Symbolic Abstract Domains

3 / 3

Laurent Mauborgne

École Normale Supérieure

Interprétation abstraite, MPRI 2–6, année 2007-2008

Lesson Plan
Thirs Session

Finite Sets of Symbols

Graphs and Infinity

Mixing Symbolic and Numeric Properties

Graphs and Infinity

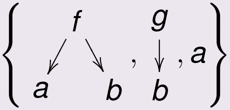
1. Classic Representations for Infinite Sets of Symbols
2. Incremental Maximal Sharing
3. Relations
4. Sets of Trees
   - First Approximation: tree skeletons
   - Adding Links: the Tree Schemata
   - In Practice

Introduction of a choice node

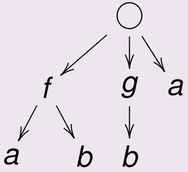
Set of trees = tree

Just add a root with special label, and children the elements of the set.

Example



would be represented by



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

Unicity of the Skeleton

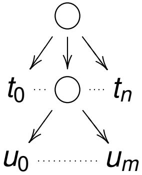
To have a maximal sharing representation:

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

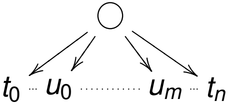
$\Rightarrow$ not all sets of trees can be represented by a skeleton.

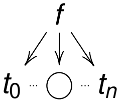
Obvious Restrictions

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.

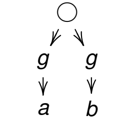
Simplifications

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

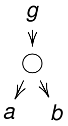
⇒ **All subtrees of a choice node have a different root label.**

⇒ the unicity problem is solved!

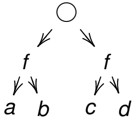
Simplification Examples



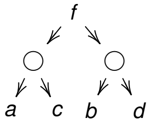
will be represented by



.



will be approximated by

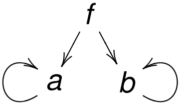


.

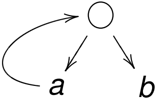
Expressive Power of Tree Skeletons

- 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.

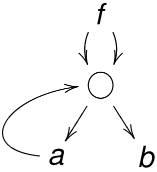
Examples of Skeletons



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

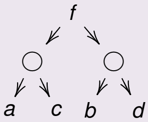
- Tree skeletons are simple and efficient;
- Can be used as an abstract domain to over-approximate sets of trees;
- Intersection of 2 skeletons is representable by a skeleton, but not union;
- There exists a best approximation for finite union, and a widening for infinite union;
- First approximation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

Graphs and Infinity

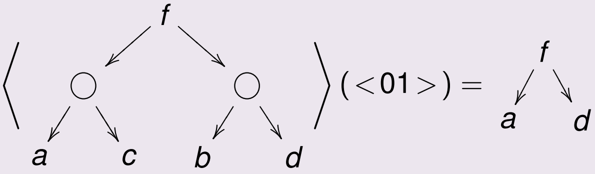
The Choice Space

- Choice space of a skeleton = set of choice possibilities;
- Skeleton = function : choice space $\rightarrow$ trees.

Example

The choice space of the skeleton 

is $\{0, 1\} \times \{0, 1\}$.



The Links

- To refine skeletons, just forbid some elements of the choice space;
- A subset of a cartesian product is a relation;

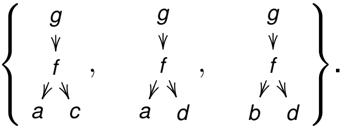
$\Rightarrow$ We force a relation on the choices of the skeleton.

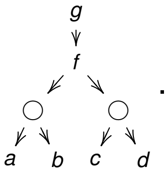
The links of the schema represent that relation.

Links are Local

- Relation decomposed in independant parts: to gain memory;
- Local links: the schemata are more incremental;
- Link = relation and choices attached to a given link
- Notion of entry in the relations.

