# Complexity Analysis by Graph Rewriting

Martin Avanzini


Computational Logic
Faculty of Computer Science, University of Innsbruck

25. November 2009

# Outline

# Complexity Analysis by Term Rewriting

# Term Rewriting

- abstract model of computation rooted in equational theory
- underlies much of declarative programming

# Term Rewriting

- abstract model of computation rooted in equational theory
- underlies much of declarative programming

## Term Rewrite System (TRS)

TRS $\mathcal{R}_{\mathsf{D}}$ consists of the rewrite rules

① $\mathsf{D}(c) \to 0$

② $\mathsf{D}(x \times y) \to \mathsf{D}(x) \times y + x \times \mathsf{D}(y)$

③ $\mathsf{D}(x + y) \to \mathsf{D}(x) + \mathsf{D}(y)$

④ $\mathsf{D}(x - y) \to \mathsf{D}(x) - \mathsf{D}(y)$

# Term Rewriting

- abstract model of computation rooted in equational theory
- underlies much of declarative programming

## Term Rewrite System (TRS)

TRS $\mathcal{R}_{\mathsf{D}}$ consists of the rewrite rules

① $\mathsf{D}(\mathsf{c}) \to 0$

② $\mathsf{D}(x \times y) \to \mathsf{D}(x) \times y + x \times \mathsf{D}(y)$

③ $\mathsf{D}(x + y) \to \mathsf{D}(x) + \mathsf{D}(y)$

④ $\mathsf{D}(x - y) \to \mathsf{D}(x) - \mathsf{D}(y)$

## Rewriting

$$\mathsf{D}(\,\mathsf{c} + (\,\mathsf{c} \times \mathsf{c}\,))$$

# Term Rewriting

- abstract model of computation rooted in equational theory
- underlies much of declarative programming

## Term Rewrite System (TRS)

TRS $\mathcal{R}_{\mathsf{D}}$ consists of the rewrite rules

① $\mathsf{D}(\mathsf{c}) \to 0$

② $\mathsf{D}(x \times y) \to \mathsf{D}(x) \times y + x \times \mathsf{D}(y)$

③ $\mathsf{D}(x + y) \to \mathsf{D}(x) + \mathsf{D}(y)$

④ $\mathsf{D}(x - y) \to \mathsf{D}(x) - \mathsf{D}(y)$

## Rewriting

$$\mathsf{D}(\,\mathsf{c} + (\,\mathsf{c} \times \mathsf{c}\,))$$

# Term Rewriting

- abstract model of computation rooted in equational theory
- underlies much of declarative programming

## Term Rewrite System (TRS)

TRS $\mathcal{R}_{\mathsf{D}}$ consists of the rewrite rules

① $\mathsf{D}(\mathsf{c}) \to 0$

② $\mathsf{D}(x \times y) \to \mathsf{D}(x) \times y + x \times \mathsf{D}(y)$

③ $\mathsf{D}(x + y) \to \mathsf{D}(x) + \mathsf{D}(y)$

④ $\mathsf{D}(x - y) \to \mathsf{D}(x) - \mathsf{D}(y)$

## Rewriting

$$\mathsf{D}(\,\mathsf{c} + (\,\mathsf{c} \times \mathsf{c}\,)) \to_{\mathcal{R}_{\mathsf{D}}} \mathsf{D}(\,\mathsf{c}\,) + \mathsf{D}(\,\mathsf{c} \times \mathsf{c}\,)$$

# Term Rewriting

- abstract model of computation rooted in equational theory
- underlies much of declarative programming

## Term Rewrite System (TRS)

TRS $\mathcal{R}_{\mathsf{D}}$ consists of the rewrite rules

① $\mathsf{D}(c) \to 0$

② $\mathsf{D}(x \times y) \to \mathsf{D}(x) \times y + x \times \mathsf{D}(y)$

③ $\mathsf{D}(x + y) \to \mathsf{D}(x) + \mathsf{D}(y)$

④ $\mathsf{D}(x - y) \to \mathsf{D}(x) - \mathsf{D}(y)$

## Rewriting

$$\begin{aligned}\mathsf{D}(\,c + (\,c \times c\,)) &\to_{\mathcal{R}_{\mathsf{D}}} \mathsf{D}(\,c\,) + \mathsf{D}(\,c \times c\,) \\ &\to_{\mathcal{R}_{\mathsf{D}}} 0 + \mathsf{D}(c \times c)\end{aligned}$$

# Term Rewriting

- abstract model of computation rooted in equational theory
- underlies much of declarative programming

## Term Rewrite System (TRS)

TRS $\mathcal{R}_{\mathsf{D}}$ consists of the rewrite rules

① $\mathsf{D}(c) \to 0$

② $\mathsf{D}(x \times y) \to \mathsf{D}(x) \times y + x \times \mathsf{D}(y)$

③ $\mathsf{D}(x + y) \to \mathsf{D}(x) + \mathsf{D}(y)$

④ $\mathsf{D}(x - y) \to \mathsf{D}(x) - \mathsf{D}(y)$

## Rewriting

$$\begin{aligned}\mathsf{D}(\,c + (\,c \times c\,)) &\to_{\mathcal{R}_{\mathsf{D}}} \mathsf{D}(\,c\,) + \mathsf{D}(\,c \times c\,)\\ &\to_{\mathcal{R}_{\mathsf{D}}} 0 + \mathsf{D}(c \times c)\\ &\to_{\mathcal{R}_{\mathsf{D}}} 0 + \mathsf{D}(c) \times c + c \times \mathsf{D}(c)\end{aligned}$$

# Term Rewriting

- abstract model of computation rooted in equational theory
- underlies much of declarative programming

## Term Rewrite System (TRS)

TRS $\mathcal{R}_{\mathsf{D}}$ consists of the rewrite rules

① $\mathsf{D}(c) \to 0$

② $\mathsf{D}(x \times y) \to \mathsf{D}(x) \times y + x \times \mathsf{D}(y)$

③ $\mathsf{D}(x + y) \to \mathsf{D}(x) + \mathsf{D}(y)$

④ $\mathsf{D}(x - y) \to \mathsf{D}(x) - \mathsf{D}(y)$

## Rewriting

$$\begin{aligned}
\mathsf{D}(\,c + (\,c \times c\,)) &\to_{\mathcal{R}_{\mathsf{D}}} \mathsf{D}(\,c\,) + \mathsf{D}(\,c \times c\,) \\
&\to_{\mathcal{R}_{\mathsf{D}}} 0 + \mathsf{D}(c \times c) \\
&\to_{\mathcal{R}_{\mathsf{D}}} 0 + \mathsf{D}(c) \times c + c \times \mathsf{D}(c) \\
&\to^{2}_{\mathcal{R}_{\mathsf{D}}} 0 + 0 \times c + c \times 0
\end{aligned}$$

# Term Rewriting

- abstract model of computation rooted in equational theory
- underlies much of declarative programming

## Term Rewrite System (TRS)

TRS $\mathcal{R}_{\mathsf{D}}$ consists of the rewrite rules

① $\mathsf{D}(\mathsf{c}) \to 0$

② $\mathsf{D}(x \times y) \to \mathsf{D}(x) \times y + x \times \mathsf{D}(y)$

③ $\mathsf{D}(x + y) \to \mathsf{D}(x) + \mathsf{D}(y)$

④ $\mathsf{D}(x - y) \to \mathsf{D}(x) - \mathsf{D}(y)$

## Rewriting

$$\mathsf{D}(\,\mathsf{c} + (\,\mathsf{c} \times \mathsf{c}\,)) \to^{!}_{\mathcal{R}_{\mathsf{D}}} 0 + 0 \times \mathsf{c} + \mathsf{c} \times 0$$

# Term Rewriting

- abstract model of computation rooted in equational theory
- underlies much of declarative programming

## Term Rewrite System (TRS)

TRS $\mathcal{R}_{\mathsf{D}}$ consists of the rewrite rules

① $\mathsf{D}(\mathsf{c}) \to 0$

② $\mathsf{D}(x \times y) \to \mathsf{D}(x) \times y + x \times \mathsf{D}(y)$

③ $\mathsf{D}(x + y) \to \mathsf{D}(x) + \mathsf{D}(y)$

④ $\mathsf{D}(x - y) \to \mathsf{D}(x) - \mathsf{D}(y)$

## Computation

$$\mathsf{D}(\,\mathsf{c} + (\,\mathsf{c} \times \mathsf{c}\,)) = 0 + 0 \times \mathsf{c} + \mathsf{c} \times 0$$

- $\mathcal{R}_{\mathsf{D}}$ computes differentiation of arithmetical expressions

# Term Rewriting

- abstract model of computation rooted in equational theory
- underlies much of declarative programming

## Term Rewrite System (TRS)

TRS $\mathcal{R}_{\mathsf{D}}$ consists of the rewrite rules

① $\mathsf{D}(c) \to 0$

② $\mathsf{D}(x \times y) \to \mathsf{D}(x) \times y + x \times \mathsf{D}(y)$

③ $\mathsf{D}(x + y) \to \mathsf{D}(x) + \mathsf{D}(y)$

④ $\mathsf{D}(x - y) \to \mathsf{D}(x) - \mathsf{D}(y)$

## Computation

Let $\mathcal{R}$ denote a confluent and terminating TRS, and let $\mathsf{f}$ be a defined symbol. Then $\mathcal{R}$ computes the function $f$ defined by

$$f(s_1, \ldots, s_n) = t \qquad \Longleftrightarrow \qquad \mathsf{f}(s_1, \ldots, s_n) \to^{!}_{\mathcal{R}} = t$$

for values $s_1, \ldots, s_n$ and $t$. value . . . term build from constructor symbols

# Complexity Analysis by Rewriting

- natural complexity measure
  - ☞ number of rewrite steps

# Complexity Analysis by Rewriting

- natural complexity measure

  ☞ number of rewrite steps

- derivation length

$$\mathsf{dl}(t, \to) = \max\{\ell \mid \exists(t_1, \ldots, t_\ell).\ t \to t_1 \to \cdots \to t_\ell\}$$

$$\mathsf{dl}(n, T, \to) = \max\{\mathsf{dl}(t, \to) \mid t \in T \text{ and } |t| \leqslant n\}$$

