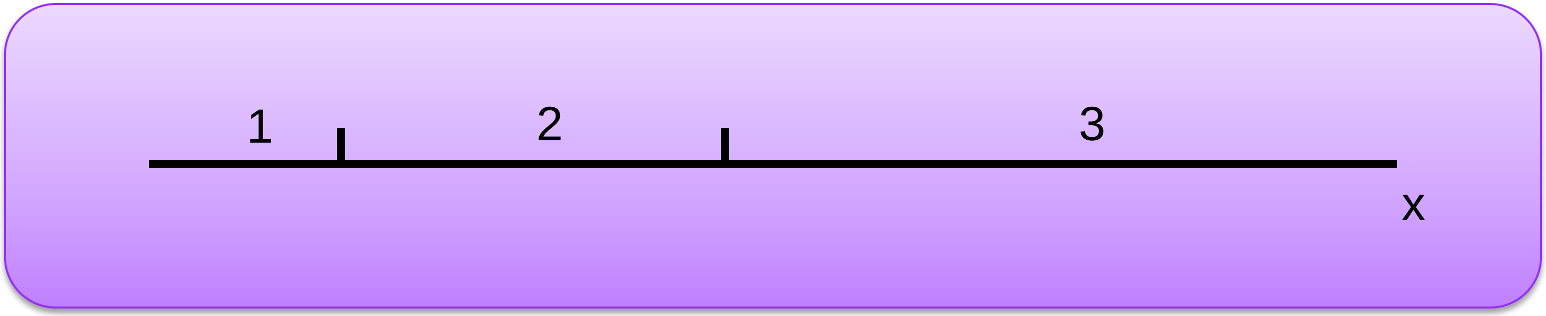
A function on the reals $\mathbb{R}$ is a *step function* if it can be written as a *finite* linear combination of semi-open intervals

$$f(x) = \alpha_1 f_1 (x) + \cdots + \alpha_n f_n (x)$$

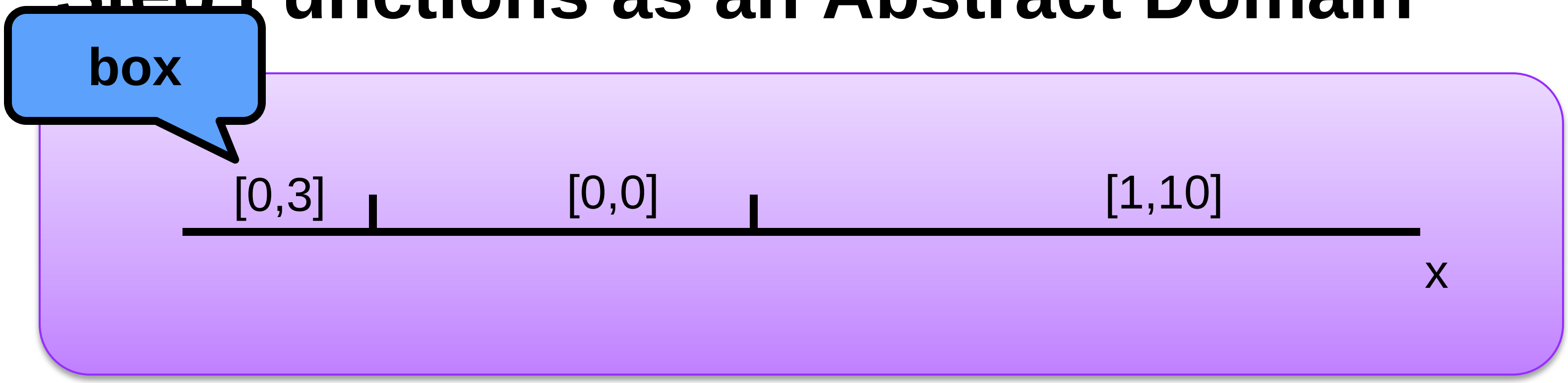
where $f_i \in \mathbb{R}$ and $\alpha_i(x)=1$ if $x \in [a_i, b_i)$ and 0 otherwise, for $i=1,\ldots,n$

Weisstein, Eric W. "Step Function." From *MathWorld*--A Wolfram Web Resource.
http://mathworld.wolfram.com/StepFunction.html