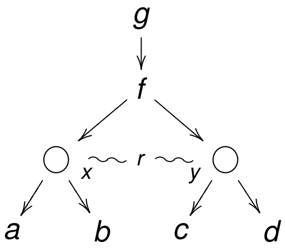
Tree Schema Example



The best skeleton to approximate it is: 

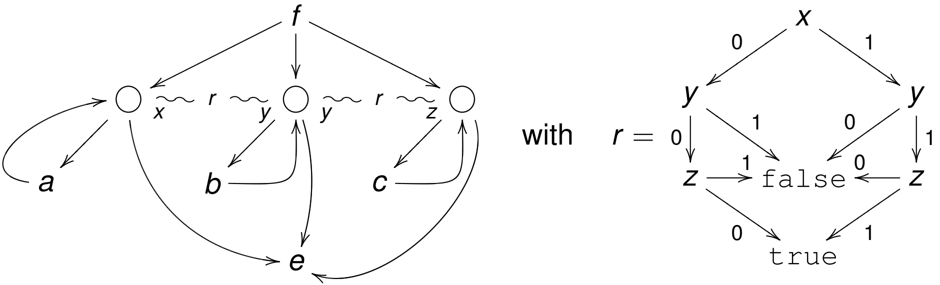
Tree Schema Example (2)

The corresponding tree schama would be



where r is the relation true when x is 0 or x and y are 1.

An Example that Cannot be Represented by a Tree Automaton



Note that r is the equality relation

Represents $\{ f(a^n e, b^n e, c^n e) \mid n \in \mathbb{N}\}$

Tree Schemata Semantics
By Pseudo-decision Process

Principle

- Decision to know if a tree $\in$ schema
- Go through the tree and the schema in parallel
- Each link is associated with a stack of partially evaluated relations

Pseudo-algorithm

- Each label of the children must correspond
- In the choice nodes, a choice must correspond to the current label in the tree
- The choice is possible iff first partially evaluated relation not evaluated on that entry allows that choice.
- If yes, partial evaluation of that relation
- If necessary, stack a new relation for the link

Graphs and Infinity

Properties

- Union and intersection of two schemata are representable by a tree schema;
- Inclusion is decidable;
- Projection of a schema is representable by a schema.

Sharing

- No unicity, except for $\bigcirc$
- But unicity of the underlying skeleton
- Depending on the choice for relations, sharing of links may be possible.

Sketch Ideas for the Algorithms

Principle stages of the algorithms

- First, fast algorithm on underlying skeletons
- If necessecary, relations are extended
- Algorithm on relations

Precision/speed trade-off

Choice of the class of relations

- Nothing (= just skeletons)
- Equalities
- Finite relations
- ω-deterministic
- Büchi...

The Framework (reminder)

Static Analysis by Abstract Interpretation

$P \longrightarrow$ behavior (= $I \rightarrow L$, L language)

Collecting $P \longrightarrow \wp(I \rightarrow L)$

Formal Languages $P \longrightarrow I \rightarrow \wp(L)$

Fixpoint of F, definied by meta-language

Meta-language

$$e ::= \quad \mathcal{X} \quad | \quad \left\{ T' : T_1 \in e_1, \ldots, T_n \in e_n \right\} \quad | \quad e_1 \cup e_2$$

$\mathcal{X} \in \mathcal{V}$ $T' \in \mathcal{H}(F \cup v)$, v variable and T' of finite variability
$T_i \in \mathcal{H}(F \cup v)$ quelconques

intersection $\{x : x \in e_1, x \in e_2\}$

projection $\left\{ x : \begin{array}{c} f \\ \swarrow \searrow \\ x \quad y \end{array} \in e \right\}$

Expressive Power

Limitations

$$\mathcal{X} = \left\{ \begin{array}{c} f \\ \swarrow \quad \searrow \\ x_0 \;\cdots\; x_{n-1} \end{array} : (x_i \in e_i)_{i<n} \right\}$$

Expressible cases

$$\mathcal{X} = \left\{ \begin{array}{c} f \\ \swarrow \quad \searrow \\ x \qquad y \end{array} \right\}$$

$$\mathcal{X} = \left\{ x : \begin{array}{c} f \\ \swarrow \quad \searrow \\ x \qquad y \end{array} \in \mathcal{X} \right\}$$

Example: Weak Fairness (1/5)

Parallel programme:

$$X = \texttt{true}; [\![\texttt{while } X \texttt{ do skip} \| X = \texttt{false}]\!]$$

The positions in the program are:

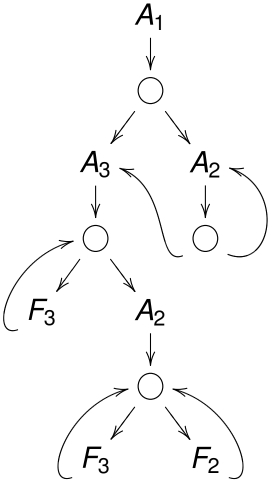
$$A_1 [\![A_2 \| A_3]\!]$$

with F_2 and F_3 the ends of A_2 and A_3.

