Matching integer intervals by minimal sets of binary words with *don't cares*

Wojciech Fraczak[1,3] Wojciech Rytter[2,4] Reza Yazdani[3]

1. Universit du Qubec en Outaouais, Canada
2. Warsaw University, Poland
3. Carleton University, Ottawa, Canada
4. Copernicus University, Torun, Poland

CPM 2008, Pisa, Italy

Motivation

- TCAM - *Ternary Content Addressable Memory* is used for performing very fast (constant time) table look-up operations.
- Problem of interval representation by sets of binary words with *don't cares* (i.e., in TCAM) appears in the context of network packet processing:
  the header fields of each IP packet (e.g., source address, destination address, port number, etc.) should be matched under strict time constraints against the entries of an Access Control List (ACL)

Binary words with *don't cares*

- $E : \{0,1\}^n \hookrightarrow \{0,1,\ldots,2^n-1\}$
  — encoding of integers by n-bit strings.
- Set represented by binary words with "$*$"
  — e.g., "*template*" representation of interval $[1,14]$:

1	0001
2	001$*$
3	01$**$
4	10$**$
5	110$*$
6	1110

- **Observation.** The template corresponds to a boolean formula in disjunctive normal form, in which variable x_i determines whether the i-th bit is one or zero (negation of x_i).

Statement of the problem

Problem: Finding a minimal *template* representation of a given interval of positive integers (bit encoded).

Other (and smaller) *templates* for the interval $[1, 14]$:

1	$0{*}{*}1$
2	$10{*}{*}$
3	${*}10{*}$
4	${*}{*}10$

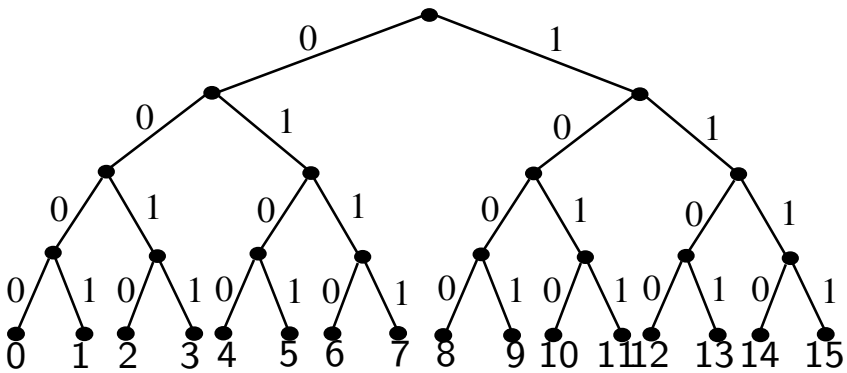
1	$01{*}{*}$
2	$1{*}0{*}$
3	${*}0{*}1$
4	${*}{*}10$

1	$0{*}1{*}$
2	$10{*}{*}$
3	${*}1{*}0$
4	${*}{*}01$

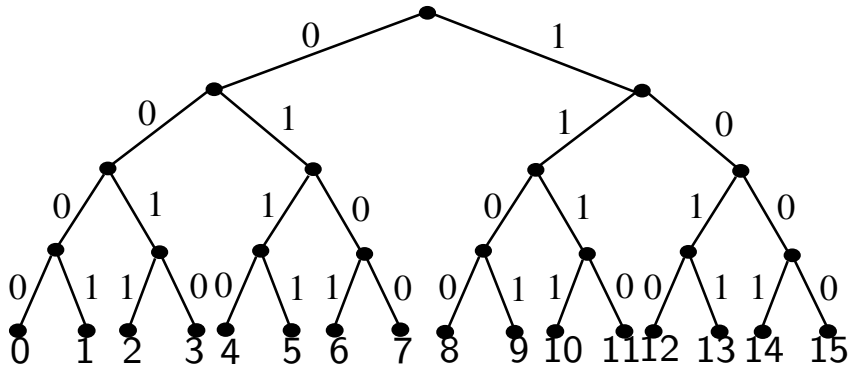
1	$01{*}{*}$
2	$1{*}{*}0$
3	${*}01{*}$
4	${*}{*}01$

