Minimizing Finite Automata with Graph Programs

Detlef Plump[1] Robin Suri[2] Ambuj Singh[3]

[1]The University of York

[2]Indian Institute of Technology Roorkee

[3]Indian Institute of Technology Kanpur

Automata Minimization

Problem

Input: A deterministic finite automaton $A = (Q, \Sigma, \delta, q_0, F)$.

Output: $A' = (Q', \Sigma, \delta', q_0', F')$ such that $\mathrm{L}(A) = \mathrm{L}(A')$ and $|Q'|$ is minimal.

Automata Minimization

Problem

Input: A deterministic finite automaton $A = (Q, \Sigma, \delta, q_0, F)$.

Output: $A' = (Q', \Sigma, \delta', q_0', F')$ such that $\mathrm{L}(A) = \mathrm{L}(A')$ and $|Q'|$ is minimal.

Definition

States p and q are *equivalent* if for all strings w,

$$\delta^*(p, w) \in F \text{ if and only if } \delta^*(q, w) \in F.$$

Algorithm of Hopcroft, Motwani and Ullman

Marking phase

Stage 1:

for each $p \in F$ and $q \in Q - F$ **do** mark the pair $\{p, q\}$

Stage 2:

repeat
  for each non-marked pair $\{p, q\}$ **do**
    for each $a \in \Sigma$ **do**
      if $\{\delta(p, a), \delta(q, a)\}$ is marked **then** mark $\{p, q\}$
until no new pair is marked

Algorithm of Hopcroft, Motwani and Ullman

Marking phase

Stage 1:
for each $p \in F$ and $q \in Q - F$ **do** mark the pair $\{p, q\}$

Stage 2:
repeat
  for each non-marked pair $\{p, q\}$ **do**
    for each $a \in \Sigma$ **do**
      if $\{\delta(p, a), \delta(q, a)\}$ is marked **then** mark $\{p, q\}$
until no new pair is marked

Lemma (HMU 07)

Two states p,q are equivalent if and only if the pair $\{p, q\}$ is not marked by the marking phase.

Algorithm of Hopcroft, Motwani and Ullman (cont'd)

<u>Merging phase</u>

Construct $A' = (Q', \Sigma, \delta', q_0', F')$ as follows:

- $Q' = \{[p] \mid p \in Q\}$, where $[p]$ is the equivalence class of p
- $q_0' = [q_0]$
- $\delta'([p], a) = [\delta(p, a)]$, for each $[p] \in Q'$ and $a \in \Sigma$
- $F' = \{[p] \in Q' \mid [p] \cap F \neq \emptyset\}$

Algorithm of Hopcroft, Motwani and Ullman (cont'd)

Merging phase

Construct $A' = (Q', \Sigma, \delta', q_0', F')$ as follows:

- $Q' = \{[p] \mid p \in Q\}$, where $[p]$ is the equivalence class of p
- $q_0' = [q_0]$
- $\delta'([p], a) = [\delta(p, a)]$, for each $[p] \in Q'$ and $a \in \Sigma$
- $F' = \{[p] \in Q' \mid [p] \cap F \neq \emptyset\}$

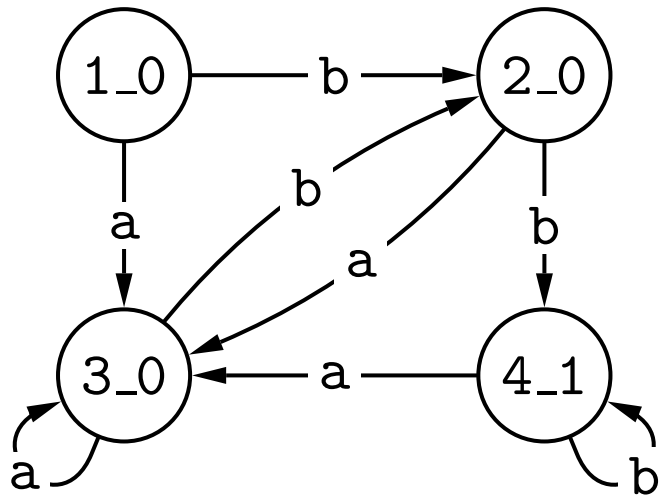
Theorem (HMU 07)

The automaton A' accepts the same language as A and is minimal.

Implementation in GP

Automata are given by their transition diagrams, where

- final/non-final states are labelled i_1 resp. i_0, with $i \in \mathbb{Z}$,
- only q_0 is labelled 1_b (with $b \in \{0, 1\}$),
- the symbols in Σ are character strings,
- all states are reachable from q_0.



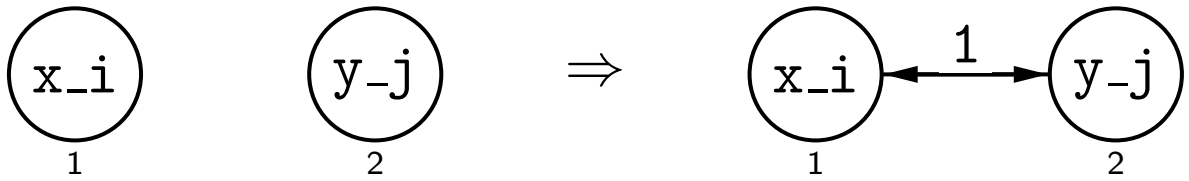
Minimization program

```
main    =  mark; merge; clean_up
```

```
mark    =  distinguish!; propagate!; equate!
merge   =  init; add_tag!; (choose; add_tag!)!;
           disconnect!; redirect!
clean_up  =  remove_edge!; remove_node!; untag!
```