Example: Weak Fairness (2/5)

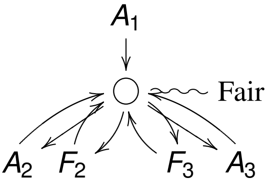
Interleaving trace semantics:

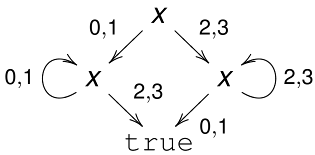
$T =$



Example: Weak Fairness (3/5)

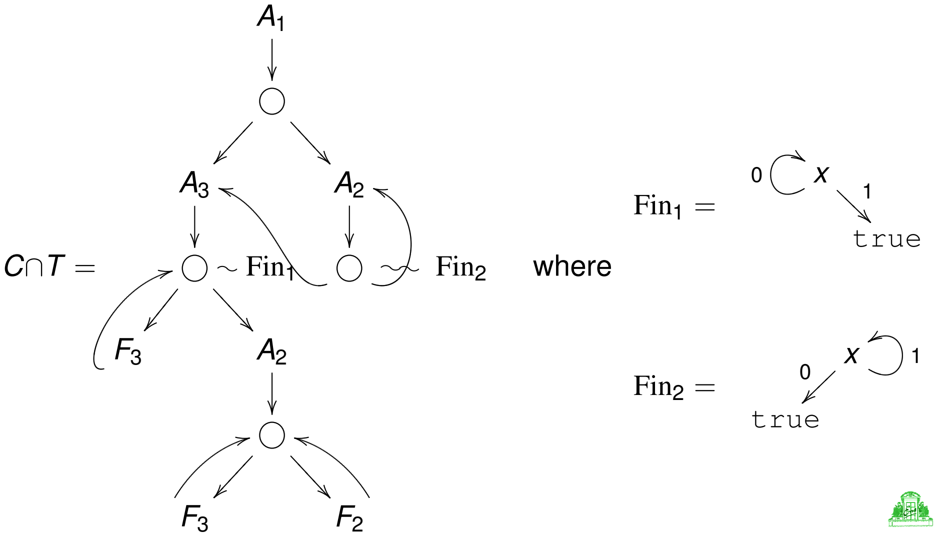
Controler forcing equity:

$C =$ 

where $\mathrm{Fair} =$ 

Example: Weak Fairness (4/5)

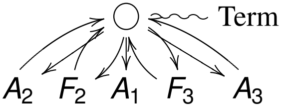
Controler is introduced by intersection:



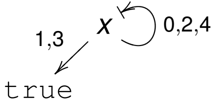
Example: Weak Fairness (5/5)

Now, we can prove the the program always terminates, under weak fairness assumption, by showing the inclusion of $C \cap T$ into

$F =$



where Term $=$



Part III

Mixing Symbolic and Numeric Properties

Mixing Symbolic and Numeric Properties

Mixing Symbolic and Numeric Properties

Numeric Domains to Help for Symbolic Properties

- What Makes Symbolic Properties Hard?
  - Lack of structures
  - In some cases, hierarchy, ...
  - Generic remedy: try to discover some structure on the fly
- Extract numbers from symbolic properties
  - Then represent numbers with abstract domains (relational or not)
  - Maybe a structure can emerge?
  - Widenings on the numbers
- Two main paths:
  - Approximate symbolic properties by numeric values
  - Introduce counters in representations

Approximating Words by Numbers

Approximation Principle

- A word will be represented by a vector of numbers
- So a vector of numbers represents a set of words
- To make use of numerical abstract domains: limit the size of the vectors

$$\mathcal{P}(A^\star) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \mathcal{P}(\mathcal{N}^k)$$

- Example: length of the words
- Example: Parikh vector
  - word represented by number of occurence of each letter

$$aabaab \quad \longrightarrow \quad \langle 4, 2 \rangle$$

  - Approximation of free algebra
  - Can give nice results with relational numerical domains

Word Automata with Counters

Definition of Counter Automata

- Automata (Q, δ) with a finite set of counters $\{x_1, \ldots, x_k\}$
- Transition function: $\delta \subset Q \times \Phi \times Q$, where Φ is the set of Presburger formulae over $\{x_1, x_1', \ldots x_k, x_k'\}$
- $q, q' \in Q$ and v, v' valuation of variables, $(q, v) \longrightarrow (q', v')$ iff
  - $\exists (q, \phi, q') \in \delta$
  - and a substitution of σ of the variables in ϕ, valid for ϕ
  - such that $\sigma(x_i) = v(x_i)$, $\sigma(x_i') = v'(x_i)$, when σ defined
  - otherwise $v(x_i) = v'(x_i)$

