Symbolic Abstract Domains

2 / 3

Laurent Mauborgne

École Normale Supérieure

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

Lesson Plan
Second session

Finite Sets of Symbols

Graphs and Infinity

1. Classic Representations for Infinite Sets of Symbols
2. Incremental Maximal Sharing
3. Relations

Mixing Symbolic and Numeric Properties

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

Graphs and Infinity

Sharing and Incrementality

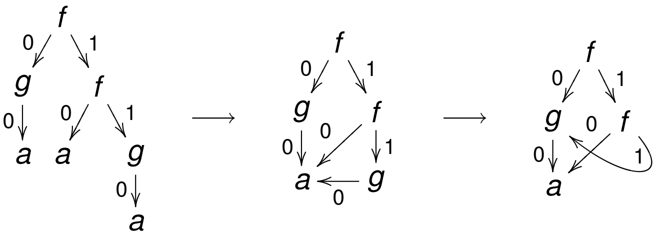
Sharing

- 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**

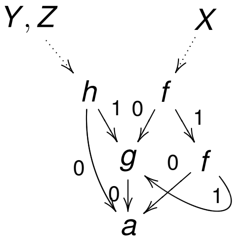
The Easy Case

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

Unicity

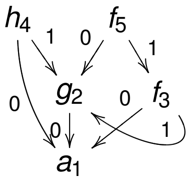


Mechanism

Dictionary + key

Key = label + sub-trees id

$$\begin{array}{rcl} a & : & 1 \\ g(1) & : & 2 \\ f(1,2) & : & 3 \\ h(1,2) & : & 4 \\ f(2,3) & : & 5 \end{array}$$

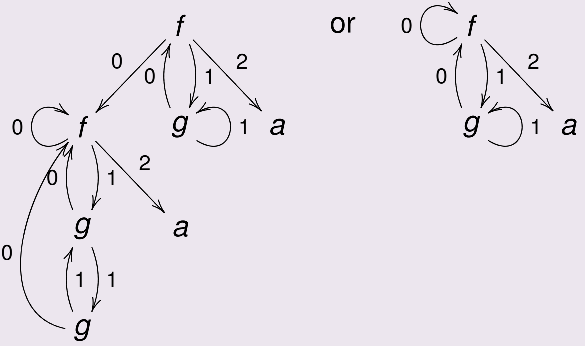


Graphs and Infinity

Regular Trees

Regular = *finite* number of distinct sub-trees

Example



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

Equivalent Graphs

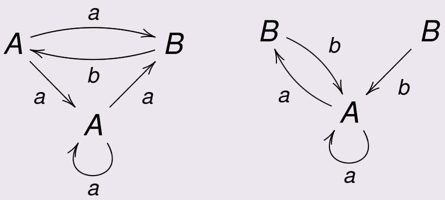
- First determine the semantic equality
- Idea: all what we can observe of a graph is
  - Node labels
  - Follow edges by specifying labels (=paths)

Equivalent graphs

- Two nodes can be distinguished iff there is a path starting from each node, with same edge labels and leading to nodes with different 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

⇒ 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

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 algorithme than partition refinment for such graphs...

A Partition Refinment 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 splited blocks in W

Graphs and Infinity

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

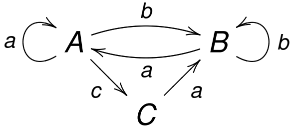
$\Rightarrow$ find a caracterictic key?

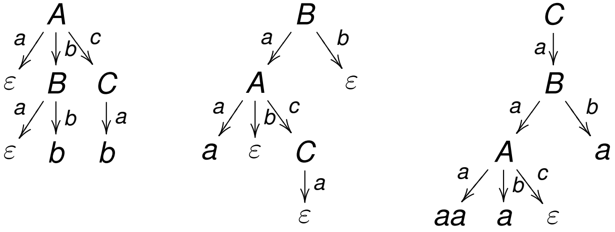
Caracteristic property

Isomorphic cycles have the same set of labeled paths

Caracteristic Set of Trees for a Strongly Connected Graph

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





Keys for Strongly Connected Graphs

- The caracteristic set of trees can be big (quadratic cost)
- We can either
  - represent all paths (by a set of trees)
  - or just the paths starting from one node, and try all nodes

Better Solution

Distinguish one node through sorting

- Naïve is quadratic
- But can use partitioning algorithm!
- So the key is a tree (same size as the graph)
- Key comparison is constant time!