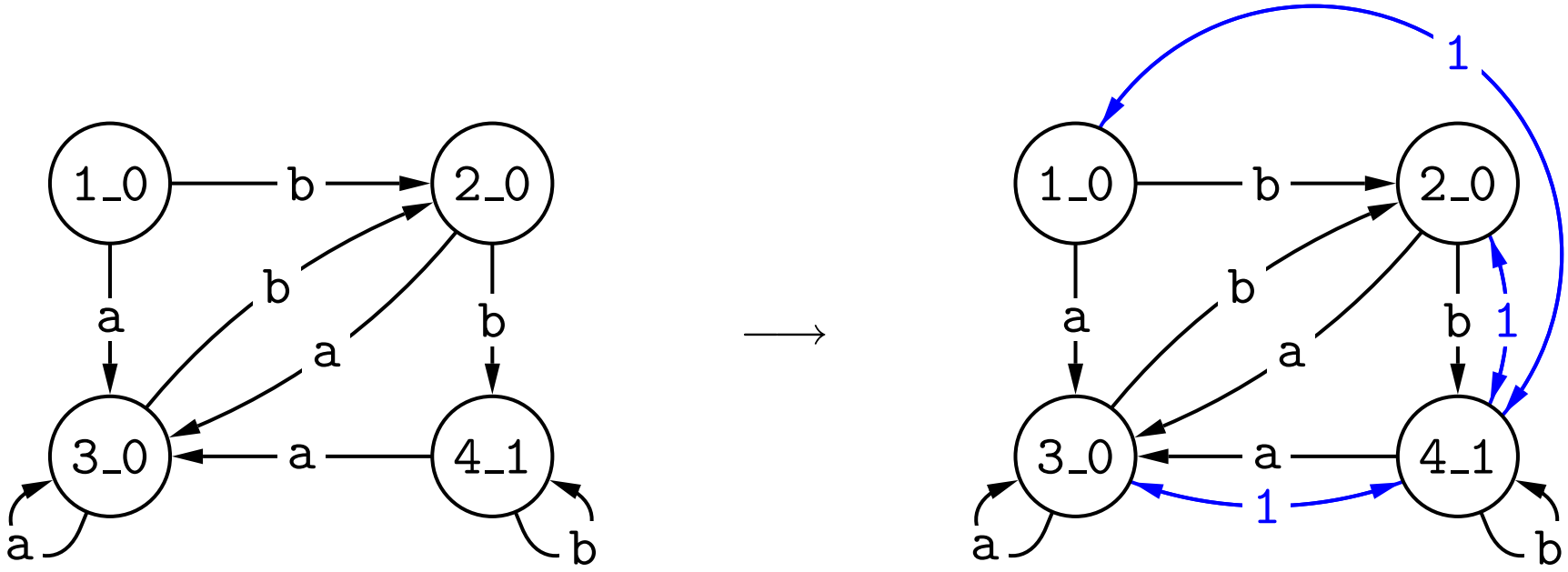
mark = distinguish!; propagate!; equate!

distinguish(x, y, i, j : int)



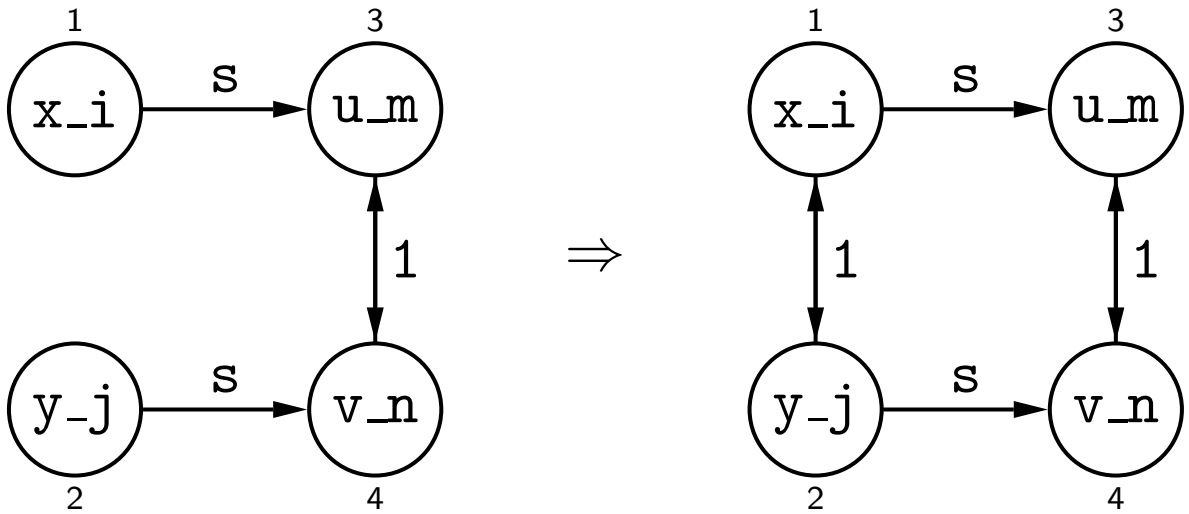
where $i \neq j$ and not edge(1, 2, 1)

Example: execution of `distinguish`!



mark = distinguish!; propagate!; equate!

propagate(x, y, u, v, i, j, m, n: int; s: str)

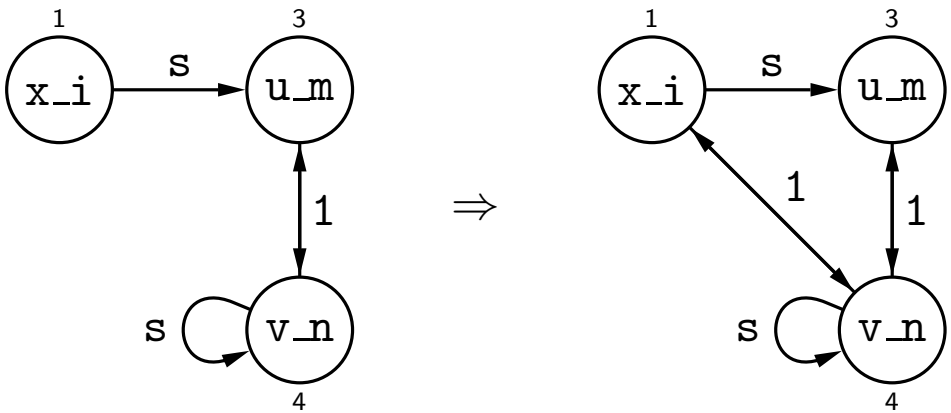


where not edge(1, 2, 1)

all matches

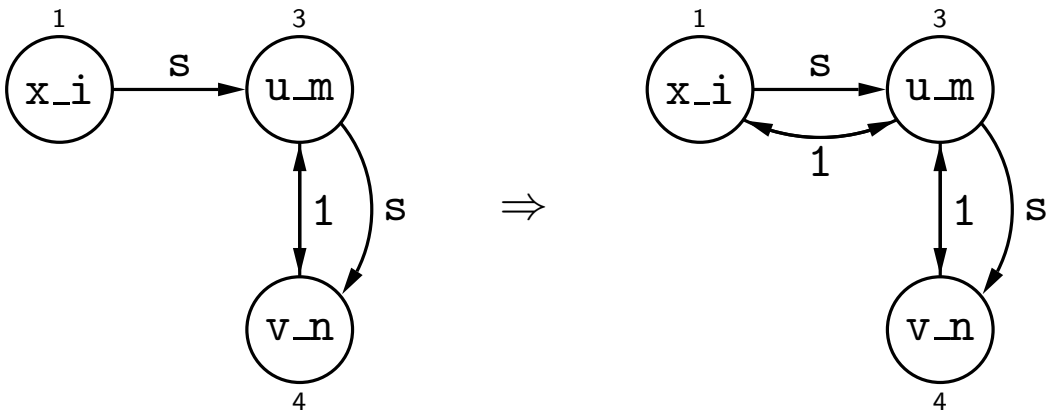
Quotients of propagate

propagate_2(x, u, v, i, m, n: int; s: str)