# Complexity Analysis by Rewriting

- natural complexity measure

  ☞ number of rewrite steps

- derivation length

$$\mathsf{dl}(t, \to) = \max\{\ell \mid \exists(t_1, \ldots, t_\ell).\ t \to t_1 \to \cdots \to t_\ell\}$$

$$\mathsf{dl}(n, T, \to) = \max\{\mathsf{dl}(t, \to) \mid t \in T \text{ and } |t| \leqslant n\}$$

- runtime complexity

$$\mathsf{rc}_{\mathcal{R}}(n) = \mathsf{dl}(n, \mathcal{B}, \to_{\mathcal{R}})$$

where $\mathcal{B} := \{f(t_1, \ldots, t_n) \mid f \text{ defined and } t_i \text{ values}\}$ denotes the set of basic terms

$$\mathsf{D}(\mathsf{c}) \in \mathcal{B} \qquad \mathsf{D}(\mathsf{c} \times \mathsf{c} + \mathsf{c}) \in \mathcal{B} \qquad \mathsf{D}(\mathsf{D}(\mathsf{D}(\mathsf{c}))) \notin \mathcal{B}$$

# Complexity Analysis by Rewriting

- natural complexity measure

  ☞ number of rewrite steps

- derivation length

$$\mathsf{dl}(t, \to) = \max\{\ell \mid \exists(t_1, \ldots, t_\ell).\ t \to t_1 \to \cdots \to t_\ell\}$$

$$\mathsf{dl}(n, T, \to) = \max\{\mathsf{dl}(t, -$$

apply rule only if arguments are irreducible

- innermost runtime complexity

$$\mathsf{rc}^{\mathrm{i}}_{\mathcal{R}}(n) = \mathsf{dl}(n, \mathcal{B}, \xrightarrow{\mathrm{i}}_{\mathcal{R}})$$

where $\mathcal{B} := \{f(t_1, \ldots, t_n) \mid f \text{ defined and } t_i \text{ values}\}$ denotes the set of basic terms

$$\mathsf{D}(\mathsf{c}) \in \mathcal{B} \qquad \mathsf{D}(\mathsf{c} \times \mathsf{c} + \mathsf{c}) \in \mathcal{B} \qquad \mathsf{D}(\mathsf{D}(\mathsf{D}(\mathsf{c}))) \notin \mathcal{B}$$

# Automated Complexity Analysis by Rewriting

- direct techniques

  arctic interpretations, polynomial interpretations, matrix interpretations, match- and raise-bounds, polynomial path orders

- transformation techniques

  (weak) dependency pairs (+ argument filtering, dependency graph, reduction pairs, subterm criterion, usable rules, ...), dummy elimination, freezing, mirror, narrowing, reverse, rewriting right-hand sides, (restricted) semantic labeling (root labeling, finite semantic labeling), size change termination, split, strip, type introduction, uncurrying

  incomplete list

# Automated Complexity Analysis by Rewriting

- direct techniques

  arctic interpretations, polynomial interpretations, matrix interpretations, match- and raise-bounds, polynomial path orders

- transformation techniques

  (weak) dependency pairs (+ argument filtering, dependency graph, reduction pairs, subterm criterion, usable rules, ...), dummy elimination, freezing, mirror, narrowing, reverse, rewriting right-hand sides, (restricted) semantic labeling (root labeling, finite semantic labeling), size change termination, split, strip, type introduction, uncurrying

  incomplete list

- above techniques induce polynomial (innermost) runtime-complexity

# Automated Complexity Analysis by Rewriting

- direct techniques

  arctic interpretations, polynomial interpretations, matrix interpretations, match- and raise-bounds, polynomial path orders

- transformation techniques

  (weak) dependency pairs (+ argument filtering, dependency graph, reduction pairs, subterm criterion, usable rules, ...), dummy elimination, freezing, mirror, narrowing, reverse, rewriting right-hand sides, (restricted) semantic labeling (root labeling, finite semantic labeling), size change termination, split, strip, type introduction, uncurrying

  incomplete list

- above techniques induce polynomial (innermost) runtime-complexity
- do they assess polynomial computational complexity?

# This Talk

- runtime-complexity is a reasonable cost model for rewriting

# This Talk

- runtime-complexity is a reasonable cost model for rewriting

**Theorem**

*Let $\mathcal{R}$ be a confluent and terminating TRS.*
*If $\mathcal{R}$ admits polynomial bounded (innermost) runtime-complexity then the functions computed by $\mathcal{R}$ are polytime-computable.*

$$\mathrm{rc}^{\mathrm{i}}_{\mathcal{R}}(n) \leqslant n^k \Rightarrow f \text{ computed by } \mathcal{R} \text{ is computable in time } \mathrm{O}(n^{5*(k+1)})$$

# This Talk

- runtime-complexity is a reasonable cost model for rewriting

Example

① $\mathsf{D}(c) \to 0$

② $\mathsf{D}(x \times y) \to \mathsf{D}(x) \times y + x \times \mathsf{D}(y)$

③ $\mathsf{D}(x + y) \to \mathsf{D}(x) + \mathsf{D}(y)$

④ $\mathsf{D}(x - y) \to \mathsf{D}(x) - \mathsf{D}(y)$

- above TRS $\mathcal{R}_{\mathsf{D}}$ admits polynomial (linear) runtime-complexity

# This Talk

- runtime-complexity is a reasonable cost model for rewriting

## Example

① $\mathsf{D}(c) \to 0$

② $\mathsf{D}(x \times y) \to \mathsf{D}(x) \times y + x \times \mathsf{D}(y)$

③ $\mathsf{D}(x + y) \to \mathsf{D}(x) + \mathsf{D}(y)$

④ $\mathsf{D}(x - y) \to \mathsf{D}(x) - \mathsf{D}(y)$

- above TRS $\mathcal{R}_{\mathsf{D}}$ admits polynomial (linear) runtime-complexity
- symbolic differentiation as defined by $\mathcal{R}_{\mathsf{D}}$ is polytime computable

# Main Obstacle

- a single rewrite step may copy arbitrary large terms

  ☞ terms may grow exponential in size during reduction

# Main Obstacle

- a single rewrite step may copy arbitrary large terms

  ☞ terms may grow exponential in size during reduction

① $\mathsf{D}(c) \to 0$

② $\mathsf{D}(x \times y) \to \mathsf{D}(x) \times y + x \times \mathsf{D}(y)$

③ $\mathsf{D}(x + y) \to \mathsf{D}(x) + \mathsf{D}(y)$

④ $\mathsf{D}(x - y) \to \mathsf{D}(x) - \mathsf{D}(y)$

# Main Obstacle

- a single rewrite step may copy arbitrary large terms

  ☞ terms may grow exponential in size during reduction

① $\mathsf{D}(c) \to 0$

② $\mathsf{D}(x \times y) \to \mathsf{D}(x) \times y + x \times \mathsf{D}(y)$

③ $\mathsf{D}(x + y) \to \mathsf{D}(x) + \mathsf{D}(y)$

④ $\mathsf{D}(x - y) \to \mathsf{D}(x) - \mathsf{D}(y)$

$$\mathsf{D}(c) = 0$$

# Main Obstacle

- a single rewrite step may copy arbitrary large terms

  ☞ terms may grow exponential in size during reduction

① $\mathsf{D}(c) \to 0$

② $\mathsf{D}(x \times y) \to \mathsf{D}(x) \times y + x \times \mathsf{D}(y)$

③ $\mathsf{D}(x + y) \to \mathsf{D}(x) + \mathsf{D}(y)$

④ $\mathsf{D}(x - y) \to \mathsf{D}(x) - \mathsf{D}(y)$

$$\mathsf{D}(c) = 0$$
$$\mathsf{D}(c \times c) = 0 \times c + c \times 0$$

# Main Obstacle

- a single rewrite step may copy arbitrary large terms

  ☞ terms may grow exponential in size during reduction

① $\mathsf{D}(c) \to 0$

② $\mathsf{D}(x \times y) \to \mathsf{D}(x) \times y + x \times \mathsf{D}(y)$

③ $\mathsf{D}(x + y) \to \mathsf{D}(x) + \mathsf{D}(y)$

④ $\mathsf{D}(x - y) \to \mathsf{D}(x) - \mathsf{D}(y)$

$$\mathsf{D}(c) = 0$$
$$\mathsf{D}(c \times c) = 0 \times c + c \times 0$$
$$\mathsf{D}((c \times c) \times c) = (0 \times c + c \times 0) \times c + (c \times c) \times 0$$

# Main Obstacle

- a single rewrite step may copy arbitrary large terms

  ☞ terms may grow exponential in size during reduction

① $\mathsf{D}(c) \to 0$

② $\mathsf{D}(x \times y) \to \mathsf{D}(x) \times y + x \times \mathsf{D}(y)$

③ $\mathsf{D}(x + y) \to \mathsf{D}(x) + \mathsf{D}(y)$

④ $\mathsf{D}(x - y) \to \mathsf{D}(x) - \mathsf{D}(y)$

$$
\begin{aligned}
\mathsf{D}(c) &= 0\\
\mathsf{D}(c \times c) &= 0 \times c + c \times 0\\
\mathsf{D}((c \times c) \times c) &= (0 \times c + c \times 0) \times c + (c \times c) \times 0\\
\mathsf{D}((c \times c) \times (c \times c)) &= ((0 \times c + c \times 0) \times c + (c \times c) \times 0) \times (c \times c)\\
&\quad + (c \times c) \times ((0 \times c + c \times 0) \times c + (c \times c) \times 0)\\
&\vdots
\end{aligned}
$$