- Can define word automata that counts the number of times we take an edge
- With just one counter, $\subset$ indecidable
- With restriction, quite a litterature on model checking with counter automata (because Presburger formulae can be represented as automata)

Mixing Symbolic and Numeric Properties

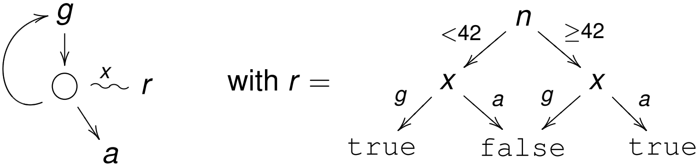
First Step: Introducing New Variables

- Tree schemata $\Rightarrow$ natural decomposition into tree structure + relations.
- Relation: easy to add new variables

New variables

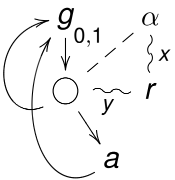
- So far, variable = choice nodes
- But sole constraint is variable over a finite domain
- Inifite domain? $\Rightarrow$ finite partition of the domain ($x \leq 42$, $x > 42$) will do!

Second Step: Counters in the Skeletons

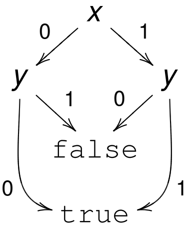


- Only usefull count: time spent in loops
- Counters: count the number of times we go through a choice node
- Counter domain: a partitioning function of $\mathbb{N}$
- Decidability kept: just operations on partitions
- Interest: memory gain and widening help
- Drawback: more possibilies of redundancies

Example with counter

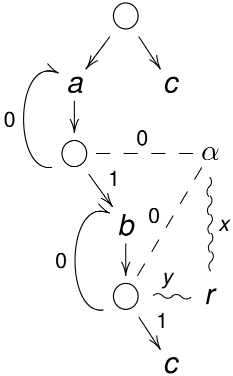


with

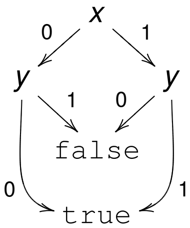
$r =$ 

and $\alpha(i, j) = 1$ iff $(i + j) \equiv 41 \bmod 42$

Example with counter (2)



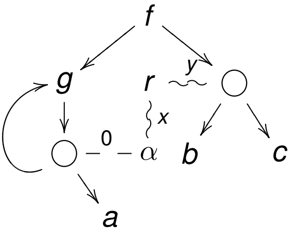
with $r =$



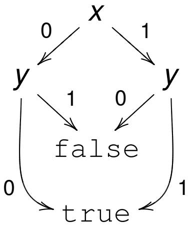
and $\alpha(i,j) = 0$ iff $i > j$

Counters Cannot be Used Everywhere

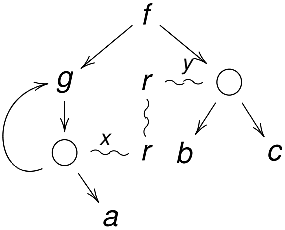
As for Links

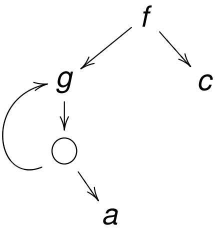


with $r =$



and $\begin{cases} \alpha(0) & = & 1 \\ \alpha(n+1) & = & 0 \end{cases}$





Applications for Counters

- Variables coming from systems to analyse (hybrid representation)
- Automatic detection of loops (creation of counters during iteration)
- Widening by numerical analysis of the counters
- Exact acceleration in some cases

Mixing Symbolic and Numeric Properties

A Simple Example

Mixing Symbolic and Numeric Properties

Example

```
  b = (x>0);•
  if (b) sign = 1;•
  else sign = -1;•
• x = x / sign;•
  assert(x>=0);
```

With set of interval arrays

- $b \in \{0\}$ and $x \leq 0$, or $b \in \{1\}$ and $x > 0$
- $b \in \{1\}$ and $x > 0$ and $sign \in \{1\}$
- $b \in \{0\}$ and $x \leq 0$ and $sign \in \{-1\}$
- $b \in \{1\}$ and $x > 0$ and $sign \in \{1\}$, or $b \in \{0\}$ and $x \leq 0$ and $sign \in \{-1\}$
- $x > 0$ and $sign \in \{1\}$, or $x \geq 0$ and $sign \in \{-1\}$

- Approximations due to convexity $\Rightarrow$ use disjunctions?
- Approximations due to non relational domains $\Rightarrow$ use disjunctions or relational domains?