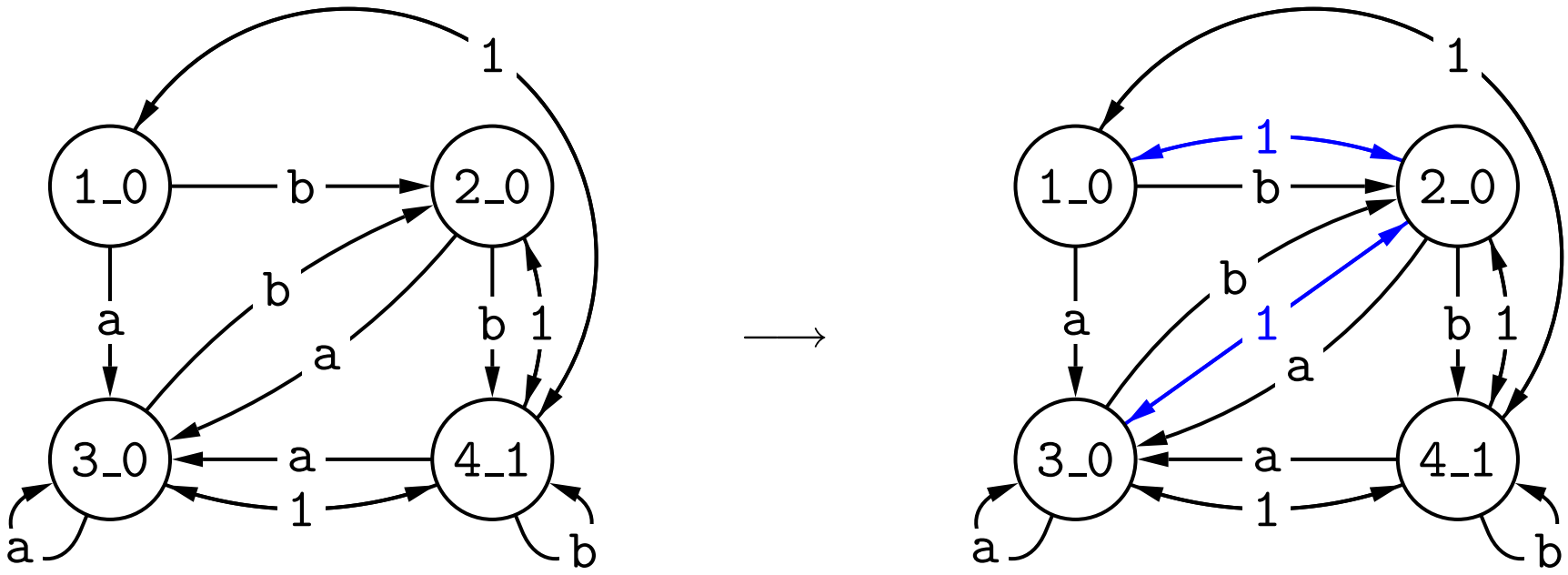
where not edge(1, 4, 1)

propagate_3(x, u, v, i, m, n: int; s: str)



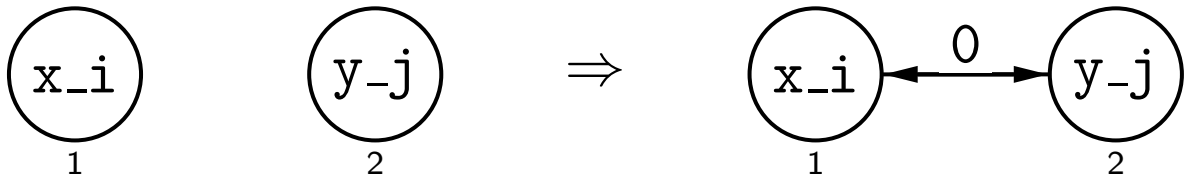
where not edge(1, 3, 1)

Example: execution of `propagate!`



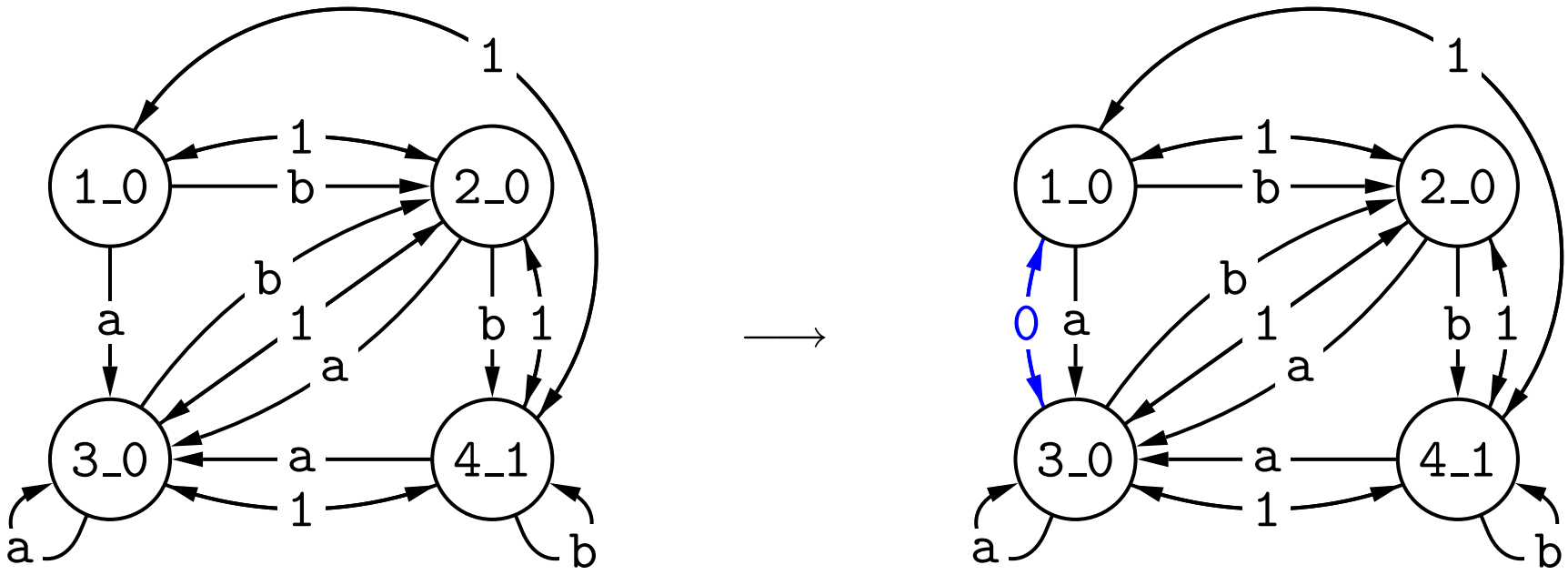
mark = distinguish!; propagate!; equate!

equate(x, y, i, j : int)



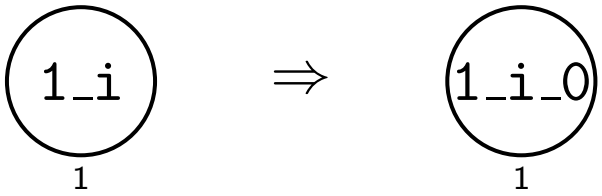
where not edge(1, 2, 1) and not edge(1, 2, 0)

