Symbolic Abstract Domains

Laurent Mauborgne

IMDEA Software

Interprétation abstraite, MPRI 2–6, année 2012-2013

Abstract Domains
(Reminder)

Goal:

Represent and manipulate sets of values

In practice:

- the representation should be compact
- operations should be fast

In abstract interpretation

- we can approximate
- …but not too much (false alarms!)

Efficient abstract domains
Reminder again

What operations should be efficient?

- Sets of value computed by fixpoint iteration

$\Rightarrow$ needs efficient inclusion testing

- Each iteration adds an increment

$\Rightarrow$ incremental structures

- Each individual instruction that is evaluated only modifies a small part of the environment

$\Rightarrow$ needs a mechanism to perform local modifications and avoid copying the whole environment.

Symbolic or Numeric?

Why another course on abstract domains?

- Representation of (big) sets of values $\Rightarrow$ symbolic representations
- Programs manipulate symbolic values or numeric values
  - everything is a number *in fine*, but
  - sets of `enum` not well approximated by intervals
  - or $\mathcal{V} \rightarrow \mathbb{B}$ not well approximated by polyhedra
  - idem for memory structure

Symbolic values of Programs

Sets of value without arithmetic structure

- Symbolic properties (about programs)
  - so-called necessary variables
  - reasoning about traces
  - temporal properties

Lesson Plan

Sets, Relations and Boolean Functions

- Consider a finite set of symbols (= `enum`, properties . . .)

Example

Values of a variable `x`

```
enum {Blue Green
Red}x;
```

Example

Properties of a variable x such as

- $p1 = x$ is reachable from variable y
- $p2 = x$ is necessary for function f

- Abstract property = set of symbols

$\Rightarrow$ bit vector

Exact representation

Set of bit vectors
(Coded as sequences of bits)

- Logical formula
- Relation
- Boolean function

Logical Formulæ

First Order Logic

Definition

Logical formula ::=		x	boolean variable
	$\mid$	$f \wedge f \mid f \vee f \mid \neg f$	logical connectors
	$\mid$	$\forall\, \mathrm{x}.f \mid \exists \mathrm{x}.f$	quantifiers

Interpretation

- $f(x, y, z)$ represents the set of boolean vectors $< b_0, b_1, b_2 >$ such that $f(b_0, b_1, b_2)$ is true
- Formula = algorithm of a function $\mathbb{B}^n \to \mathbb{B}$

Set Membership Algorithm

Going through the formula tree

Example

Let $f(x, y, z) = (y \wedge z) \vee (\neg x \wedge \neg y \wedge \neg z)$

The tree is

$$\begin{tikzcd}
 & & & \vee \arrow[dll] \arrow[drr] & & & \\
 & \wedge \arrow[dl] \arrow[dr] & & & & \wedge \arrow[dl] \arrow[d] \arrow[dr] & \\
y & & z & & \neg x & \neg y & \neg z
\end{tikzcd}$$

Bottom up traversal

Inclusion Testing

- Set of $f \subset$ set of g iff $f \Rightarrow g$
- It's often the construction ordering in static analysis

SAT solvers

- Computes if a formula is satisfiable, and when it is, gives an element
- State of the art software very efficient (but needs fine tuning)
- Very much used in hardware verification

For static analysis

- SAT($f \wedge \neg g$) for inclusion
- Problems :
  - negation expensive (because of normal forms)
  - formulæ can grow unboundedly

Relations

Definition

Let $(E_i)_{i \in I}$ be a family of sets. A relation of support $(E_i)_{i \in I}$ is a sub-set of $\bigotimes_{i \in I} E_i$.

- On booleans, amounts to sets of bit vectors
- We denote the projection $R_{(J)}$
- and partial evaluation $R_{:i=b}$

Example

$\{000, 011, 111\}$

0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

A formula

$f(x, y, z) = (y \wedge z) \vee (\neg x \wedge \neg y \wedge \neg z)$

Another formula

$f(x, y, z) = (\exists t. t \wedge y \wedge z) \vee \neg(x \vee y \vee z)$

Many other formulæ

as big as you like...

Boolean Relations as Abstract Domain

- How can we be efficient?
- For which operations?
  - abstract transfer functions
  - fixpoint testing (implications)
  - testing emptiness
  - union, but with a lot of recomputations

A Possible Solution

Sharing and incremental (whenever possible) representation.

- Sharing $\Rightarrow$ constant emptiness testing
- Sharing $\Rightarrow$ memoization

Decision Trees
or Shannon trees

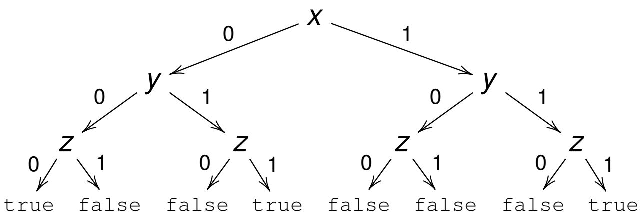
Definition

Shannon's identity: $f = x \wedge f_{:x=\texttt{true}} \vee \neg x \wedge f_{:x=\texttt{false}}$

Let f be the set $\{000, 011, 111\}$.

$$f(x, y, z) = (y \wedge z) \vee (\neg x \wedge \neg y \wedge \neg z)$$

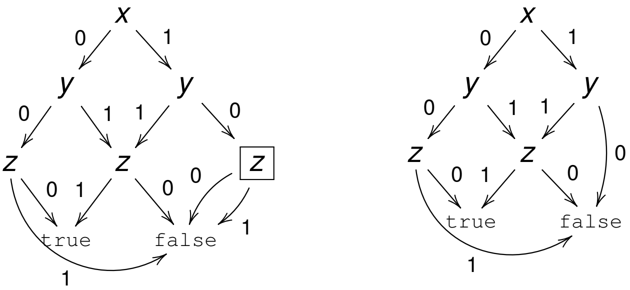
The decision tree of f pour the (ordered) variables x, y, z:



BDDs
Binary Decision Diagrams

Definition

The BDD of f for the variables $\mathcal{V}$ is the decision tree of f for those variables, with sub-tree sharing and redundant nodes elimination.



