The undecidability of Aliasing

G. Ramalingam

Presented by:
Ory Samorodnitzky

Language Computation

$$Reg \subsetneq CFL \subsetneq R \subsetneq RE \subsetneq L$$

$\{a^n b^n | n \geq 0\}$

$\{a^n b^n c^n | n \geq 0\}$

$Halt$

Alphabets/Languages Properties

- $A = Alphabet$
- $A^* = \{w | w \epsilon A\}$
- Countable Set:
  - $\exists f,\ f: S \to \mathbb{N}, f$ is bijective
- A^* is countable
- $|A^*| = \aleph_0$
- Lemma1:
  - $|X| = \aleph_0$
  - $Y \subseteq X$
  - ⇨ Y is finite, or countable

Alphabets/Languages Properties

- Cantor's Theorem:
  - $|P(X)| \coloneqq |2^X| > |X|$
  - ⇨ Let $Z = \{L | L \subseteq A^*\}$
  - ⇨ $|Z| > |A^*| = \aleph_0$
  - ⇨ $|Z| = 2^{\aleph_0} = \aleph$

Turing Machine

- $M = \langle Q, \Gamma, b, \Sigma, \delta, q_0, F \rangle$
- Q- Finite set of states
- Γ- Finite set of the tape alphabet
- $b \in \Gamma$ – blank symbol
- $\Sigma \subseteq \Gamma \setminus \{b\}$ – set of input symbols
- $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, N\}$ - transition function
- $q_0 \in Q$ – initial state
- $F \subseteq Q$ – set of accepting states

Turing Machine

- $M = \langle Q, \Gamma, b, \Sigma, \delta, q_0, F\rangle$
- MA – Turing Machine Alphabet
- L - is the language accepted by M
- We denote:
  - $L = L(M)$
- $\mathcal{M} = \{L | L = L(M)\}$
  - Set of all Turing Machines
  - $\mathcal{M} \subseteq (MA)^*$
- From *Lemma*1:
- $|\mathcal{M}| = |(MA)^*| = \aleph_0$

Undecidable Languages
Existence Proof

- $Z = \{L | L \subseteq A^*\}$
- $|Z| = 2^{\aleph_0} = \aleph$
- $|\mathcal{M}| = \aleph_0$

⇨ $\exists L \in A^*, \forall M \quad L \neq L(M)$

Recursively Enumerable Set

- Definition:

$$S \in \mathbb{N}, \exists f(x) = \begin{cases} 0, & x \in S \\ undefined, & x \notin S \end{cases}$$

- S – Recursively Enumerable, Computabely Enumerable

Halting Problem

Description of a program

Finite input

The program finishes running or will run forever

Halting Problem

$$h(i,x) = \begin{cases} 1, & prog\ i\ halts\ on\ x \\ 0, & otherwise \end{cases}$$

- Let f be a total computable function

$$g(i,x) = \begin{cases} 0, & f(i,i) = 0 \\ undefined, & otherwise \end{cases}$$

- g is also computable
  - ➔ There exists a program e which computes g

Halting Problem

$$g(i,x) = \begin{cases} 0, & f(i,i) = 0 \\ undefined, & otherwise \end{cases}$$

- Exactly one of the following cases holds:
  - $g(e) = f(e,e) = 0$
    ⇨ $h(e,e) = 1$
  - $g(e)$ is undefined, and $f(e,e) \neq 0$
    ⇨ $h(e,e) = 0$
- In either case $f \neq h$

Halting Problem

$w_i \epsilon \Sigma^*$

$\mathcal{M}_i$

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

Halting Problem

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

Halting Problem

$$g(e) = f(e, e) = 0$$

	1	0	0	1	0	0
	0	0	1	1	0	0
	1	0	1	0	1	0
$e \longrightarrow$	0	0	0	0	0	0
	1	0	0	1	0	0
	0	0	1	1	0	1

