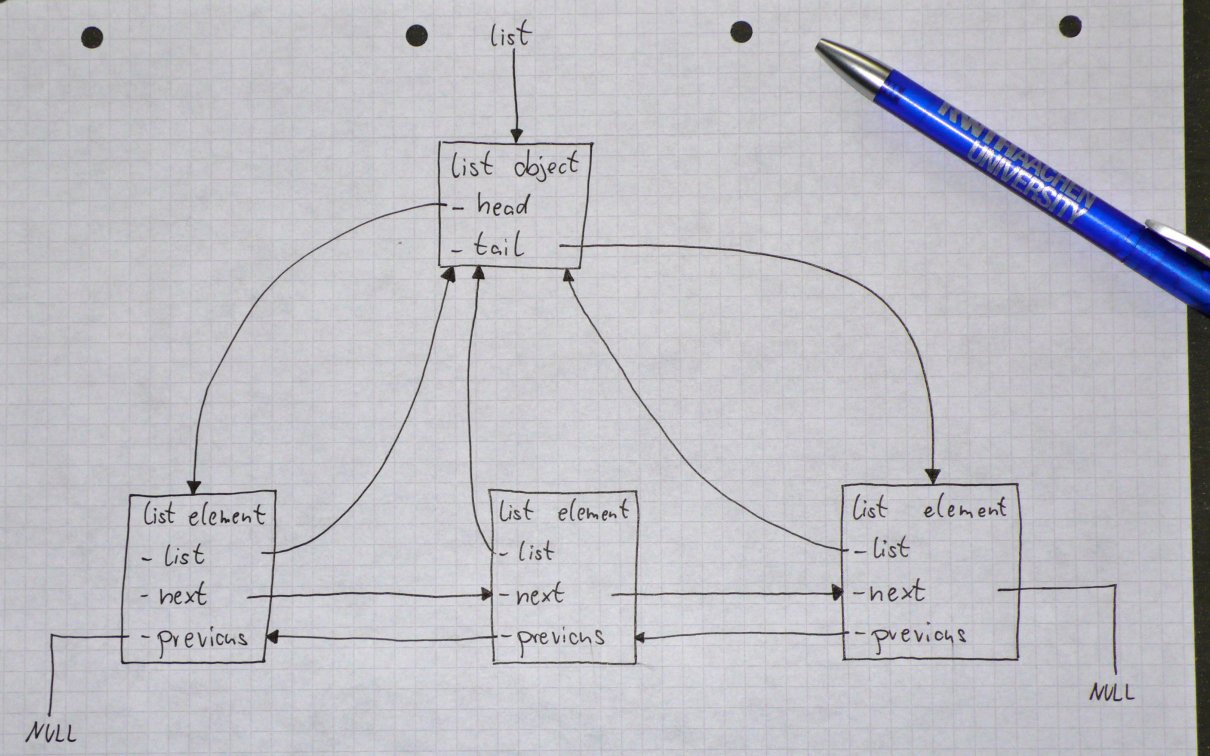
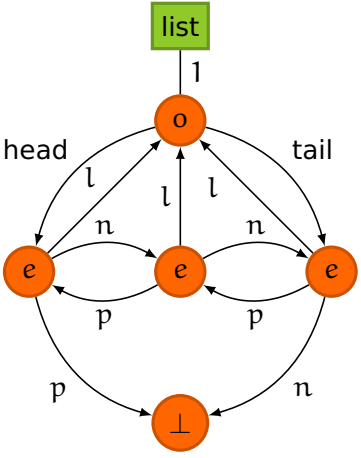
Automata-Based Detection of Hypergraph Embeddings

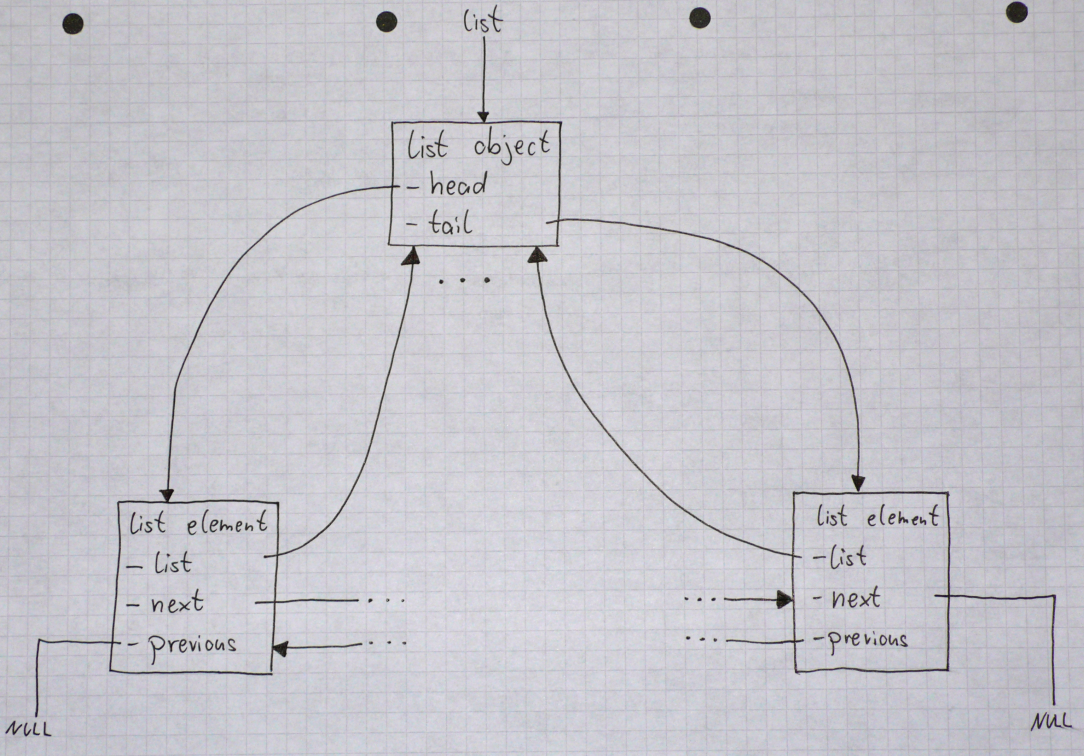
Henrik Barthels

October 13, 2011

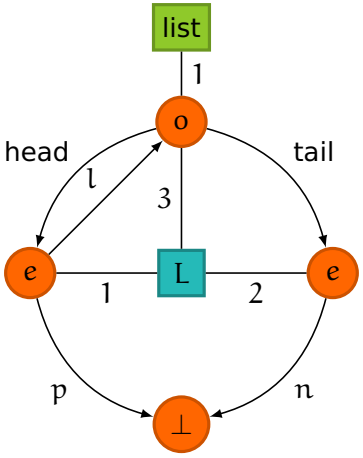

list
list object
- head
- tail
list element
- list
- next
- previous
list element
- list
- next
- previous
list element
- list
- next
- previous
NULL
NULL

Heap Configuration

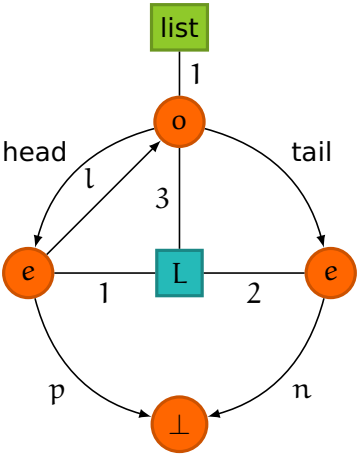



list
list object
- head
- tail
list element
- list
- next
- previous
list element
- list
- next
- previous
NULL
NULL