Dense-tree encodings of integers

Some encodings can be represented by binary trees.
Two examples, lexicographic and reflected Gray encodings:



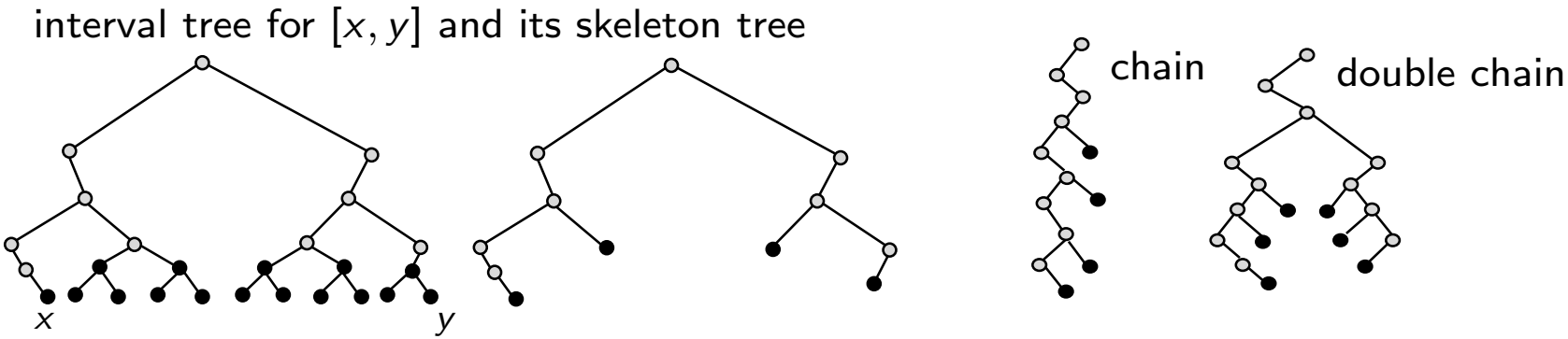
$\mathtt{Lex}_4$



$\mathtt{Gray}_4$

Subset representation by skeleton trees

- A tree representation for subset $[x, y]$ and the corresponding *skeleton tree* (in a dense-tree encoding)



- Lemma: A skeleton tree for an interval is a *chain* or a *double-chain*.

A rule is called a prefix rule if all non-star symbols occur as a prefix of it, e.g., 0101**** is a prefix rule.

Theorem

Let $X \subseteq \{0,1\}^n$ be an interval-set of a dense-tree encoding T_n. The minimum number of prefix rules covering X equals the number of leaves in the skeleton tree of X in T_n.

Theorem

Let the skeleton tree of $X \subseteq \{0,1\}^n$ in a dense-tree encoding T_n be a chain with k leaves. The minimal size of a template for X is k.

Corollary

In an n-bit dense-tree encoding, representing each of the intervals $[1, 2^n - 1]$ and $[0, 2^n - 2]$ need exactly n rules.

Essential rules

- Let $X \subseteq \{0,1\}^n$
- A rule r is called an X-*limited* rule if $L(r) \subseteq X$.
- An X-limited rule r is said to be *X-essential* if there is no other X-limited rule that covers r (i.e., r is a "*prime implicant*" of X)
- Any coverage of X by a template of size k can be turned into a coverage by k X-essential rules. Therefore, one can consider only X-essential rules in the process of finding a minimal coverage of X.

Small number of essential rules for intervals

In general, the number of essential rules for a given set may be exponential with respect to the number of bits.