Halting Problem

$$g(e) = f(e,e) = 0$$

	1	0	0	1	0	0
	0	0	1	1	0	0
	1	0	1	0	1	0
e	0	0	0	0	0	0
	1	0	0	1	0	0
	0	0	1	1	0	1

$$h(e,e) = 1$$

Halting Problem

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

Halting Problem

$g(e)\ is\ undefined \qquad f(e,e) = 1$

	1	0	0	1	0	0
	0	0	1	1	0	0
$e \longrightarrow$	1	0	1	0	1	0
	0	0	0	0	0	0
	1	0	0	1	0	0
	0	0	1	1	0	1

Halting Problem

$g(e)$ *is undefined* $\quad f(e, e) = 1$

	1	0	0	1	0	0
	0	0	1	1	0	0
$e \longrightarrow$	1	0	1	0	1	0
	0	0	0	0	0	0
	1	0	0	1	0	0
	0	0	1	1	0	1

$$h(e, e) = 0$$

Halting Problem – Alt. Proof

$$g^{'(x)} = \begin{cases} 1, & f(i,i) = 0 \\ 0, & f(i,i) = 1 \\ f(i,j) & i \neq j \end{cases}$$

⇨ No $\mathcal{M}_i$ corresponds to $g^{'}$

⇨ $g^{'} \neq L_n \quad \forall n \in \mathbb{N}$

⇨ g' is **not** RE

Halting Problem – Alt. Proof

Lemma 2 – Let $L \subseteq \Sigma^*$.

L is Recursive ⟺ *L* is RE and *co-L* is RE

- g' is not RE
- $co\_g' = Halt'$

⇨ $Halt'$ is not R

Rice’s Theorem

- The question of whether a given algorithm computes a partial function with a non-trivial property is undecidable.
- May/Must-alias problem is not-trivial.
- Rice’s Theorem says nothing about properties of **machines**.
- May/Must-alias is not a property of an algorithm/Language.

Post correspondence problem - Definition

given:

$A, B \subseteq \{0,1\}^+$

$|A| = |B| = r$

$A = w_1, w_2, w_3, \dots, w_r$

$B = z_1, z_2, z_3, \dots, z_r$

decide:

$\exists I = i_1, i_2, i_3, \dots, i_k \quad k > 0$

s.t.

$w_{i_1}, w_{i_2}, w_{i_3}, \cdots, w_{i_k} = z_{i_1}, z_{i_2}, z_{i_3}, \cdots, z_{i_k}$

Post correspondence problem - Example

w_1: *aba*	w_2: *bbb*	w_3: *aab*	w_4: *bb*
z_1: *a*	z_2: *aaa*	z_3: *abab*	z_4: *babba*

Post correspondence problem - Example

w_1 *aba* w_4 *bb* w_3 *aab* w_1 *aba*

$I = i_1, i_4, i_3, i_1$ *ababbaababa*

z_1 *a* z_4 *babba* z_3 *abab* z_1 *a*

Post correspondence problem

- Undecidable
  - Hopcroft and Ullman, 1979
  - PCP is simpler than Halting problem
    - Often used in proofs of undecidability

Alias Analysis

- Two pointers are said to be **aliased** if they point to the same location
- Aliasing scenarios:
  - Two variables cannot alias
  - Two variables must alias
  - Two variables may alias. Cannot be determined at compile-time.
- Applications
  - More accurate (less conservative) memory dependence analysis
  - More accurate data flow analysis
  - Better optimizations and scheduling