Hypergraphs



Hypergraphs



Definition

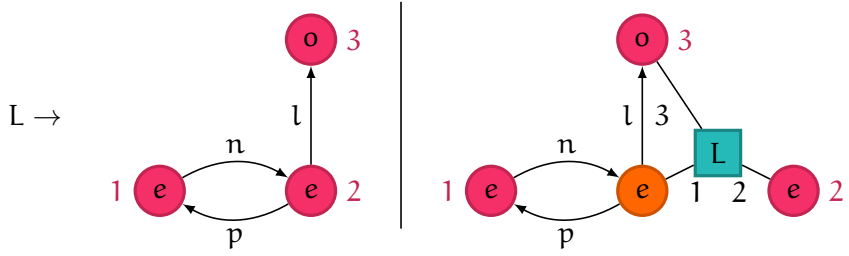
Labeled and annotated *hypergraph* over Λ and $\Sigma = \mathrm{Var}_\Sigma \uplus \mathrm{Sel}_\Sigma$: $H = (V, E, \mathrm{att}, \mathrm{lab}, \mathrm{type}, \mathrm{ext})$

- Σ: Finite ranked alphabet.
- Rank $\mathrm{rk} : \Sigma \to \mathbb{N}$
- Λ: Set of types.
- Attachment function $\mathrm{att} : E \to V^*$
- Hyperedge-labeling function $\mathrm{lab} : E \to \Sigma$
- Vertex-annotation function $\mathrm{type} : V \to \Lambda$
- Pairwise distinct external vertices: $\mathrm{ext} \in V^*$

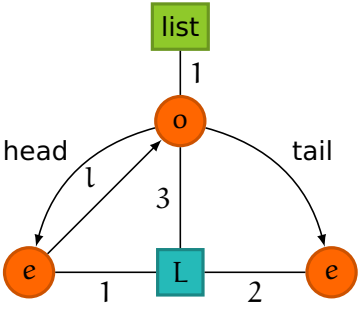
Hyperedge Replacement Grammar

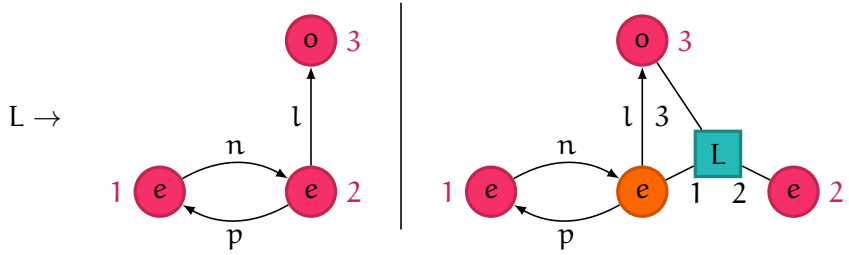
Definition

A *hyperedge replacement grammar* (HRG) over alphabets $\Gamma_N = (\Sigma_N, \Lambda)$ is a set of production rules of the form $X \to H$, with $X \in N$ and $H \in HG_{\Gamma_N}$ where $|\mathrm{ext}_H| = \mathrm{rk}(X)$.