Step Functions as an Abstract Domain



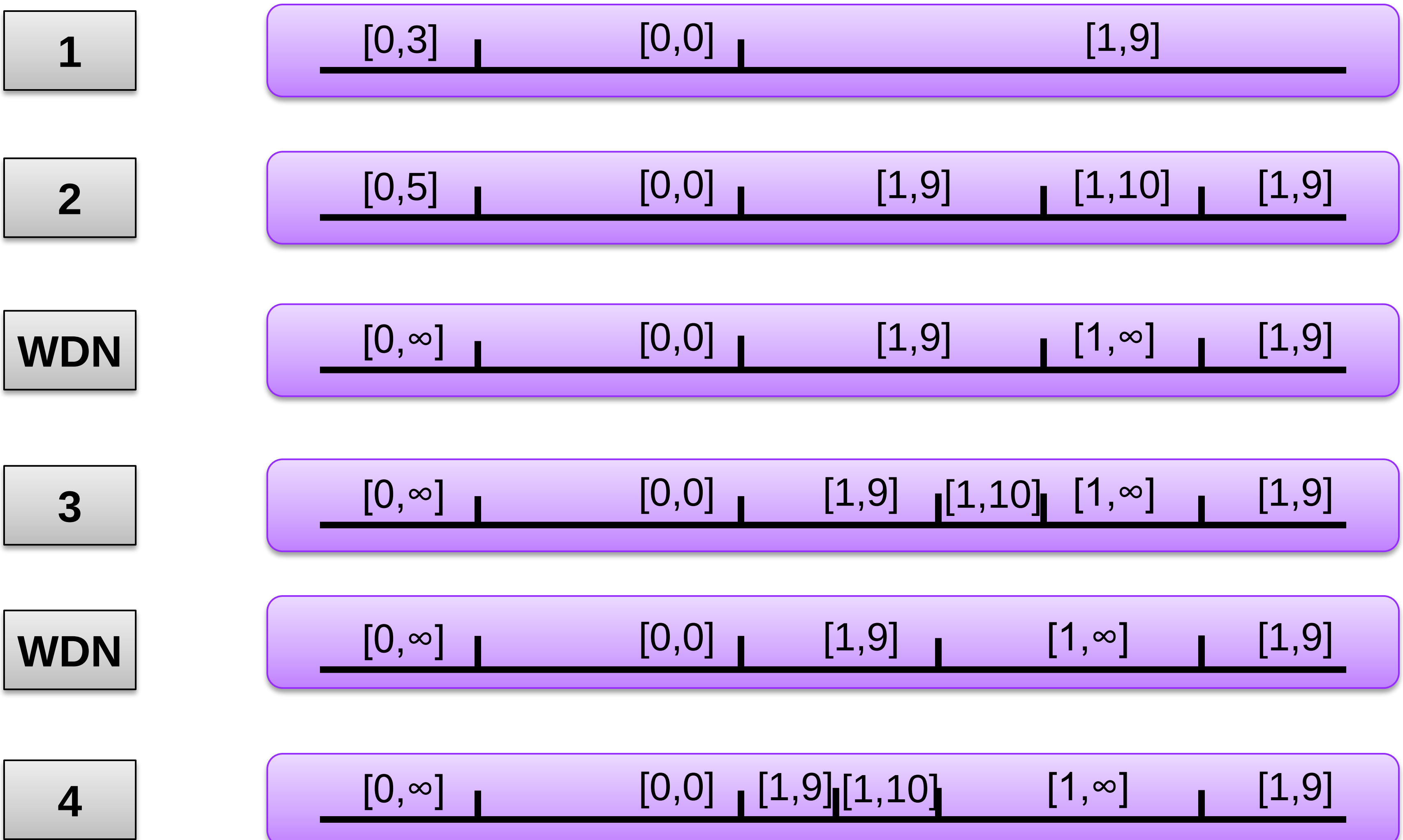
Step Functions as an Abstract Domain



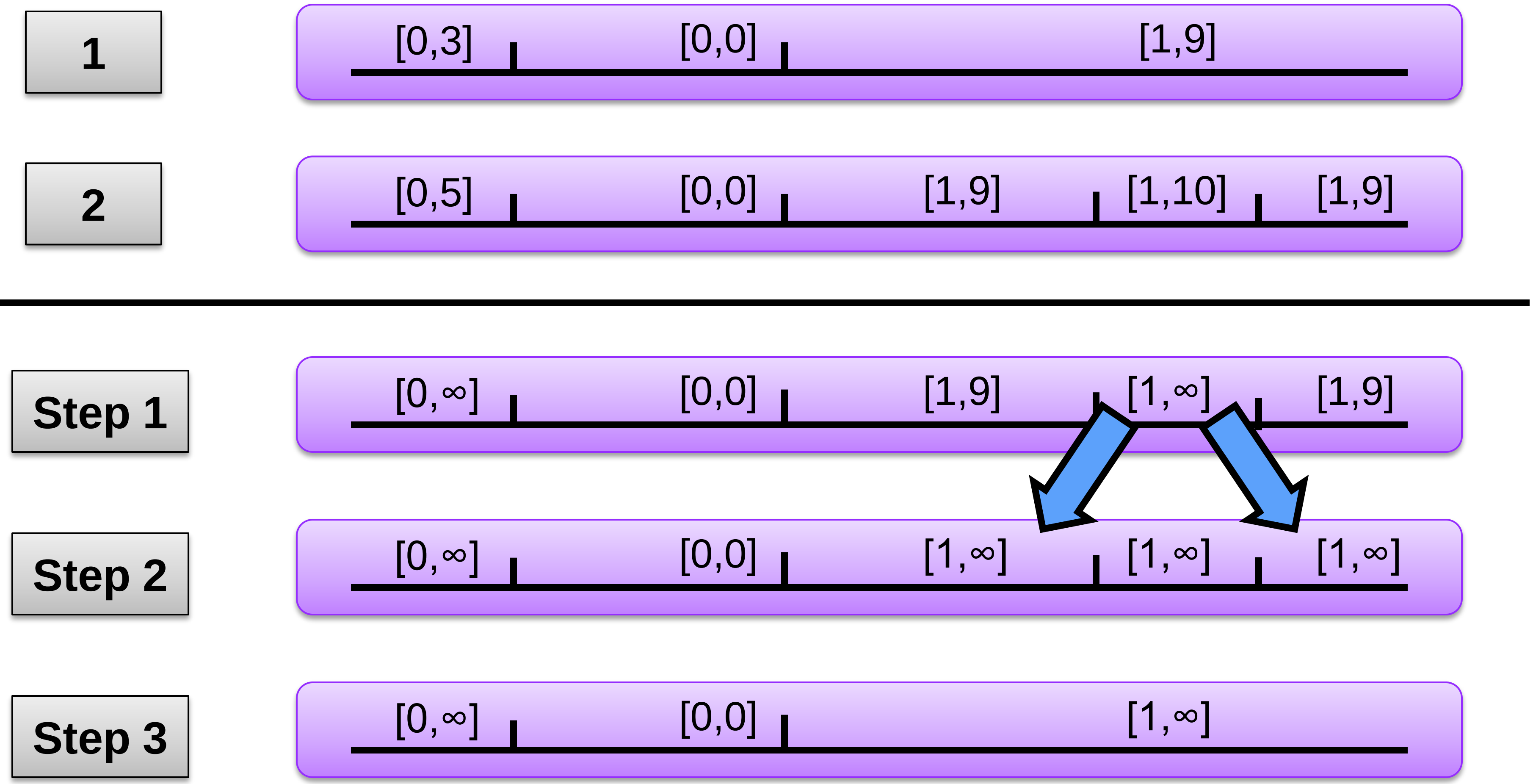
STEP(D) an abstract domain of step functions over an abstract domain D

- elements are step functions $\mathbb{R} \to D$
- order is pointwise: $f \sqsubseteq g$ iff $\forall x \,.\, f(x) \sqsubseteq_D g(x)$
- join is pointwise: $f \sqcup g$ is $\lambda x \,.\, f(x) \sqcup_D g(x)$
- meet is pointwise: $f \sqcap g$ is $\lambda x \,.\, f(x) \sqcap_D g(x)$
- ~~widen is pointwise: $f \nabla g$ is $\lambda x \,.\, f(x) \nabla_D g(x)$~~ ????

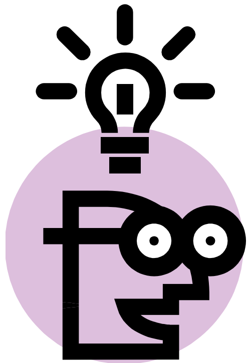
Pointwise Extension of Widen Diverges



Widening for Step Functions



Back to Boxes



Boxes are Step functions!