# Main Obstacle

- a single rewrite step may copy arbitrary large terms

  ☞ terms may grow exponential in size during reduction

## Solution

1. replace copying by sharing

   - encode terms as graphs
   - perform rewriting on graphs

# Main Obstacle

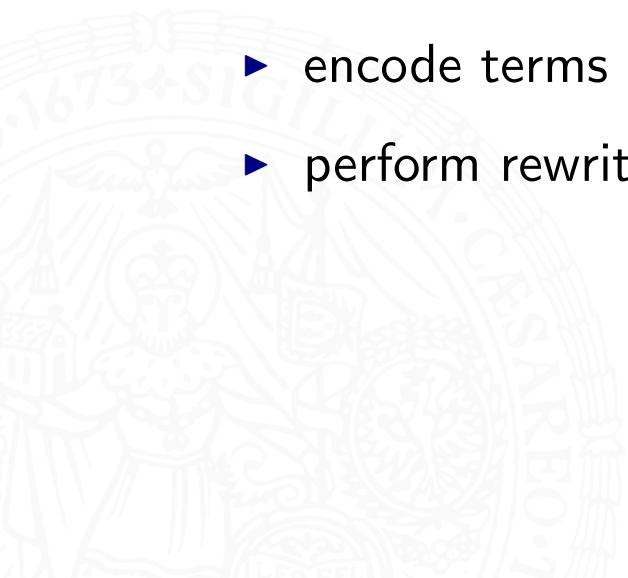
- a single rewrite step may copy arbitrary large terms

  ☞ terms may grow exponential in size during reduction

## Solution

1. replace copying by sharing — graph rewriting

   - encode terms as graphs
   - perform rewriting on graphs



# Main Obstacle

- a single rewrite step may copy arbitrary large terms

  ☞ terms may grow exponential in size during reduction

## Solution

1. replace copying by sharing — graph rewriting
   - encode terms as graphs
   - perform rewriting on graphs
2. implement graph rewriting on a Turing machine

# Graph Rewriting

# Graph Rewriting

## Labeled and Directed Graph

A labeled and directed graph $G = (\mathsf{V}_G, \mathsf{succ}_G, \mathsf{L}_G)$ over labels $\mathcal{L}$ is a structure such that

- $\mathsf{V}_G$ is a finite set of nodes (or vertices)
- $\mathsf{succ}_G : \mathsf{V}_G \to \mathsf{V}_G^*$ maps nodes $n$ to an ordered sequence of nodes, the successors of $n$
- $\mathsf{L}_G : \mathsf{V}_G \to \mathcal{L}$ labels nodes with elements from $\mathcal{L}$

# Graph Rewriting

## Labeled and Directed Graph

A labeled and directed graph $G = (\mathrm{V}_G, \mathrm{succ}_G, \mathrm{L}_G)$ over labels $\mathcal{L}$ is a structure such that

- $\mathrm{V}_G$ is a finite set of nodes (or vertices)
- $\mathrm{succ}_G : \mathrm{V}_G \to \mathrm{V}_G^*$ maps nodes $n$ to an ordered sequence of nodes, the successors of $n$
- $\mathrm{L}_G : \mathrm{V}_G \to \mathcal{L}$ labels nodes with elements from $\mathcal{L}$

# Graph Rewriting

## Labeled and Directed Graph

A labeled and directed graph $G = (\mathrm{V}_G, \mathrm{succ}_G, \mathrm{L}_G)$ over labels $\mathcal{L}$ is a structure such that

- $\mathrm{V}_G$ is a finite set of nodes (or vertices)
- $\mathrm{succ}_G : \mathrm{V}_G \to \mathrm{V}_G^*$ maps nodes $n$ to an ordered sequence of nodes, the successors of $n$
- $\mathrm{L}_G : \mathrm{V}_G \to \mathcal{L}$ labels nodes with elements from $\mathcal{L}$

# Graph Rewriting

## Labeled and Directed Graph

A labeled and directed graph $G = (\mathsf{V}_G, \mathsf{succ}_G, \mathsf{L}_G)$ over labels $\mathcal{L}$ is a structure such that

- $\mathsf{V}_G$ is a finite set of nodes (or vertices)
- $\mathsf{succ}_G : \mathsf{V}_G \to \mathsf{V}_G^*$ maps nodes $n$ to an ordered sequence of nodes, the successors of $n$
- $\mathsf{L}_G : \mathsf{V}_G \to \mathcal{L}$ labels nodes with elements from $\mathcal{L}$

# Graph Rewriting

## Labeled and Directed Graph

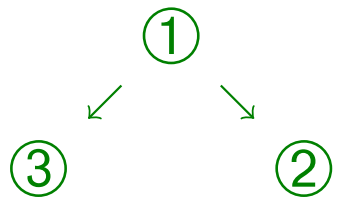
A labeled and directed graph $G = (\mathsf{V}_G, \mathsf{succ}_G, \mathsf{L}_G)$ over labels $\mathcal{L}$ is a structure such that

- $\mathsf{V}_G$ is a finite set of nodes (or vertices)
- $\mathsf{succ}_G : \mathsf{V}_G \to \mathsf{V}_G^*$ maps nodes $n$ to an ordered sequence of nodes, the successors of $n$
- $\mathsf{L}_G : \mathsf{V}_G \to \mathcal{L}$ labels nodes with elements from $\mathcal{L}$

## Example

Let $\mathsf{V} := \{1, 2, 3\}$ and $\mathsf{L} := \{1 \mapsto ①, 2 \mapsto ②, 3 \mapsto ③\}$.


$\mathsf{succ} := \{1 \mapsto [2, 3], 2 \mapsto [], 3 \mapsto []\}$




$\mathsf{succ} := \{1 \mapsto [3, 2], 2 \mapsto [], 3 \mapsto []\}$

# Graph Rewriting

## Term Graph

A term graph with respect to a signature $\mathcal{F}$ and variables $\mathcal{V}$ is an acyclic and rooted graph $S = (\mathsf{V}_S, \mathsf{succ}_S, \mathsf{L}_S)$ over labels $\mathcal{F} \cup \mathcal{V}$.

1. if $\mathsf{L}_S(n) = f$ is a $k$-ary function symbol then $\mathsf{succ}_S(n) = [n_1, \ldots, n_k]$
2. if $\mathsf{L}_S(n) = x$ is a variable then $\mathsf{succ}_S(n) = []$
3. variables are represented by unique nodes

# Graph Rewriting

## Term Graph

A term graph with respect to a signature $\mathcal{F}$ and variables $\mathcal{V}$ is an acyclic and rooted graph $S = (\mathsf{V}_S, \mathsf{succ}_S, \mathsf{L}_S)$ over labels $\mathcal{F} \cup \mathcal{V}$.

1. if $\mathsf{L}_S(n) = f$ is a $k$-ary function symbol then $\mathsf{succ}_S(n) = [n_1, \ldots, n_k]$
2. if $\mathsf{L}_S(n) = x$ is a variable then $\mathsf{succ}_S(n) = []$
3. variables are represented by unique nodes

# Graph Rewriting

## Term Graph

A term graph with respect to a signature $\mathcal{F}$ and variables $\mathcal{V}$ is an acyclic and rooted graph $S = (\mathsf{V}_S, \mathsf{succ}_S, \mathsf{L}_S)$ over labels $\mathcal{F} \cup \mathcal{V}$.

1. if $\mathsf{L}_S(n) = f$ is a $k$-ary function symbol then $\mathsf{succ}_S(n) = [n_1, \ldots, n_k]$
2. if $\mathsf{L}_S(n) = x$ is a variable then $\mathsf{succ}_S(n) = []$
3. variables are represented by unique nodes

# Graph Rewriting

## Term Graph

A term graph with respect to a signature $\mathcal{F}$ and variables $\mathcal{V}$ is an acyclic and rooted graph $S = (\mathsf{V}_S, \mathsf{succ}_S, \mathsf{L}_S)$ over labels $\mathcal{F} \cup \mathcal{V}$.

1. if $\mathsf{L}_S(n) = f$ is a $k$-ary function symbol then $\mathsf{succ}_S(n) = [n_1, \ldots, n_k]$
2. if $\mathsf{L}_S(n) = x$ is a variable then $\mathsf{succ}_S(n) = []$
3. variables are represented by unique nodes

# Graph Rewriting

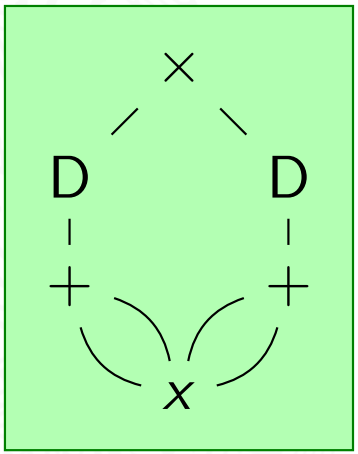
## Term Graph

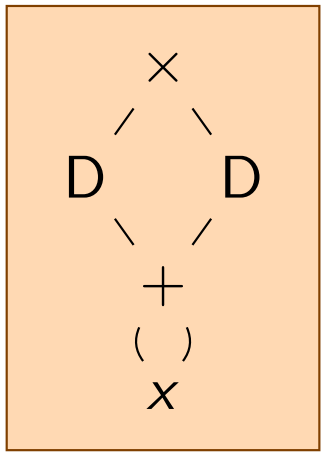
A term graph with respect to a signature $\mathcal{F}$ and variables $\mathcal{V}$ is an acyclic and rooted graph $S = (\mathsf{V}_S, \mathsf{succ}_S, \mathsf{L}_S)$ over labels $\mathcal{F} \cup \mathcal{V}$.