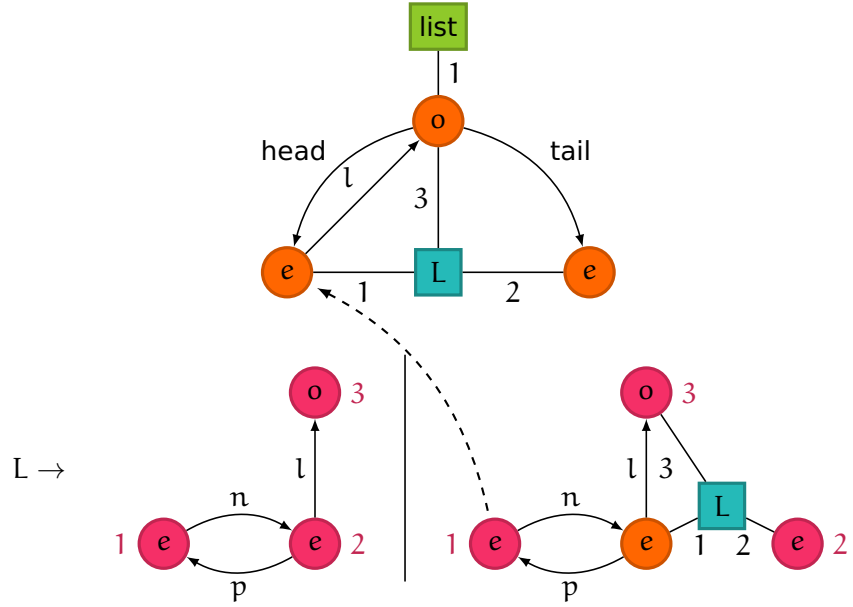


Hyperedge Replacement Grammar

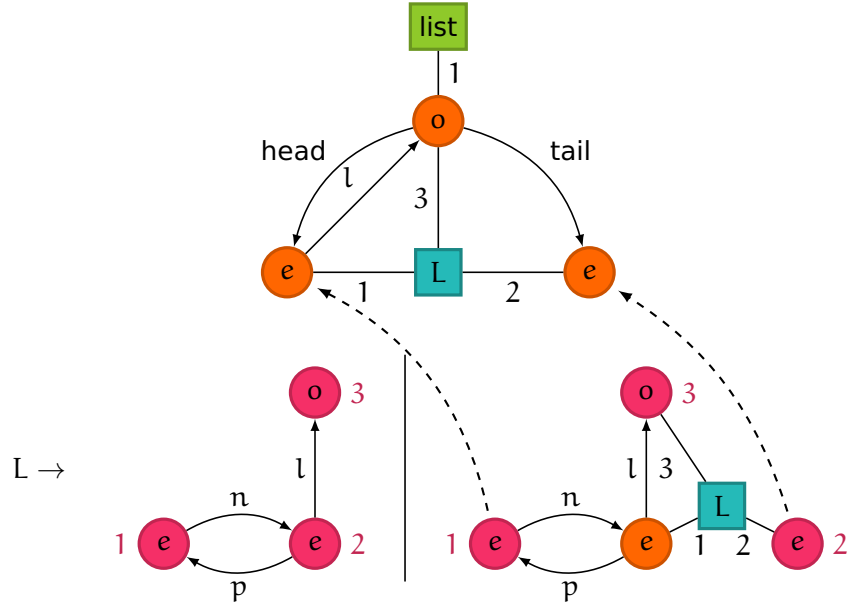




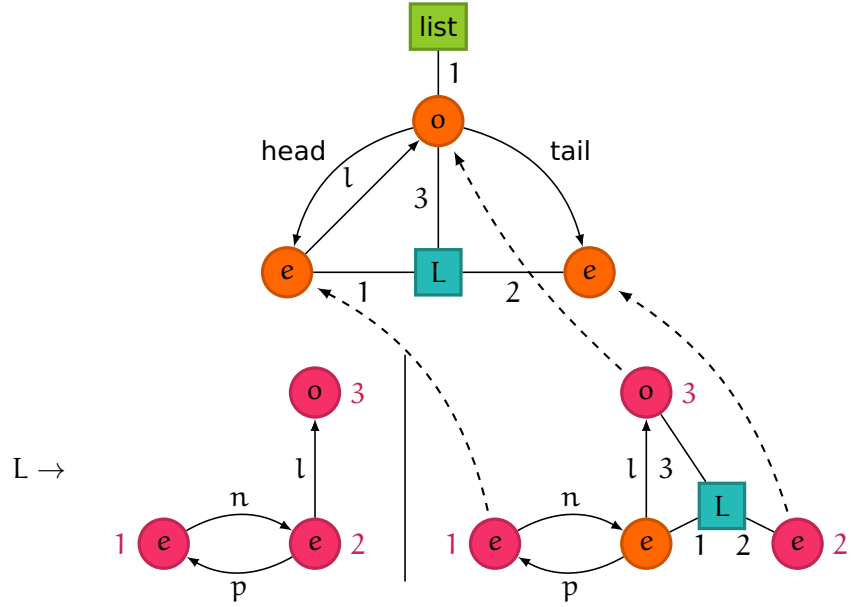
Hyperedge Replacement Grammar



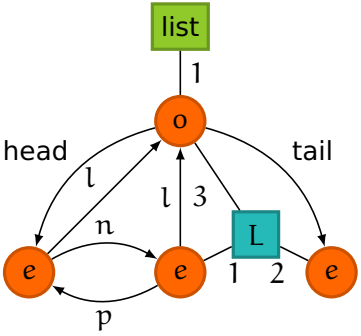
Hyperedge Replacement Grammar

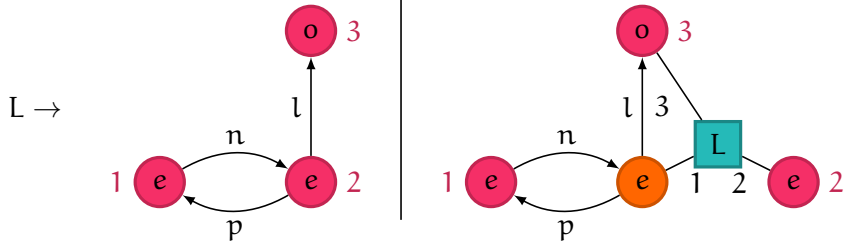


Hyperedge Replacement Grammar

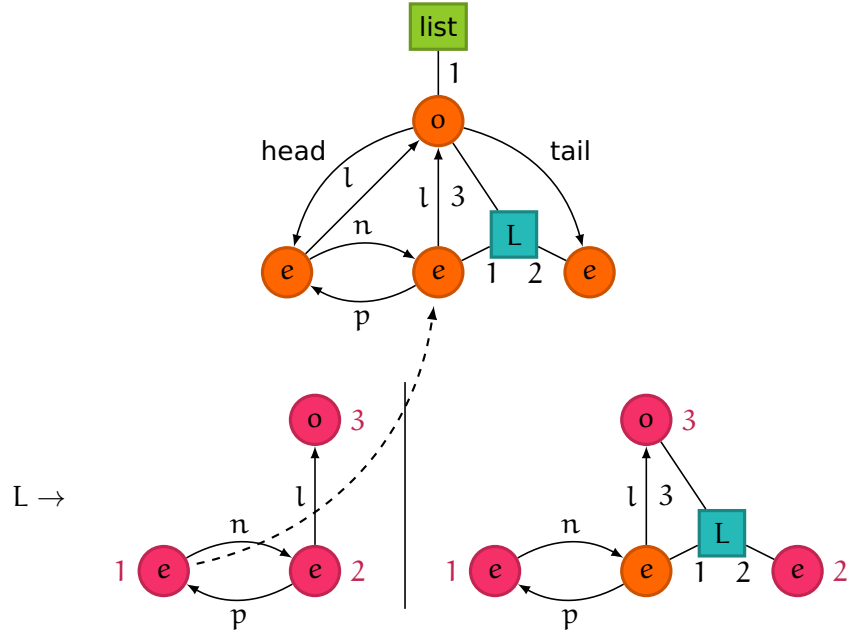


Hyperedge Replacement Grammar

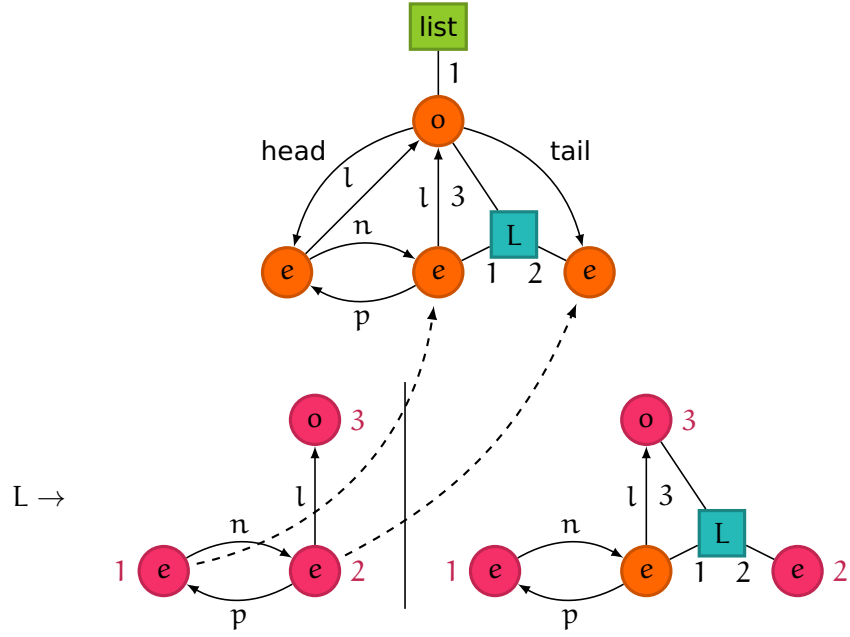




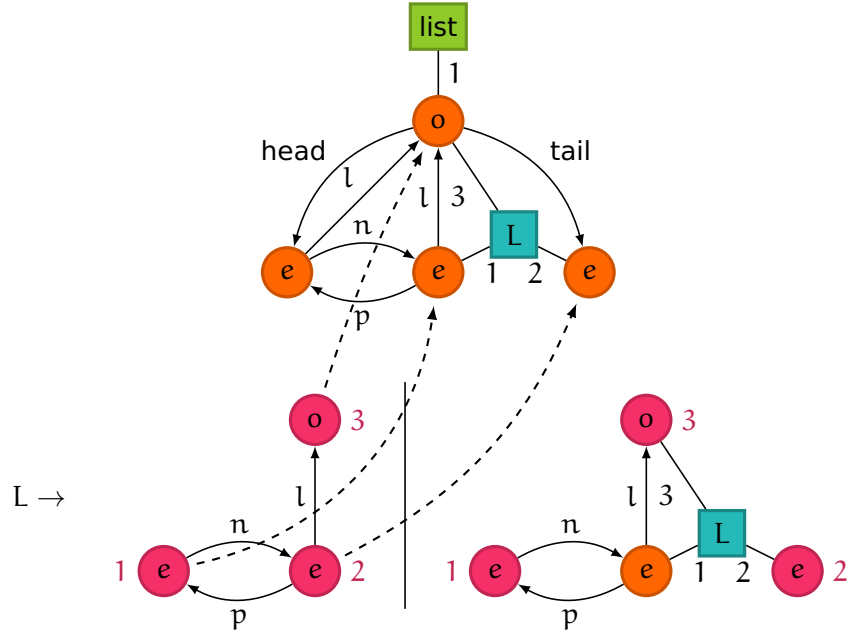
Hyperedge Replacement Grammar



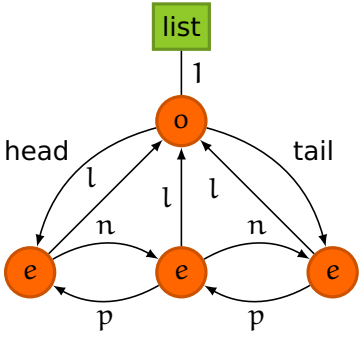
Hyperedge Replacement Grammar

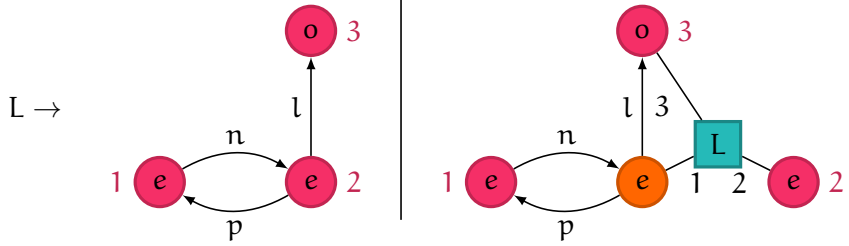


Hyperedge Replacement Grammar

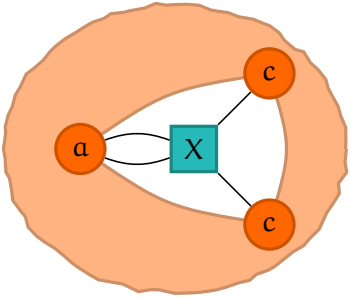


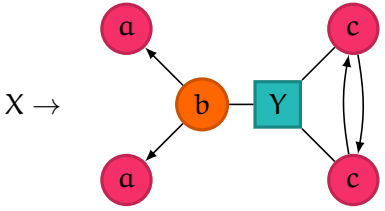
Hyperedge Replacement Grammar



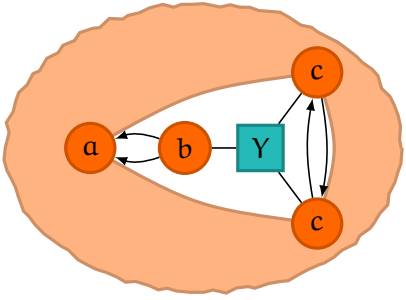


Embeddings

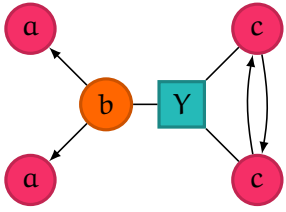




Embeddings



$X \to$



Embeddings

Definition

Let $\{X \to H\} \in DSG_{\Gamma_N}$ and $G \in HC_{\Gamma_N}$. H is *embedded* in G if there are functions $f_V : V_H \to V_G$ and $f_E : E_H \to E_G$ such that

$$lab_H(e) = lab_G(f_E(e)) \forall e \in E_H$$

$$type_H(v) = type_G(f_V(v)) \forall v \in V_H \setminus [ext_H]$$