Hashconsing

Unique representation: $t_1 = t_2 \Leftrightarrow t_1 == t_2$

- Nodes numbering
- Dictionary (hash table):

  (variable, left id, right id) -> id
- Incremental construction.
- Basic operation: `if` x `then` f_1 `else` f_0.
- Memoization. Worst case cost of binary operations is quadratic.

BDD Complexity

- Worst case size: exponential in the number of variables
- Average size: exponential
- Average gain compared to an array: linear factor (which comes from the sharing)
- The elimination of redundant nodes allows the manipulation of different functions in the same dictionary.
- But in practice, most of the time, very big gain
- BDD exploits the structure the problem
- In abstract interpretation, approximations are possible. . .

Approximating BDDs for space

- BDDs size can change with variable ordering, but
  - The problem of finding an optimal variable ordering is NP-hard
  - For some classes of functions, all variable orderings yield an exponential size BDD

  $\Rightarrow$ needs to change the function to obtain tractability

Problem

Given a function f find a function f' such that $f \Rightarrow f'$ and the BDD representing f' is smaller than the BDD representing f.

- Solution: $f' = \texttt{true}$
- Add a new constraint:
  - the model (number of vectors evaluating to $\texttt{true}$) of f' should be as small as possible.
  - but balance that with the gain in size...

Density of a BDD

Definition

- A minterm of a boolean function *f* is an assignment to the variables of *f* that evaluates to `true`.
- The density of a BDD is the number of nodes in the BDD, divided by the number of minterms of the boolean function it represents.

- A density driven algorithm will try heuristics at each node of the BDD and estimate the gain in density
- When the density reaches a predefined threshold, the algorithm terminates
- Two such algorithms are available in a standard BDD package (CUDD)

Two Simple Heuristics

Heavy Branch

- Compute the number of minterms at each node
- Starting from the root, at each node, replace the direct child with the most minterms by `true`
- Until the size of the BDD is below a given threshold

$\Rightarrow$ Biased towards BDD with first variables having a child `true`

$\Rightarrow$ Depends on the variable ordering (not semantic)

Shortest Path

- idea: shortest paths give better density
- Compute the length of the shortest path starting at each node
- Replace each node with too big a shortest path by `true`

$\Rightarrow$ Not much control over the desired size of the BDD

$\Rightarrow$ Not very predictable algorithm

Both techniques can be modified to allow sharing of direct children (replacing $N.l$ and $N.r$ by their union).

Dual Prime Implicants

Definition

- A clause is a disjunction of variables or negation of variables (called literals)
- A clause c is a `dual prime implicant` of a boolean function f if
  - $f \Rightarrow c$
  - There is no clause c' (other than c) such that $f \Rightarrow c' \Rightarrow c$
- We denote $primes(f)$ the set of dual prime implicants of f.

Property

For all boolean function,

$$f = \bigwedge primes(f)$$

Approximation based on dual prime implicants[a]

[a] Based on Neil KETTLE's thesis

- A set of dual prime implicants is a sound approximation
- The smaller the clauses, the denser
- Deterministic approximation
  - compute the dual prime implicants of length at most k
  - take their conjunction
  - in practice much better than other heuristics, because semantic based
- Randomized approximation
  - randomly select a path to `false` in the BDD
  - extract a dual prime implicant c
  - collect the conjunction of such clauses
  - before selecting next path, can transform f into $f \wedge \neg c$
  - probability to select a given clause = $2^{n-|c|}$

Cartesian Approximation

Exact representation of boolean relations	=	exponential size

Definition

$$\wp\left(\bigotimes_{i\in I} E_i\right) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \bigotimes_{i\in I} \wp\left(E_i\right)$$

$$\alpha(V) \stackrel{\text{def}}{=} \bigotimes_{i\in I} V_{(i)}$$

The cost becomes linear!

Example

$$\alpha\left(\{000, 011\}\right) = \{0\}.\{0, 1\}.\{0, 1\}$$

Smash Product

- Let $\bigotimes_{i \in I} V_{(i)}$ be a cartesian approximation
- If one V_i is $\emptyset$, then the product is empty too

Smash

More efficient if just one possible representation for $\emptyset$

- In a bit vector, we needed 2 bits per boolean variable
- but the sequence 00 $\Rightarrow \emptyset$

Approximation using classic logic

Only 1 bit per boolean variable
$\Rightarrow$ either 0 = {0} and 1 = {0,1}, either 0 = {0,1} and 1 = {1}

First Example: Predicate Abstraction

- Given a set of predicates, $\mathcal{P}$
- Approximate a set of states by the set of predicates in $\mathcal{P}$ which are `true` for all states in the set
  - $\alpha_{\mathbb{P}}(Q) \stackrel{\text{def}}{=} \{p \in \mathbb{P} \mid Q \subseteq \mathcal{I}\llbracket p \rrbracket\}$
  - $\gamma_{\mathbb{P}}(P) \stackrel{\text{def}}{=} \bigcap \{\mathcal{I}\llbracket p \rrbracket \mid p \in P\}$

  $\Rightarrow$
$$\langle \wp(M), \subseteq \rangle \underset{\alpha_{\mathbb{P}}}{\overset{\gamma_{\mathbb{P}}}{\leftrightarrows}} \langle \wp(\mathbb{P}), \supseteq \rangle$$

$\Rightarrow$ just keep the set of predicates which are true, represented by bit vector

- So, in this representation, 1 represents $\{1\}$ and 0 represents $\{0, 1\}$

Second Example: Strictness Analysis

- Property about the program: parameter $\mathtt{x}$ evaluates or not (either because of error or non-termination)
- To know if $\mathtt{x}$ is strict:

Deduction rule

if ($\mathtt{x}$ does not terminate or produces an error $\Rightarrow$ `f(x)` too), then $\mathtt{x}$ is strict in `f`.

- Approximation:

Only errors are for sure

- $\alpha(\mathrm{x}) \stackrel{\text{def}}{=} 0$ if x does not terminate
- $\alpha(\mathrm{x}) \stackrel{\text{def}}{=} 1$ represents all cases

Kleene's Logic

$\emptyset$ is superfluous, but we keep $\{0\}$, $\{1\}$ and $\{0, 1\}$.