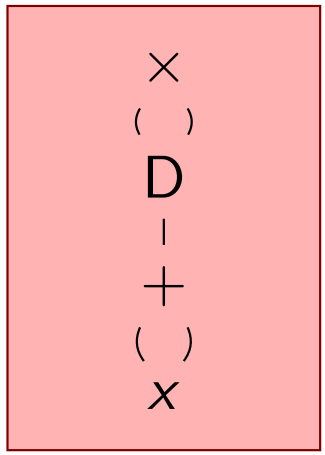
1. if $\mathsf{L}_S(n) = f$ is a $k$-ary function symbol then $\mathsf{succ}_S(n) = [n_1, \ldots, n_k]$
2. if $\mathsf{L}_S(n) = x$ is a variable then $\mathsf{succ}_S(n) = []$
3. variables are represented by unique nodes

## Example

Let $t = \mathsf{D}(x + x) \times \mathsf{D}(x + x)$. It is represented by



# Graph Rewriting

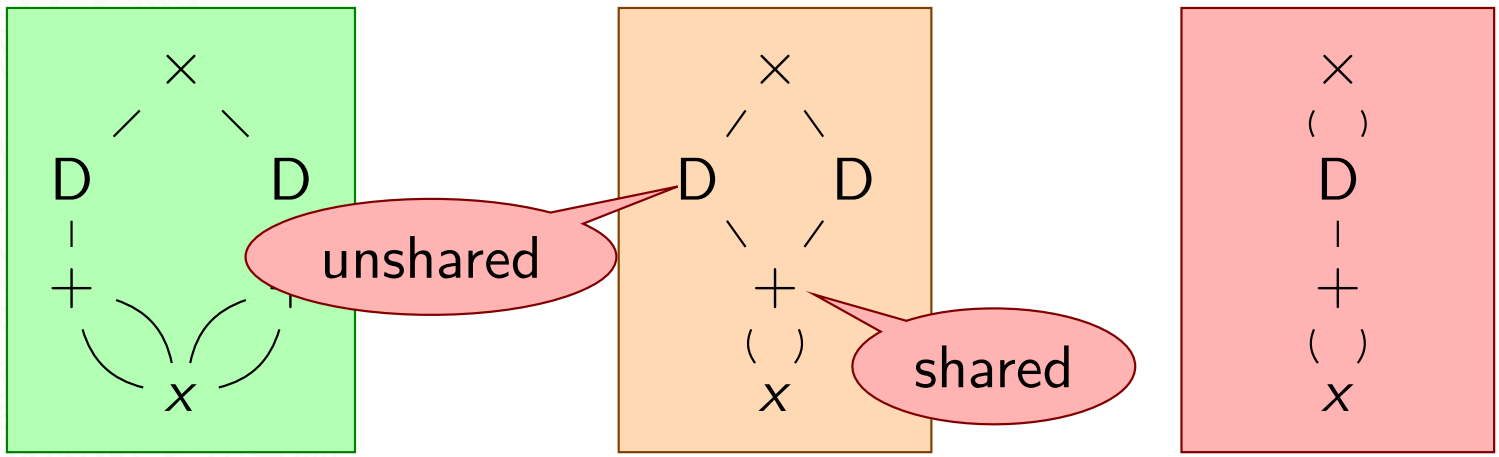
## Term Graph

A term graph with respect to a signature $\mathcal{F}$ and variables $\mathcal{V}$ is an acyclic and rooted graph $S = (\mathsf{V}_S, \mathsf{succ}_S, \mathsf{L}_S)$ over labels $\mathcal{F} \cup \mathcal{V}$.

1. if $\mathsf{L}_S(n) = f$ is a $k$-ary function symbol then $\mathsf{succ}_S(n) = [n_1, \ldots, n_k]$
2. if $\mathsf{L}_S(n) = x$ is a variable then $\mathsf{succ}_S(n) = []$
3. variables are represented by unique nodes

## Example

Let $t = \mathsf{D}(x + x) \times \mathsf{D}(x + x)$. It is represented by



# Graph Rewriting

## Term Graph

A term graph with res … is an acyclic and rooted graph $S =$ …

1. if $\mathsf{L}_S(n) = f$ is a … $[n_1, \ldots, n_k]$
2. if $\mathsf{L}_S(n) = x$ is a …

…que …odes

**minimally sharing**

- only variable nodes are shared

**$(\mathcal{R}_\mathsf{D})$ normal-form sharing**

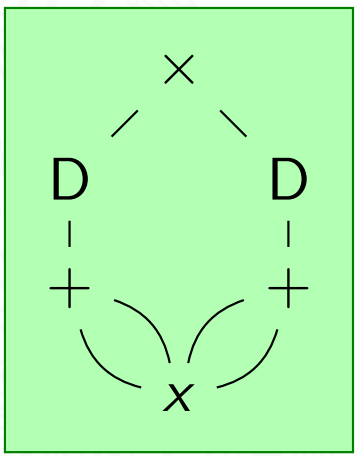
- nodes representing normal-forms maximally shared
- redex nodes unshared

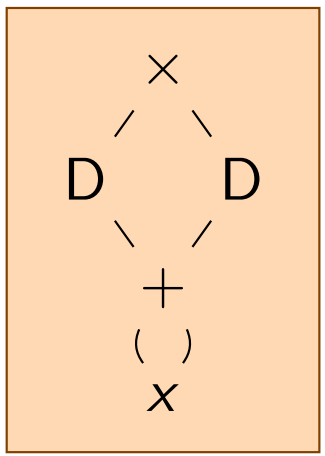
**maximally sharing**

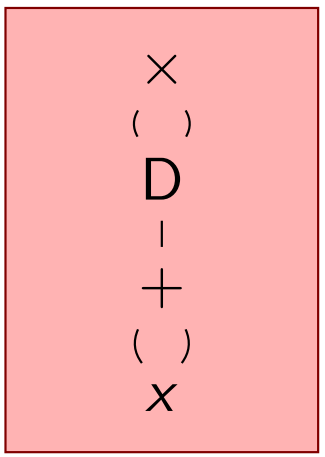
- sharing where possible

Example

Let $t = \mathsf{D}(x + x) \times \mathsf{D}(x + x)$. It is represented by



# Graph Rewriting

## Term Graph

A term graph with res … is an acyclic and rooted graph $S = $ …

1. if $\mathsf{L}_S(n) = f$ is a … $[n_1, \ldots, n_k]$
2. if $\mathsf{L}_S(n) = x$ is a …

…que …odes

**minimally sharing**

- only variable nodes are shared

**$(\mathcal{R}_\mathsf{D})$ normal-form sharing**

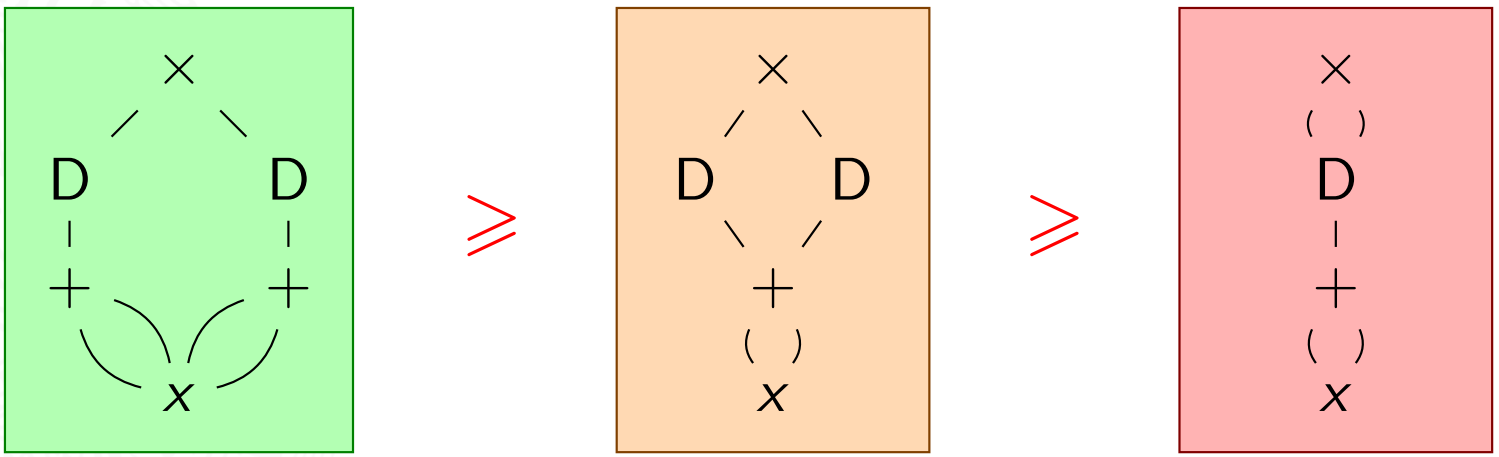
- nodes representing normal-forms maximally shared
- redex nodes unshared

**maximally sharing**

- sharing where possible

Example

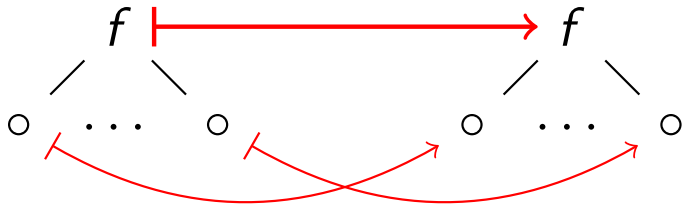
Let $t = \mathsf{D}(x + x) \times \mathsf{D}(x + x)$. It is represented by



# Graph Rewriting

## Term Graph Morphism

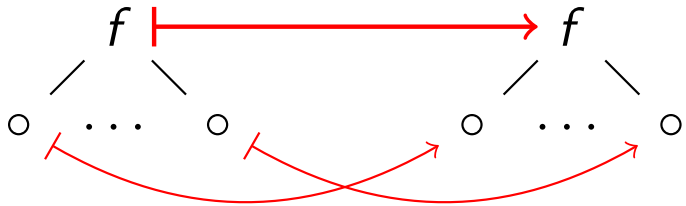
- a term graph morphism $m\colon S \to T$ is a function $m\colon \mathsf{V}_S \to \mathsf{V}_T$
  1. sending the root of $S$ to the root of $T$
  2. preserving labels and successors for all non-variable nodes