Graphs and Infinity

The 3 Cases

- We want to minimize $\mathcal{G}$, knowing that $\mathcal{U}$, subgraph of $\mathcal{G}$ is minimal.
- Go through $\mathcal{G}$ and see what happens when reaching $\mathcal{U}$
- If all outgoing edges of a node $n \in \mathcal{G} \backslash \mathcal{U}$ are in $\mathcal{U}$
  - Classical hash-consing determines if n is equivalent to a node in $\mathcal{U}$

  $\Rightarrow$ if no cycle in $\mathcal{G} \backslash \mathcal{U}$, we are done!
  - So we keep a dictionary for (label, children id) $\longrightarrow$ id
- Otherwise, we have a strongly connected component with all outgoing edges in $\mathcal{U}$. 3 cases:
  1. No node is equivalent to a node in $\mathcal{U}$
  2. One node is equivalent do a direct child
  3. All nodes are equivalent to a node in $\mathcal{U}$, but none is a direct child of the strongly connected component.
- Case n° 1 yelds when we don't have either 2 or 3. So we just recognize cases 2 and 3.

Partial Keys

Recognizing strongly connected components when the candidate has no node equivalent to a direct child

- Let $\mathcal{N}$ a strongly connected subgraph of $\mathcal{G} \backslash \mathcal{U}$ such that it is equivalent to a subgraph $\mathcal{V}$ of $\mathcal{U}$
- Suppose moreover all outgoing edges of $\mathcal{N}$ lead to $\mathcal{U} \backslash \mathcal{V}$,
- Then $\forall n \in \mathcal{N}$, n is equivalent to a node v of $\mathcal{V}$, and for each edge label l, if $n.l$ is in $\mathcal{U}$, then

$$n.l = v.l$$

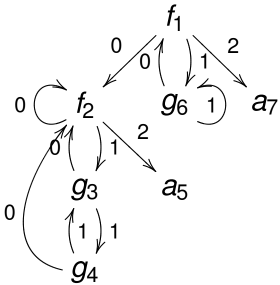
$\Rightarrow$ If we relace labels in $\mathcal{N}$ and $\mathcal{V}$ by a couple (label, id of children in $\mathcal{U} \backslash \mathcal{V}$, then the two graphs become Equivalent

Partial key = label + id of already treated children

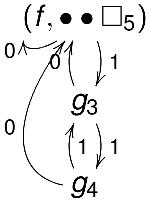
- So we just minimise the graph with partial keys
- and use keys for strongly connected graphs with partial keys

$\Rightarrow$ we have a recursive procedure for incremental minimisation!

Exemple I

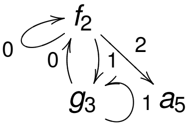


$$\begin{array}{rcl} D & = & \emptyset \\ D_G & = & \emptyset \end{array}$$



$$\begin{array}{rcl} D & = & \{a \to 5\} \\ D_G & = & \emptyset \end{array}$$

Exemple II



$$D = \{a \to 5,\ f(2,3,5) \to 2,\ g(2,3) \to 3\}$$

$$D_G = \left\{ \begin{tikzcd} & (f, \bullet\bullet\Box_5) \arrow[dl] \arrow[d] & \\ \varepsilon & (g, \bullet\bullet) \arrow[dl] \arrow[dr] & \\ \varepsilon & & 1 \end{tikzcd} \to 2 \right\}$$

Exemple III

$$\begin{tikzcd}
(f,\square_2\bullet\bullet) \arrow[d, bend left, "1"] \arrow[dr, "2"] & \\
g_6 \arrow[u, bend left, "0"] \arrow[loop right, "1"] & a_7
\end{tikzcd}$$

$$D = \{a \to 5,\ f(2,3,5) \to 2,\ g(2,3) \to 3\}$$

$$D_G = \left\{ \begin{tikzcd}
& (f,\bullet\bullet\square_5) \arrow[dl] \arrow[d] & \\
\varepsilon & (g,\bullet\bullet) \arrow[dl] \arrow[dr] & \\
\varepsilon & & 1
\end{tikzcd} \to 2 \right\}$$

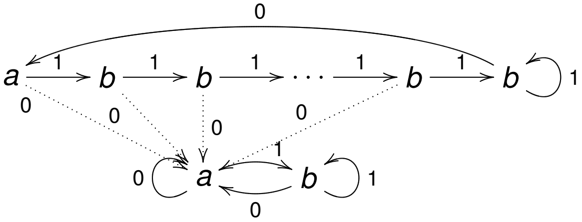
Exemple IV

$$\begin{tikzcd}
(f, \square_2 \bullet \square_5) \arrow[d, bend left, "1"] \\
g_6 \arrow[u, bend left, "0"] \arrow[loop right, "1"]
\end{tikzcd}
\qquad D = \{a \to 5,\ f(2,3,5) \to 2,\ g(2,3) \to 3\}$$

$$D_G = \left\{ \begin{tikzcd}[column sep=small]
 & (f, \bullet \bullet \square_5) \arrow[dl] \arrow[d] & \\
\varepsilon & (g, \bullet\bullet) \arrow[dl] \arrow[dr] & \\
\varepsilon & & 1
\end{tikzcd} \to 2 \right\}$$

Root Unrolling

The last case, when a child is equivalent to a node of the strongly connected component:



- Naive: compare each node to each child in $\mathcal{U}$.
- Each comparison is quadratic...

