Attrattori di Stringhe: Utilizzo di Walnut per Dimostrazioni Automatiche e Implementazione di Algoritmi di Ricerca e Verifica

Claudio Simonelli N86003781

18 Marzo 2025

Introduzione

Obiettivo: Studiare gli attrattori di stringhe e verificarne la presenza tramite Walnut.

- Definizione formale di parole, parole sturmiane, attrattori, k-attrattori.
- Implementazione di macro **Walnut** per la verifica della presenza e ricerca di attrattori e k-attrattori all'interno delle parole bi-infinite di Fibonacci e Thue-Morse.
- Implementazione di algoritmi per la verifica e ricerca di 2-attrattori.
- Struttura dati per l'estrazione di una sottostringa tramite attrattori.

Parole Matematiche

Sia Σ un alfabeto finito.

- **Parola finita**: funzione $w : \{0, 1, \ldots, n-1\} \to \Sigma$.
- **Parola infinita**: funzione $w : \mathbb{N} \to \Sigma$.
- **Parola bi-infinita**: funzione $w : \mathbb{Z} \to \Sigma$.

Parole Sturmiane

Una parola Sturmiana è una parola infinita w con **complessità fattoriale**:

$$p_w(n) = n + 1, \quad \forall n \geq 0.$$

Ovvero ha esattamente $n + 1$ sottostringhe distinte di lunghezza n. Un esempio noto è la parola di Fibonacci

$$f = 0100101001001010010100100101001 0 \cdots$$

Attrattori di Stringhe

Un **attrattore di stringhe** per una stringa $\mathrm{T} \in \Sigma^n$ è un insieme di γ posizioni $\Gamma = \{j_1, ..., j_\gamma\}$ tale che ogni sottostringa $T[i..j]$ ha almeno un'occorrenza $T[i^{'}..j^{'}] = T[i..j]$ con $j_k \in [i^{'}, j^{'}]$, per qualche $j_k \in \Gamma$.
Span di un attrattore:

$$\mathrm{span}(\Gamma) = \max \Gamma - \min \Gamma.$$

Esempio:

$$\mathrm{CDA\underline{B}CC\underline{D}ABC\underline{CA}}$$

Ha attrattore $\Gamma = \{4, 7, 11, 12\}$.

k-Attrattori di Stringhe

Data una stringa $T \in \Sigma^n$, un k-attrattore è un insieme $\Gamma \subseteq [1..n]$ tale che ogni sottostringa $T[i..j]$ con lunghezza minore o uguale a k ha almeno un'occorrenza $T[i'..j'] = T[i..j]$ tale che esista almeno una posizione $j'' \in [i'..j']$ che appartiene a Γ.
Esempio:

$$\underline{A}ABBC\underline{BC}ABA$$

Ha 2-attrattore $\Gamma = \{0, 4, 5\}$.

Uso di Walnut

Walnut: software per verificare proprietà combinatorie esprimibili in logica del primo ordine secondo l'aritmetica di Büchi.

- Definizione della **parola bi-infinita di Fibonacci**.
- Definizione della **parola bi-infinita di Thue-Morse**.
- Verifica di **attrattori e k-attrattori**.

Definizione della parola di Fibonacci su Walnut

reg negfib msd_neg_fib "$0*(0|10)*1$" :
combine NF negfib:

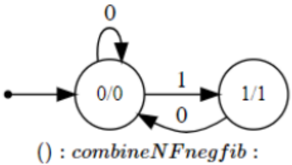


Figure: DFAO Fibonacci

Definizione della parola di Thue-Morse su Walnut

def tmn "T[n-1]=@1":
combine T2 tmn:
rsplit T21 [+] T:
rsplit T22 [-] T2:
join TM21 T21[x] T22[x]:

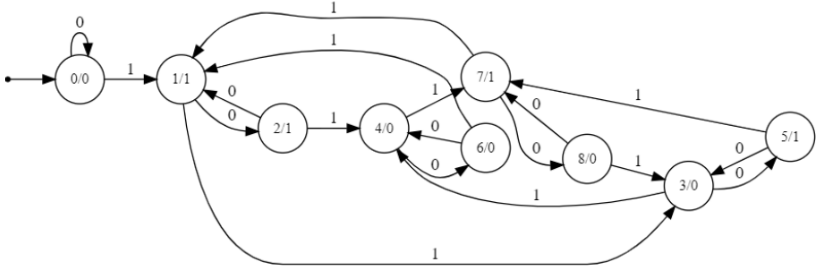


Figure: DFAO Thue-Morse bi-infinita

Predicato Walnut per la verifica di attrattori

eval string_attractor "?msd_neg_fib Ai,m(($m > 0$) $\implies$ (Ej ((($j >= 1 - m$) & ($j <= 1$)) & (Ak ((0 $<= k$) & ($k < m$) $\implies$ (NF[i+k]=NF[j+k])))))))";

computed :1 states - 25ms

computed :1 states - 2ms

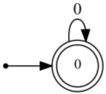
computed :2 states - 2ms

TRUE

- $A = \forall$
- $E = \exists$
- $w[i..j]$ = sottostringa di w che inizia in i e finisce in j