$$v = ext_H(i), type_H(v) \neq type_G(f_V(v)) \Rightarrow type_G(f_V(v)) = \bot, ts(X)(i) \in \Lambda_0$$

$$att_G(f_E(e)) = f_V(att_H(e)) \, \forall e \in E_H$$

$$v_1, v_2 \in V_H, v_1 \neq v_2, v_1 \notin [ext_H] \Rightarrow f_V(v_1) \neq f_V(v_2)$$

$$ts(X)(i), ts(X)(j) \in \Lambda \Rightarrow f_V(ext_H(i)) \neq f_V(ext_H(j)), i, j \in [1, rk(X)], i \neq j$$

$$\forall e \in E_G \setminus f_E(E_H), i \in [1, rk(e)], v \in V_H \setminus [ext_H] : \neg \exists e' : f_E(e') = e \Rightarrow att_G(e)(i) \neq f_V(v)$$

Embeddings

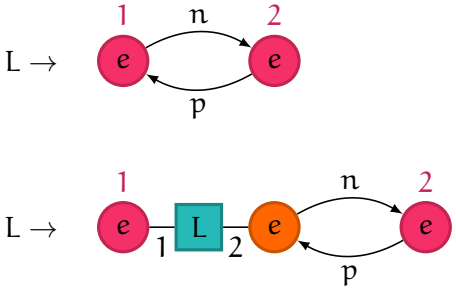
Theorem

Detecting embeddings is NP-complete.

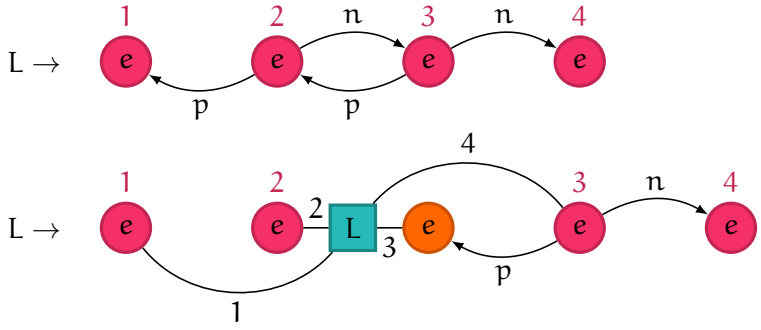
Heap Configurations

- Hypergraphs are unnecessarily general.
- Objects have types (at least in Java, C++, . . .).
- Objects of the same type always have the same pointers.
- Garbage will not be considered.

Hyperedge Replacement Grammar: Doubly-Linked List



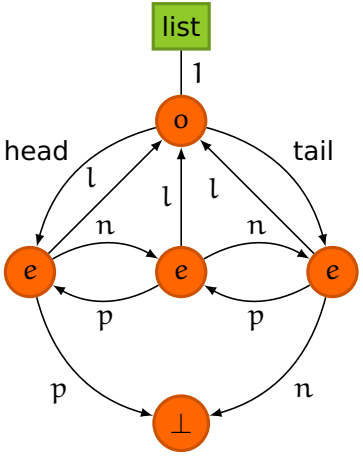
Data Structure Grammar: Doubly-Linked List



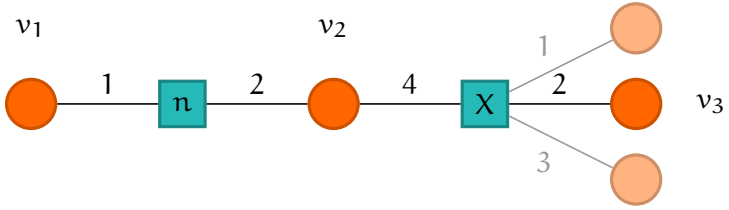
- Selectors have to match type.
- Vertices have either all or none of their selectors (possibly abstracted).

Paths

```
list.head
list.head.next
list.head.previous
list.head.next.previous
```