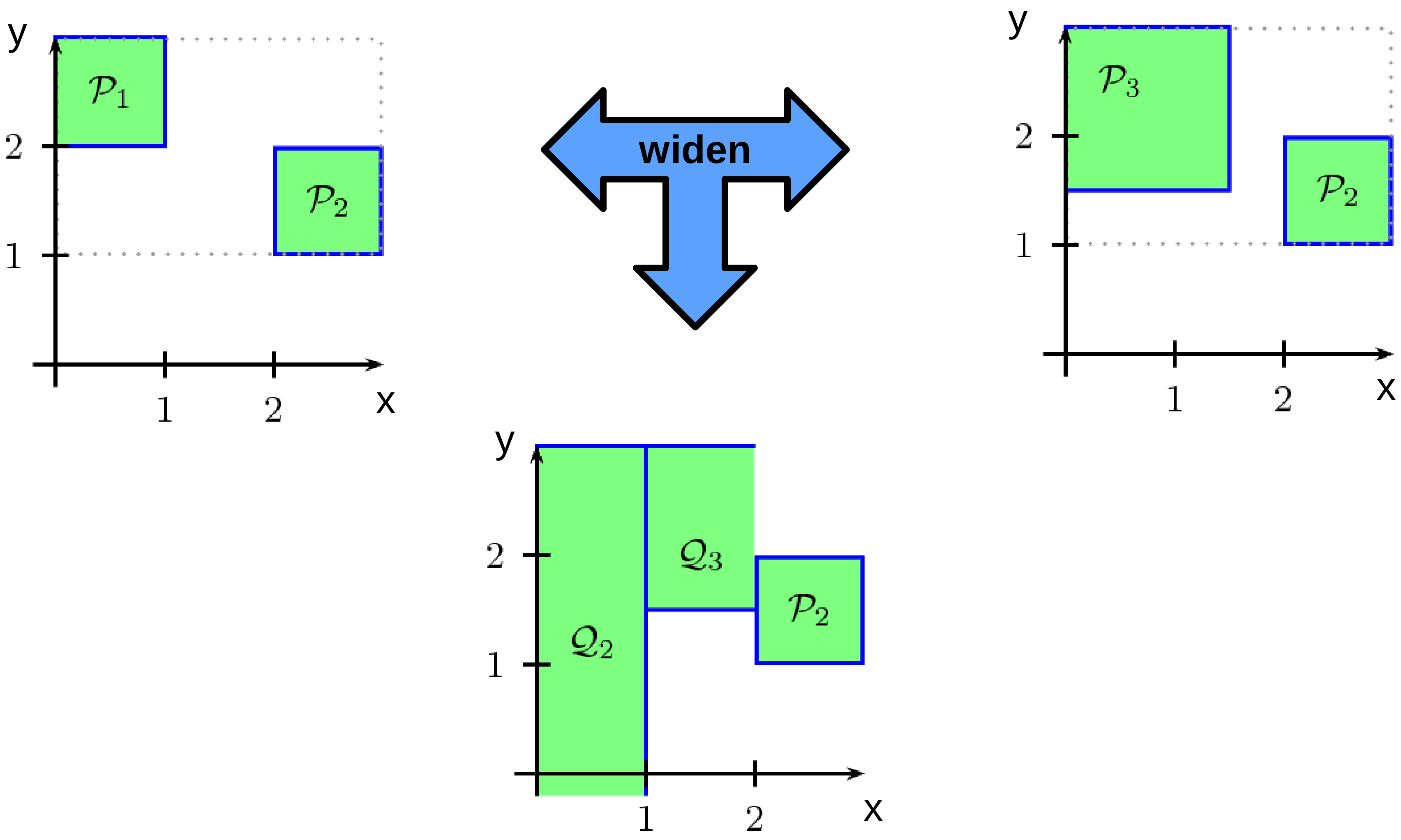
- 1-dim Boxes are STEP($\{\bot,\top\}$) $\mathbb{R}\rightarrow\{\bot,\top\}$
- 2-dim Boxes are STEP (STEP ($\{\bot,\top\}$) $\mathbb{R}\rightarrow\mathbb{R}\rightarrow\{\bot,\top\}$
- n-dim Boxes are STEP^n ($\{\bot,\top\}$) $\mathbb{R}^n\rightarrow\{\bot,\top\}$