Kleene notation

- $0 \stackrel{\text{def}}{=} \{0\}$
- $1 \stackrel{\text{def}}{=} \{1\}$
- $\frac{1}{2} \stackrel{\text{def}}{=} \{0, 1\}$

Approximation Ordering

$$\begin{tikzcd} & \frac{1}{2} & \\ 0 \arrow[ur] & & 1 \arrow[ul] \end{tikzcd}$$

Logical Ordering

$$\begin{tikzcd} 1 \\ \frac{1}{2} \arrow[u] \\ 0 \arrow[u] \end{tikzcd}$$

With that ordering, logical connectors and quantifiers on Kleene's logic are a sound approximation of the operators on sets of booleans.

TVLA
Three Values Logic Analyzer

- Static analysis tool by abstract interpretation
- Developed at Tel Aviv University, by Mooly SAGIV et al.
- Parameterized by a finite set of predicates (but predicates with arguments $\Rightarrow$ not finite...
- Mainly used to determine the *shape* of the heap during program execution
- Can represent unbounded heaps, thanks to "summary nodes"

Adding Predicates and Functions to Formulæ

$$\begin{array}{rclll}
\mathtt{x}, \mathtt{y}, \mathtt{z}, \ldots & \in & \mathbb{x} & & \\
\mathtt{a}, \mathtt{b}, \mathtt{c}, \ldots & \in & \mathbb{f}^0 & & \\
\mathtt{f}, \mathtt{g}, \mathtt{h}, \ldots & \in & \mathbb{f}^n & & \\
t & \in & \mathbb{T}(\mathbb{x}, \mathbb{f}) & t ::= & \mathtt{x} \mid \mathtt{c} \mid \mathtt{f}(t_1, \ldots, t_n) \\
\mathtt{p}, \mathtt{q}, \mathtt{r}, \ldots & \in & \mathbb{p}^n, \quad \mathbb{p} \triangleq \bigcup_{n \geqslant 0} \mathbb{p}^n & & \\
a & \in & \mathbb{A}(\mathbb{x}, \mathbb{f}, \mathbb{p}) & a ::= & \mathtt{false} \mid \mathtt{p}(t_1, \ldots, t_n) \mid \neg a \\
e & \in & \mathbb{E}(\mathbb{x}, \mathbb{f}, \mathbb{p}) & \triangleq & \mathbb{T}(\mathbb{x}, \mathbb{f}) \cup \mathbb{A}(\mathbb{x}, \mathbb{f}, \mathbb{p}) \\
\varphi & \in & \mathbb{C}(\mathbb{x}, \mathbb{f}, \mathbb{p}) & \varphi ::= & a \mid \varphi \wedge \varphi \\
\\
\Psi & \in & \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p}) & \Psi ::= & a \mid \neg \Psi \mid \Psi \wedge \Psi \mid \exists \mathtt{x} : \Psi
\end{array}$$

Plus special predicate for equality

Interpretations

Definition

Interpretation set of values + meanings of predicates and functions

$$I = \langle I_{\mathcal{V}}, I_{\gamma} \rangle \qquad \in \mathfrak{I}$$

Environment $\eta \in \mathcal{R}_I \stackrel{\text{def}}{=} \mathbb{x} \to I_{\mathcal{V}}$

$$
\begin{array}{rcl@{\qquad}rcl}
I \models_\eta a & \triangleq & [\![a]\!]_I\,\eta & I \models_\eta \Psi \wedge \Psi' & \triangleq & (I \models_\eta \Psi) \wedge (I \models_\eta \Psi') \\
I \models_\eta \neg\Psi & \triangleq & \neg(I \models_\eta \Psi) & I \models_\eta \exists \mathrm{x} : \Psi & \triangleq & \exists v \in I_{\mathcal{V}} : \; I \models_{\eta[\mathrm{x}\leftarrow v]} \Psi
\end{array}
$$

Natural meaning

$$\gamma^{\mathfrak{a}}(\Psi) \triangleq \{\langle I, \eta \rangle \mid I \models_\eta \Psi\}$$

Theories and Models

Definition

- Sentence = formula without free variables
- Theory = set of sentences + signature
- Model = interpretation on which a sentence is true

Idea: Restrict the possible meanings to those that make the sentences true.

A theory can be

- deductive,
- defined by a set of axioms,
- complete,
- the theory of an interpretation

$\mathfrak{M}(\mathcal{T})$ = set of interpretations of $\mathcal{T}$

Satisfiability, Validity and Decidability

- Ψ satisfiable iff $\exists\, I \in \mathfrak{I} : \exists \eta : I \models_\eta \Psi$
- satisfiable in $\mathcal{T}$: replace $\mathfrak{I}$ by models of $\mathcal{T}$
- $\mathcal{T}$ decidable iff there is an algorithm deciding if a sentence is in $\mathcal{T}$.

$$\text{decide}_{\mathcal{T}}(\exists \vec{x}_\Psi : \Psi) \quad \Longrightarrow \quad \text{satisfiable}_{\mathcal{T}}(\Psi)$$

Equivalence when theory is *complete* only.

Comparison of theories

- $\mathcal{T}_1$ more general than $\mathcal{T}_2$ iff $\mathfrak{M}(\mathcal{T}_2) \subseteq \mathfrak{M}(\mathcal{T}_1)$

$\Rightarrow$ $\text{satisfiable}_{\mathcal{T}_2}(\Psi) \Longrightarrow \text{satisfiable}_{\mathcal{T}_1}(\Psi)$

$\Rightarrow$ We can use decisions in $\mathcal{T}_2$ to approximate satisfiability in $\mathcal{T}_1$

- $\mathcal{T}_1 \cup \mathcal{T}_2$ is the combination of $\mathcal{T}_1$ and $\mathcal{T}_2$

Axiomatic Semantics

- Gives a semantics to program in terms of logical formulæ
- Ordered by $\implies$
- Approximation of the concrete semantics (but often exact)

Example

$$
\begin{array}{rcl}
\mathsf{f}_{\mathfrak{a}} & \in & (\mathbb{x} \times \mathbb{T}(\mathbb{x}, \mathbb{f})) \to \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p}) \to \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p}) \\
\mathsf{f}_{\mathfrak{a}} [\![\mathrm{x} := t]\!] \Psi & \triangleq & \exists x' : \Psi[\mathrm{x} \leftarrow x'] \land \mathrm{x} = t[\mathrm{x} \leftarrow x'] \\
\mathsf{b}_{\mathfrak{a}} & \in & (\mathbb{x} \times \mathbb{T}(\mathbb{x}, \mathbb{f})) \to \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p}) \to \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p}) \\
\mathsf{b}_{\mathfrak{a}} [\![\mathrm{x} := t]\!] \Psi & \triangleq & \Psi[\mathrm{x} \leftarrow t] \\
\mathsf{p}_{\mathfrak{a}} & \in & \mathbb{C}(\mathbb{x}, \mathbb{f}, \mathbb{p}) \to \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p}) \to \mathcal{B} \\
\mathsf{p}_{\mathfrak{a}} [\![\varphi]\!] \Psi & \triangleq & \Psi \land \varphi
\end{array}
$$