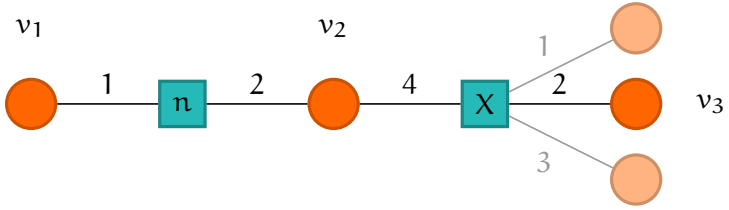


Paths



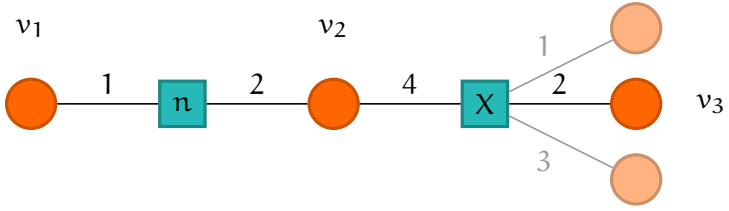
$$\pi = (n, 2)(X, 2)$$

Paths



$$\pi = n(X, 2)$$

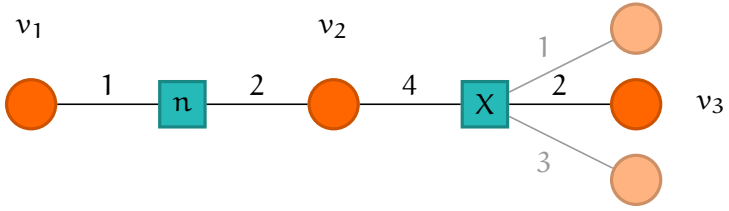
Paths



$$\pi = n(X, 2)$$

$$\text{eval}(\pi, G, v_1) = v_3$$

Paths



$$\pi = n(X, 2)$$

$$\langle \pi \rangle_{(G, v_1)} = v_3$$

Path-Based Embeddings

$$
\begin{aligned}
\mathrm{type}_H(\langle\pi\rangle_{(H,\nu_H)}) &= \mathrm{type}_G(\langle\pi\rangle_{(G,\nu_G)}) \forall \pi \in \Pi_{DFS} \\
\mathrm{ot}(\langle\pi\rangle_{(H,\nu_H)}) &= \mathrm{ot}(\langle\pi\rangle_{(G,\nu_G)}) \forall \pi \in \Pi_{DFS}, \langle\pi\rangle_{(H,\nu_H)} \in V \setminus [\mathrm{ext}_H] \\
\mathrm{ot}(\langle\pi\rangle_{(H,\nu_H)}) &= \mathrm{ot}(\langle\pi\rangle_{(G,\nu_G)}) \\
&\quad \forall \pi \in \Pi_{DFS}, \langle\pi\rangle_{(H,\nu_H)} \in [\mathrm{ext}_H], \mathrm{ot}(\langle\pi\rangle_{(H,\nu_H)}) \neq \emptyset \\
\mathrm{ot}(\langle\pi\rangle_{(G,\nu_G)}) &\in \{\mathrm{ot}(\nu') \mid \nu' \in V_G, \mathrm{type}_G(\nu') = \mathrm{type}_H(\nu)\} \\
&\quad \forall \pi \in \Pi_{DFS}, \langle\pi\rangle_{(H,\nu_H)} \in [\mathrm{ext}_H], \mathrm{ot}(\langle\pi\rangle_{(H,\nu_H)}) = \emptyset \\
\deg(\langle\pi\rangle_{(H,\nu_H)}) &= \deg(\langle\pi\rangle_{(G,\nu_G)}) \forall \pi \in \Pi_{DFS}, \langle\pi\rangle_{(H,\nu_H)} \in V \setminus [\mathrm{ext}_H] \\
\deg(\langle\pi\rangle_{(H,\nu_H)}) &\leqslant \deg(\langle\pi\rangle_{(G,\nu_G)}) \forall \pi \in \Pi_{DFS}, \langle\pi\rangle_{(H,\nu_H)} \in [\mathrm{ext}_H] \\
\mathrm{ts}(X)(i), \mathrm{ts}(X)(j) \in \Lambda, \langle\pi_1\rangle_{(H,\nu_H)} &= \mathrm{ext}_H(i), \langle\pi_2\rangle_{(H,\nu_H)} = \mathrm{ext}_H(j) \\
&\quad \Rightarrow \langle\pi_1\rangle_{(G,\nu_G)} \neq \langle\pi_2\rangle_{(G,\nu_G)}, i, j \in [1, \mathrm{rk}(X)], i \neq j \\
\langle\pi_1\rangle_{(H,\nu_H)} = \langle\pi_2\rangle_{(H,\nu_H)} &\Rightarrow \langle\pi_1\rangle_{(G,\nu_G))} = \langle\pi_2\rangle_{(G,\nu_G))} \forall \pi_1, \pi_2 \in \Pi_{DFS}
\end{aligned}
$$

Path-Based Embeddings

Theorem

*There is a path-based embedding if and only if there is an embedding **and** there is one external vertex from which every other vertex can be reached.*

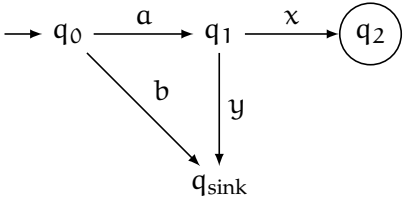
Path-Based Embeddings

Theorem

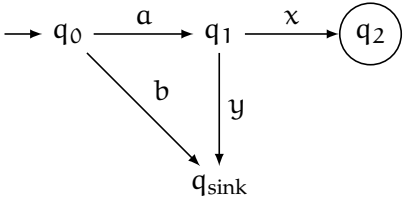
*There is a path-based embedding if and only if there is an embedding **and** there is one external vertex from which every other vertex can be reached.*

How to avoid checking properties more often than necessary if we search for more than one embedding?

Function Automata

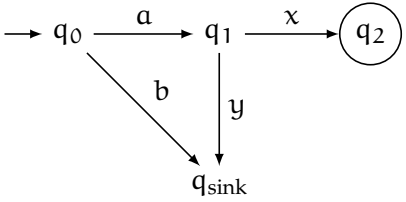


Function Automata



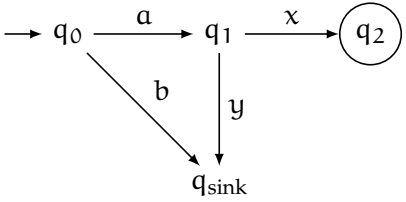
$$\lambda(q_0) = f : D \to \{a, b\}$$
$$\lambda(q_1) = g : D \to \{x, y\}$$

Function Automata



$$\lambda(q_0) = f : D \to \{a, b\}$$
$$\lambda(q_1) = g : D \to \{x, y\}$$
$$\lambda(q_2) = \text{end}$$
$$\lambda(q_{sink}) = \text{end}$$

Function Automata



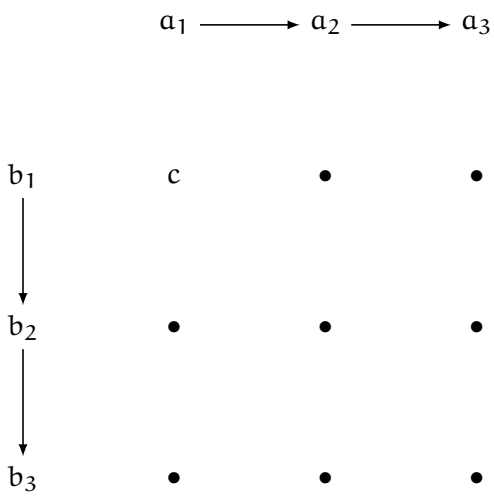
$$\lambda(q_0) = f : D \to \{a, b\}$$
$$\lambda(q_1) = g : D \to \{x, y\}$$
$$\lambda(q_2) = \text{end}$$
$$\lambda(q_{sink}) = \text{end}$$

Definition

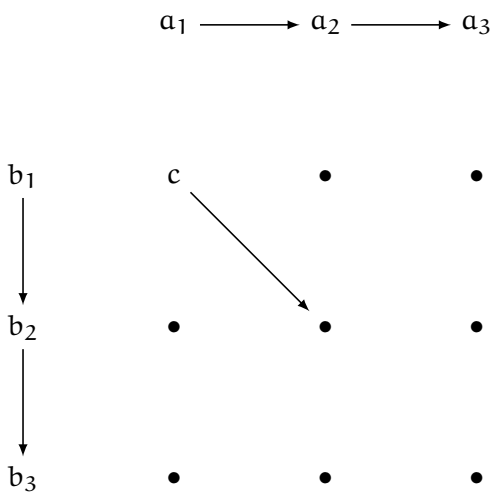
Cycle-free deterministic finite state machine: $\mathcal{A} = (Q, \mathfrak{F}, \lambda, \delta, q_0, F)$.

- Set of states Q.
- Universe of functions $\mathfrak{F}$ with $\text{end} \in \mathfrak{F}$.
- $\lambda : Q \to \mathfrak{F}$ assigns a function to each state.
- $\Omega = \bigcup_{f \in \mathfrak{F}} \Omega_f$ with $f : D \to \Omega_f \in \mathfrak{F}$.
- Transition function $\delta : Q \times \Omega \to Q$.
- Initial state $q_0 \in Q$.
- Set of accepting states $F \subseteq \{q \in Q \mid \lambda(q) = \text{end}\}$.
- $\lambda(q) \preccurlyeq \lambda(\delta(q, \omega)) \forall \omega \in \Omega_{\lambda(q)}$.