Finding if Loop Unrolling

Improvements

- If two outgoing edges lead to two different strongly connected components, only the latest included in $\mathcal{U}$ can be equivalent do $\mathcal{N}$
- If $n \in \mathcal{N}$ and l an edge label such that $n.l$ is in the latest strongly connected component of $\mathcal{U}$, then just compare n with the v in the component of $n.l$, such that $v.l = n.l$
- Comparison between a node of $\mathcal{U}$ and a node of $\mathcal{N}$ is linear.

Still quadratic in the worst case, but very rare

Complexity results

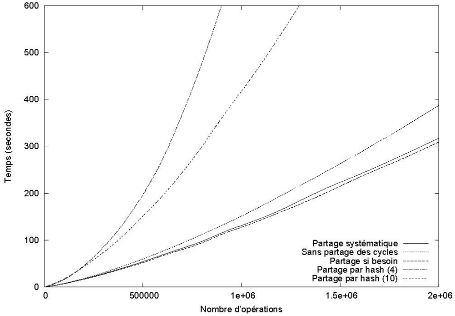
	shared	direct representation
testing $t_1 = t_2$	$\mathcal{O}(1)$	$\mathcal{O}\left((n_1 + n_2)\log(n_1 + n_2)\right)$
testing t_1 subtree of t_2	$\mathcal{O}(n_2)$	$\mathcal{O}\left((n_1 + n_2)\log(n_1 + n_2)\right)$
building $t_{[p]}$	$\mathcal{O}(\vert p\vert)$	$\mathcal{O}(\vert p\vert)$
root constructor	$\mathcal{O}(1)$	$\mathcal{O}(1)$
cycle constructor	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$

- n may be far bigger in the unshared case.
- We don't have to share systematicaly

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



Graphs and Infinity

Application to Word Automata

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

Static Analysis Application

Approximate the messages on channels between parallel processes

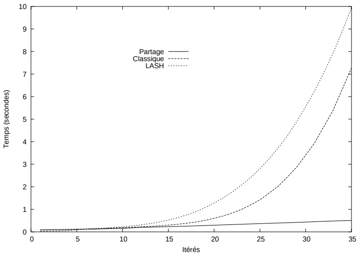
Approximation

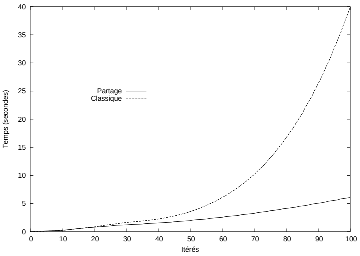
Using Q-automata: encodes a sequence of languages by a regular language

Experimental Results for Message Analysis

- Fixpoint computation
- Without minimisation, automara 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

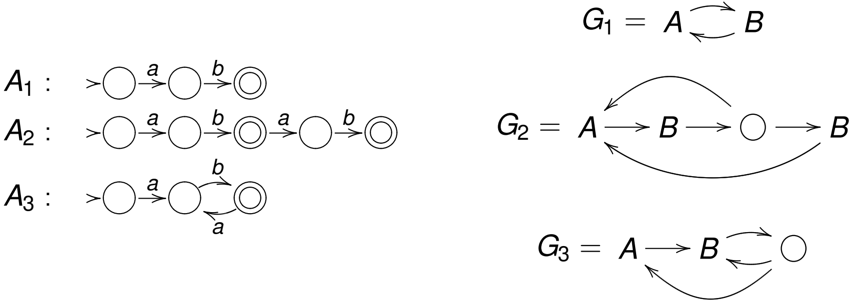
Widening

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

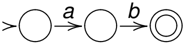
- Graph folding
  - Try to replace a new node by an old one with the same label
  - Only if this old one represents more values
- Path extrapolation
  - Repeat infinitely a newly added edge (or path).
  - Approximates $\{ a^n b^n \mid n \in \mathbb{N}\}$ by $a^k a^* b^k b^*$
- Size limiting
  - After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

Examples of Graph Folding

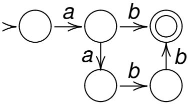


Examples of Path Extrapolation

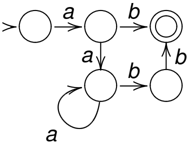
A_1 :



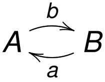
A_2 :



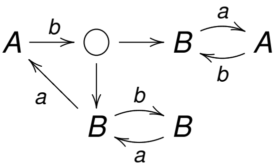
A_3 :