Example of program

```
x=0;
while(true)
  x = incr(x)
```

$F_{\mathfrak{a}}[\![\mathtt{P}]\!](\Psi) \triangleq (\mathtt{x}=\mathtt{0}) \vee (\exists x' : \Psi[\mathtt{x} \leftarrow x'] \wedge \mathtt{x} = \mathtt{incr(x)}[\mathtt{x} \leftarrow x']) \iff (\mathtt{x}=\mathtt{0}) \vee (\exists x' : \Psi[\mathtt{x} \leftarrow x'] \wedge \mathtt{x} = \mathtt{incr}(x'))$

- $F_{\mathfrak{a}}[\![\mathtt{P}]\!]^0 \triangleq \mathtt{false}$
- $F_{\mathfrak{a}}[\![\mathtt{P}]\!]^1 \triangleq F_{\mathfrak{a}}[\![\mathtt{P}]\!](F_{\mathfrak{a}}[\![\mathtt{P}]\!]^0) =$ $(\mathtt{x}=\mathtt{0}) \vee (\exists x' : \mathtt{false}[\mathtt{x} \leftarrow x'] \wedge \mathtt{x} = \mathtt{incr}(x'))$
- $F_{\mathfrak{a}}[\![\mathtt{P}]\!]^2 = (\mathtt{x}=\mathtt{0}) \vee (\exists x_2 : (x_2 = \mathtt{0}) \wedge \mathtt{x} = \mathtt{incr}(x_2))$
- ...

No least fixpoint, even though theory is decidable.

Multi-interpreted Semantics

- Give semantics in *a set of interpretations*
- Could correspond e.g. to different platforms of execution, loose specification of language, ...

$$
\begin{array}{lcll}
\mathcal{R}_I & & & \text{program observables} \\
\mathcal{P}_\mathcal{I} & \triangleq & I \in \mathcal{I} \not\rightarrow \wp(\mathcal{R}_I) & \text{interpreted properties} \\
 & \simeq & \wp(\{\langle I, \eta\rangle \mid I \in \mathcal{I} \wedge \eta \in \mathcal{R}_I\}) &
\end{array}
$$

Example

For imperative programs, $\mathcal{R}_I = \mathbb{x} \to I_\mathcal{V}$ and

$$
\begin{array}{rcll}
\mathsf{f}_\mathcal{I}\,[\![\mathrm{x} := e]\!]\,P & \triangleq & \{\langle I, \eta[\mathrm{x} \leftarrow [\![e]\!]_I\,\eta]\rangle \mid I \in \mathcal{I} \wedge \langle I, \eta\rangle \in P)\} & \text{post-condition} \\
\mathsf{b}_\mathcal{I}\,[\![\mathrm{x} := e]\!]\,P & \triangleq & \{\langle I, \eta\rangle \mid I \in \mathcal{I} \wedge \langle I, \eta[\mathrm{x} \leftarrow [\![e]\!]_I\,\eta]\rangle \in P\} & \text{pre-condition} \\
\mathsf{p}_\mathcal{I}\,[\![\varphi]\!]\,P & \triangleq & \{\langle I, \eta\rangle \in P \mid I \in \mathcal{I} \wedge [\![\varphi]\!]_I\,\eta = \mathit{true}\} & \text{test}
\end{array}
$$

Abstractions between Multi-interpretations

We must consider

- $\mathcal{I}$ the set of interpretations for which the program is defined
- and $\mathcal{I}^\sharp$ the set of interpretations used in the analysis

Then we have the Galois connections (for the $\subseteq$ ordering):

$$\langle \mathcal{P}_{\mathcal{I}}, \subseteq \rangle \underset{\alpha_{\mathcal{I} \to \mathcal{I}^\sharp}}{\overset{\gamma_{\mathcal{I}^\sharp \to \mathcal{I}}}{\leftrightarrows}} \langle \mathcal{P}_{\mathcal{I}^\sharp}, \subseteq \rangle$$

where

$$\begin{aligned}
\alpha_{\mathcal{I} \to \mathcal{I}^\sharp}(P) &\triangleq P \cap \mathcal{P}_{\mathcal{I}^\sharp} \\
\gamma_{\mathcal{I}^\sharp \to \mathcal{I}}(Q) &\triangleq \left\{ \langle I, \eta \rangle \mid I \in \mathcal{I} \wedge \left(I \in \mathcal{I}^\sharp \implies \langle I, \eta \rangle \in Q \right) \right\}
\end{aligned}$$

Example of abstractions

- Uniform abstraction: forget about the interpretations

$$\langle \mathcal{P}_{\mathcal{I}}, \subseteq \rangle \underset{\alpha_{\mathcal{I}}}{\overset{\gamma_{\mathcal{I}}}{\leftrightarrows}} \langle \cup_{I \in \mathcal{I}} \mathcal{R}_I, \subseteq \rangle$$

$$\begin{array}{rcl} \gamma_{\mathcal{I}}(E) & \triangleq & \{ \langle I, \eta \rangle \mid \eta \in E \} \\ \alpha_{\mathcal{I}}(P) & \triangleq & \{ \eta \mid \exists I : \langle I, \eta \rangle \in P \} \end{array}$$

  - ASTRÉE does that for rounding errors of floating points computations
- Abstraction by a theory: only keep interpretations in the theory
  - theories used to represent an infinite number of interpretations
  - Necessarily an approximation when we have just *one* interpretation
  - But no best interpretation (Gödel's first incompleteness theorem)

Logical Abstract Domains

Difficult points

- Computing (or approximating) the least fixpoint
- Checking that the invariant is strong enough to prove desired property

Solutions

- Restrict the set of formulæ to enforce ascending chain condition
- Use a decidable theory