Widen for $\{\bot,\top\}$ is trivial

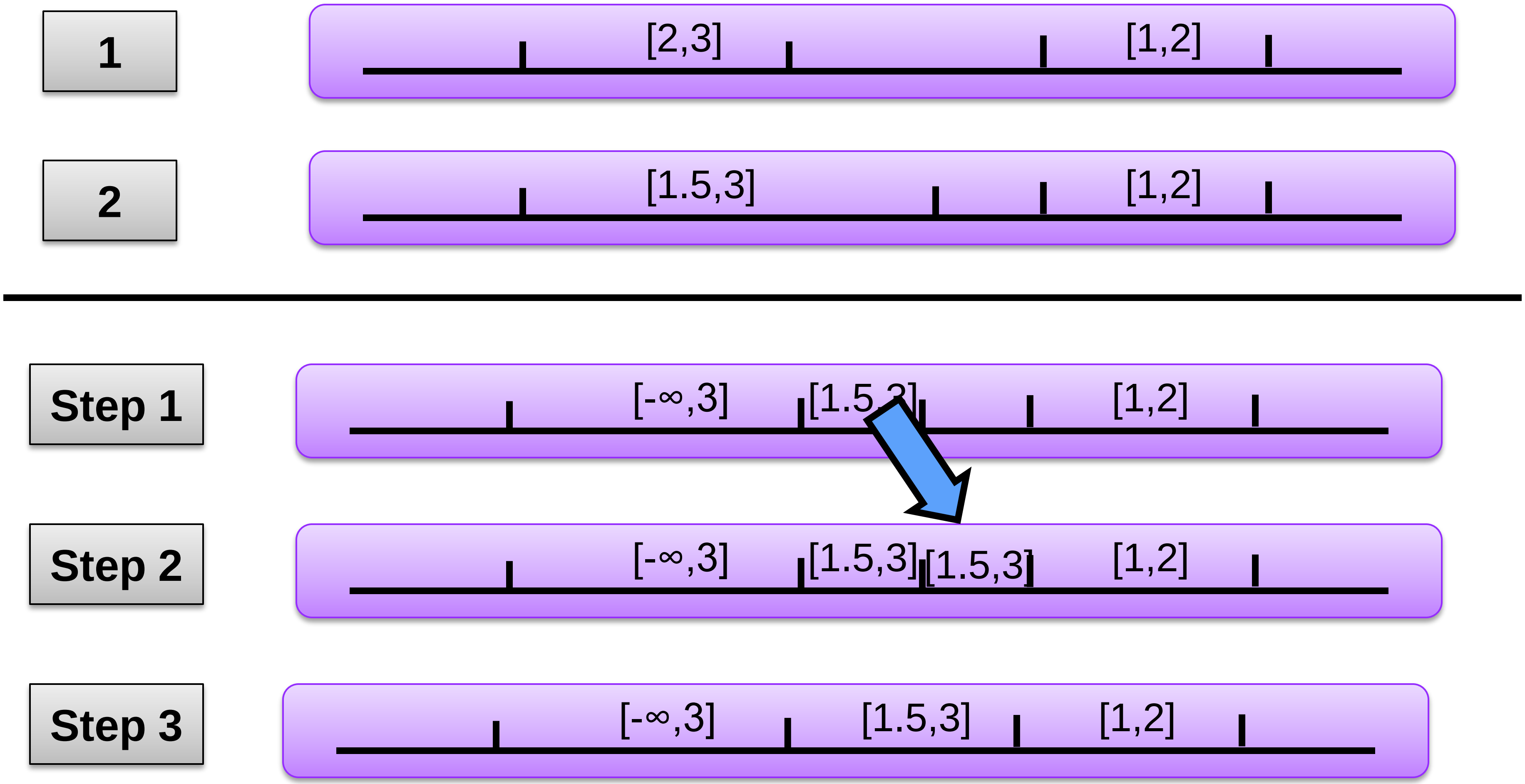
Widen for n-dim Boxes is defined recursively on dimensions

We give a polynomial time algorithm that implements this widen operator directly on LDDs. (See paper for details)