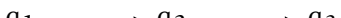
Union of Function Automata



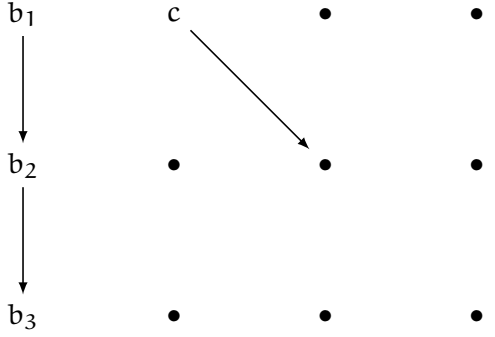
Union of Function Automata



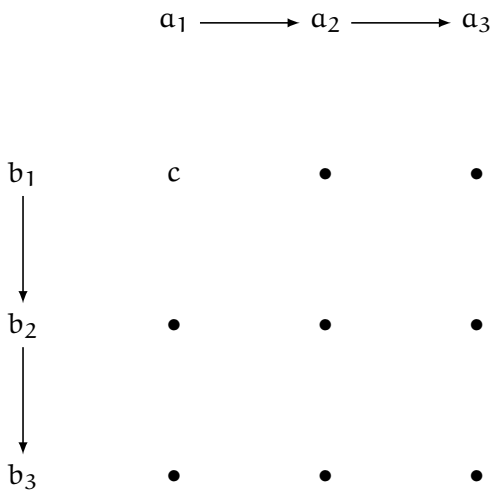
Union of Function Automata



$$\lambda(a_1) = \lambda(b_1)$$
$$\lambda(c) = \lambda(a_1) = \lambda(b_1)$$

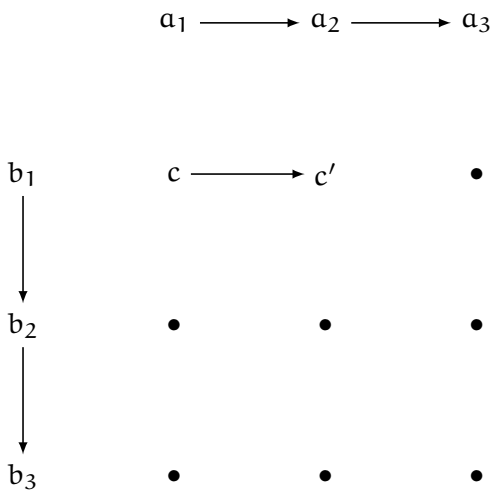


Union of Function Automata



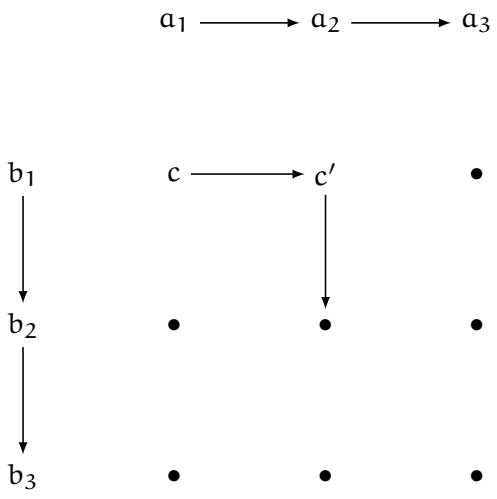
$$\lambda(a_1) \prec \lambda(b_1)$$

Union of Function Automata



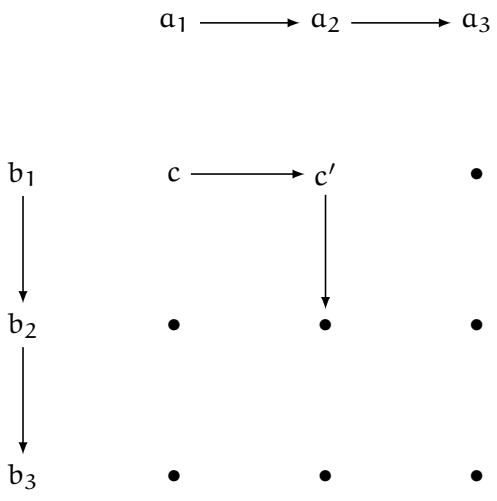
$$\lambda(a_1) \prec \lambda(b_1)$$
$$\lambda(c) = \lambda(a_1)$$

Union of Function Automata



$$\lambda(a_1) \prec \lambda(b_1)$$
$$\lambda(c) = \lambda(a_1)$$
$$\lambda(a_2) \succ \lambda(b_1)$$
$$\lambda(c') = \lambda(b_1)$$

Union of Function Automata



$$\lambda(a_1) \prec \lambda(b_1)$$
$$\lambda(c) = \lambda(a_1)$$
$$\lambda(a_2) \succ \lambda(b_1)$$
$$\lambda(c') = \lambda(b_1)$$

$\lambda(a_x) \succcurlyeq \lambda(b_1)$ eventually holds because of $\lambda(q) \preccurlyeq \lambda(\delta(q, \omega))$.

Embedding Detection Automata

Input: $D = \{(G, v) \mid G \in HC_{\Gamma_N}, v \in V_G\} = I_{\Gamma_N}$

Embedding Detection Automata

Input: $D = \{(G, v) \mid G \in HC_{\Gamma_N}, v \in V_G\} = I_{\Gamma_N}$
Functions:

$$\mathrm{type}(\langle\pi\rangle_{(G,v)})$$
$$\deg_{\geqslant n}(\langle\pi\rangle_{(G,v)})$$
$$\mathrm{ot}(\langle\pi\rangle_{(G,v)})$$
$$(\langle\pi\rangle_{(G,v)} = \langle\pi'\rangle_{(G,v)})$$

Embedding Detection Automata

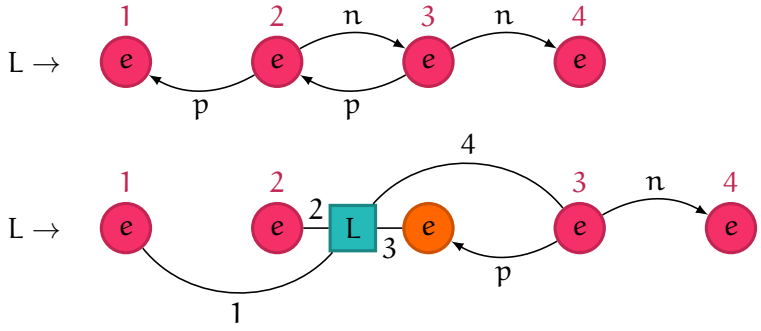
Input: $D = \{(G, v) \mid G \in HC_{\Gamma_N}, v \in V_G\} = I_{\Gamma_N}$
Functions:

$$\begin{aligned}
&\mathrm{type}(\langle\pi\rangle_{(G,v)})\\
&\deg_{\geqslant n}(\langle\pi\rangle_{(G,v)})\\
&\mathrm{ot}(\langle\pi\rangle_{(G,v)})\\
&(\langle\pi\rangle_{(G,v)} = \langle\pi'\rangle_{(G,v)})
\end{aligned}$$

λ-function order: $\mathrm{type}(\langle\pi\rangle_g) \prec \deg_{\geqslant n}(\langle\pi\rangle_g) \prec \mathrm{ot}(\langle\pi\rangle_g) \prec (\langle\pi\rangle_g = \langle\pi'\rangle_g) \prec \mathrm{end}$

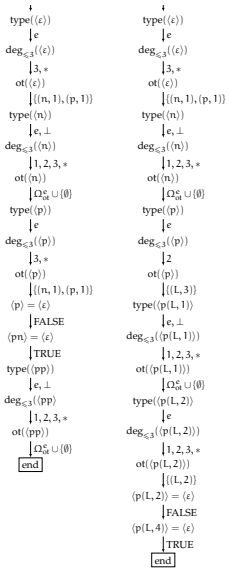
$$f(\langle\pi_1\rangle_{(G,v)}) \prec f(\langle\pi_2\rangle_{(G,v)}) \Leftrightarrow \pi_1 \prec \pi_2$$

Example: Double-Linked List

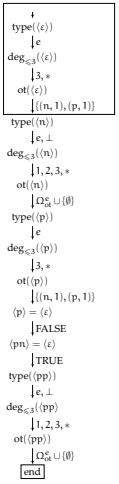


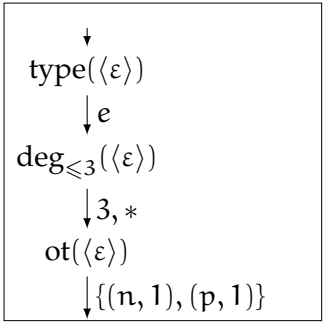
Initial vertices $\mathrm{ext}(3)$.