Definition

Logical Abstract Domain = set of formulæ + a theory

Ordering is $(\Psi \sqsubseteq \Psi') \triangleq ((\forall\, \vec{x}_\Psi \cup \vec{x}_{\Psi'} : \Psi \implies \Psi') \in \mathcal{T})$

Abstraction to Logical Abstract Domain

- Can use context-independent $\text{alpha}_A^{\mathcal{I}} \in \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p}) \to A$
- Soundness: $\forall \Psi \in \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p}), \forall I \in \mathcal{I} : I \models \Psi \implies \text{alpha}_A^{\mathcal{I}}(\Psi)$
- Assignment then becomes $\mathsf{f}^\sharp [\![\mathrm{x} := t]\!] \varphi \triangleq \text{alpha}_A^{\mathcal{I}}(\mathsf{f} [\![\mathrm{x} := t]\!] \varphi)$

Example: Literal Elimination

- $A = \mathbb{F}(\mathbb{x}, \mathbb{f}_A, \mathbb{p}_A)$, $\mathbb{f}_A \subseteq \mathbb{f}$ and $\mathbb{p}_A \subseteq \mathbb{p}$A
- $\Psi[t, \dots, t]$, where $t \in \mathbb{f} \setminus \mathbb{f}_A$ is approximated by $\exists x : \Psi[x, \dots, x]$

Example: Quantifier Elimination

- A is quantifier-free
- Quantifiers can be eliminated without loss of precision in some theories (but size blow-up)
- But approximations, using heuristics are possible (Simplify, ...)

Other Abstract Operations

Examples of Widenings

- Widen to finite sub-domain
- Limit the size of formulæ, eliminating new literals (in conjunctive form)
- Reduce only the evolving parts, comparing syntactic evolution
- Make generalizations ($I(1) \vee I(2) \vee \ldots$ implies $\exists k \geqslant 0 : I(k)$)

- Can be composed with other abstract domains
- Nelson-Oppen procedure is an instance of domain reduction

$\Rightarrow$ Reuse of existing, well tested and efficient SMT solvers

- Satisfiability can be approximated

Structures

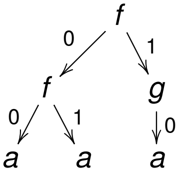
- To describe an infinite set, need a structure or algebra
- The most general:
  - uninterpreted symbols
  - combined
  - $\Rightarrow$ trees (Herbrand model), or if possible graphs
- $\Rightarrow$ Representing sets of trees
- For what usage?...

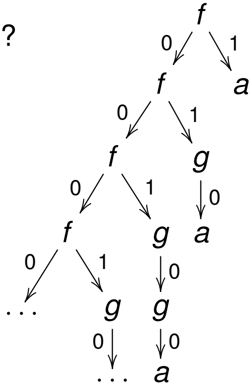
What Usage?

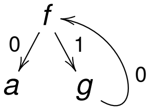
- Static analysis :
  - sets of traces
  - memory shapes
  - protocol analysis
  - any non-linear property (term algebra);
- *Computation* of a set of terms:
  - abstract transfer functions
  - fixpoint testing (inclusion)
  - testing emptiness
  - union, but with a lot of recomputations

What trees?

- Labeled trees;
- Finite number of children (finite arity of children, but not compulsory);
- Ordered children;
- Possibly infinite trees?







What graphs?

Definition

- An oriented graph is a set of nodes V and a set of edges $E \subseteq V \times V$
- An oriented labeled multigraph is a set of nodes V, a node labeling function $(V \rightarrow F)$, and a set of labeled edges $E \subseteq V \times V \times L$.

- Example: program heap structure
  - Node = memory location
  - Node label = data
  - Labeled edge = named field pointing to another memory location
- From now on: graph = oriented labeled multigraph

What Tree language?

Representing *everything* is impossible.
$\Rightarrow$ Each representation defines a class of tree languages.

Relevance of the class

- What trees (infinite, regular...)?
- True branching or linearity?
- If branching, what level of relationship between subtrees?

Operations closure

- In general, yes for boolean operations
- In general, no for limits of sequences of languages

$\Rightarrow$ Approximating tree languages (smartly?)

1. Boolean Relations
   - Boolean Formulæ
   - Decision Trees
   - BDD approximation
2. Cartesian Approximation
   - Classic Logic
   - Kleene's Logic
3. More Interpretations to Logical Formulæ
   - Satisfiability Modulo Theory
   - First Order Logic as Abstract Domains
4. Graphs and Trees
   - Classic Representations
   - Example of Representation Designed for AI

A few examples using variables:

- Tree Grammars:
  - simple and easy to understand (good descriptive tools),
  - unsuccessful attempts to use them in static analysis (bad tools for automatic manipulation);
- Set constraints:
  - with $\cup$ and $\cap$, emptiness testing is EXPTIME,
  - possibility to add infinite trees using coinductive definitions;
- μ-calculus:
  - powerful tool to describe languages over possibly infinite trees,
  - too powerful for a practical usage.

Usage as a Representation for Automatic Manipulation

- Inherent default of representations using expressions:
  - renaming and increasing number of variables;
  - looking for normal (or just simplified) forms.
- Lesson: the more operations we use in expressions ($\cup$, $\cap$), the more equality testing is difficult;
- in practice :
  - if representation not too powerful, translated into an automaton,
  - if too powerful, restrain to a proper subset, then translate into an automaton.

Definition of Tree Automata

- Invented to show the decidability of a logic;
- Natural extension to word automata;
- Word automata are a good representation

$\Rightarrow$ using tree automata for practical representation

But there are differences between the two classes of automata

Definition

- A: alphabet (or labels),
- Q: set of states,
- $\delta \subset Q \times A \times Q^n$: transition relations ($n = 1$ for words),
- $I, F \subset Q$: sets of starting states and ending states.

Comparing words/trees

Word automata

- Defines rational languages, quite expressive in practice.
- Same class if δ is deterministic $(Q, A) \to Q$.

Tree automata

- Trees can be read bottom-up or top-down
- Not the same class for top-down deterministic $((Q, A) \to Q^n$ not isomorphic to $(Q^n, A) \to Q)$
- Complexity: $\mathcal{A}_1 \equiv \mathcal{A}_2$ is EXPTIME
- Expressivity: cannot express $\begin{tikzcd}[column sep=tiny] & f \arrow[dl] \arrow[dr] & \\ x & & x \end{tikzcd}$ and infinite trees.