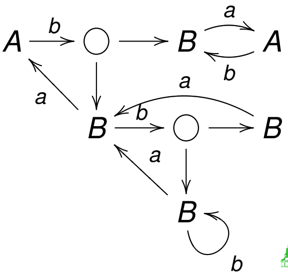
$G_1 =$



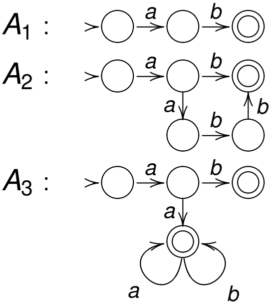
$G_2 =$

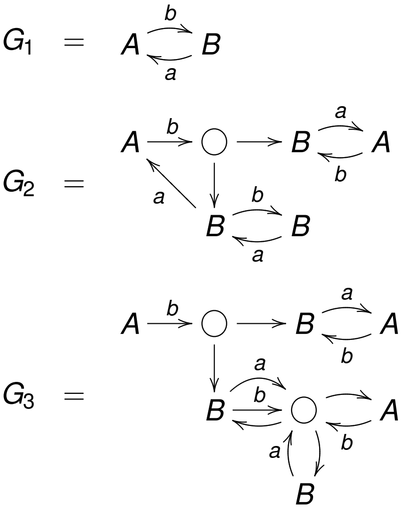


$G_3 =$



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?

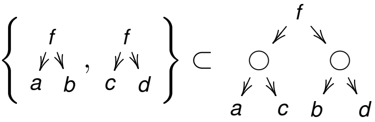
- Equivalence is not the equality of paths
- Unless normal form?
- Potential problem of cartesian approxiation

- A set of tuples (of the same size) is a relation
- A subset of a cartesian approximation is a relation

First take a look at representation of relations

Graphs and Infinity

Motivations

Relational domain of trees: 

What we need

- Define the possible sequences
- Keep track of what we link
- Maybe link ∞ decisions

Relations
Reminder

Definition

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

- Relation $\equiv$ language, except for negation and the operations
  - $u.v$ the concatenation between vectors of disjoint supports and $u.R \stackrel{\text{def}}{=} \{ u.v \mid v \in R \}$
  - projection $R_{(J)}$ $(0_0 1_1 \neq 0_0 1_2)$
  - and partial evaluation $R_{:i=b}$
- Let $R \simeq S$ iff same underlying language
- Relation $\equiv$ function

$$
\begin{aligned}
R(u) &= \texttt{true} \text{ or } \texttt{false} \text{ if } u \in \bigotimes_{i \in I} E_i \\
&= \{ v \mid u.v \in R \} \text{ if } u \in \bigotimes_{i \in J \subset I} E_i
\end{aligned}
$$

Representations

- A binary relation is a graph, but not the same notion of equivalence: nodes cannot be merged
- In the finite case, see first part
- If the sets are finite, we can always use a boolean encoding
- What remains to explore are the

infinitary relations

ω-regular languages

Linear Temporal Logic (LTL)

Temporal logic ::=		$\mathtt{p}$	atomic proposition
formula	|	$f \wedge f \mid f \vee f \mid \neg f$	logic connectors
	|	$G.f \mid F.f \mid f.U.f \mid f.R.f$	temporal operators

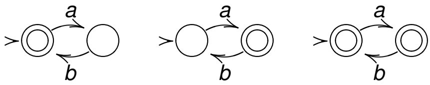
Büchi Automata

- Defined by (Q, E, I, F) i.e. (states, transitions, initial, final)
- Infinite word recognized $\Leftrightarrow$ goes infinitely through F
- Closed by $\cup$, $\cap$ and $\neg$
- ω-regular = finite union of $U.V^\omega$, U and V regular.

Sharing Büchi Automata?

Properties of Büchi Automata

- Deterministic Büchi is less expressive than non-deterministic
- In general, there is no minimal Büchi automaton



Even in the restricted cases:

- Not same notion of node equivalence
- Final states can be redundant (hard to detect)

$\Rightarrow$ define sub-classes based on properties of the final state?

Problems with Classical ω-regular Languages in Static Analysis

Translating Formulae into Automata

Formula with n sub-formulae $\Rightarrow$ automaton with $n.2^n$ states

Complexity

Emptyness testing: PSPACE-complete

Usage

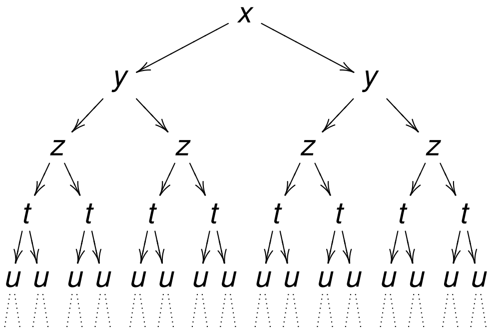
Not efficient in case of non-linear access:

- partial evaluation
- projection

What about a decision tree approach?

Decision Trees for Infinitary Relations

Problems if I is infinite...



∞ variables

regularity?

trees more than ∞

Simplifications

- Each E_i can be encoded by $\mathbb{B} \times \ldots \times \mathbb{B}$
  $\Rightarrow$ we will just consider $\mathbb{B}$
- Just considere the case $I \equiv \mathbb{N}$.