Approximations

- Common approximations, necessary to scale up:
  - Cartesian (intervals, congruences, ...)
  - Convexity (intervals, polyhedra, octagons, Karr...)
- Where they approximate
  - Unions
  - And some transfer functions, only if there is more than one concrete state

A Natural Solution

Let's use disjunctions

Disjunctive Completion

- There is a theoretical definition for a domain precise on unions

Definition

Let $\mathcal{A}$ a Moore family on S (defining a closure operator $\rho_{\mathcal{A}}$, or a Galois connection), the *disjunctive completion* of $\mathcal{A}$ is

$$\mathit{lfp}_{\subseteq}^{\mathcal{A}} \lambda \mathcal{X} \bullet \mathit{Moore}\left(\mathcal{X} \cup \left\{ \bigcup_{P \in S} \rho_{\mathcal{X}}(P) \;\middle|\; \rho_{\mathcal{X}}\left(\bigcup_{P \in S} \rho_{\mathcal{X}}(P)\right) \neq \bigcup_{P \in S} \rho_{\mathcal{X}}(P) \right\}\right)$$

- It is a constructive definition, and it can be approximated
- In practice, $\simeq$ unions of abstract states, without redundancy

Disjunctive Completion in Practice

- Works with infinite domains (with widening)
- But not easy to tune the cost
- Widening is a complex matter
- Not always easy to reuse existing domains
- Often, big memory cost (especially with relational domains)
- And computation cost (normal forms?)

For big programs analysis

Too costly (both implementation and usage)

Semantic Disjunction

- No systematic disjunction

⇒ Question: when do we perform disjunctions?

- 2 principles

Semantic Criteria

- Identify where precision is needed
  - via a first analysis
  - or dynamically
- perform unions (lose precision) when leaving critical parts

Reuse Abstract Domains

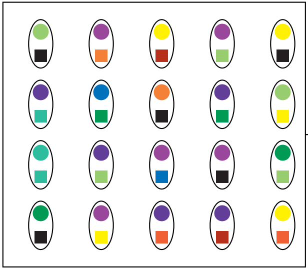
- Forget normalization
- Use abstract domains as parameters

Mixing Symbolic and Numeric Properties

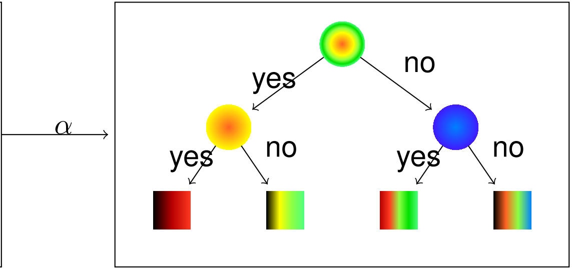
Disjunctions Based on Value Cases

- Disjunction criterion: finite number of disjoint cases covering the space of values of an expression
- Representation: Decision tree
  - Internal nodes = cases
  - leaves = abstract elements of parameter domain

Set of Reachable States



Disjunction based on round colors



Back to the Simple Example

Example

```
  b = (x>0);•
  if (b) sign = 1;•
  else sign = -1;•
• x = x / sign;•
  assert(x>=0);
```

With Disjunction Based on *b*

- if $b \in \{1\}$ then $x > 0$ else $x \leq 0$
- $b \in \{1\}$ and $x > 0$ and $sign \in \{1\}$
- $b \in \{0\}$ and $x \leq 0$ and $sign \in \{-1\}$
- if $b \in \{1\}$ then $x > 0$ and $sign \in \{1\}$, else $x \leq 0$ and $sign \in \{-1\}$
- if b then $x > 0$ and $sign \in \{1\}$, else $x \geq 0$ and $sign \in \{-1\}$

A More Complex Example

Example

```
while(1) {
  b = [0,1];•
  if(b^b') {
    if(b) t = 20;
    else  t = 10;
• } else {
•   if(t>0) t--;
    if(b) x = t/20;
    else  x = t/10;
• }
• b' = b;•
}
```