Alias Analysis

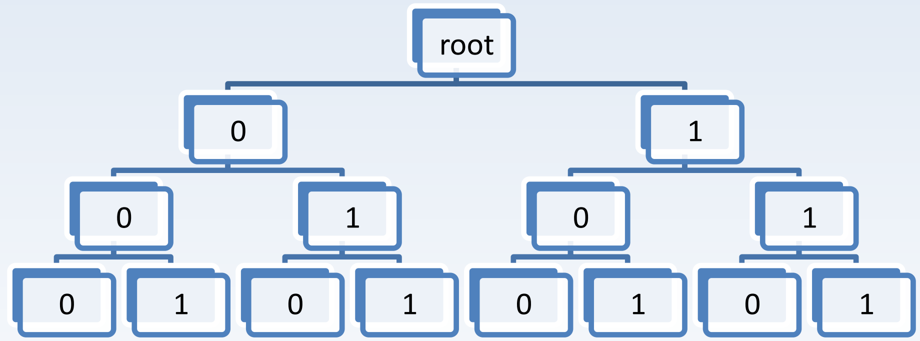
- Decision Version:
  - Given:
    - Program point - *P*
    - Two names – *u,v*
  - Decide:
    - The may-alias relation holds between *u,v* at *P*
- Theorem 1:
  - The intraprocedural may-alias problem is **undecidable** for languages with if statements, loops, dynamic storage, and recursive data structures

Alias Analysis

- Proof by reduction
- Reduction Buildup:
  - Binary Tree
    - *branch(0) – Left*
    - *branch(1) – Right*
  - Strings as Paths:
    - For binary string $b_1, b_2, b_3, \cdots, b_n$

      $Path(b_1, b_2, b_3, \cdots, b_n) =$

      $branch(b_1) \rightarrow branch(b_2) \rightarrow \cdots \rightarrow branch(b_n)$

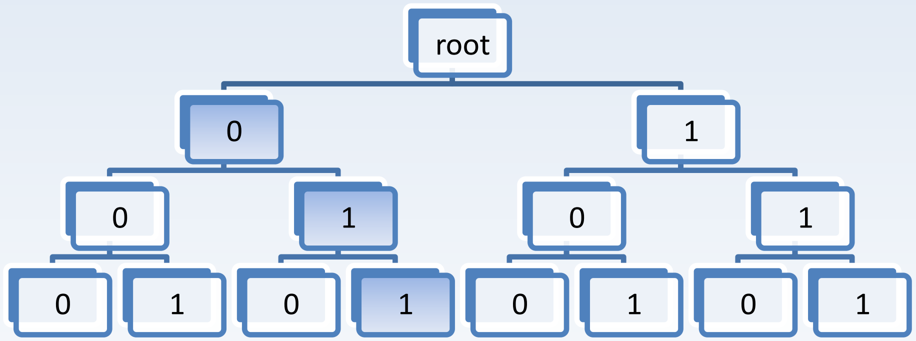


Alias Analysis

- Proof by reduction
- Reduction Buildup:
  - Binary Tree
    - *branch(0) – Left*
    - *branch(1) – Right*
  - Strings as Paths:
    - For binary string $b_1, b_2, b_3, \cdots, b_n$

      $Path(b_1, b_2, b_3, \cdots, b_n) =$

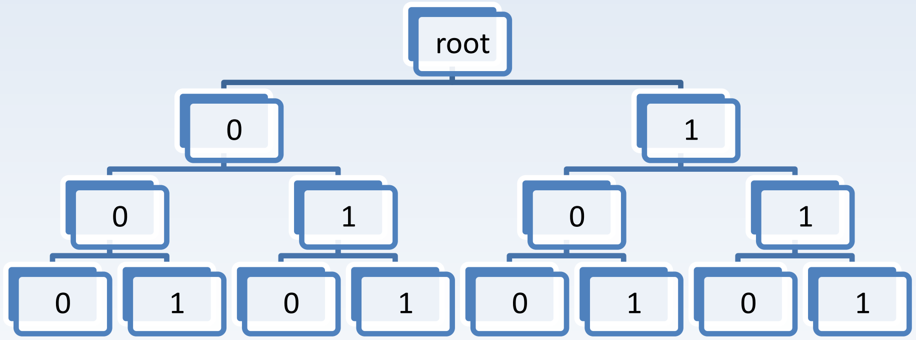
      $branch(b_1) \rightarrow branch(b_2) \rightarrow \cdots \rightarrow branch(b_n)$



S = 011

Alias Analysis

- Proof by reduction
- Reduction Buildup:



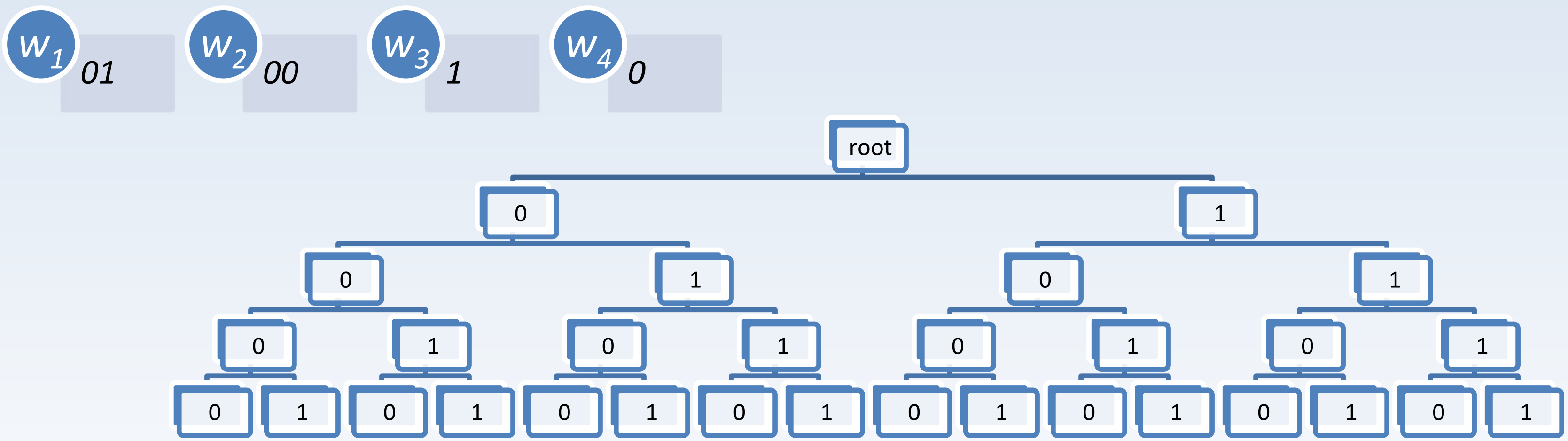
Let α, β be two binary strings

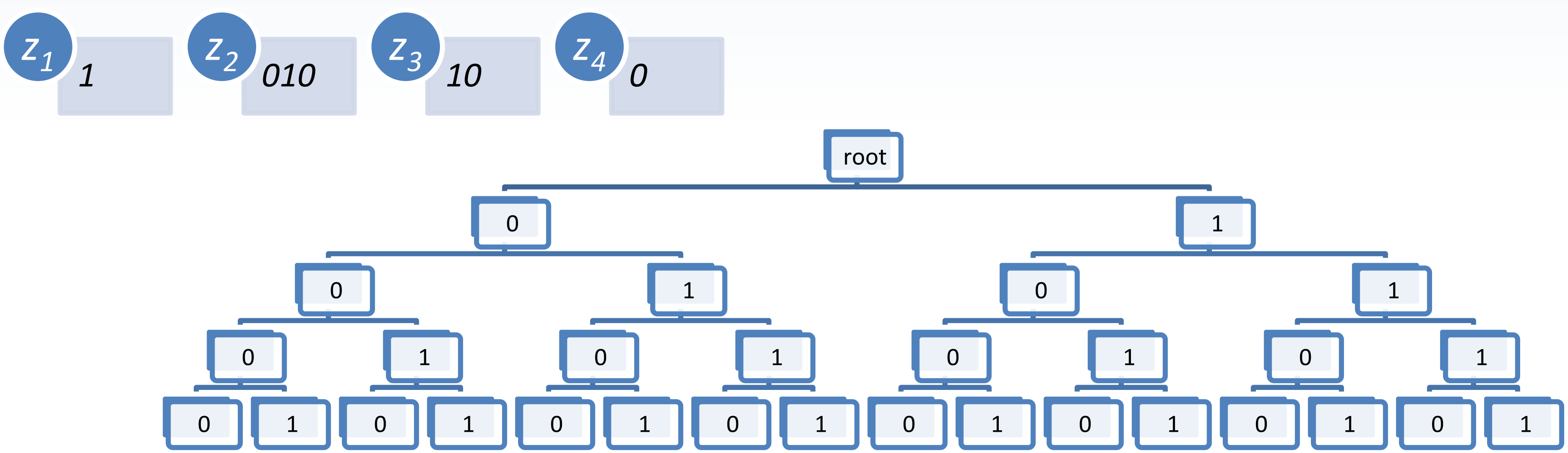
$\alpha = \beta \Leftrightarrow$

$root \rightarrow path(\alpha)$ and $root \rightarrow path(\beta)$

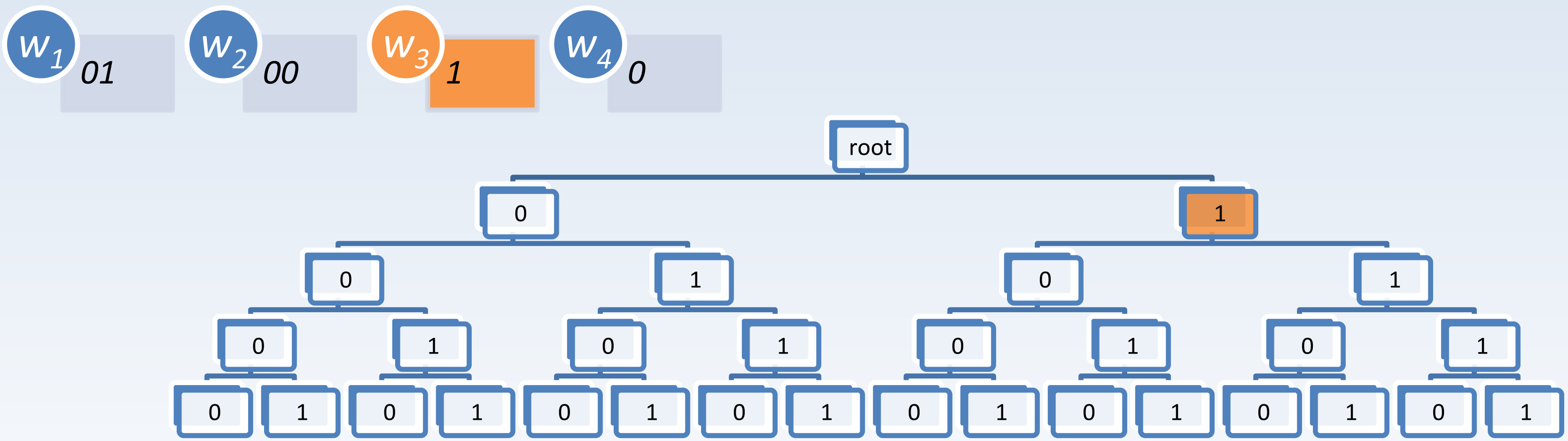
refer to the same node in the tree

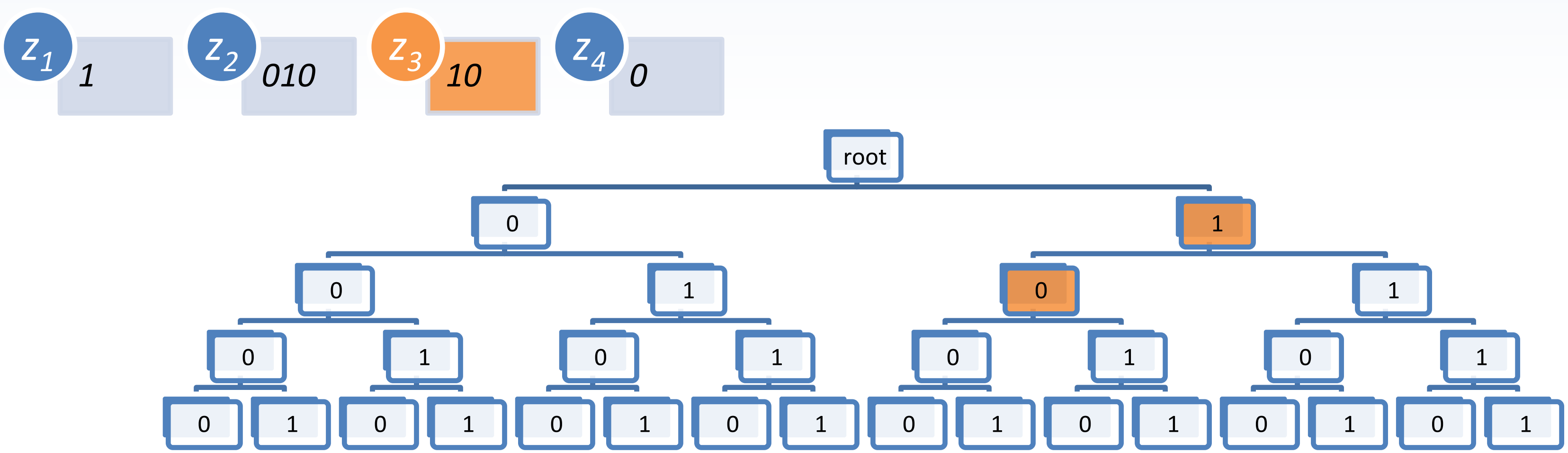