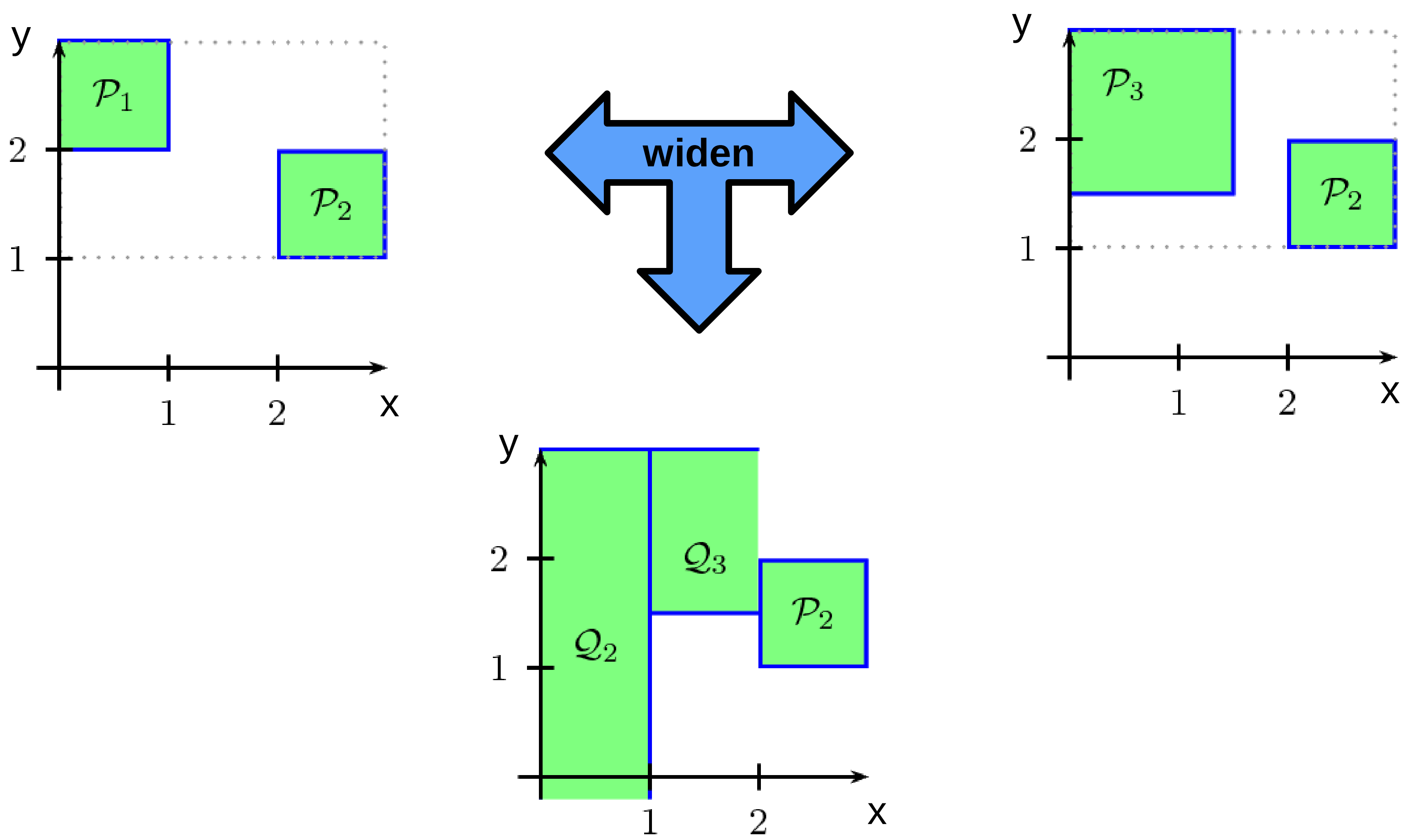
Widen: An Example



Widening: Example



Widen: An Example



Boxes versus Finite Powersets

Bagnara et al.
Parma Polyhedra Library (PPL)

Our work

	Boxes	Finite Powerset
Base domain	Box	Any
Representation	Decision Diagram	Set / DNF
Domain order	semantic	syntactic
Complexity	polynomial in representation	polynomial in representation
Singleton Widen	Box	base domain
Widen	Step Function	Multiple Choices

Experiments: Invariant Computation

Abstract Domains

- LDD Boxes – Our Boxes domain using LDDs
- PPL Boxes – **`Pointset_Powerset<Rational_Box>`** of PPL

Analyzer

- custom analyzer on top of LLVM compiler infrustructure
- computes loop invariants for all loops over all SSA variables in a function

Benchmark

- from open source software: mplayer, CUDD, make, …
- Stats: 5,727 functions

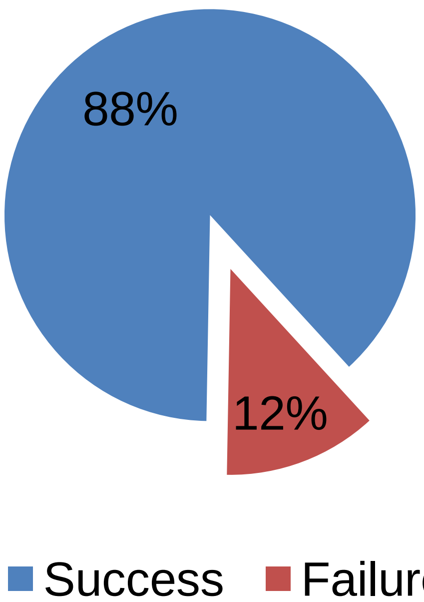
  9 – 9,052 variables (avg. 238, std. 492)

  0 – 241 loops (avg. 7, std. 12)

Results: Time

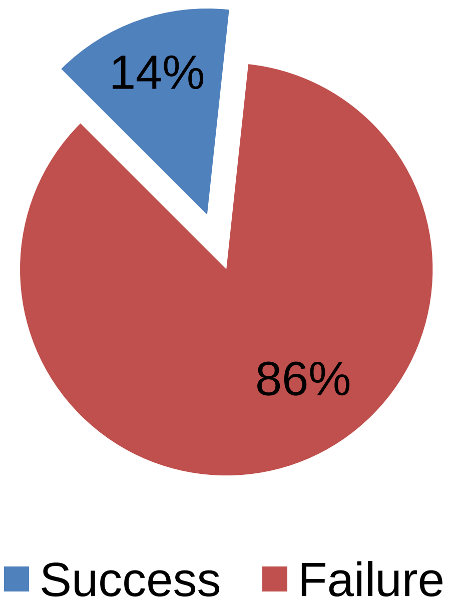
5727 functions, each run with a time limit of 60s