At the beginning, all variables are initialized to 0

Show that $x < 1$

Disjunctions Based on b and b'

b

$\downarrow 1$

• b'

$0 \downarrow$

$t \in \{20\}$

b

$0 \downarrow$

b'

$0 \downarrow$

$t \in \{0\}$

Mixing Symbolic and Numeric Properties

Implementation Problems

- Testing if shared costly $\Rightarrow$ opportunistic sharing
- Still exponential cost

Solution

- Keep only needed disjunctions
- Group variables in packs

Local Packs Strategy

- Assumption: the invariants needed to prove the code are local
- Identify expressions that are well represented by a domain
- Aggregate locally
- This is the strategy for octagon packs in ASTRÉE
- But not enough for boolean disjunctions...

Global Packs Strategy

- Identify expressions that are well represented by a domain, but put them in tentative packs
  - `b = (x>0)`
  - `if(b) x=...`
- Aggregate according to variable dependence, with a limiter
- Validate tentative packs by expressions where we know the domain can give some precision
  - `if(b) read(x)`
- Possibility to add dynamically some unvalidated tentative packs

Pack Creation Example

Example

```
while(1) {
  b = [0,1];
  if(b^b') {
•   if(b) t = 20;
    else  t = 10;
  } else {
    if(t>0) t--;
•   if(b) x = t/20;
    else  x = t/10;
  }
• b' = b;
}
```

- Creation of the tentative pack (b, t)
- Validation of the tentative pack (b, t)
- Aggregation of b' to the pack (b, t)
- The final pack is (b, b', t)

Results

Inside ASTRÉE, on industrial size embedded codes

Program	test 1		test 2		test 3	
Code size (LOCs)	69 997		273 803		485 663	
♯ of Packs	78		737		1 313	
♯ variables in Packs	247		2 268		3 672	
Average Pack size	3.84		4.15		4.15	
Iterations	146	**150**	146	**146**	158	**158**
Analysis time (min.)	133	**149**	398	**414**	717	**758**
Memory peak (Mb)	672	**676**	1 396	**1 404**	1 733	**1 803**
Alarms	560	**549**	5 214	**5 189**	7 497	**7 464**

Value Based Disjunctions are not Satisfying

- Too costly if we use the entire abstract domains at the leaves
- Keeps unnecessary relations in that case

$\Rightarrow$ Not used on expressions in ASTRÉE (just variables)

- Relational informations may be kept too long

Mixing Symbolic and Numeric Properties

Towards Trace Properties as Disjunction Criteria

- In avionic code, one big loop, but a boolean variable tells if in initialization mode
- Behavior quite different in initialization and permanent modes
- Union of the two behaviors loses information
- But using the boolean as a disjunction criterion is too costly
- initialization = first 24 loop turns

⇒ unroll the loop

Loop Unrolling

Compute the fixpoint of the loop after *k* iterations
⇒ Same as disjunctions based on number of iteration!

Example of Array Initialization

Example

```
i = 0;
while(i < n)•{
  t[i] = i;•
  i++;•
}
```

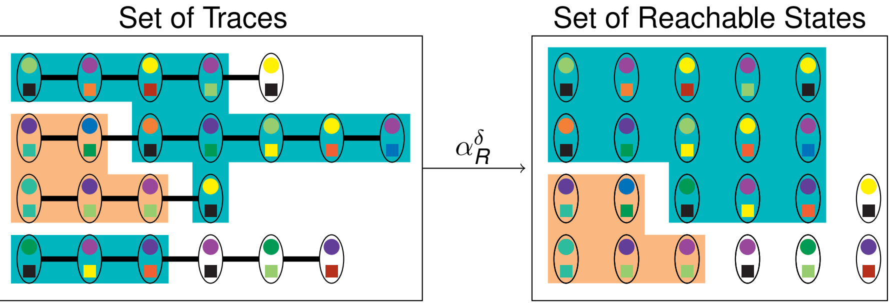
With Loop Unrolling

- $i \in \{0\}$
- $t[0] \in \{0\}$
- $i \in \{1\}$
- No union, because we unroll the loop: after one iteration, *i* is exactly 1
- Strong update $t[1] \in \{1\}$

- Even if the loop not fully unrolled, first *k* values are kept precise
- And the rest is more precise
- Requires to accumulate the false guards

Trace Discrimination

- Add a structure before doing reachability
- Set of Traces S -> Function $E \rightarrow \mathcal{P}(S)$ -> Function $E \rightarrow S^\sharp$



- If size of E is 1, same as classical reachability
- Thm: More precise if the function is a partition
- Thm: The finer the partition, the more precise

Mixing Symbolic and Numeric Properties

Examples of Trace Discrimination I

Final Control State

- In fact, very common
- Consists in keeping one set of states for each control state

Example

```
x = 1;
if (x>0) y = 1/x;
x = -1;
if (x<0) y = 1/x;
```

Examples of Trace Discrimination II

Control Flow

- Partition traces according to the history of choices
- $\times 2$ partition for each test
- Partition not finite for loops
- Really too costly!