Tree Automata in Practice

Efficient Representation of δ

Representation of the decision process using compressed tables [Börstler, Moncke and Wilhelm 1991] or BDDs: each $A \rightarrow Q$ is represented by BDD [MONA, par Klarlund].

Guided Automata (MONA)

- Idea: Top-down deterministic automata are less complicated

$\Rightarrow$ Divide the tree space using a deterministic top-down automaton, then in each space, use bottom-up automata.

- Automaton is run in 2 steps: first marking top-down, then finer automata.
- Minimisation complex.

Extensions of Tree Automata

Infinite Trees

- Diversity of automata (Rabin, Büchi, Streett)
- For each of them, heavy complexity: $\emptyset$ is PSPACE, determinisation doubly exponential .

$\Rightarrow$ Not used in practice.

Automata with constraints between subtrees

- Add constraints ($=$ and $\neq$) to production rules;
- $\emptyset$ undecidable
- $\emptyset$ decidable if constraint between brothers only
- practical application?

Finding a Good Data Structure for Symbolic Properties

In the unbounded case

- Most general structures for symbolic properties:
  - Trees, graphs
  - Sets of trees or even sets of graphs?
- Classical representations
  - Expressions, using variables, seem a bad idea
  - Automata are not well tailored to static analysis

New Representation for Sets of Trees

- Expressive enough
- Efficient for incremental computations
- Can take advantage of approximations

Sharing and Incrementality

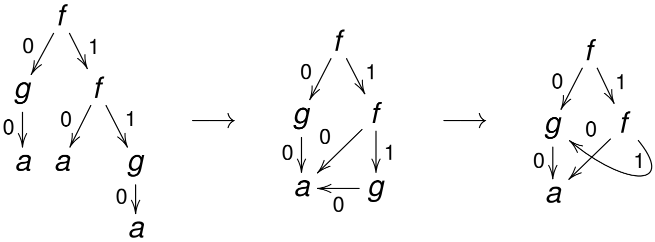
Sharing

- Objects are represented by a data structure
- This data structure is stored at a given memory address
- Representation shared iff no two memory address contain data structures representing semantically equal objects

- Gain in memory
- Constant time equality $\Rightarrow$ easy memoization
- But hidden cost: when computing a new object
  - must be compared with all other represented objects
  - can be made efficient with hash-like techniques
  - but what is the interest compared with on-demand equality testing?
- Only interesting if highly **incremental**

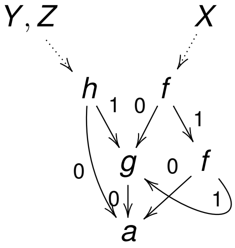
The Easy Case

The most classical representation with sharing is hash-consing of trees:



- Bottom-up process
- *Incremental*: not need to compute everything again at each tree modification

Uniqueness

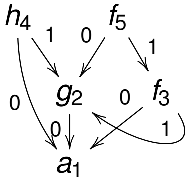


Mechanism

Dictionary + key

Key = label + sub-trees id

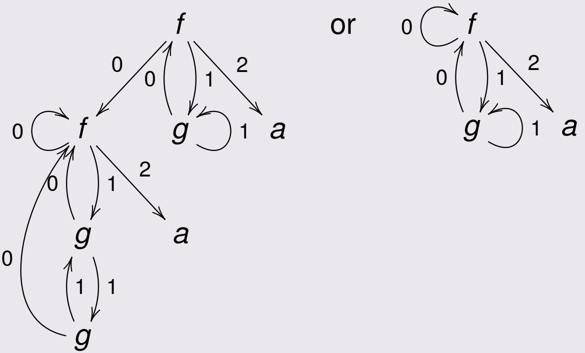
a	:	1
$g(1)$	:	2
$f(1,2)$	:	3
$h(1,2)$	:	4
$f(2,3)$	:	5



Regular Trees

Regular = *finite* number of distinct sub-trees

Example



- Same complexity as oriented labeled multigraphs
- Question: how to extend hash-consing to graphs?

Equivalent Graphs

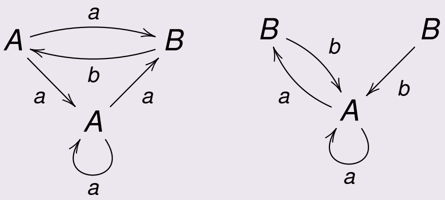
- First determine the semantic equality
- Idea: all what we can observe of a graph is
  - Node labels
  - Follow edges by specifying labels (=paths)

Equivalent graphs

- Two nodes can be distinguished iff there is a path starting from one of the nodes, such that there is no path starting from the other with same edge labels and leading to nodes with same labels
- Two edges can be distinguished iff different label or link distinguishable nodes.
- Two graphs are equivalent iff each node of each graph is undistinguishable from a node of the other graph.

Example of equivalent graphs

Example



Minimal graph

Definition

A graph is minimal iff all its nodes are distinguishable.

- If we store all the graphs encountered in an analysis
- Then it forms a big graph
- If it is minimal, then no redundancy

$\Rightarrow$ We can easily reuse previous computations
  - To recognize if a graph argument has already been encountered, just compare the nodes (= memory locations).
  - Notion of maximal sharing.

- But systematic sharing might not be profitable

How to compute a minimal graph?

- Finding the minimal graph amounts to a graph partitioning problem

$\Rightarrow$ Can be done in $O(n \log n)$.

- Algorithm similar to Hopcroft for automata (refine a partition)
- But not incremental at all.

The Incremental Minimality Problem

- Suppose a minimal graph $\mathcal{U}$ (i.e. uniquely represented graphs)
- Let $\mathcal{G}$ be a graph containing $\mathcal{U}$.
- Extend $\mathcal{U}$ in a minimal graph $\mathcal{U}'$ such that all nodes of $\mathcal{G}$ is equivalent to a node of $\mathcal{U}'$.