Example: execution of `equate!`

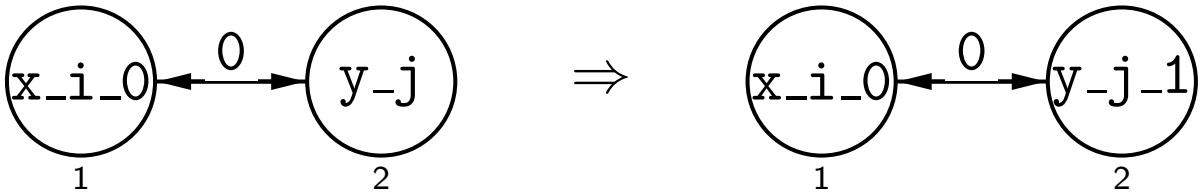


$$\texttt{merge} \;=\; \texttt{init};\ \texttt{add\_tag!};\ (\texttt{choose};\ \texttt{add\_tag!})!;\ \texttt{disconnect!};\ \texttt{redirect!}$$

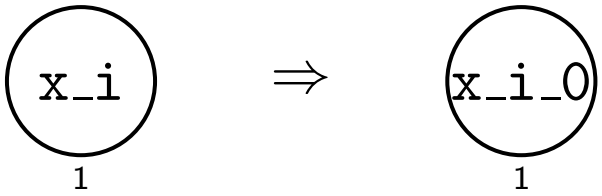
$\texttt{init}(\texttt{i}\colon \texttt{int})$



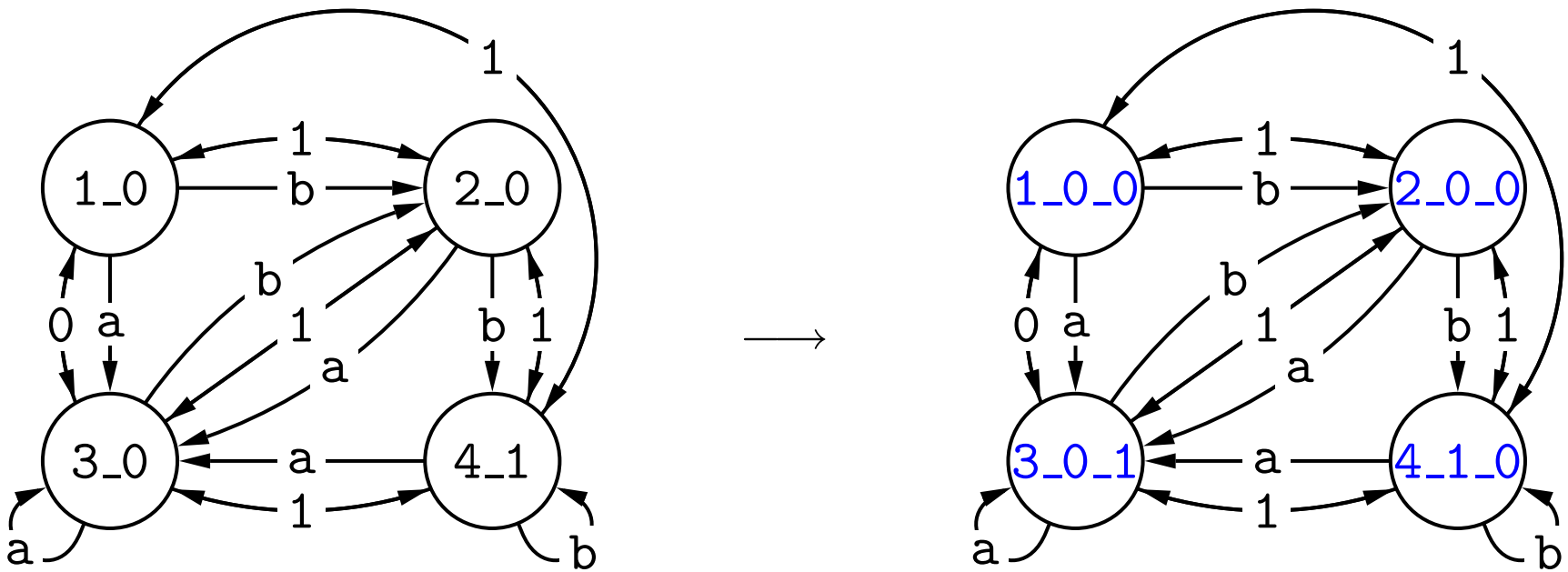
$\texttt{add\_tag}(\texttt{x}, \texttt{y}, \texttt{i}, \texttt{j}\colon \texttt{int})$



$\texttt{choose}(\texttt{x}, \texttt{i}\colon \texttt{int})$

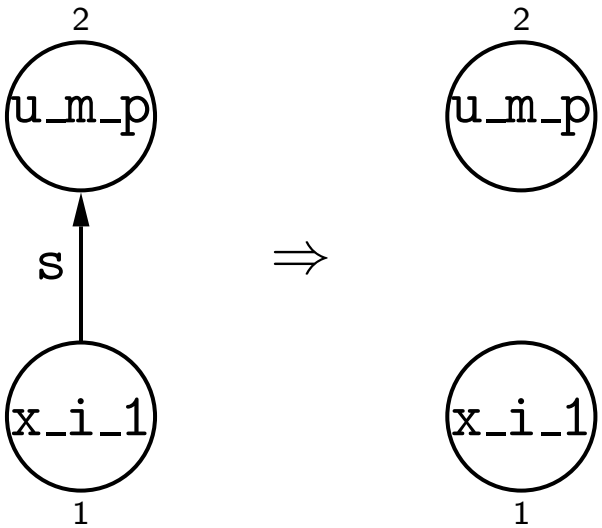


Example:
execution of `init`; `add_tag!`; (`choose`; `add_tag!`)!



$$\texttt{merge} \;=\; \texttt{init};\ \texttt{add\_tag!};\ (\texttt{choose};\ \texttt{add\_tag!})!;\ \texttt{disconnect!};\ \texttt{redirect!}$$