Reduction - Example



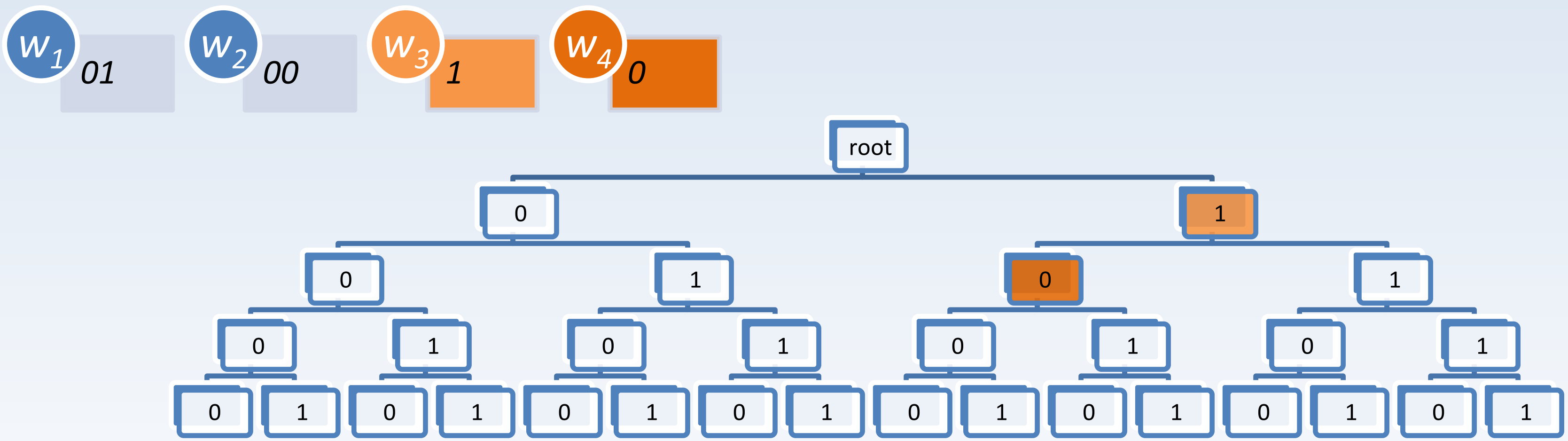


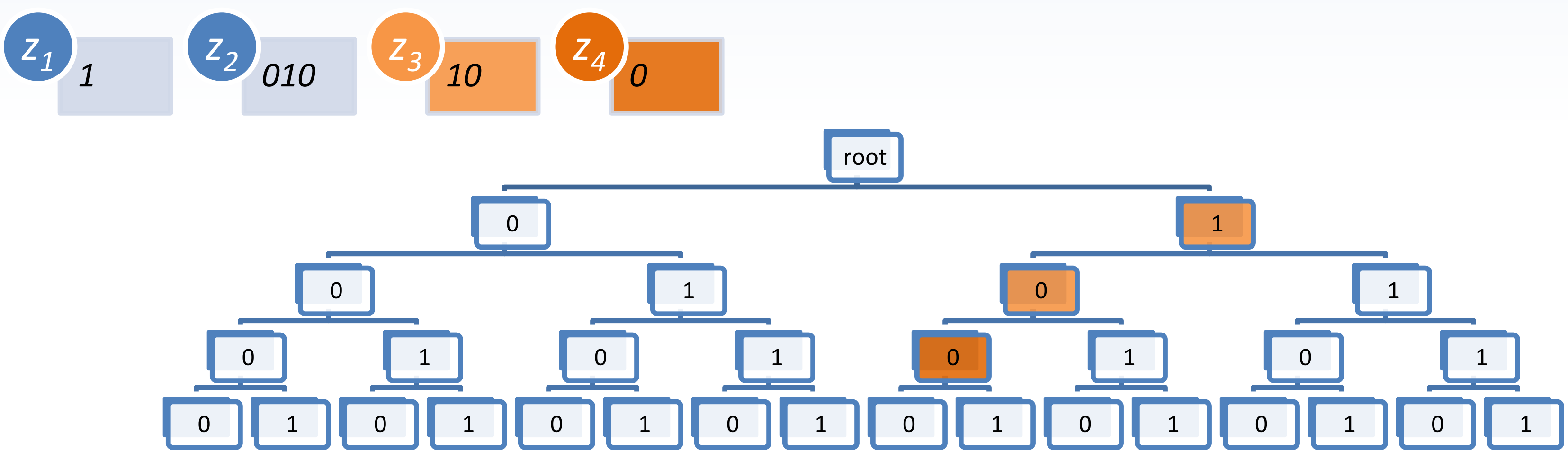
Reduction - Example



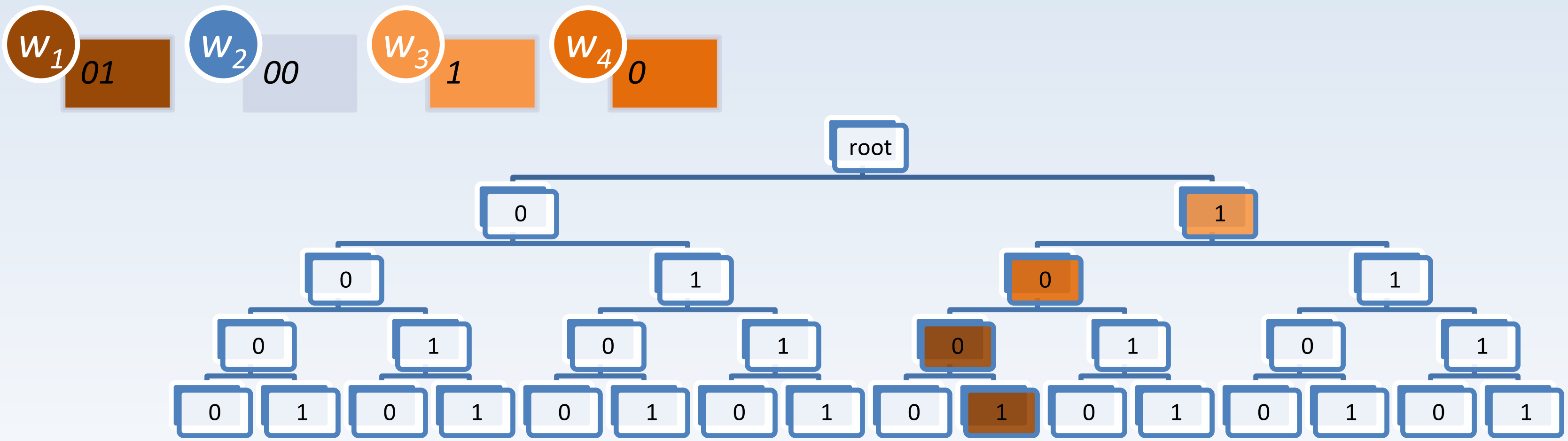


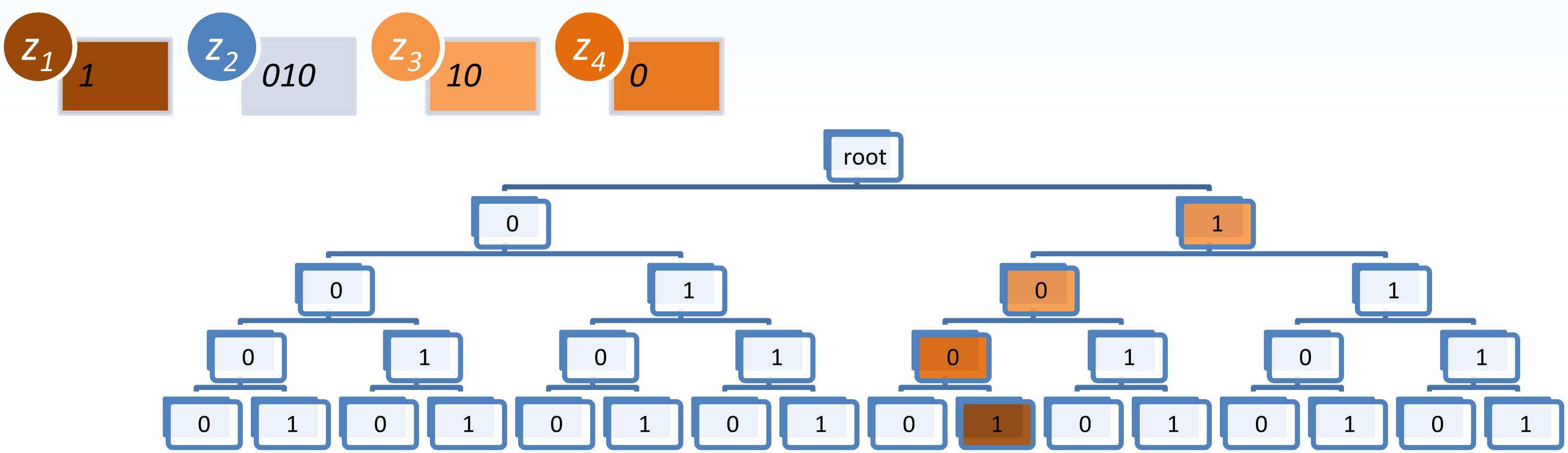
Reduction - Example





Reduction - Example





Reduction

- After finishing the loop
  - ***p***->*left* = &*node* ; *undefined.left* = &*undefined*
  - The given PCP has an affirmative answer iff: *(***q***->*left*) **may-alias** *node*

  - PCP ≤ may-alias
  - PCP is not Recursive

  ➔ may-alias is **not Recursive**

Reduction

- may-alias is not Recursive. But is it RE?
- What about **must-alias**?
- *Theorem 2*: The intraprocedual **must-alias** relation is not even RE
- Proof:
  - We'll use **must**-alias information to compute **may**-alias information. (Line 40)
  - not may-alias ≤ must-alias
  - may-alias is RE but not R

  ➔(from *Lemma2*): co-may-alias is not RE

  ➔must-alias is not RE

Conclusion

- The intraprocedural may-alias problem is **undecidable** for languages with if statmenets, loops, dynamic storage, and recursive data structures.
- The intraprocedural must-alias problem is **not even RE**.
- ■
- In the absence of recursively defined data structures, various versions of the aliasing problems become decidable, but remain difficult.