# Graph Rewriting

## Term Graph Morphism

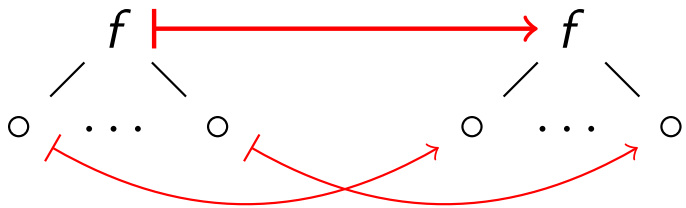
- a term graph morphism $m\colon S \to T$ is a function $m\colon \mathsf{V}_S \to \mathsf{V}_T$
  1. sending the root of $S$ to the root of $T$
  2. preserving labels and successors for all non-variable nodes



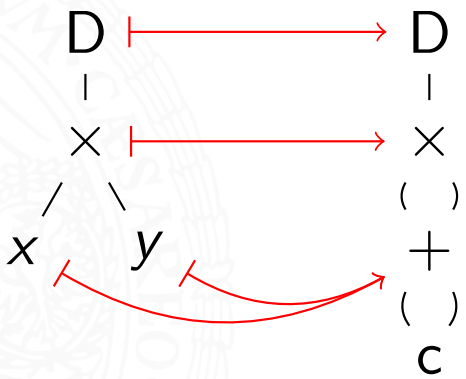
# Graph Rewriting

## Term Graph Morphism

- a term graph morphism $m\colon S \to T$ is a function $m\colon \mathsf{V}_S \to \mathsf{V}_T$
  1. sending the root of $S$ to the root of $T$
  2. preserving labels and successors for all non-variable nodes



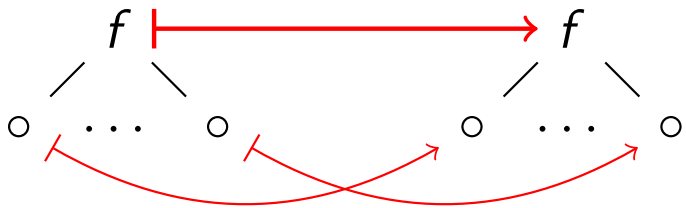

## Example



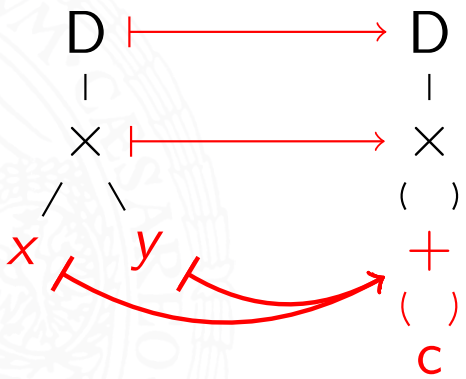
# Graph Rewriting

## Term Graph Morphism

- a term graph morphism $m\colon S \to T$ is a function $m\colon \mathsf{V}_S \to \mathsf{V}_T$
  1. sending the root of $S$ to the root of $T$
  2. preserving labels and successors for all non-variable nodes



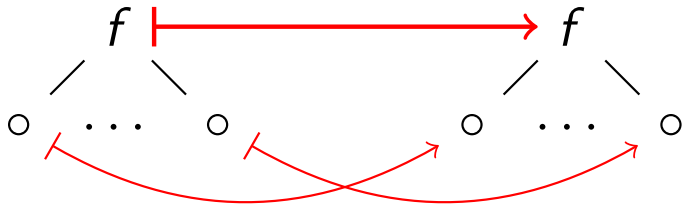

## Example



# Graph Rewriting

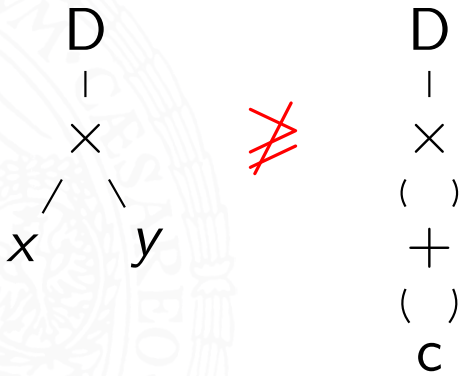
## Term Graph Morphism

- a term graph morphism $m\colon S \to T$ is a function $m\colon \mathsf{V}_S \to \mathsf{V}_T$
  1. sending the root of $S$ to the root of $T$
  2. preserving labels and successors for all non-variable nodes




- $S \geqslant_m T$ ($S \geqslant T$) if there exists a morphism $m\colon S \to T$ preserving also labels and successors of variable nodes

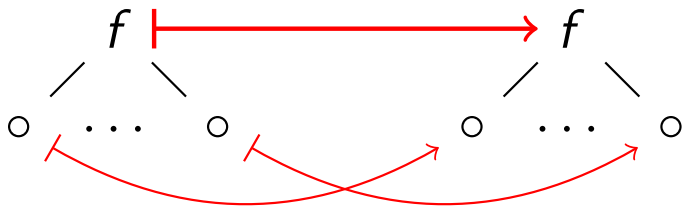
## Example



# Graph Rewriting

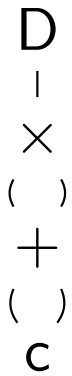
## Term Graph Morphism

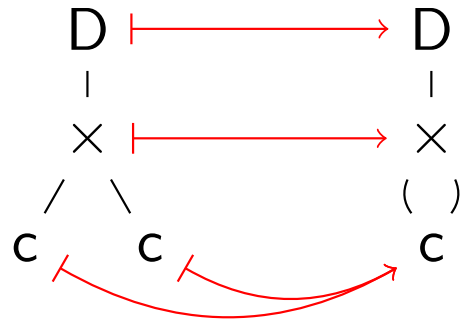
- a term graph morphism $m\colon S \to T$ is a function $m\colon \mathsf{V}_S \to \mathsf{V}_T$
  1. sending the root of $S$ to the root of $T$
  2. preserving labels and successors for all non-variable nodes




- $S \geqslant_m T$ ($S \geqslant T$) if there exists a morphism $m\colon S \to T$ preserving also labels and successors of variable nodes

## Example

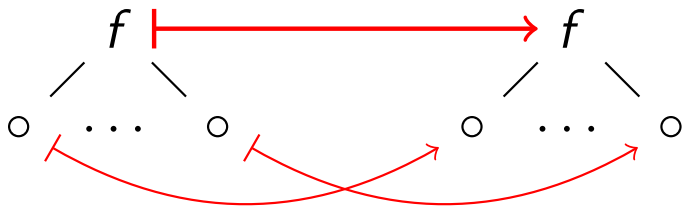



# Graph Rewriting

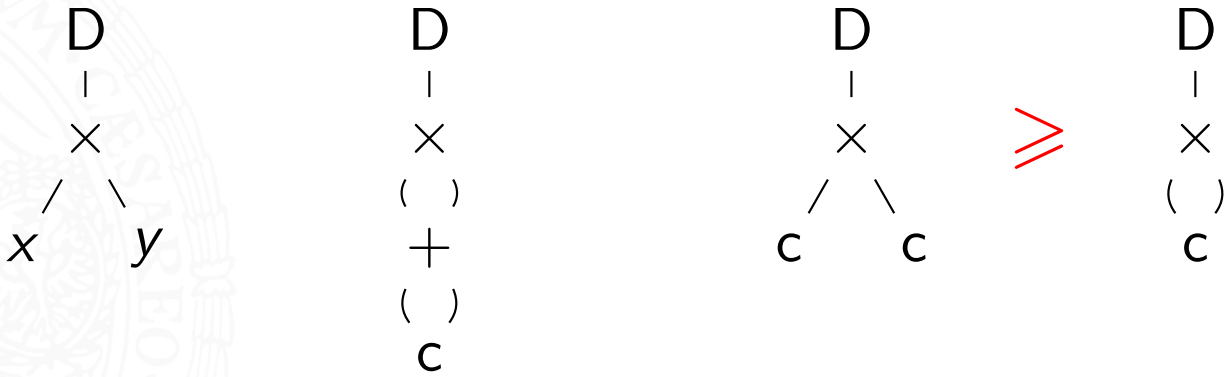
## Term Graph Morphism

- a term graph morphism $m\colon S \to T$ is a function $m\colon \mathrm{V}_S \to \mathrm{V}_T$
  1. sending the root of $S$ to the root of $T$
  2. preserving labels and successors for all non-variable nodes




- $S \geqslant_m T$ ($S \geqslant T$) if there exists a morphism $m\colon S \to T$ preserving also labels and successors of variable nodes

## Example



# Graph Rewriting

## Graph Rewrite Rule, Graph Rewrite System, Redex

- a rule is a pair of graphs $L, R$, denoted by $L \longrightarrow R$
- a graph rewrite system $\mathcal{G}$ is a set of rules
- A redex in a term graph $S$ is a node $n \in \mathrm{V}_S$ such that
  - there exists a rule $L \longrightarrow R \in \mathcal{G}$
  - there exists a morphism $m\colon L \longrightarrow S \restriction n$

$S \restriction n$ . . . subgraph of $S$ rooted at $n$

# Graph Rewriting

## Graph Rewrite Rule, Graph Rewrite System, Redex

- a rule is a pair of graphs $L, R$, denoted by $L \longrightarrow R$
- a graph rewrite system $\mathcal{G}$ is a set of rules
- A redex in a term graph $S$ is a node $n \in \mathsf{V}_S$ such that
  - there exists a rule $L \longrightarrow R \in \mathcal{G}$
  - there exists a morphism $m\colon L \longrightarrow S \restriction n$

$S \restriction n$ . . . subgraph of $S$ rooted at $n$

# Graph Rewriting

## Graph Rewrite Rule, Graph Rewrite System, Redex

- a rule is a pair of graphs $L, R$, denoted by $L \longrightarrow R$
- a graph rewrite system $\mathcal{G}$ is a set of rules
- A redex in a term graph $S$ is a node $n \in \mathrm{V}_S$ such that
  - there exists a rule $L \longrightarrow R \in \mathcal{G}$
  - there exists a morphism $m\colon L \longrightarrow S \restriction n$