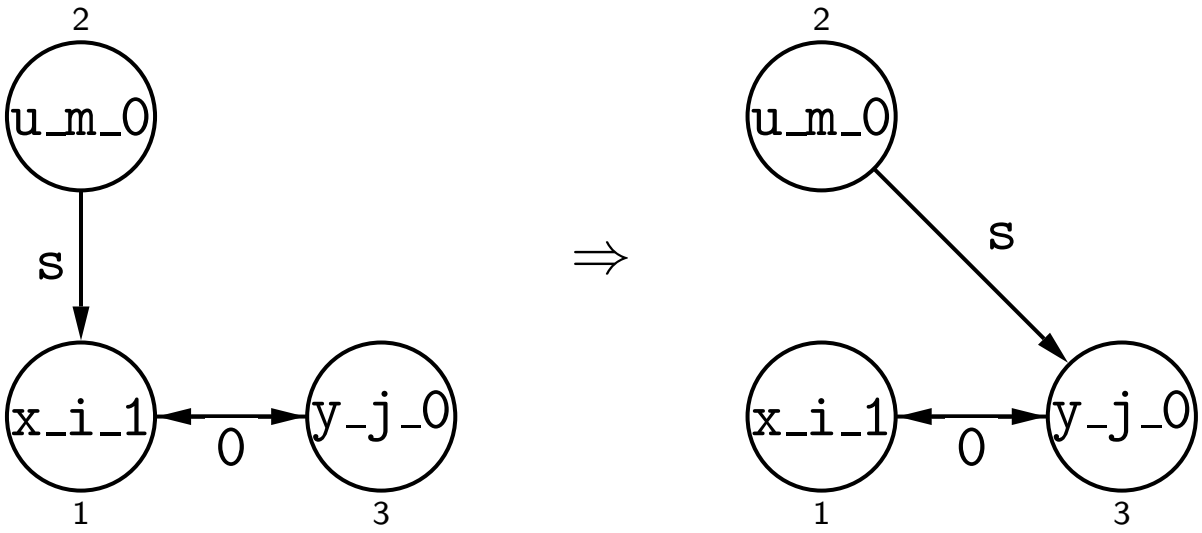
$\texttt{disconnect}(\texttt{x}, \texttt{u}, \texttt{i}, \texttt{m}, \texttt{p}\colon \texttt{int};\ \texttt{s}\colon \texttt{str})$



all matches

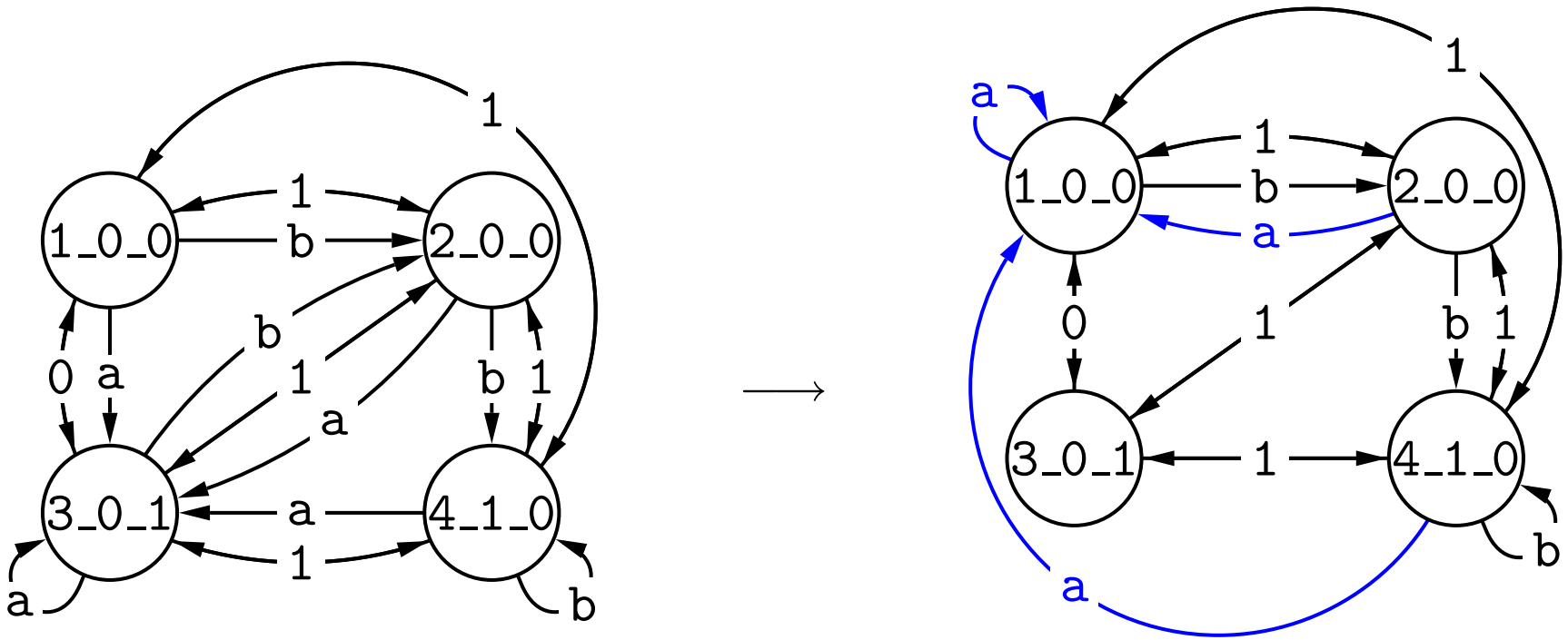
$$\texttt{merge} = \texttt{init}; \texttt{add\_tag}!; (\texttt{choose}; \texttt{add\_tag}!)!; \texttt{disconnect}!; \texttt{redirect}!$$

redirect(x, y, u, i, j, m: int; s: str)



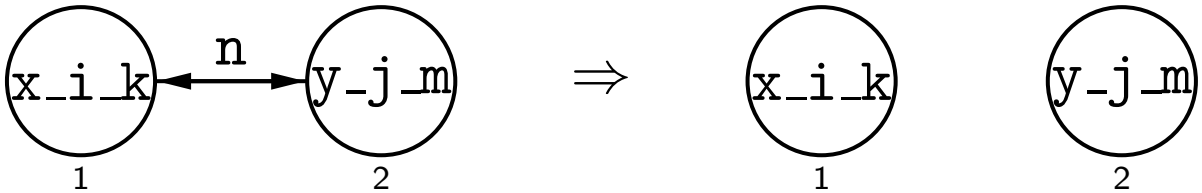
all matches

Example: execution of `disconnect!`; `redirect!`

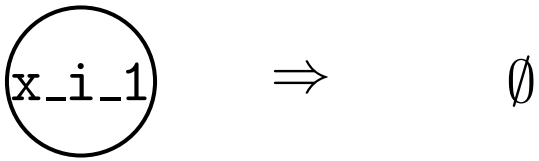


clean_up = remove_edge!; remove_node!; untag!