Versione con posizione variabile libera

eval string_attractor_NF "?msd_neg_fib Ai,m(($m > 0$) $\implies$ (Ej ((($j >= 1 - m + p$) & ($j <= p + 1$)) & (Ak (($0 <= k$) & ($k < m$)) $\implies$ (NF[i+k]=NF[j+k])))))))";



(p): ?msd_neg_fib Ai,m((m>0) => (Ej (((j >= 1-m+p) & (j <= p+1)) & (Ak ((0<=k) & (k<m) => (NF[i+k]=NF[j+k]))))))

Figure: Automa restituito dal predicato per la ricerca di uno string attractor sulla parola bi-infinita di Fibonacci

Macro per la verifica di attrattori con posizione e span come parametri

$$\text{macro check\_attractor\_at\_p\_l "?\%0 Ai,m}((m > 0) \implies \text{Ej}((j > \%2 - m) \;\&\; (j <= \%2 + \%3) \;\&\; \text{Ak}((0 <= k) \;\&\; (k < m) \implies (\%1[\text{i+k}] = \%1[\text{j+k}]))))\text{";}$$

Esempio di chiamata:

eval check_attractor_at_p_l_example
"#check_attractor_at_p_l(msd_neg_fib, NF, 5,6)";

Output = FALSE

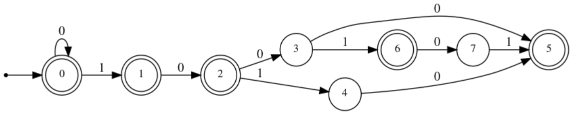
- %0 = sistema di numerazione
- %1 = automa accettante una parola
- %2 = posizione
- %3 = span

Versione con posizione libera e span come parametro

$$\text{macro check\_attractor\_only\_l "?\%0 Ai,m}((m > 0) \implies \text{Ej}((j > p - m)\ \&\ (j <= p + \%2)\ \&\ \text{Ak}((0 <= k)\ \&\ (k < m) \implies (\%1[\text{i+k}] = \%1[\text{j+k}]))))\text{";}$$

Esempio di chiamata:

eval check_attractor_only_l_example_2
"#check_attractor_only_l(msd_neg_fib, NF, 6)";



(p): #check_attractor_only_l(msd_neg_fib, NF, 6)

Figure: Automa accettante le posizioni che fungono da attrattore con span 6 sulla parola di Fibonacci

- %0 = sistema di numerazione
- %1 = automa accettante una parola
- %2 = span

Macro Walnut per verifica di k-attrattori

macro check_k_attractor_at_p_l "?%0 Ai,m(($m > 0$ & $m <$ %4) $\implies$ Ej(($j >$ %2 $- m$) & ($j <=$ %2 + %3) & Ak(($0 <= k$) & ($k < m$) $\implies$ (%1[i+k] = %1[j+k])))))";

Esempio di chiamata:

eval check_k_attractor_at_pl_example "#check_k_attractor_at_p_l(msd_neg_fib, NF, 5, 16, 10)";

Output:

computed :1 states - 39ms
computed :1 states - 2ms
computed :2 states - 3ms
____
TRUE

- %0 = sistema di numerazione
- %1 = automa accettante una parola
- %2 = posizione
- %3 = span
- %4 = k

Altro esempio con Thue-Morse

eval check_k_attractor_only_l_example2
"#check_k_attractor_only_l(msd_neg_2, TM21, 16, 10)";

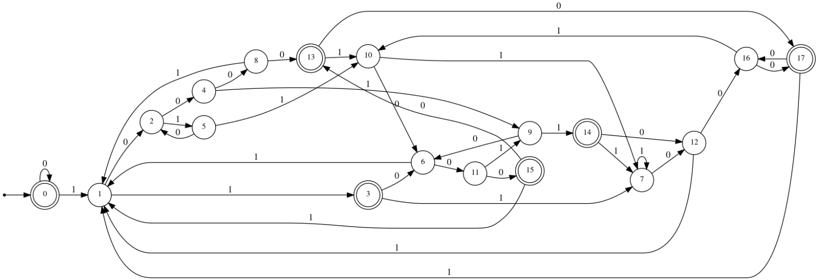


Figure: Automa accettante le posizioni che fungono da 10-attrattore con span 16 per Thue-Morse