Example: Double-Linked List

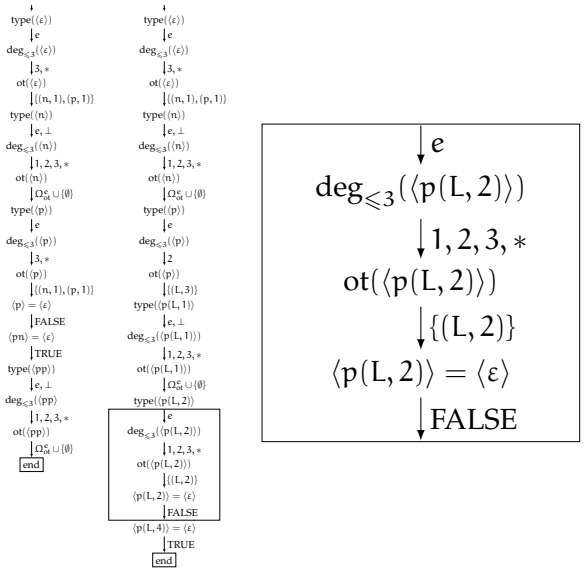


Example: Double-Linked List

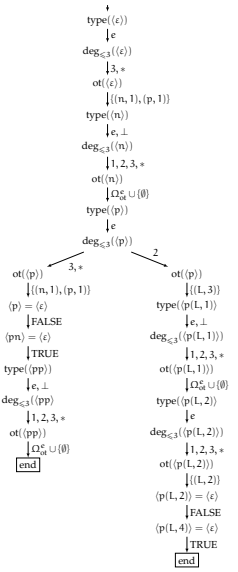


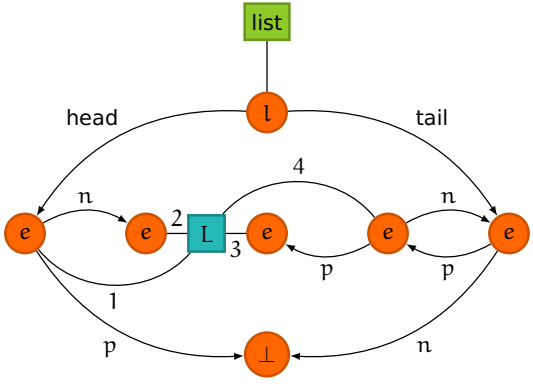


Example: Double-Linked List

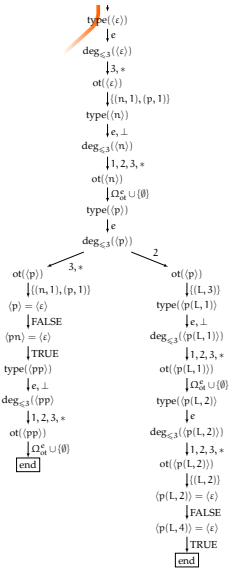


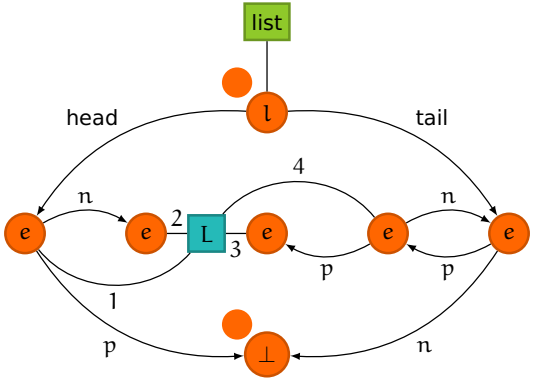
Example: Double-Linked List



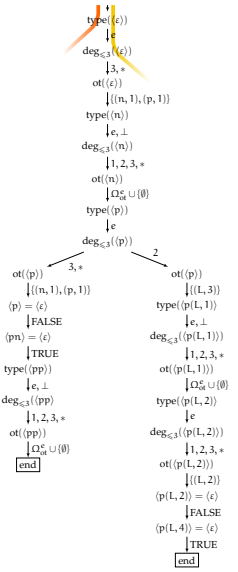


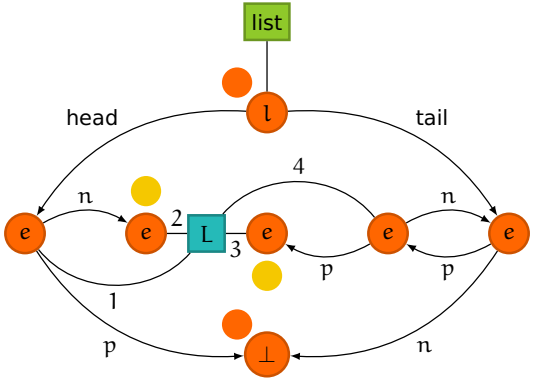
Example: Double-Linked List



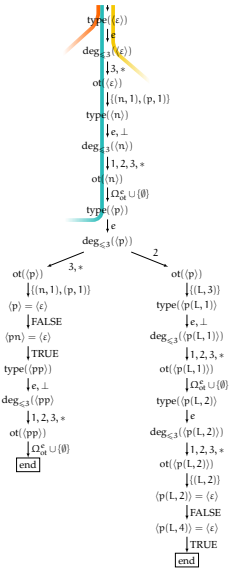


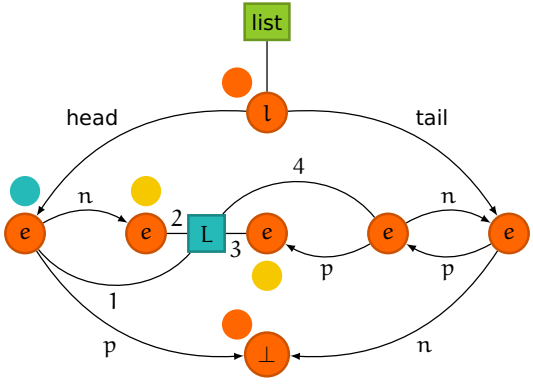
Example: Double-Linked List



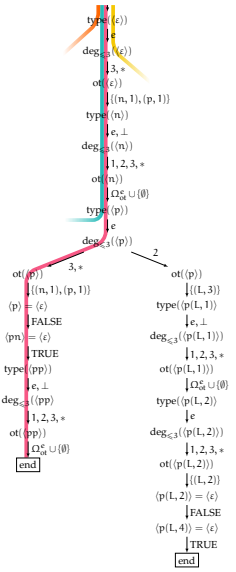


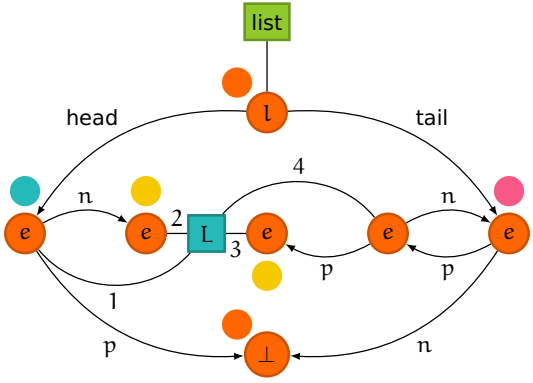
Example: Double-Linked List



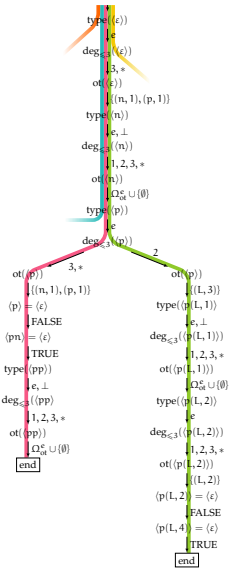


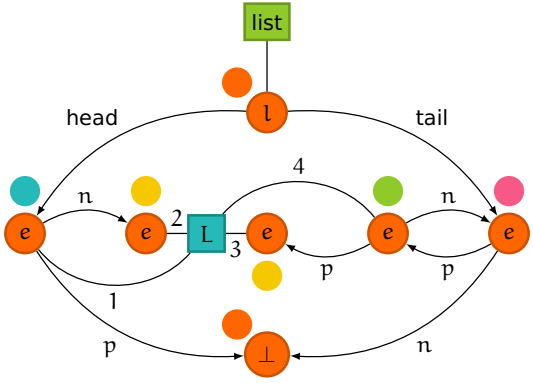
Example: Double-Linked List





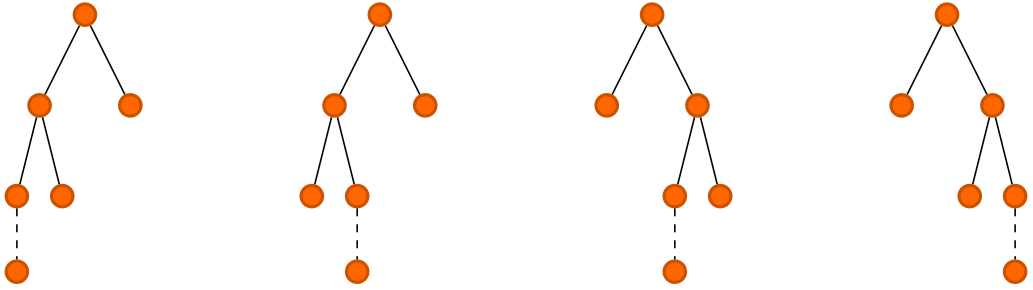
Example: Double-Linked List





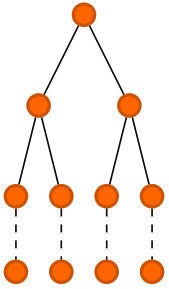
Time Complexity

Length of the longest run. Worst case:



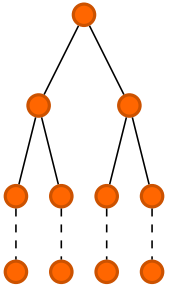
Time Complexity

Length of the longest run. Worst case:



Time Complexity

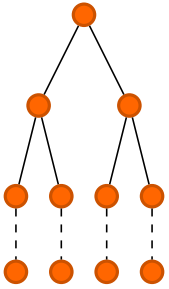
Length of the longest run. Worst case:



$$\mathcal{O}\left(|\mathcal{A}| \cdot |V_H| \cdot B_H \cdot |\,\mathrm{ext}_H\,|\right)$$

Time Complexity

Length of the longest run. Worst case:

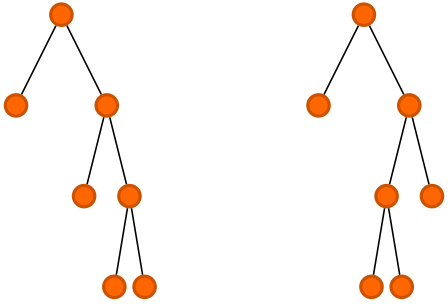