Theorem

In the case of the n-bit lexicographic encoding and n-bit reflected Gray encoding, an interval-set $X \subseteq \{0,1\}^n$ *admits no more than* $n(n-1)+1$ *and* $2n$ X*-essential rules, respectively.*

Is finding optimal templates for intervals simpler than finding optimal templates for any subset?

Are prefix rules good enough?

- The minimum number of prefix rules required for representing an interval is equal to the number of leaves in its corresponding skeleton tree.
- Can we do better by not limiting ourselves to prefix rules?

  YES! For some intervals, this number can be significantly reduced by using non-prefix rules.
- Actually, there is a T_n and an interval I such that we need $2n-2$ prefix rules while n rules is enough.

Bounds ...

Let $T_n \in \{\mathtt{Lex}_n, \mathtt{Gray}_n\}$: $\max(n, 2n-4)$

- There is an interval $I = [x, y]$ which cannot be represented with less than $\max(n, 2n-4)$ rules.
- Every interval $I = [x, y]$ can be represented by $\max(n, 2n-4)$ rules.

General case: $\max(n, 2n-3)$

- For each $n \geq 1$ there exists a T_n and an interval $I = [x, y]$ which needs $\max(n, 2n-3)$ rules.
- Every interval $I = [x, y]$ (in any T_n) can be represented by $\max(n, 2n-3)$ rules.

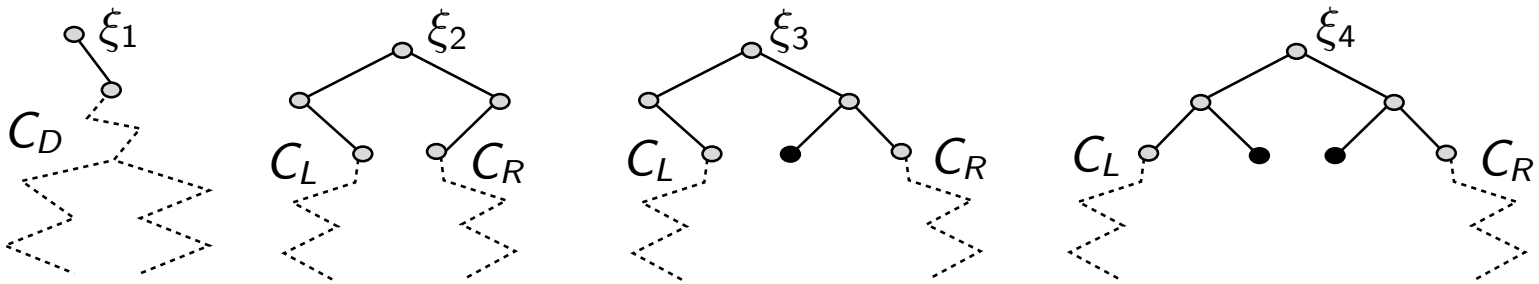
- Motivated by a practical problem:
  - We introduced the notion of dense-tree encoding schemes which includes the lexicographic encoding and the binary reflected Gray encoding.
  - We showed that the number of prime implicants (called here the *essential rules*) of a given interval set $X \subseteq \{0,1\}^n$ is *small*.
  - We provided the strict bounds for the number of rules we need in case of $\mathtt{Lex}_n, \mathtt{Gray}_n$, and unknown T_n.
  - ...

So, do we know how to find those optimal templates for intervals?

- Yes, we do (in linear time) for Gray encoding!
- Yes, we do (in linear time) for Lexicographic encodings!