Algoritmi sui 2-Attrattori

- **Algoritmo di verifica**: data una parola, un insieme di posizioni e uno span l, verifica se tale insieme è un **2-attrattore** con span l per la parola data in input.
- **Algoritmo di ricerca**: data una parola e uno span l, restituisce l'insieme di posizioni che fungono da **2-attrattore** con span l per la parola in input.

Algoritmo di verifica

Algorithm 1 hasDuoAttractor

Input: $w \in \Sigma^n$, $p \in \mathbb{N}$, $l \in \mathbb{N}^*$
Output: $\{true, false\}$
$length \leftarrow |w|$
if $p < 0 \vee p \geq length \vee p + l \geq length$ **then**
    return false
end if
if $p = 0$ **then**
    $attractorSubstring \leftarrow w[p..p + l]$
else
    $attractorSubstring \leftarrow w[p - 1..p + l]$
end if
for $i = 0$ **to** $length$ **step** $+1$ **do**
    $couple \leftarrow w[i..i + 1]$
    if $couple \notin attractorSubstring$ **then**
        return false
    end if
end for
return true

Complessità temporale $O(nm)$.

Algoritmo di ricerca

Algorithm 2 getDuoAttractorPositions

Input: $w \in \Sigma^n,\ l \in \mathbb{N}^*$
Output: $P \subseteq \mathbb{N}$
$\mathit{length} \leftarrow |w|$
$P = \emptyset$
for $p = 1$ **to** $\mathit{length} - 1$ **step** $+1$ **do**
    if $p + l > \mathit{length}$ **then**
        return P
    else if $\mathit{hasDuoAttractor}(w, p, l) = \mathit{true}$ **then**
        $P \leftarrow P \cup p$
    end if
end for
return P

Complessità temporale $O(n^2 m)$.

Implementazione di una struttura dati per estrazione di sottostringhe

Struttura dati presentata da Kempa e Prezza(2018). Definita in Java a partire dagli attributi:

- **T**: Parola di input;
- **n**: Lunghezza della parola T;
- Σ: Alfabeto della parola T;
- σ: Dimensione dell'alfabeto Σ;
- Γ: Attrattore per la parola T;
- γ: Dimensione dell'attrattore T;
- **w**: Dimensione della parola di memoria del sistema;
- **l**: Lunghezza della sottoparola che si vuole estrarre da T;

Implementazione di una struttura dati per estrazione di sottostringhe

- τ: Parametro intero fissato al momento della costruzione della struttura dati;
- s_i: Lunghezza delle sottoparole di contesto al livello i, calcolata come $\frac{n}{\gamma\tau^{i-1}}$";
- α: Numero di caratteri packed che la struttura dati supporta di estrarre in $O(log_t(\frac{n}{\gamma}))$, calcolato come $w\frac{log_\tau(\frac{n}{\gamma})}{log(\alpha)}$;
- **i***: Livello in cui i caratteri delle sottostringhe di contesto vengono memorizzati esplicitamente. È definito come il più piccolo numero tale che $s_{i^*+1} < 2\alpha = 2w\frac{log_\tau(\frac{n}{\gamma})}{log(\alpha)}$;

Complessità temporale: $O(log_\tau(\frac{n}{\gamma}) + l\frac{log(\alpha)}{w})$
Complessità spaziale: $O(\gamma\tau log_\tau(\frac{n}{\gamma}))$

Implementazione di una struttura dati per estrazione di sottostringhe

Utilizza un attrattore di dimensione γ per organizzare la struttura in $O(log_t(\frac{n}{\gamma}))$ livelli:

- Livello 0 (radice);
- Livelli intermedi (da 1 a $i^* - 1$);
- Livello i* (foglia).

Metodi della classe Java StringAttractorRandomAccess

- extractSubstring(int i, int l);
- buildStructure();
- calculateIStar();
- StringAttractorRandomAccess(String T, Set Γ, int τ, int w).

Bibliografia

1. Kempa Dominik, Prezza Nicola. *"At the roots of dictionary compression: String attractors."* Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing. 2018.
2. Pierre Béaur et al. arXiv e-prints, 2024, arXiv: 2403.13449.