Graphs and Infinity

Entry names

Considering relation $\equiv$ function $\mathbb{B}^{\mathbb{N}} \longrightarrow \mathbb{B}$

Definition

The entries of a function are the rank of the arguments of that function

Names of a relation

For a given computation algorithm, the entry names are the variables associated with the arguments

Definition

Named relation = relation R + $\text{name}_R : I \longrightarrow \text{ename}(R)$

Equivalent Entries

finite $\sharp$ of variables $\Rightarrow$ sharing entry names

Entries i and j are equivalent

$$\begin{array}{ccccccc} & i & & j & & \\ \ldots 010 & a & 10\ldots 00 & b & 001\ldots & \in f \\ \ldots 010 & b & 10\ldots 00 & a & 001\ldots & \in f \end{array}$$

Definition

$\forall \sigma$ permutation of I, $\forall u \in \bigotimes_{i \in I} E_i$, $R(u) = R(\sigma(u))$

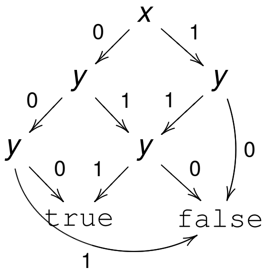
Idea

- $R_{:x=a}$ is not ambiguous
- In the decision tree, nodes with the same label x are equivalent

Example (finite case)

Let $R = \{000, 011, 111\}$. Entries 1 and 2 are equivalent (but not 0 and 2 as $R(011) \neq R(110)$).

So we can use the following BDD to represent R:



Speeds up some projection operations ($R_{:2=\mathtt{false}}$)

Elimination of redundant nodes

Theorem

If $R(u.0) = R(u.1)$, then $\forall v$ such that $name_R(|u.v|) = name_R(|u|)$, $R(u.v.0) = R(u.v.1)$

Proof.

Let $v = a.w$.
Then $R(u.a.w.0) = R(u.0.w.a) = R(u.1.w.a) = R(u.a.w.1)$ $\square$

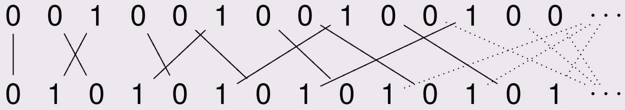
Infinitely many equivalent entries

Caution!

The notion "is equivalent entry" is not infinitely transitive

Example

- Let f true iff u contains ∞ many 00
- $(001)^\omega \in f$, but $(01)^\omega \notin f$
- All pairs of entries are equivalent
- but...



Consequences when all entries are equivalent

Let $R \subset \bigotimes_{i \in \mathbb{N}} E$ such that all entries of R are equivalent

Theorem

$\forall v \in E^n$, $\forall b$ letter of v, $\forall \alpha \in E^\omega$ containing b ∞ often,

- $v^\omega \in R \Leftrightarrow (v.b)^\omega \in R$
- $\alpha \in R \Leftrightarrow b.\alpha \in R$

Equivalent Vectors of Entries

- To represent greater classes of functions
- In particular, the encoding of E into $\mathbb{B} \times \ldots \times \mathbb{B}$
- Just keep the permutations that change the whole vector at a time (keeping the ordering of the vector)

Relations Representation

$$\begin{array}{ccc} \text{decision tree} & + & \text{naming} \\ \downarrow & & \downarrow \\ \text{regular tree} & & \text{regular with sharing?} \end{array}$$

Representation

Definition

The entry names of R are ultimately periodic iff $\exists k, j$ such that $\forall i > j$, $\text{name}_R(i) = \text{name}_R(i + k)$

- Classical case: $x < y < z$...
- Infinite case, representable, infinite word $xyxxyz^\omega$
- In the sequel, mainly considere relations with ultimately periodic entry names
- It is a true restriction:

Restricting to Ultimately Periodic Entry Names

Theorem

The ω-regular languages such that $\exists$ ultimately periodic naming of the entries are a strict subset, closed under $\cap$, $\cup$ and $\neg$

Example

- $\{O, 11\}^\omega$ is ω-regular,
- but $\nexists i < j$ such that i equivalent to j,
- because $0^j 110^\omega \in R$, but not $0^i 10^{j-i} 10^\omega$

Graphs and Infinity

Regularity

Definition

Let R a named relation. R is prefix-regular iff its entry names are ultimately periodic and the $\sharp$ of $R(u)$ modulo $\simeq_n$ is finite.