Merging

- To make control flow partition tractable
- Keep local discrimination only: discriminate according to bounded past

Application to First Example

Example

```
  if (x>0) sign = 1;•
  else sign = -1;•
• x = x / sign;
• assert(x>0);•
```

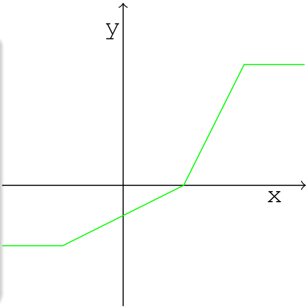
With Intervals and Trace Discrimination

- $x > 0$ and *sign* $\in \{1\}$
- $x < 0$ and *sign* $\in \{-1\}$
- $x > 0$ and *sign* $\in \{1\}$, or $x < 0$ and *sign* $\in \{-1\}$
- $x > 0$ and *sign* $\in \{1\}$, or $x > 0$ and *sign* $\in \{-1\}$
- Then we merge traces: $x > 0$ and *sign* $\in [-1, 1]$

A More Realistic Example

Example

```
 float[] tc = { 0;   0.5;   2; 0};
 float[] tx = { 0;    -1;   1; 2};
 float[] ty = {-1; -0.5; -1; 2};
 int i = 0;
•while (i<3 && x>tx[i+1])
i++;•
•y=tc[i]*(x-tx[i])+ty[i];•
```



- **Intervals:** $i \in \{0\}$ and $x \in [-2, 2]$ $i \in \{1\}$ and $x \in [-1, 2]$ $i \in [0, 1]$ and $x \in [-2, 2]$ $i \in [1, 2]$ and $x \in [-1, 2]$ Fixpoint: $i \in [0, 3]$ and $x \in [-2, 2]$ While output: $i \in [0, 2]$ and $x \in [-2, 2]$ $i \in [0, 2]$ and $x \in [-2, 2]$ and $y \in [-4, 4]$
- **Trace discrimination:** $i \in \{0\}$ and $x \in [-2, 2]$ Loop 1: $i \in \{1\}$ and $x \in [-1, 2]$
  - Input iteration 0: $i \in \{0\}$ and $x \in [-2, 2]$
  - Input iteration 1: $i \in \{1\}$ and $x \in [-1, 2]$
  - Loop 1: $i \in \{1\}$ and $x \in [-1, 2]$

Examples of Trace Discrimination III

Value Based

- Partition traces according to the values of an expression at a certain point
- But can be very costly! (less than value case disjunction)

Example

```
int r = 0;
float x = 0.0;
while(true){
  r = rand(0, 50);
• x = (x * r + rand(-10, 10))
                   / (r + 1);•
}
```

- With intervals:
  - $x \in [-10, 10]$
  - $x \in [-510, 510]$
- Partitioning traces according to r:
  - $x \in [-10, 10]$
  - for each r, $x \in [-10, 10]$

Mixing Symbolic and Numeric Properties

Trace Discrimination Abstract Domain

Elements of the domain

Covering $\times$ Value on covering

- Possibility to
  - start from finest covering,
  - approximate covering (getting coarser)
  - widen the covering
  - then narrow them to get more precise results

  $\Rightarrow$ much more precise than precomputing the discrimination
- Widening: $(\delta_0, S_0^\sharp)\nabla(\delta_1, S_1^\sharp)$
  - compute $\delta = \delta_0 \nabla_{\mathfrak{R}} \delta_1$
  - then widen approximation of $S_0^\sharp$ and $S_1^\sharp$ on δ

Implementation issues

Implemented in the ASTRÉE analyzer

- Because of the industrial constraints
  - not the full trace discrimination domain is implemented
  - mechanisms to make it local
  - first inclusion of partitioning directives
  - then automatizing of the directives inclusions.