$$\mathcal{O}\left(|A| \cdot |V_H| \cdot B_H \cdot |\,\mathrm{ext}_H\,|\right)$$

Embedding detection: $\mathcal{O}(|V_G| \cdot |A| \cdot |V_H|^2 \cdot B_H \cdot |\,\mathrm{ext}_H\,|)$

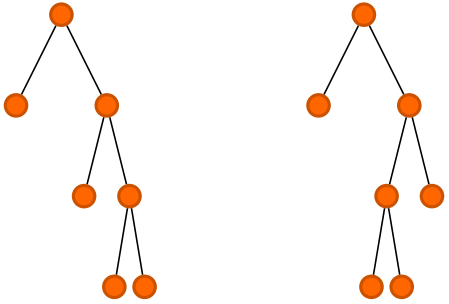
Time Complexity

Alternative approach: Fixed $|V_H|$.



Time Complexity

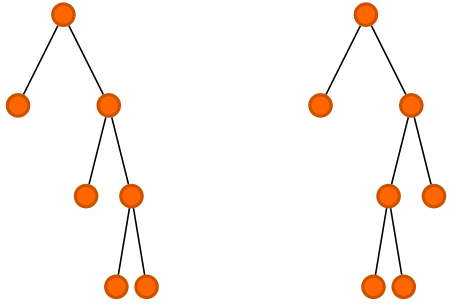
Alternative approach: Fixed $|V_H|$.



$$\sum_{i=0}^{h-1} B^i = \frac{B^h - 1}{B - 1}$$

Time Complexity

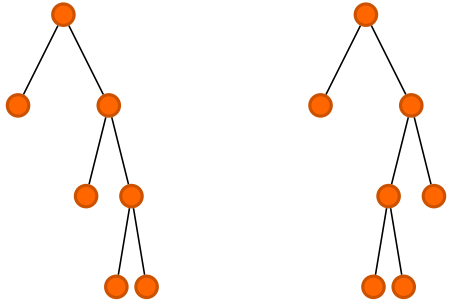
Alternative approach: Fixed $|V_H|$.



$$\sum_{i=0}^{h-1} B^i = \frac{B^h - 1}{B - 1} \approx \frac{B^{\frac{|V_H|}{B}} - 1}{B - 1}$$

Time Complexity

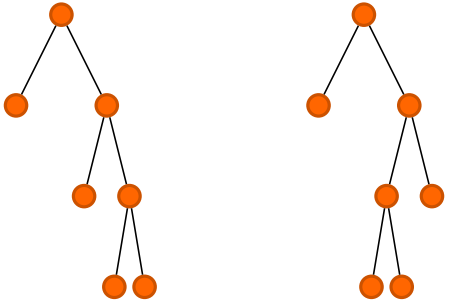
Alternative approach: Fixed $|V_H|$.



$$\sum_{i=0}^{h-1} B^i = \frac{B^h - 1}{B - 1} \approx \frac{B^{\frac{|V_H|}{B}} - 1}{B - 1} \leqslant B^{\frac{|V_H|}{B}} - 1$$

Time Complexity

Alternative approach: Fixed $|V_H|$.



$$\sum_{i=0}^{h-1} B^i = \frac{B^h - 1}{B - 1} \approx \frac{B^{\frac{|V_H|}{B}} - 1}{B - 1} \leqslant B^{\frac{|V_H|}{B}} - 1 \qquad \mathcal{O}\left(e^{\frac{|V_H|}{e}}\right) = \mathcal{O}\left(\sqrt[e]{e}^{|V_H|}\right) \approx \mathcal{O}\left(1.44^{|V_H|}\right)$$

Conclusion

Embedding detection is possible in polynomial time in many cases.

Future work:

- Implementation.
- Type hierarchy.
- More efficient path evaluation.
- Explore properties of function automata.