$S \restriction n$ . . . subgraph of $S$ rooted at $n$

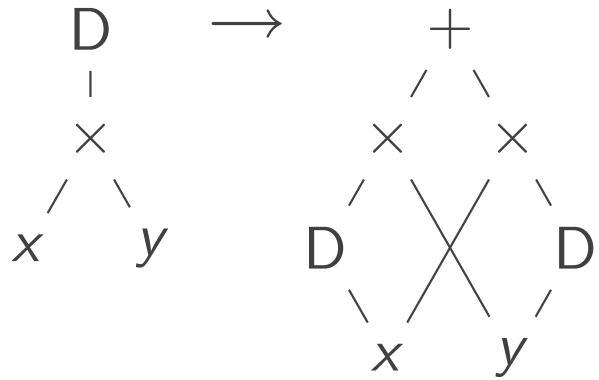
# Graph Rewriting

## Graph Rewrite Rule, Graph Rewrite System, Redex

- a rule is a pair of graphs $L, R$, denoted by $L \longrightarrow R$
- a graph rewrite system $\mathcal{G}$ is a set of rules
- A redex in a term graph $S$ is a node $n \in \mathsf{V}_S$ such that
  - there exists a rule $L \longrightarrow R \in \mathcal{G}$
  - there exists a morphism $m\colon L \longrightarrow S \restriction n$

$S \restriction n$ . . . subgraph of $S$ rooted at $n$

## Example

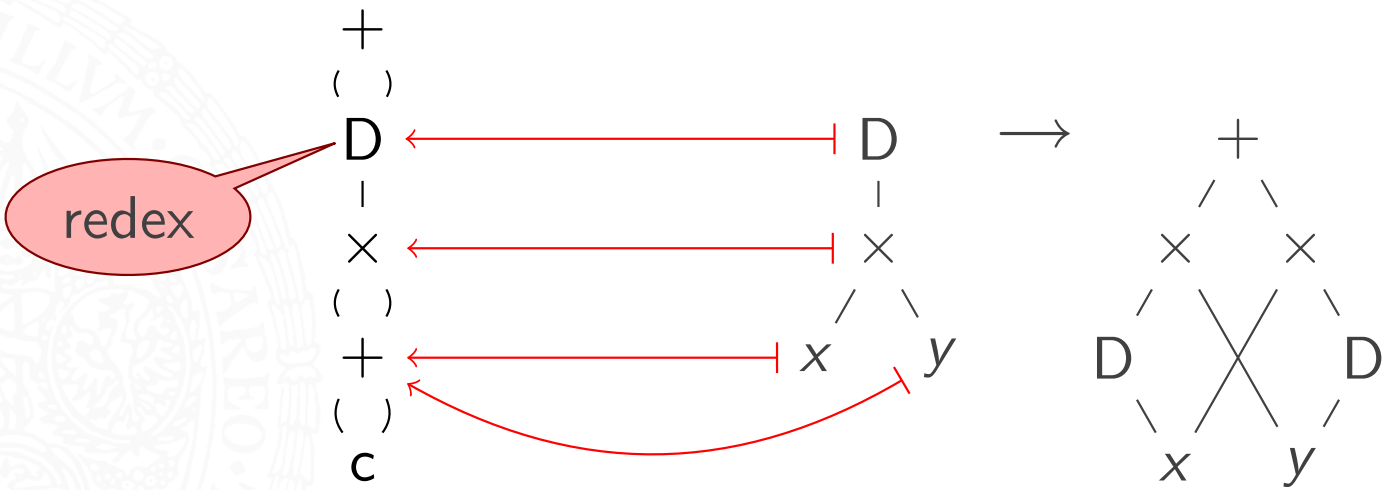
# Graph Rewriting

## Graph Rewrite Rule, Graph Rewrite System, Redex

- a rule is a pair of graphs $L, R$, denoted by $L \longrightarrow R$
- a graph rewrite system $\mathcal{G}$ is a set of rules
- A redex in a term graph $S$ is a node $n \in \mathrm{V}_S$ such that
  - there exists a rule $L \longrightarrow R \in \mathcal{G}$
  - there exists a morphism $m\colon L \longrightarrow S \restriction n$

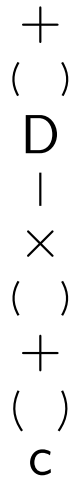
$S \restriction n$ . . . subgraph of $S$ rooted at $n$

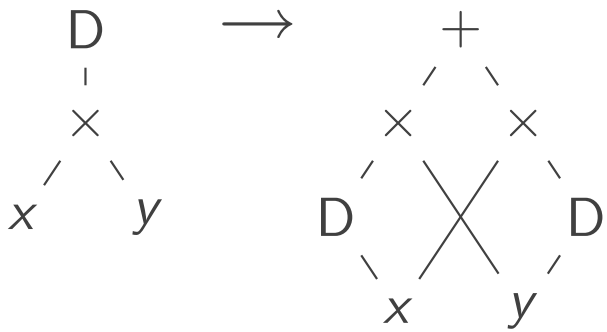
## Example



# Graph Rewriting

## Rewrite Step

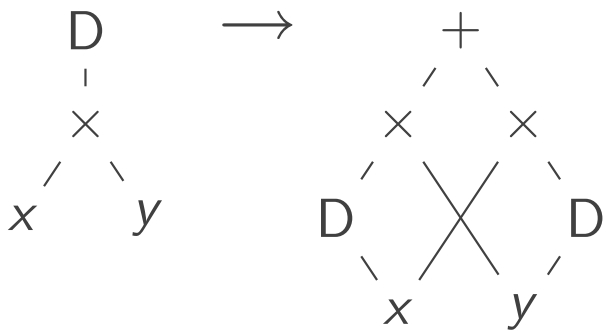


# Graph Rewriting

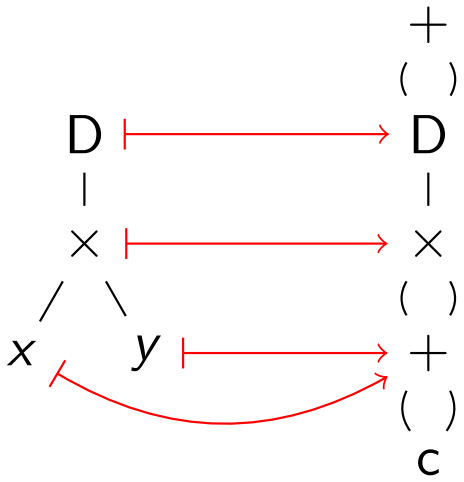
## Rewrite Step


1. identify redex
2. add a copy of the right-hand side
3. redirect edges pointing to the redex
4. redirect edges pointing to variable-nodes
5. remove inaccessible nodes

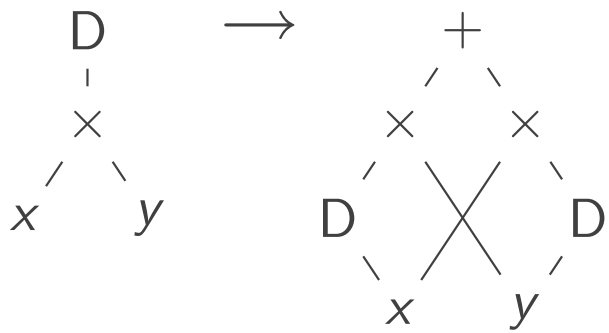


# Graph Rewriting

## Rewrite Step

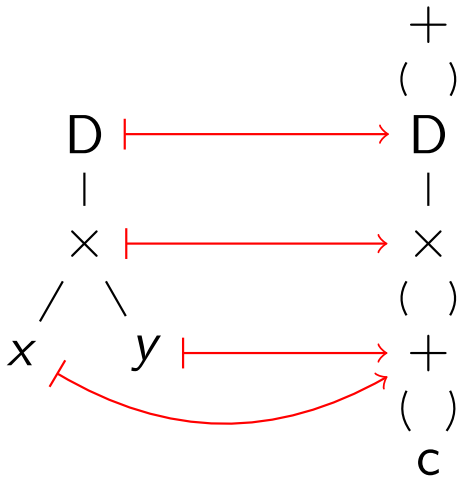


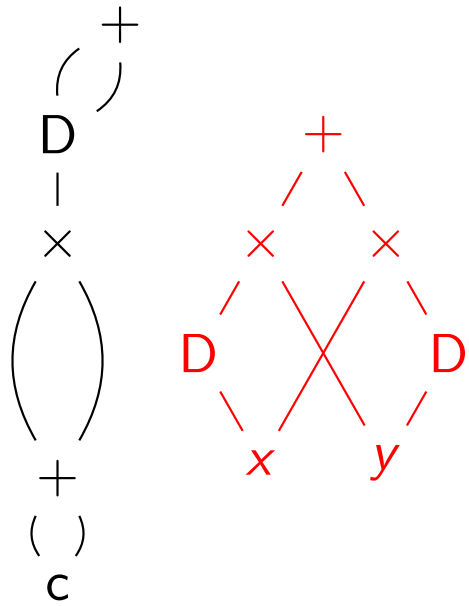

1. identify redex
2. add a copy of the right-hand side
3. redirect edges pointing to the redex
4. redirect edges pointing to variable-nodes
5. remove inaccessible nodes