Partition Creation

Every time we have a guard

- If partitions
  - traces where test is true
  - traces where test is false
- While loop partitions (at most N)
  - traces that went out of the loop at iteration k
  - traces that staid in the loop more than N iterations
- Integer values partitions
  - traces where expression have different values
  - if more than V values, no partition

Partition Deletion

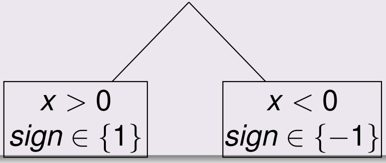
- Function return points
  - only those partitions created in the function are merged
- End of loops
  - only those partitions created in the function are merged
- Merge directives
  - either all partitions
  - or the last created

Abstract Values

- Traces are partitioned according to control point and partition directives
- At each control point, previous partition directives define a tree
  - Leaf = Underlying domain (e.g. intervals)
  - Node = Partition creation
  - One child / element in the partition

Example

```
if (x>0) sign = 1;
else sign = -1;
```



Abstract Transfer Functions

- Partition creation
  - Replace each leaf by a node with children the result of the guard
- Partition deletion
  - If merge all, replace the tree by the union of its leaves
  - Otherwise, replace terminal nodes by union of their children
- Other (non-partition related) functions
  - Replace each leaf by the result of the transfer function

Automatic Partitioning Directive Insertions

- Sources of imprecision of ASTRÉE where analyzed
- Trace partitioning was tested by manual inclusion of directives
- Then automated:
  - Partition divisions by integer if not too big
  - Partition computation of array indexes

Discrimination Strategies

Principle: looking for an identified source of imprecision

Array Access

- Array Access $\simeq$ multiple cases at once
- Critical if two subexpressions share an index

Integer Division

- Loss of precision if dividend and divisor share a variable
- But beware of the cost!

Then discriminate computation of identified variable

- if last assign inside a while, discriminate it
- otherwize discriminate the values

Experimental Results

Inside ASTRÉE, on industrial size embedded codes

Program	test 1		test 2		test 3	
Code size (LOCs)	69 997		273 803		485 663	
Partition Directives	1 037		3 684		5 886	
Iterations	150	**78**	146	**80**	158	**77**
Analysis time (min.)	149	**99**	414	**459**	758	**726**
Memory peak (Mb)	679	**694**	1 404	**1 460**	1 803	**2 240**
Alarms	406	**0**	5 189	**2**	7 464	**0**

Part IV

Conclusion

Domains Presented
Finite Case

Relation $\equiv$ Logical formula $\equiv$ Boolean function

- Possibly exact techniques
  - Formulae + SAT
  - BDDs
- With an a priori approximation
  - Cartesian approximation
  - Kleene logics (TVLA)

Domains Presented
Infinite case

Infinite case: graphs, trees and infinitary relations

graph $\equiv$ infinite tree

decision tree $\equiv$ relation

Graphs represents sets of trees or graphs and relations

Incremental sharing of graphs

- Exact techniques: too expensive
- Cartesian approximation: bottom-up tree automata
- + sharing : skeletons
- + expressive : relations between choices
- Infinitary relations:
  - close, open and quasi-open
  - ω-déterministic for more expressivity

Extracting Numbers From Symbolic Properties

- Approximating words by numbers
- Introducing counters
  - allows for compact representations
  - new widenings
  - but complexity or redundancy problems

Local disjunctions based on history of computation

- Theory:
  - Generic abstract domain construction
  - Flexible
- Practice
  - Scalable
  - Solves precision issue at low implementation cost
    $\Rightarrow$ Good alternative to the design of complex relational domains
  - But we kept the decision trees...

Some Applications

- Shape analysis (TVLA, graphs, counters)
- Protocols (trees or sets of words)
- Trace properties:
  - Simple partitioning (trees)
  - Discrimination according to temporal properties (infinitary relations)
- Typing (trees)
- Temporal Proprerties (schemata, counters + infinitary)

Many Many Paths Remain to be Explored

- Not all classical algorithmics have been explored
- Practical experimentation on the choice of expressiveness for the relations
- Using abstract domain elements as labels
  - new opportunities for sharing by label approximation
  - efficient algorithms
  - decidability?
- Explore the limits of counters
- Under-approximation, mixed with over-approximation
- Equational theories
- Approximation of negation
- Introduction of a symbol for “entire universe”
- Tree concatenation
- For trace discrimination
  - Discriminations according to number of recursive calls
  - Discriminations according to temporal properties
  - Backward analysis with trace discrimination
- Language to express temporal properties