- $R \simeq_n S$ iff $R \simeq S$ and $\text{name}_R \equiv \text{name}_S$
- does not mean the underlying language is ω-regulier!
  - Let $\mathcal{L}$ non regular

  $\Rightarrow$ $T = \{0, 1\}^*.\mathcal{L}$ not regular
  - But, $\forall u$, $T \simeq T(u)$

Open and Closed Named Relations

- Representation of prefix-regular named relations by a regular tree
- Does not describe the ∞ behavior
- Give a meaning to loops

Definition

R is open named relation iff R is prefix-regular and $\forall \alpha \in R, \exists u, \beta$ such that $\alpha = u.\beta$ and $R(u) \simeq \mathbb{B}^\omega$

Definition

R is closed named relation iff R is prefix-regular and $\forall \alpha \notin R, \exists u, \beta$ such that $\alpha = u.\beta$ and $R(u) \simeq \emptyset$

Open and Closed ω-regular Relations

- Büchi automata not easy to share because of possible redundancy of final states.
- Solution: just keep Büchi automata such that $F = Q$.
  - Contains only closed languages (for natural topology)
  - Not very interesting for temporal properties. . .
- Solution 2: only 1 final state, which is a simple loop
  - These are the complement of the previous automata
  - So contains only closed languages (for natural topology)
  - Can express "must not stay infinitely"
- Solution 3: all finite states are simple loops
  - Defines quasi-open languages:

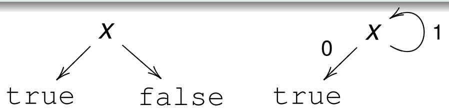
$\forall \alpha \in L, \exists u$ such that $\alpha = u.\beta$ and if A is the set of letters in β, $u.A^\omega \subset L$.

  - Can express termination
  - Cannot express fairness

Examples of Closed and Open Relations

Property

A finite relation is closed and open



var. names	$x.y^\omega$	x^ω
open	$0.\mathbb{B}^\omega$	$1^*.0.\mathbb{B}^\omega$
closed	$0.\mathbb{B}^\omega$	$1^*.0.\mathbb{B}^\omega \cup 1^\omega$

Representation

- Regular trees with sharing
- New source of non-uniqueness: $\begin{tikzcd} x \arrow[loop left, "0"] \arrow[loop right, "1"] \end{tikzcd}$
- But easy to check when sharing

$\Rightarrow$ Efficient representation

- Unique representation if elimination of redundant nodes
- But up to renaming!

Properties of Open Relations
and Closed Relations

Property

$\forall R, S$ open, $R \cap S$ and $R \cup S$ are open

Property

$\forall R, S$ closed, $R \cap S$ and $R \cup S$ are closed

Property

$\forall (R_i)_{i\in\mathbb{N}}$ open, $\bigcup_{i\in\mathbb{N}} R_i$ is open

Property

$\forall (R_i)_{i\in\mathbb{N}}$ closed, $\bigcap_{i\in\mathbb{N}} R_i$ is closed

Approximation

$\forall R$ prefix-regular relation, $\exists$ greatest open contained in R and a smallest closed containing R

Graphs and Infinity

Iteration

Idea

Use trees to represent the ∞ behavior

Let R a named relation.

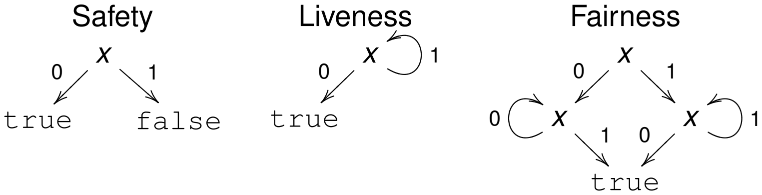
$\Omega(R) \stackrel{\text{def}}{=} \{u_0.u_1 \ldots | \; u_i$ minimal such that

- $|u_i| > 0$
- $R(u_i) = \mathbb{B}^\omega$
- $\text{name}_R(|u_i|) = \text{name}_R(0)$

Definition

R is iterative iff its names are perdiodic and $\exists S$ open such that $R = \Omega(S)$

Examples



$\emptyset$ and $\mathbb{B}^\omega$ are open and iterative

Theorem

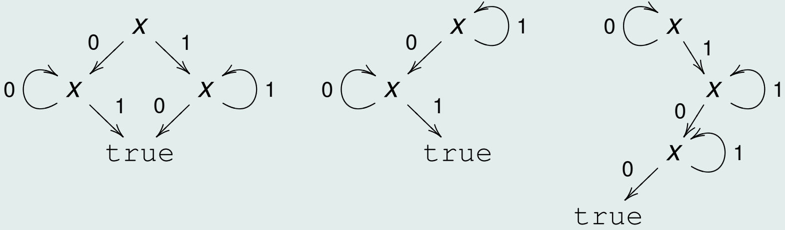
iterative $\Rightarrow$ *prefix-regular*

Proof.

$R = \Omega(S)$, and S is open, so prefix-regular. Take u such that $\exists v$, $|v| < |u|$ and $S(u) \simeq_n S(v)$. If $S(u) \neq \mathbb{B}^\omega$, $R(u) \simeq R(v)$. If $S(u) = \mathbb{B}^\omega$, $u = u_0.u_1$ with $R(u_0) = R$ so $R(u) = R(u_1)$ $\square$

Representation of Iterative Relations

Uniqueness Problem



Theorem

Let R iterative. $S = \{\, u.\alpha \mid u^\omega \in R$ and $R(u) = u \}$ is the greatest open relation such that $R = \Omega(S)$.