- Classical hash-consing algorithm?
- cannot be used: there is no bottom in a graph

Extending a minimal graph

- What we can observe of a graph is what is reachable

$\Rightarrow$ we have a notion of *bottom-up*

Definition

A graph $\mathcal{G} = (V, l, E)$ contains a graph $\mathcal{G}' = (V', l', E')$ iff

- $V' \subseteq V$
- and $\forall v \in V',\ l'(v) = l(v)$
- and $E' \subseteq E$
- and no edge in E starts in V' and ends in $V \setminus V'$
  $(\forall (v_1, v_2, a) \in E, v_1 \in V' \Rightarrow v_2 \in V')$

- A graph $\mathcal{U}'$ extends a graph $\mathcal{U}$ means that $\mathcal{U}'$ contains $\mathcal{U}$, so that no outgoing edge is added

Strongly Connected Components
à la Hopcroft Minimisation Algorithm

- A new strongly connected component is either entirely in $\mathcal{U}$ or outside it.
- There does not seem to be any better algorithm than partition refinement for such graphs...

A Partition Refinement Algorithm

- Start with a set of blocks (corresponding to a coarse partition)
- Le W be the set of (B, l), with B a block and l an edge label
- while W is not empty, take (B, l) out of W
  - Compute for each node the number of l-labeled edges leading to B
  - Split each block according to that number
  - if a block was not in W, only add the smallest split blocks in W

- Complexity: $O(n \ln(n))$

Recognizing Strongly Connected Components

Problem

- Minimizing a new strongly connected component does not share it
- Too costly to minimize $\mathcal{U}$!
- Better way to recognize a strongly connected component?

- Want to compare with as few as possible sub-graphs (limited-depth hashing?)
- Want to avoid costly equality testing

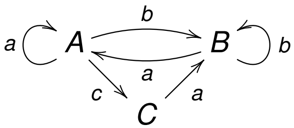
$\Rightarrow$ find a characteristic key?

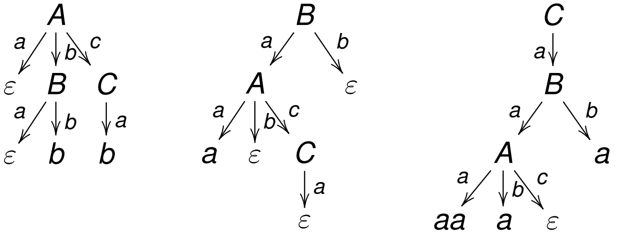
Characteristic property

Isomorphic cycles have the same set of labeled paths

Characteristic Set of Trees for a Strongly Connected Graph

The set of all paths can be described by a finite set of trees

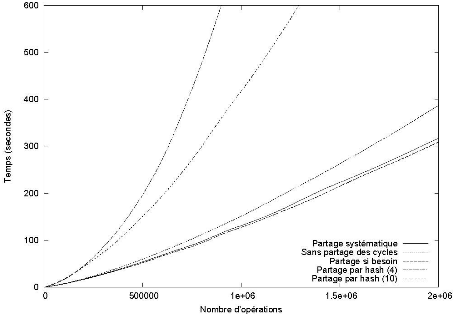


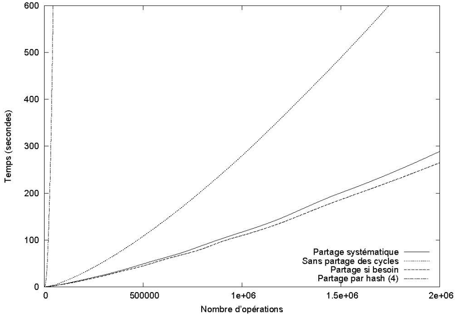


Comparison with Finite Height Hash-Consing

Experimental results on random graph incremental manipulations and equality testing show that

1. Sharing is always faster than no sharing
2. Finite height hash-consing is far less efficient than cycle hash-consing
3. Sharing on demand is slightly more efficient than systematic sharing





Application to Word Automata

- As a graph, word automata have the same equivalence notion as defined earlier, if
  - deterministic
  - and complete (no forbidden transition) or useful (all states can lead to a final state)

Static Analysis Application

Approximate the messages on channels between parallel processes

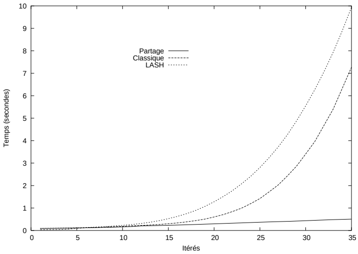
Approximation

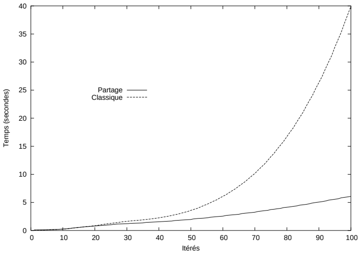
Using Q-automata: encodes a sequence of languages by a regular language

Experimental Results for Message Analysis

- Fixpoint computation
- Without minimisation, automata grow very quickly $\Rightarrow$ inclusion algorithms become very costly
- Full minimisation at each step too costly

$\Rightarrow$ substantial speed-up with shared automata





Widenings for Graph based Representations

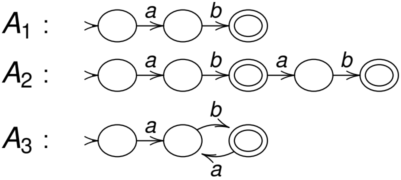
Widening

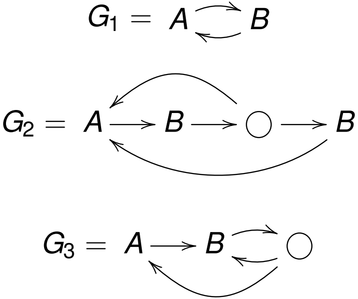
Widening is an approximation of unions used to speed-up convergence of iterations

- Essential to yield precise analysis (which demand infinite domains)
- Tries to extrapolate on successive iterates

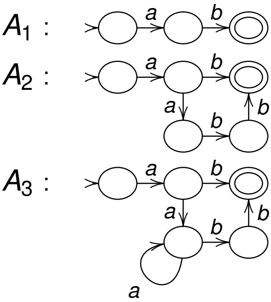
- Graph folding
  - Try to replace a new node by an old one with the same label
  - Only if this old one represents more values
- Path extrapolation
  - Repeat infinitely a newly added edge (or path).
  - Approximates $\{ a^n b^n \mid n \in \mathbb{N}\}$ by $a^k a^* b^k b^*$
- Size limiting
  - After a pre-defined size of the graph reached, replace new nodes by $\top$.
  - Enforces termination.