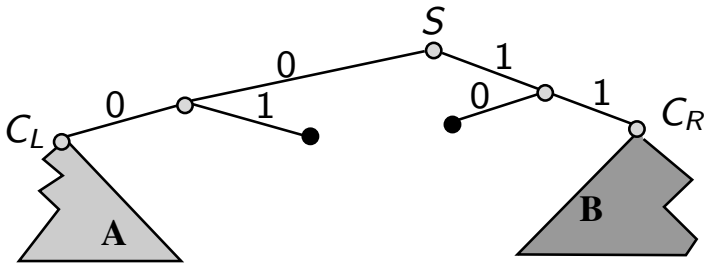
Generating optimal templates (Lex)

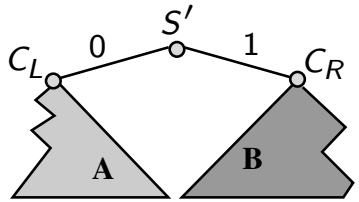
- By top down linear recursion
- INPUT: interval given in form of its skeleton tree
- Five cases: ξ_0 – chain; ξ_1 – 1 child; ξ_2 – 2 grandchildren; ξ_3 – 3 grandchildren; ξ_4 – 4 grandchildren.



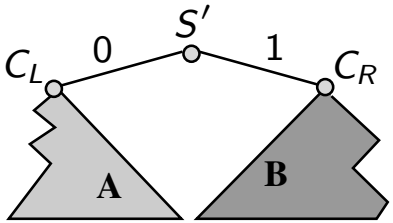
- For ξ_0, ξ_1, ξ_2 an optimal template can be easily calculated from optimal templates for the subtrees.

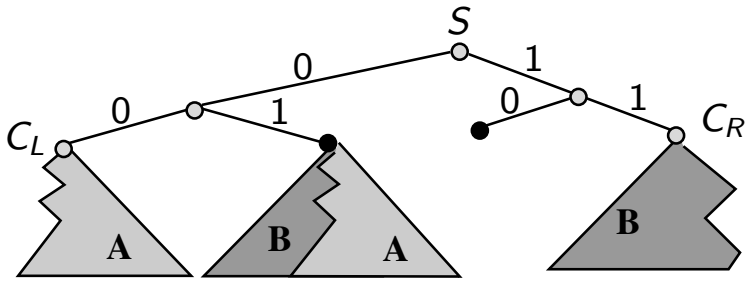
Eg, consider the case of ξ_4, i.e.:





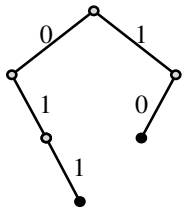
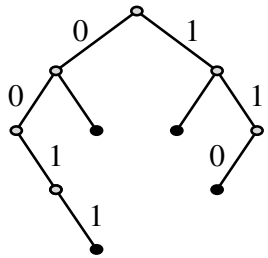
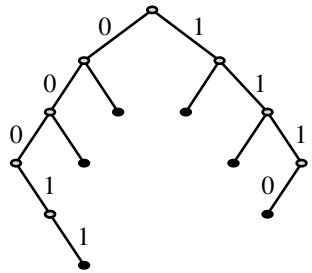
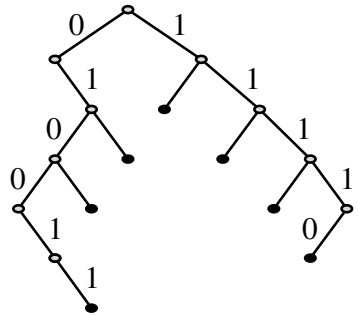
1. Firstly, we find a solution for:
2. If A and B overlap then a template for S will have one more rule than that for S'.





3. Otherwise, a template for S will have 2 more rules.

Template for $[19, 61]$ within Lex_6

$S''', 3$	$S'', 4$	$S', 5$	$S, 6$
			
ξ_2	ξ_4 (non-compl.)	ξ_4 (complementary)	ξ_3
1 011 2 10∗	1 0∗11 2 ∗10∗ 3 01∗∗ 4 10∗∗	1 0∗∗11 2 0∗1∗∗ 3 ∗10∗∗ 4 ∗∗10∗ 5 10∗∗∗	1 01∗∗11 2 01∗1∗∗ 3 1∗0∗∗∗ 4 ∗110∗∗ 5 ∗1∗10∗ 6 10∗∗∗∗