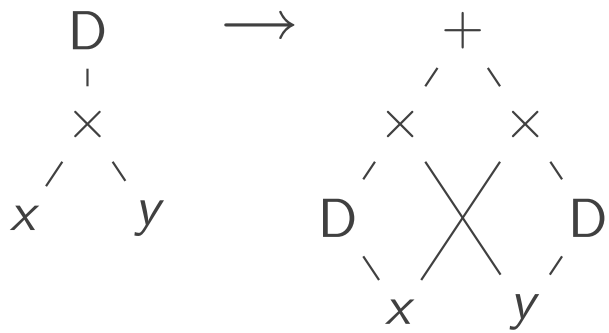
# Graph Rewriting

## Rewrite Step



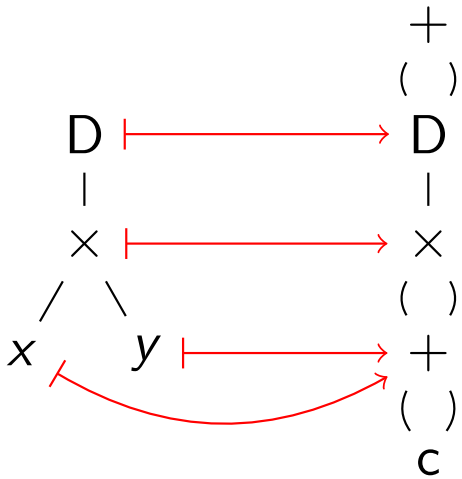



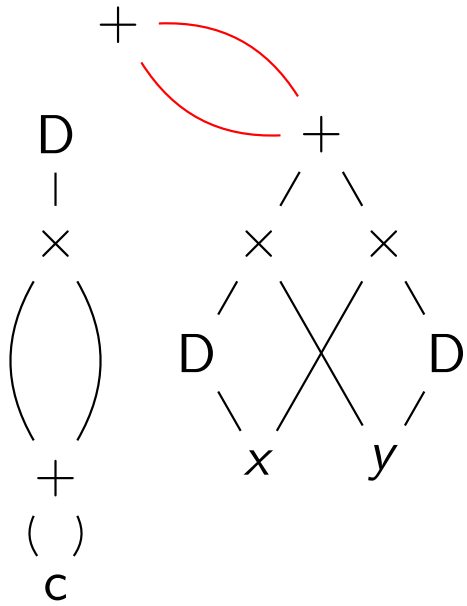

1. identify redex
2. add a copy of the right-hand side
3. redirect edges pointing to the redex
4. redirect edges pointing to variable-nodes
5. remove inaccessible nodes



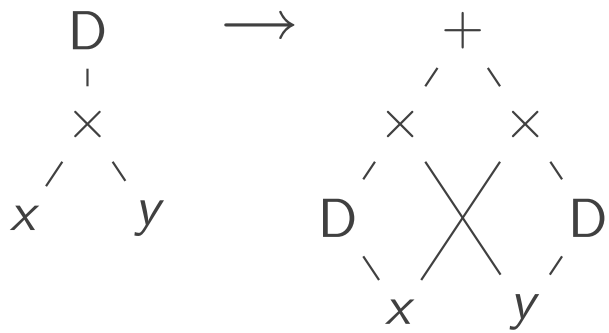
# Graph Rewriting

## Rewrite Step



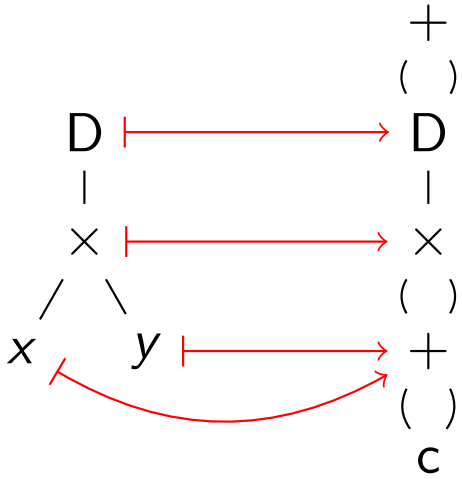



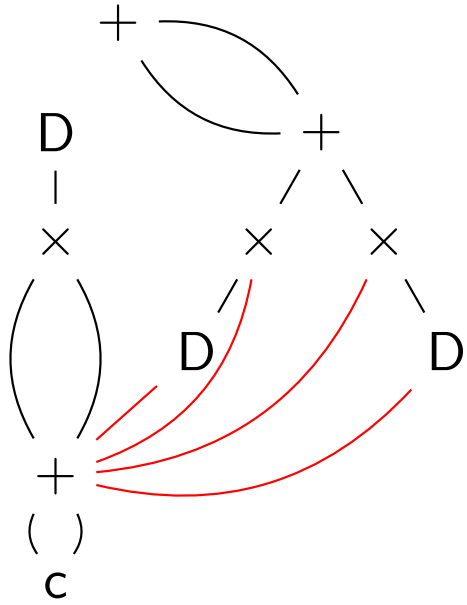

1. identify redex
2. add a copy of the right-hand side
3. redirect edges pointing to the redex
4. redirect edges pointing to variable-nodes
5. remove inaccessible nodes



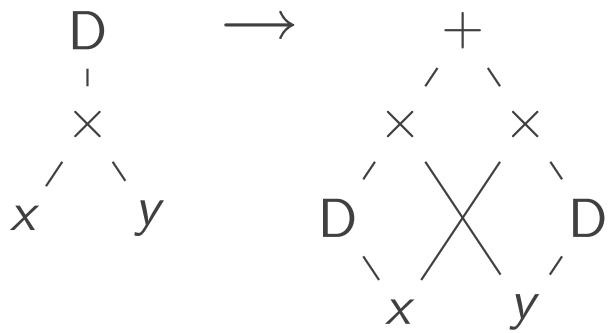
# Graph Rewriting

## Rewrite Step



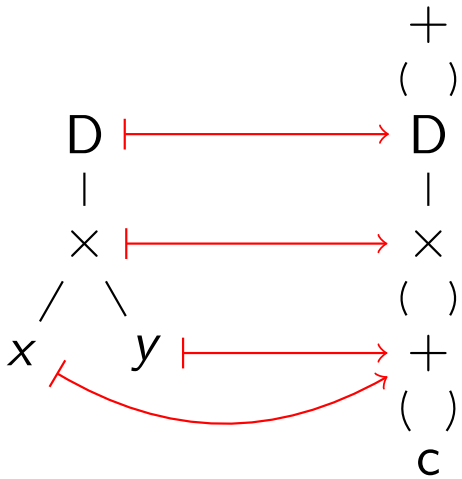



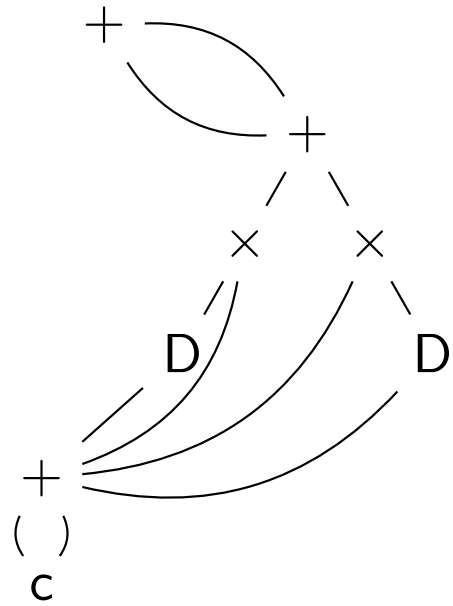

1. identify redex
2. add a copy of the right-hand side
3. redirect edges pointing to the redex
4. redirect edges pointing to variable-nodes
5. remove inaccessible nodes



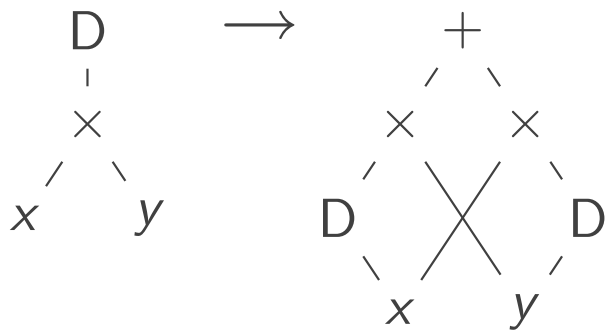
# Graph Rewriting

## Rewrite Step



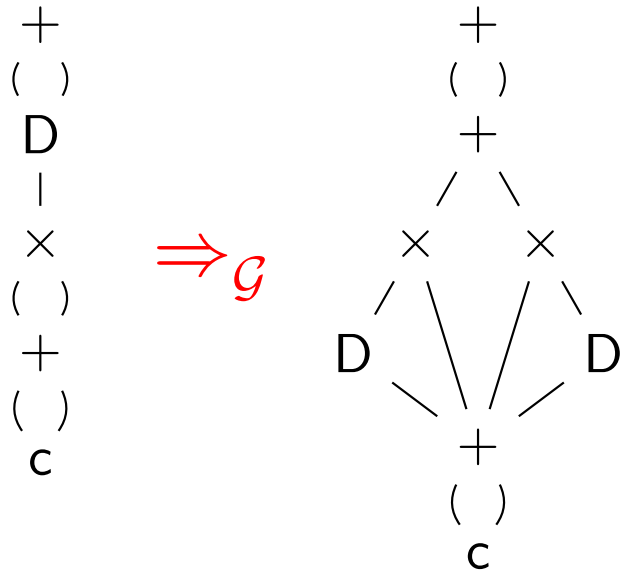




1. identify redex
2. add a copy of the right-hand side
3. redirect edges pointing to the redex
4. redirect edges pointing to variable-nodes
5. remove inaccessible nodes

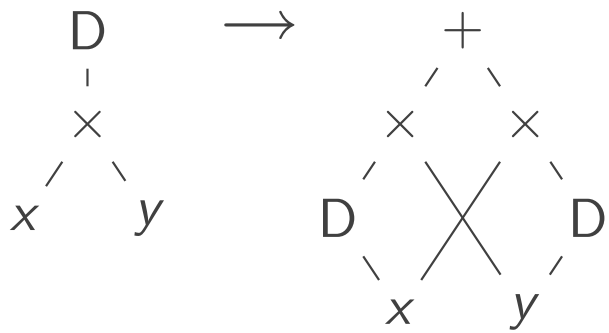


# Graph Rewriting

## Rewrite Step




1. identify redex
2. add a copy of the right-hand side
3. redirect edges pointing to the redex
4. redirect edges pointing to variable-nodes
5. remove inaccessible nodes



# Adequacy of Graph Rewriting
## Adequacy Theorem

# Adequacy Theorem

## Simulating Graph Rewrite System

Let $\mathcal{R}$ be a TRS. The simulating graph rewrite system $\mathcal{G}(\mathcal{R})$ contains for each rule $l \to r \in \mathcal{R}$ the rule $L \to R$ such that $L$ and $R$ are minimally sharing graphs representing $l$ and $r$ respectively.