LDD Boxes



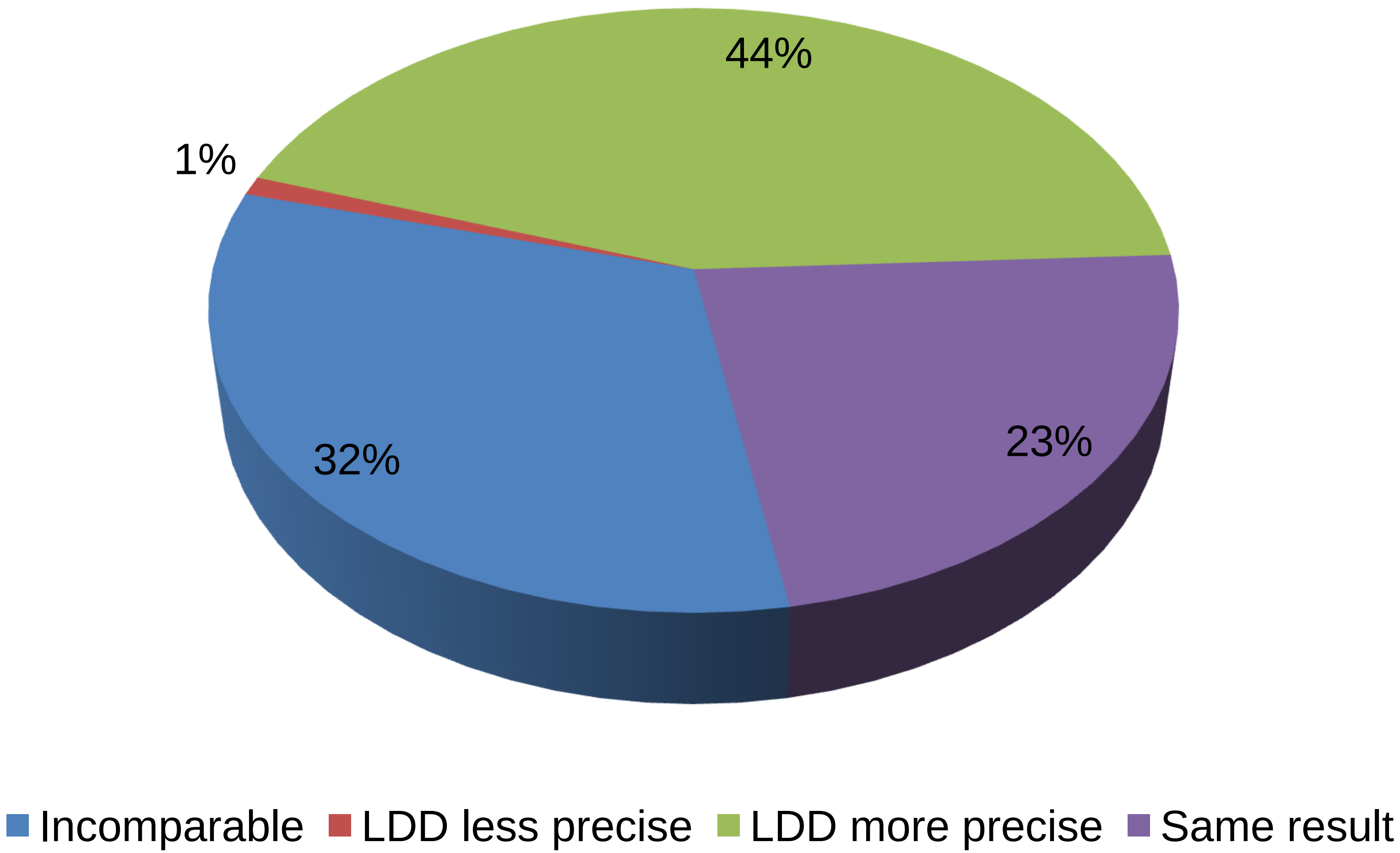
Total time = 118 min

PPL Boxes

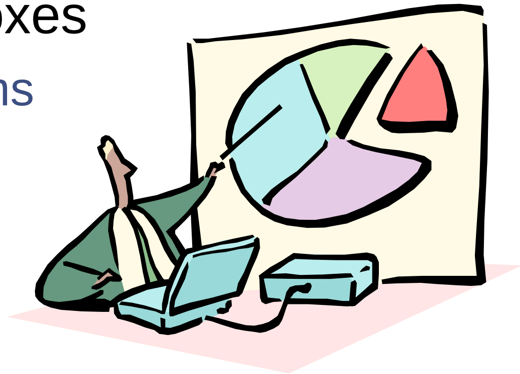


Total time = 201 min

Results: Precision



Conclusion



Boxes: A new disjunctive abstract domain of sets of boxes

- efficient representation based on Linear Decision Diagrams
- semantic order relation
- efficient operations and widening
- more precise and efficient than finite powersets of box