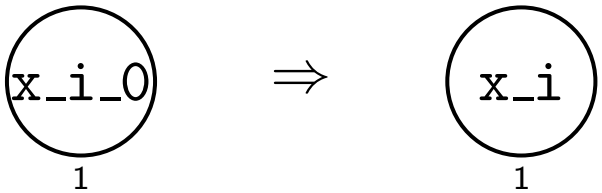
remove_edge(x, y, i, j, k, m, n: int)



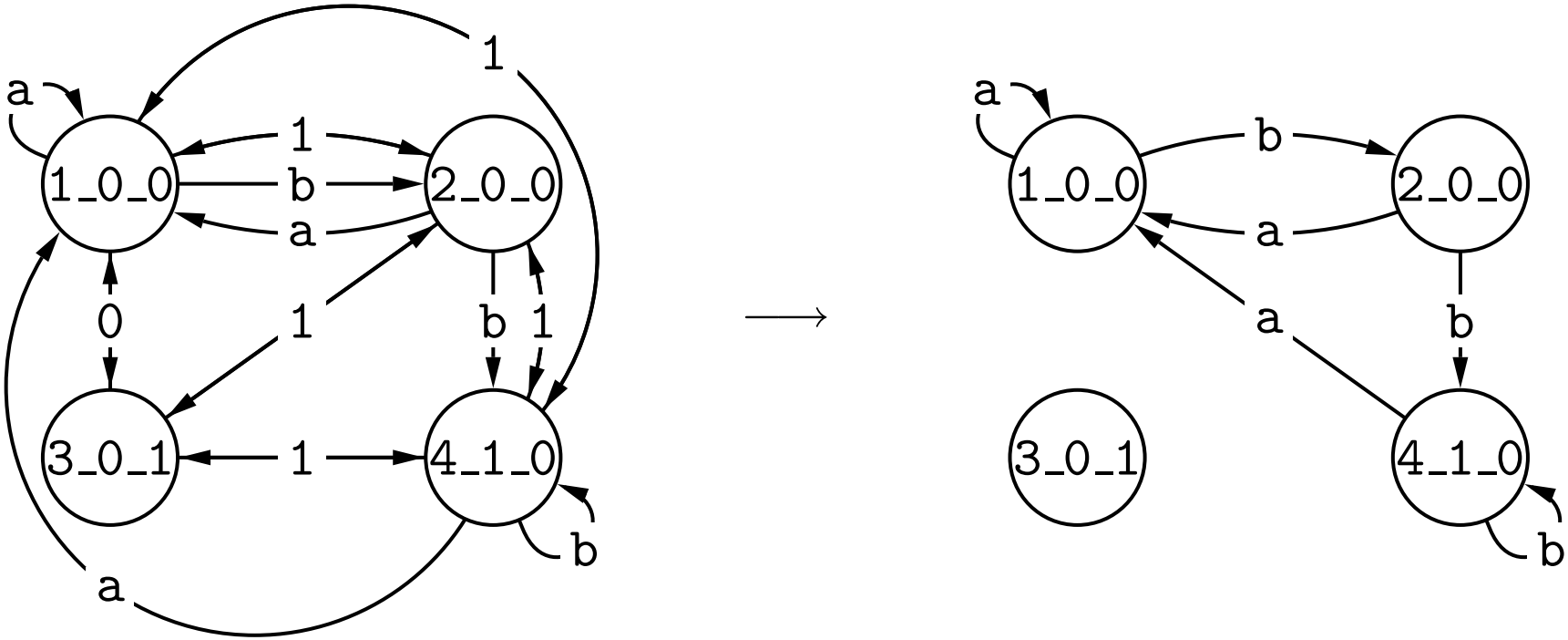
remove_node(x, i: int)



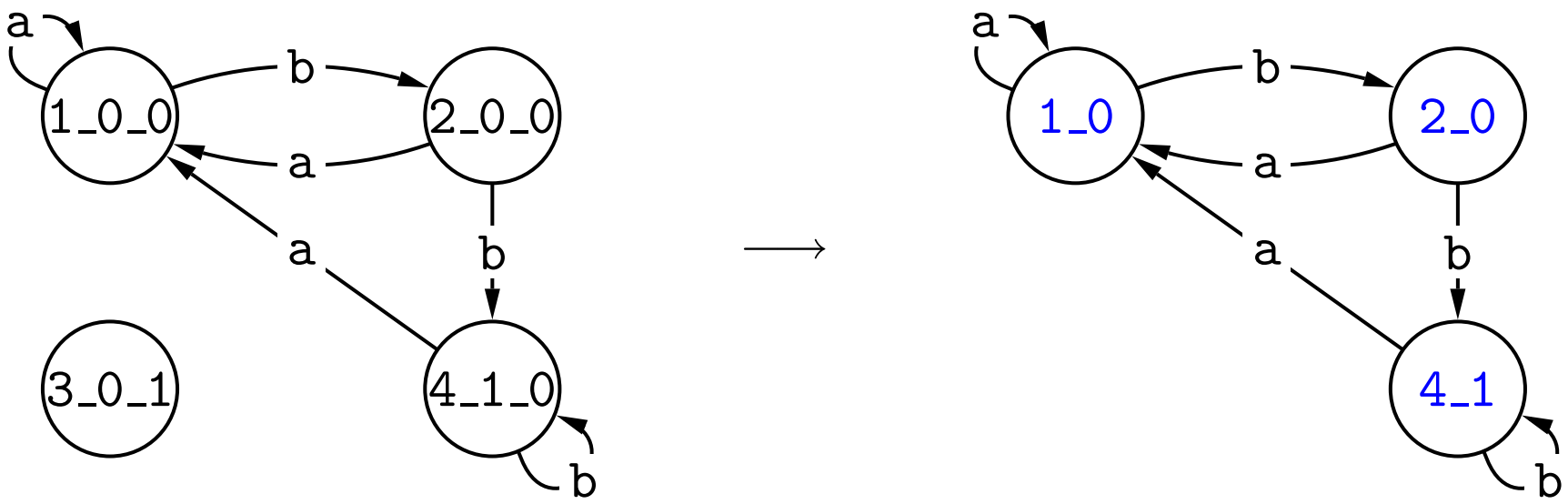
untag(x, i: int)



Example: execution of remove_edge!



Example: execution of `remove_node!`; `untag!`



Correctness of the implementation

```
main    =  mark; merge; clean_up
```

```
mark    =  distinguish!; propagate!; equate!
merge   =  init; add_tag!; (choose; add_tag!)!;
           disconnect!; redirect!
clean_up  =  remove_edge!; remove_node!; untag!
```

Proposition

For every input automaton A, the minimization program produces an automaton A' that is equivalent to A and minimal.

Time complexity

Proposition

The minimization program terminates after at most $\mathrm{O}(|\Sigma| \cdot |Q|^2)$ *rule applications.*

Time complexity

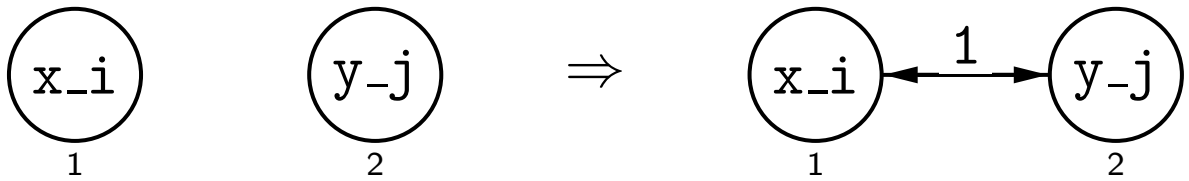
Proposition

The minimization program terminates after at most $\mathrm{O}(|\Sigma| \cdot |Q|^2)$ *rule applications.*

Note: We abstract from the complexity of rule matching.

Complexity of `distinguish`!