Lemma

To simplify: only one entry name and base domain E

Lemma

Let u such that $S(u) = E^\omega$, $|u| > 0$ and $\forall v < u$, $S(v) \neq E^\omega$. Then $u^\omega \in R$ and $R(u) \simeq R$

- Let $\alpha \in E^\omega$.
  - Lemma hypothesis: $u.\alpha \in S$
  - Definition of S: $\exists v \prec \alpha$ such that $(u.v)^\omega \in R$ and $R(u.v) \simeq R$.
  - Special case $\alpha = b^\omega$: $\exists k$ such that $(u.b^k)^\omega \in R$ and $R(u.b^k) \simeq R$
- Let b a letter of u. $(u.b^k)^\omega \in R \Rightarrow u^\omega \in R$
- Let $\beta \in R$, $\exists b$ which appears ∞ in β. $u.b^k.\beta \in R \Rightarrow u.\beta \in R$
- Let $\beta \in R(u)$ then $b^k.\beta \in R(u)$ so $\beta \in R$

Proof of the Greatest Open Theorem (1/2)

- S is open because $\forall \alpha \in S, \exists u \prec \alpha, \forall \beta, u.\beta \in R$
- R iterative $\Rightarrow \exists S'$ open such that $R = \Omega(S')$
  - Let $\alpha \in S'$, $\exists u \prec \alpha\ S'(u) = E^\omega$
  - Let u_0 the smallest such u, then $u_0^\omega \in R$ and $R(u_0) \simeq R$
  - By definition of S, $\alpha \in S$
  - $\Rightarrow$ $S' \subset S$
- $R \subset \Omega(S)$
  - Let $\alpha \in R$ such that $\alpha \notin \Omega(S)$
  - If $\exists$ smallest u such that $\alpha = u.\beta$ and $S(u) = E^\omega$, $\alpha \notin \Omega(S)$ $\Rightarrow \beta \notin \Omega(S)$. But lemma says $R(u) = R$, so we start again with β.
  - We arrive to $\nexists u \prec \alpha$ such that $S(u) = E^\omega$
  - But $\alpha \in R \Rightarrow \exists u \prec \alpha, S'(u) = E^\omega$, so $S(u) = E^\omega$.

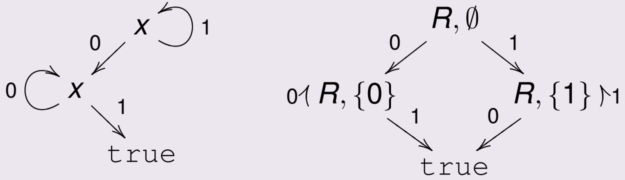
Proof of the Greatest Open Theorem (2/2)

- $\Omega(S) \subset R$
  - Let $\alpha = u_0.u_1.\ldots.u_n\ldots \in \Omega(S)$
  - $\forall i, R(u_i) = R$ and $u_i^\omega \in R$
  - $\sigma(\alpha) = v.\beta$ with β just letters ∞ often
  - $\beta \in \Omega(S)$ and $\alpha \in R \Leftrightarrow \beta \in R$ (lemma)
  - $\tau(\beta) = (v_0.v_1\ldots v_m)^\omega$
  - $R(v_i) = R$ and $v_m^\omega \in R$ so $\gamma = v_0.v_1\ldots v_{m-1}.v_m^\omega \in R$
  - $R = \Omega(S')$ so $\gamma = u'_0.u'_1\ldots u'_n\ldots$
  - $\exists j\ u'_0\ldots u'_j = v_0.v_1\ldots v_{m-1}.v_m^n.w$ with $w \prec v_m$
  - So $(v_0.v_1\ldots v_{m-1}.v_m^n.w)^\omega \in R$, so $\beta \in R$

Representation of Iterative Relations

- The theorem is constructive
- If we have a representation by an open relation, we detect the u such that $u^\omega \in R$ and $R(u) = R$

Example



Graphs and Infinity

Regular Relations

Definition

R named is regular iff prefix-regular and $\exists \mathcal{I}(R)$, $\forall S \in \mathcal{I}(R)$, $S \neq \emptyset$ and S iterative and $\forall \alpha \in R$, $\exists u$, $\exists S \in \mathcal{I}(R)$, $\alpha \in u.S$ and $S \subset R(u)$.