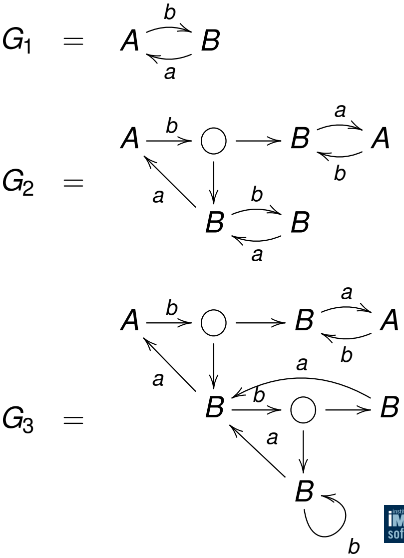
Examples of Graph Folding



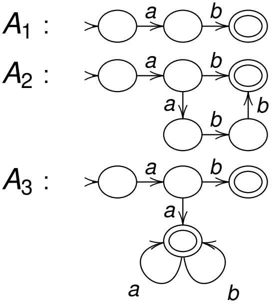


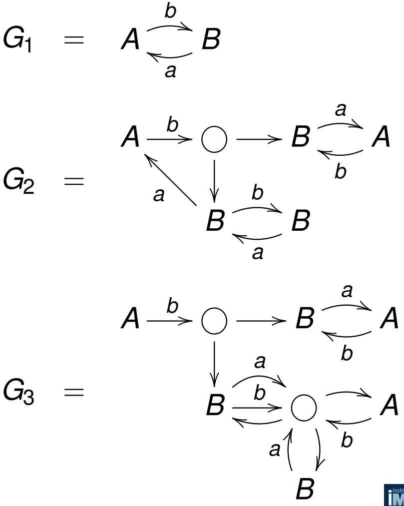
Examples of Path Extrapolation





Examples of Size Limiting





Sets of Trees

Sharing Tree Automata?

- A tree automaton is not a graph
- Hypergraph = set of nodes + set of tuples of nodes

Using a Graph + Interpreted (union) Label?

- Equivalence is not the equality of paths
- Unless normal form?
- Potential problem of cartesian approximation

Introduction of a choice node

Set of trees = tree
Just add a root with special label, and children the elements of the set.

Example

$$\left\{ \begin{array}{ccc} & f & \\ \swarrow & & \searrow \\ a & & b \end{array} \; , \; \begin{array}{c} g \\ \downarrow \\ b \end{array} \; , \; a \right\}$$

would be represented by

$$\begin{array}{ccccc} & & & \bigcirc & \\ & & \swarrow & \downarrow & \searrow \\ & f & & g & a \\ \swarrow & & \searrow & \downarrow & \\ a & & b & b & \end{array}$$

Efficient representation of trees $\Rightarrow$ Efficient representation of sets of trees (?)

Uniqueness of the Skeleton

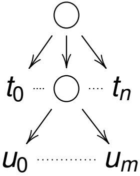
To have a maximal sharing representation:

- we must obtain uniqueness of the skeleton;
- Valid skeleton = regular tree and restrictions;

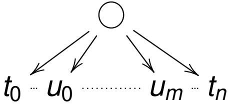
$\Rightarrow$ not all sets of trees can be represented by a skeleton.

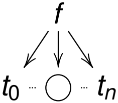
Obvious Restrictions

is equivalent to t



is equivalent to





is equivalent to  (empty set)

Conventional Restriction

Last problem: ordering the children of a choice node

- Solution: total ordering on trees
- Too expensive $\Rightarrow$ partial ordering = ordering over labels

So ordering of the children of a choice node = ordering on the labels of their roots.

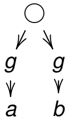
Simplifications

- Skeleton = first approximation;
- We want efficient;
- **Simplification**: share common prefixes

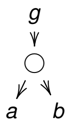
$\Rightarrow$ **All subtrees of a choice node have a different root label.**

$\Rightarrow$ the uniqueness problem is solved!

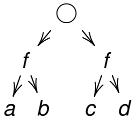
Simplification Examples



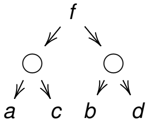
will be represented by



.



will be approximated by

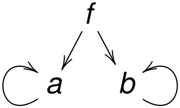


.

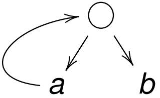
Expressive Power of Tree Skeletons

- Represent infinite trees too $\Rightarrow$ greatest fixpoint semantics;
- *i.e.* a tree skeleton represents the set of all finite and infinite trees we can form by going through the skeleton.
- If we limited to finite trees, same expressive power as deterministic top down tree automata;
- Advantage: incremental sharing.

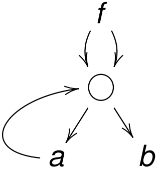
Examples of Skeletons



represents the tree $f(a^\omega, b^\omega)$.



represents the set $a^*b \cup a^\omega$.



represents the set of trees $f(a^*b, a^*b) \cup f(a^\omega, a^*b) \cup f(a^*b, a^\omega) \cup f(a^\omega, a^\omega)$. The sets of left and right children are shared.

Usage of Tree Skeletons

- Tree skeletons are simple and efficient;
- Can be used as an abstract domain to over-approximate sets of trees;
- Intersection of 2 skeletons is representable by a skeleton, but not union;
- There exists a best approximation for finite union, and a widening for infinite union;
- First approximation for more expressive tree schemata.

Bibliography

- R. E. Bryant. Graph Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers C-35*, 1986.
- L. Mauborgne. An Incremental Unique Representation for Regular Trees. *Nordic Journal of Computing 7(4)*, 2000.
- N. Kettle, A. King and T. Strzemecki. Widening ROBDDs with Prime Implicants. *TACAS*, 2006.
- A. R. Bradley and Z. Manna. The Calculus of Computation, Decision procedures with Applications to Verification. *Springer*, 2007.
- H. Comon et al. Tree Automata Techniques and Applications. 2007.
- P. Cousot, R. Cousot and L. Mauborgne. Logical Abstract Domains and Interpretations. *The Future of Software Engineering*, 2010.
- P. Cousot, R. Cousot and L. Mauborgne. The Reduced Product of Abstract Domains and the Combination of Decision Procedures. *FoSSaCS*, 2011.