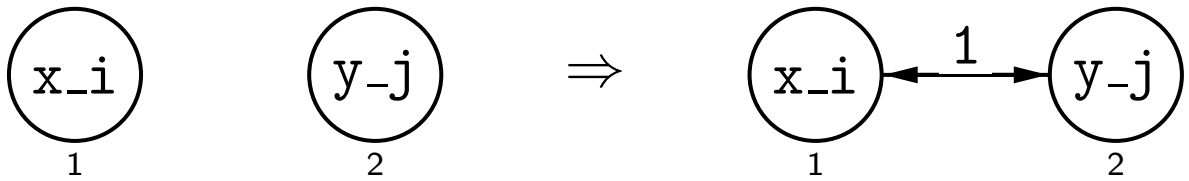
$\texttt{distinguish}(\texttt{x}, \texttt{y}, \texttt{i}, \texttt{j} : \texttt{int})$



$\texttt{where i} \neq \texttt{j and not edge}(1, 2, 1)$

Complexity of `distinguish`!

$\texttt{distinguish}(\texttt{x}, \texttt{y}, \texttt{i}, \texttt{j} : \texttt{int})$



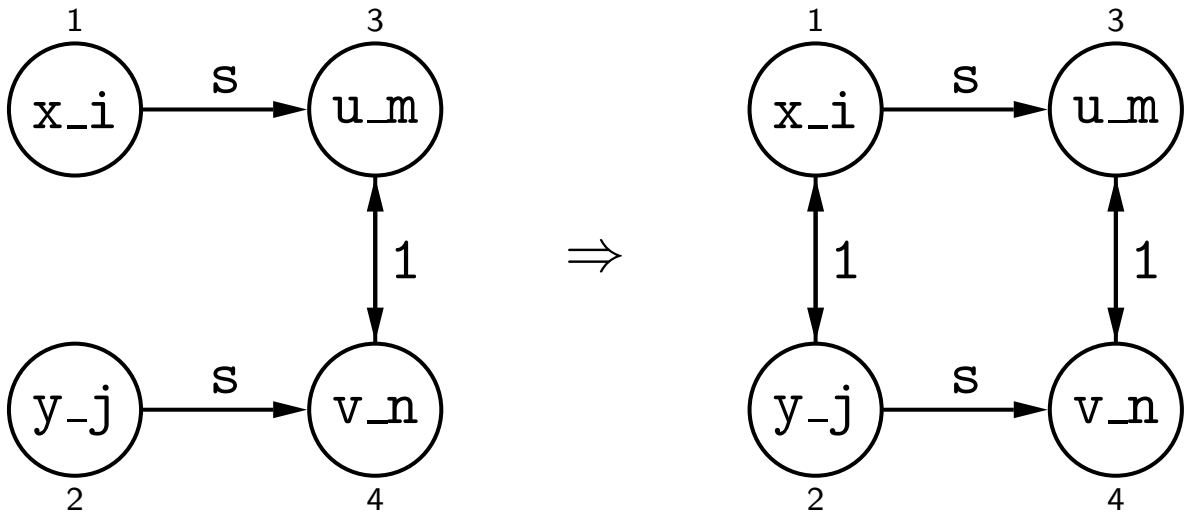
$\texttt{where i} \neq \texttt{j and not edge}(1, 2, 1)$

$$G \underset{\texttt{distinguish}}{\Longrightarrow} H \text{ implies } \#G > \#H, \text{ where}$$

$$\#X = |\{\{q, q'\} \mid q \neq q' \wedge \neg\text{edge}(q, q', 1)\}| < \frac{|Q|^2}{2}$$

Complexity of propagate!

propagate(x, y, u, v, i, j, m, n: int; s: str)

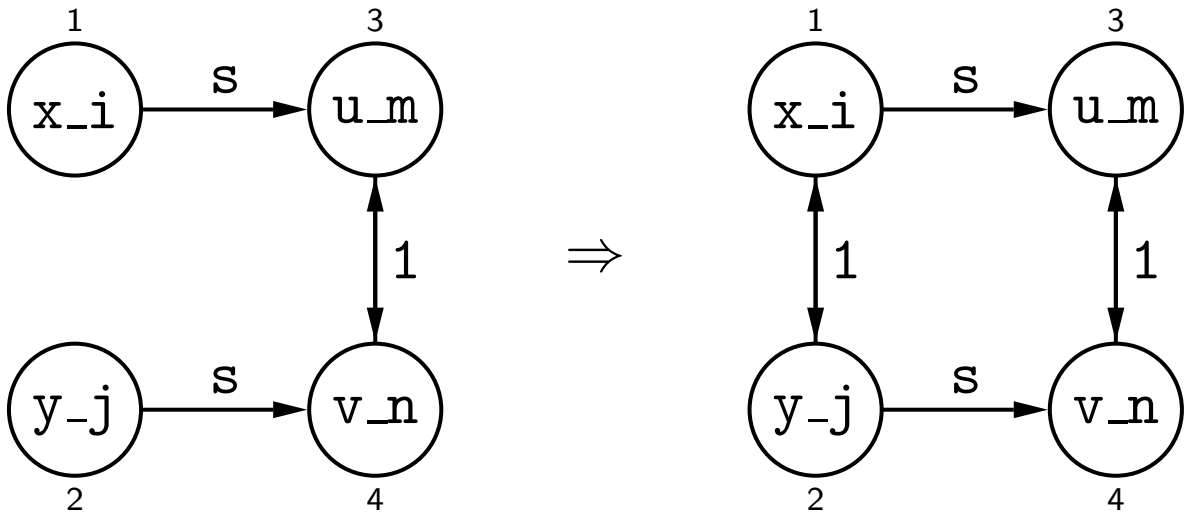


where not edge(1, 2, 1)

all matches

Complexity of propagate!

```
propagate(x, y, u, v, i, j, m, n: int; s: str)
```



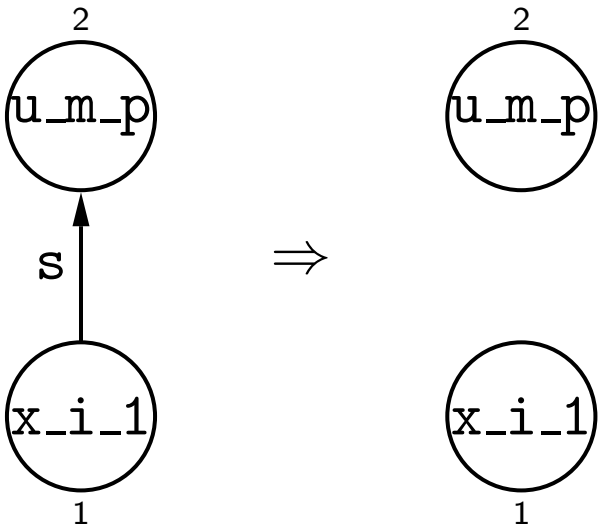

```
where not edge(1, 2, 1)
all matches
```

$G \underset{\texttt{propagate}}{\Longrightarrow} H$ implies $\#G > \#H$, where

$$\#X = |\{\{q, q'\} \mid q \neq q' \wedge \neg\mathrm{edge}(q, q', 1)\}| < \frac{|Q|^2}{2}$$

Complexity of disconnect!