A new widening scheme

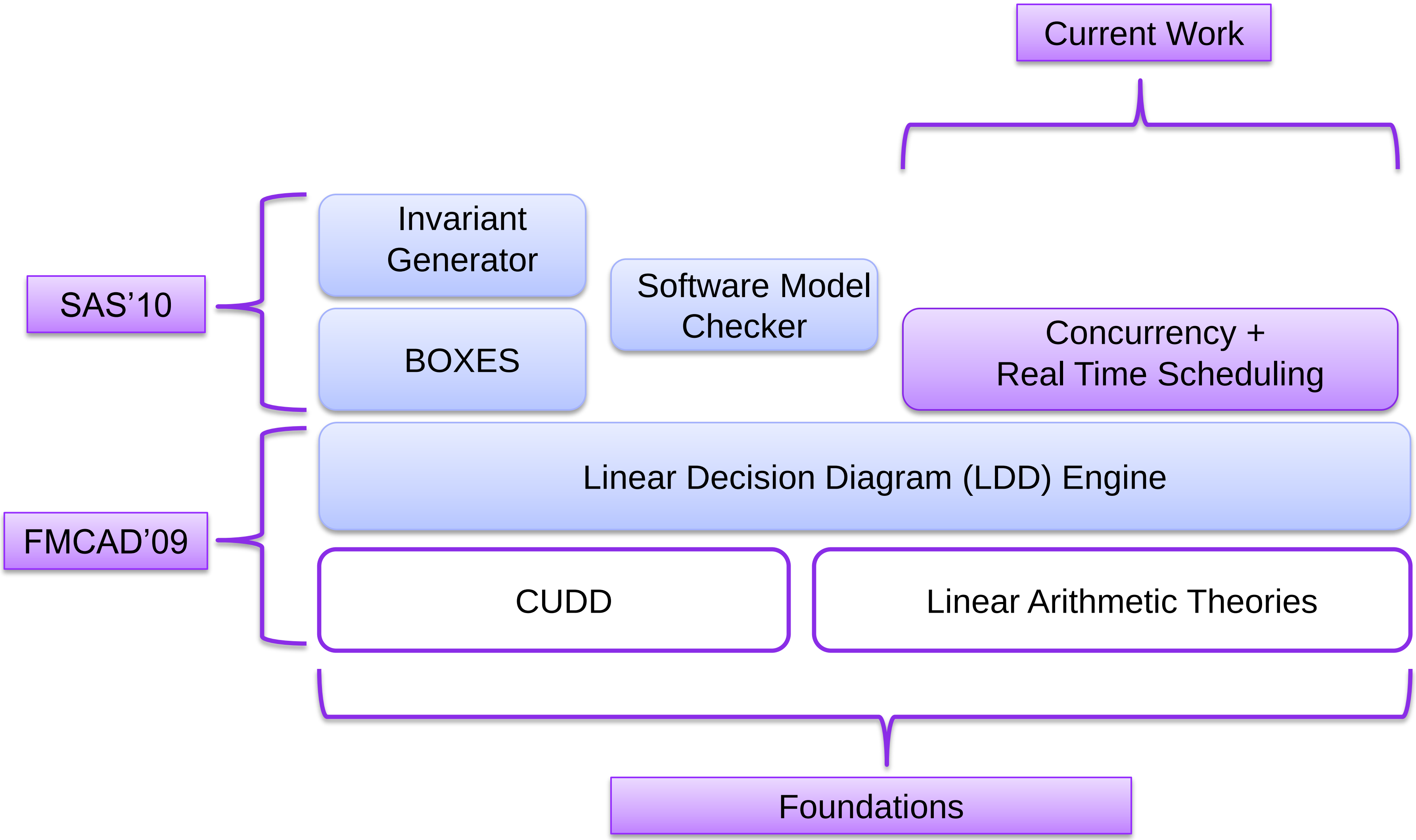
- lifting widening from a base domain to the domain of step functions

Future Work

- applications
- extending the technique to richer base domains, i.e., octagons, TVPI
  - representation and base operations are easy (already exist in LDD)
  - widening?

http://lindd.sf.net

LDD Based Analysis Infrastructure



Contact Information

Presenter
Arie Gurfinkel
RTSS
Telephone: +1 412-268-7788
Email: arie@cmu.edu

U.S. mail:
Software Engineering Institute
Customer Relations
4500 Fifth Avenue
Pittsburgh, PA 15213-2612
USA

Web:
www.sei.cmu.edu
http://www.sei.cmu.edu/contact.cfm

Customer Relations
Email: info@sei.cmu.edu
Telephone: +1 412-268-5800
SEI Phone: +1 412-268-5800
SEI Fax: +1 412-268-6257

THE END