Example

$\{0^\omega, 1^\omega\}$, or the set of vectors ending with O^ω or 1^ω

Theorem

A named relation R is regular iff entries ultimately periodic and underlying language is ω-regular

Idea: finite union of $U.V^\omega$ with U and V regular

Proof

- Let R reguliar and $\mathcal{I}(R) = (R_i)_{i \in C}$
  - Each R_i defines a regular language
    - $(Q, E, \{R_i\}, \{R_i\}$
    - $Q = \{\, R_i(u) \mid u \text{ fini}\}$
    - $E(R_i(u), b)$ = R_i if $R_i(u.b) = \mathbb{B}^\omega$ and $R_i(u.b)$ otherwise
  - So finite $\cup$ of $u.R_i$ ω-regular languages
- Let (Q, E, I, F) a Büchi such that $\exists R$ ultimately periodic
  - R_q = language of $(Q, E, \{q\}, \{q\})$
  - $R_q = \Omega\,(S_q)$ with S_q the set of α such that $\exists u \prec \alpha$, u in the finite language of $(Q, E, \{q\}, \{q\})$.

Regular Relations Usage

Corollary

R and G regular, then $R \cap G$, $R \cup G$ and $\neg R$ too.

But representation too inefficient:

- Non unicity of $\mathcal{I}(R)$ and of its representation
- Non deterministic decision process

ω-deterministic Relations

- Limit the $\sharp$ of infinite behaviors at a given point.
- $R^{\Omega}_{[u]} \stackrel{\text{def}}{=} \Omega(\{ v.\alpha \mid u.v^{\omega} \in R \text{ and } R(u.v) = R(u)\})$

Definition

R is ω-deterministic iff prefix-regular and $\forall u,\ R^{\Omega}_{[u]} \subset R(u)$

Theorem

If R ω-deterministic then R regular and $\forall u, \exists S \in \mathcal{I}(R)$, such that $S \subset R(u)$, and $\forall S' \in \mathcal{I}(R)$ behavior at u, $S' \subset S$.

$\Rightarrow$ unique representation

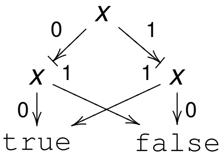
Representation of ω-deterministic Relations

Introduction of an `iter` node at u to signal $R^{\Omega}_{[u]}$ non empty

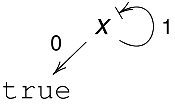
Pseudo-decision process

- Deterministicaly going through the tree and the vector to recognize
- Start with an empty stack
- If variable $\begin{array}{c} x \\ \swarrow \quad \searrow \\ t_0 \qquad t_1 \end{array}$ take t_0 or t_1 according to vector value
- If `false` the vector is not in the relation
- If $\begin{array}{c} \texttt{iter} \\ \downarrow \\ t \end{array}$, empty the stack
- If `true`, continue at the latest `iter` and stack `true`
- The vector is in the relation if the stack is infinite

Examples of ω-deterministic Decision Trees



represents the relation $\{0^\omega, 1^\omega\}$



represents the set of vectors ending with 0^ω.

open $\Rightarrow \omega$-deterministic

Properties of ω-deterministic Relations

- closed by intersection
- $\forall R$ prefix-regular, there is a smallest ω-deterministic containing R

$\Rightarrow$ possibility to approximate all operations yelding a prefix-regular relation

- Canonical and incremental representation
- If we apply the same representation to finite relation, we get exactly BDDs

Algorithms on ω-deterministic Relations

General idea for binary operations

- Go through the two trees in parallel (as for BDD)
- Go back to the previous `iter`if you come to a `true`
- Store the couples of encountered sub-trees
- If you cross again (u, v), according to the relation, see if you have been through a `true` on a side or both

To finish, must compute the biggest open (smaller v such that $v^\omega \in R$ and $R(v) = v$

For inclusion

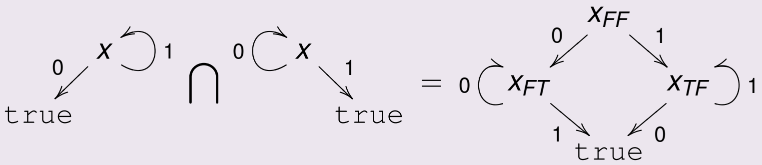
- Go through the trees, according to pseudo-decision process
- So you cross again (u, v), if u have been through `true`, v must have too (otherwise, not included)

Intersection

Algorithm

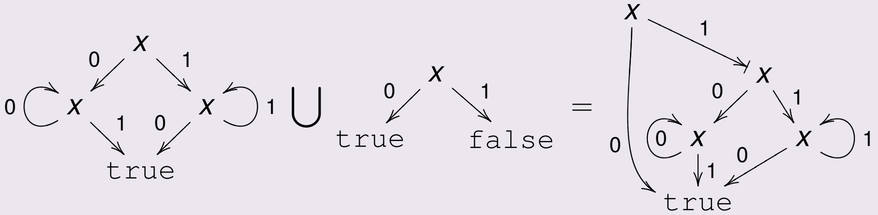
- Go through the trees, keeping which one went through `true`
- Creation of a node according to that information
- Then sharing

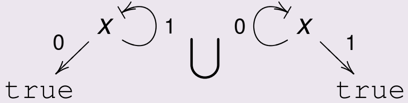
Example



Union

Example





will be approximated by `true`