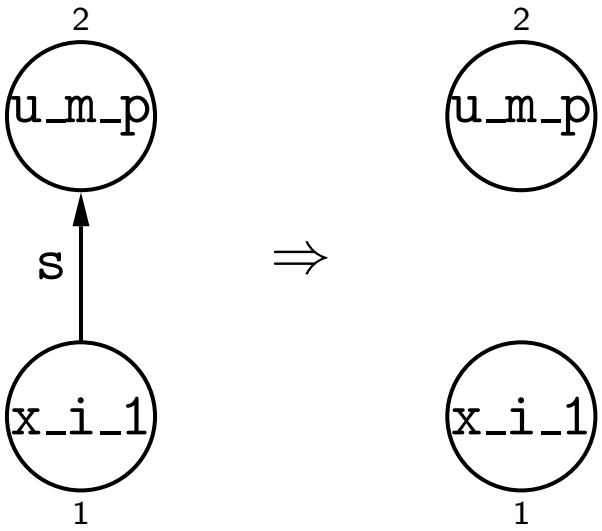
disconnect(x, u, i, m, p: int; s: str)



all matches

Complexity of `disconnect`!

`disconnect(x, u, i, m, p: int; s: str)`



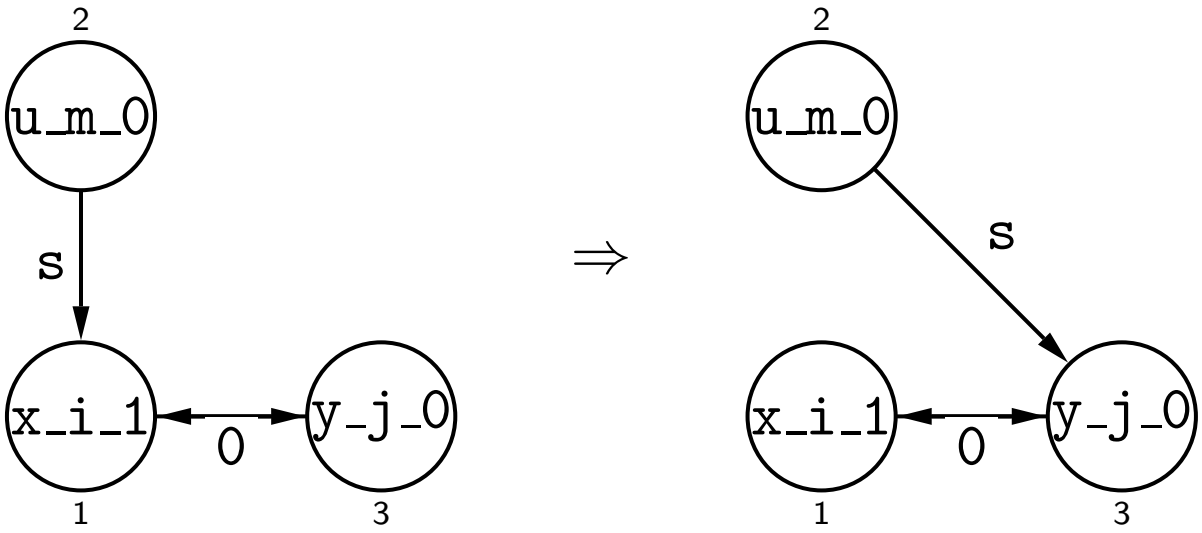
`all matches`

$G \underset{\texttt{disconnect}}{\Longrightarrow} H$ implies $\#G > \#H$, where

$$\#X = |\{e \in E_X \mid \mathrm{lab}(e) \in \Sigma\}| \leq |\Sigma| \cdot |Q|^2$$

Complexity of redirect!

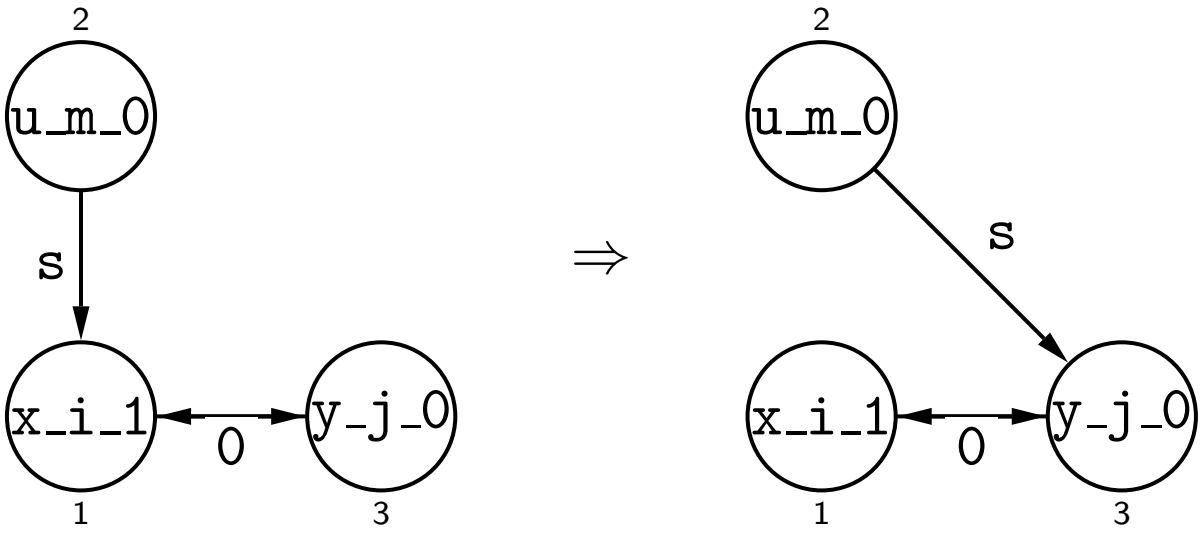
redirect(x, y, u, i, j, m: int; s: str)



all matches

Complexity of redirect!

redirect(x, y, u, i, j, m: int; s: str)



all matches

$G \underset{\texttt{redirect}}{\Longrightarrow} H$ implies $\#G > \#H$, where

$$\#X = |\{e \in E_X \mid \mathrm{lab}(e) \in \Sigma \wedge \mathrm{lab}(\mathrm{target}(e)) = x\_i\_1\}| \leq |\Sigma| \cdot |Q|^2$$

Conclusion

- Minimizing automata with rule-based, visual programming
- Direct manipulation of transition diagrams (augmented with auxiliary labels and edges); no need to implement transition functions, state tables, etc.
- Rule schemata and control constructs allow formal reasoning at a high level of abstraction
- Case study reveals convenience/necessity of certain GP constructs, e.g. 'all matches' attribute and ternary edge predicate

Possible extensions

- Implementing state merging by non-injective rules — requires GP extension
- Implementing more efficient minimization algorithms such as the n^2 algorithm of Hopcroft and Ullman or the $n\log n$ algorithm of Hopcroft
- Minimizing nondeterministic finite automata