# Adequacy Theorem

## Simulating Graph Rewrite System

Let $\mathcal{R}$ be a TRS. The simulating graph rewrite system $\mathcal{G}(\mathcal{R})$ contains for each rule $l \to r \in \mathcal{R}$ the rule $L \to R$ such that $L$ and $R$ are minimally sharing graphs representing $l$ and $r$ respectively.

## Problem ①

$$\mathcal{R} := \{\mathsf{f}(x, x) \to \mathsf{a}\}$$

$$\mathcal{G}(\mathcal{R}) := \left\{ \begin{array}{c} \mathsf{f} \\ (\ ) \\ x \end{array} \to \mathsf{a} \right\}$$

$$\mathsf{f}(\mathsf{a}, \mathsf{a}) \to_{\mathcal{R}} \mathsf{a}$$

$$\begin{array}{ccc} & \mathsf{f} & \\ / & & \backslash \\ \mathsf{a} & & \mathsf{a} \end{array} \in \mathsf{NF}(\mathcal{G}(\mathcal{R}))$$

# Adequacy Theorem

## Simulating Graph Rewrite System

Let $\mathcal{R}$ be a TRS. The simulating graph rewrite system $\mathcal{G}(\mathcal{R})$ contains for each rule $l \to r \in \mathcal{R}$ the rule $L \to R$ such that $L$ and $R$ are minimally sharing graphs representing $l$ and $r$ respectively.

## Problem ①

$$\mathcal{R} := \{\mathsf{f}(x, x) \to \mathsf{a}\}$$

$$\mathcal{G}(\mathcal{R}) := \left\{ \begin{array}{c} \mathsf{f} \\ (\ ) \\ x \end{array} \to \mathsf{a} \right\}$$

$$\mathsf{f}(\mathsf{a}, \mathsf{a}) \to_{\mathcal{R}} \mathsf{a}$$

$$\begin{array}{ccc} \mathsf{f} & & \mathsf{f} \\ / \ \backslash & \geqslant & (\ ) \Rightarrow_{\mathcal{G}(\mathcal{R})} \mathsf{a} \\ \mathsf{a} \quad \mathsf{a} & & \mathsf{a} \end{array}$$

# Adequacy Theorem

## Simulating Graph Rewrite System

Let $\mathcal{R}$ be a TRS. The simulating graph rewrite system $\mathcal{G}(\mathcal{R})$ contains for each rule $l \to r \in \mathcal{R}$ the rule $L \to R$ such that $L$ and $R$ are minimally sharing graphs representing $l$ and $r$ respectively.

## Problem ②

$$\mathcal{R} := \{\mathsf{a} \to \mathsf{b}\} \qquad \mathcal{G}(\mathcal{R}) := \{\mathsf{a} \to \mathsf{b}\}$$

$$\mathsf{f}(\mathsf{a},\mathsf{a}) \to_{\mathcal{R}} \mathsf{f}(\mathsf{a},\mathsf{b}) \qquad \begin{array}{ccc} \mathsf{f} & & \mathsf{f} \\ (\ \ ) & \Rightarrow_{\mathcal{G}(\mathcal{R})} & (\ \ ) \\ \mathsf{a} & & \mathsf{b} \end{array}$$

# Adequacy Theorem

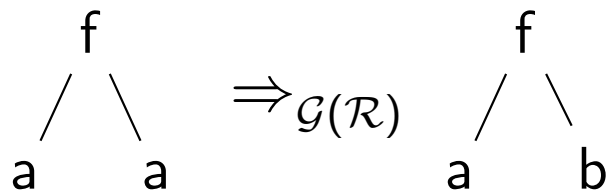
## Simulating Graph Rewrite System

Let $\mathcal{R}$ be a TRS. The simulating graph rewrite system $\mathcal{G}(\mathcal{R})$ contains for each rule $l \to r \in \mathcal{R}$ the rule $L \to R$ such that $L$ and $R$ are minimally sharing graphs representing $l$ and $r$ respectively.

## Problem ②

$$\mathcal{R} := \{\mathsf{a} \to \mathsf{b}\} \qquad \mathcal{G}(\mathcal{R}) := \{\mathsf{a} \to \mathsf{b}\}$$

$$\mathsf{f}(\mathsf{a},\mathsf{a}) \to_{\mathcal{R}} \mathsf{f}(\mathsf{a},\mathsf{b})$$



# Adequacy Theorem

## Simulating Graph Rewrite System

Let $\mathcal{R}$ be a TRS. The simulating graph rewrite system $\mathcal{G}(\mathcal{R})$ contains for each rule $l \to r \in \mathcal{R}$ the rule $L \to R$ such that $L$ and $R$ are minimally sharing graphs representing $l$ and $r$ respectively.

## Solution

① keep everything below redexes maximally shared

② keep redexes unshared

# Adequacy Theorem

## Theorem

*Let $\mathcal{R}$ be a TRS and $\mathcal{G}(\mathcal{R})$ the simulating graph rewrite system of $\mathcal{R}$.*
*Let $s$ be a term and $S$ its normal-form sharing term graph representation.*

$$s \xrightarrow{\mathsf{i}}_{\mathcal{R}} t \qquad \Longleftrightarrow \qquad S \overset{\mathsf{i}}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} \cdot \geqslant T$$

*where $T$ is a normal-form sharing term graph representing $t$.*

# Adequacy Theorem

## Theorem

*Let $\mathcal{R}$ be a TRS and $\mathcal{G}(\mathcal{R})$ the simulating graph rewrite system of $\mathcal{R}$.*
*Let $s$ be a term and $S$ its normal-form sharing term graph representation.*

$$s \xrightarrow{\mathsf{i}}_{\mathcal{R}} t \qquad \Longleftrightarrow \qquad S \overset{\mathsf{i}}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} \cdot \geqslant T$$

*where $T$ is a normal-form sharing term graph representing $t$.*

## Proof Idea

suppose $s = C[l\sigma] \xrightarrow{\mathsf{i}}_{\mathcal{R}} C[r\sigma]$

1. $S \overset{\mathsf{i}}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} T'$ where $T'$ represents $t$

2. $T' \geq T$

# Adequacy Theorem

## Theorem

*Let $\mathcal{R}$ be a TRS and $\mathcal{G}(\mathcal{R})$ the simulating graph rewrite system of $\mathcal{R}$.*
*Let $s$ be a term and $S$ its normal-form sharing term graph representation.*

$$s \xrightarrow{\mathrm{i}}_{\mathcal{R}} t \qquad \Longleftrightarrow \qquad S \overset{\mathrm{i}}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} \cdot \geqslant T$$

*where $T$ is a normal-form sharing term graph representing $t$.*

## Proof Idea

suppose $s = C[l\sigma] \xrightarrow{\mathrm{i}}_{\mathcal{R}} C[r\sigma]$

1. $S \overset{\mathrm{i}}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} T'$ where $T'$ represents $t$
   - subgraph $S \restriction p$ of $S$ representing $l\sigma$ is maximally shared
   - thus $m\colon L \to S \restriction p$ for $L \to R \in \mathcal{G}(\mathcal{R})$ representing $l \to r$
   - redex node $p$ in $S$ representing $l\sigma$ is unshared
2. $T' \geq T$

# Adequacy Theorem

## Theorem

*Let $\mathcal{R}$ be a TRS and $\mathcal{G}(\mathcal{R})$ the simulating graph rewrite system of $\mathcal{R}$.*
*Let $s$ be a term and $S$ its normal-form sharing term graph representation.*

$$s \xrightarrow{\mathrm{i}}_{\mathcal{R}} t \qquad \Longleftrightarrow \qquad S \overset{\mathrm{i}}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} \cdot \geqslant T$$

*where $T$ is a normal-form sharing term graph representing $t$.*

## Proof Idea

suppose $s = C[l\sigma] \xrightarrow{\mathrm{i}}_{\mathcal{R}} C[r\sigma]$

1. $S \overset{\mathrm{i}}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} T'$ where $T'$ represents $t$
   - subgraph $S \restriction p$ of $S$ representing $l\sigma$ is maximally shared
   - thus $m\colon L \to S \restriction p$ for $L \to R \in \mathcal{G}(\mathcal{R})$ representing $l \to r$
   - redex node $p$ in $S$ representing $l\sigma$ is unshared
2. $T' \geq T$

# Adequacy Theorem

## Theorem

*Let $\mathcal{R}$ be a TRS and $\mathcal{G}(\mathcal{R})$ the simulating graph rewrite system of $\mathcal{R}$.*
*Let $s$ be a term and $S$ its normal-form sharing term graph representation.*

$$s \xrightarrow{\mathrm{i}}_{\mathcal{R}} t \qquad \Longleftrightarrow \qquad S \overset{\mathrm{i}}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} \cdot \geqslant T$$

*where $T$ is a normal-form sharing term graph representing $t$.*

## Proof Idea

suppose $s = C[l\sigma] \xrightarrow{\mathrm{i}}_{\mathcal{R}} C[r\sigma]$

1. $S \overset{\mathrm{i}}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} T'$ where $T'$ represents $t$
   - subgraph $S \restriction p$ of $S$ representing $l\sigma$ is maximally shared
   - thus $m\colon L \to S \restriction p$ for $L \to R \in \mathcal{G}(\mathcal{R})$ representing $l \to r$
   - redex node $p$ in $S$ representing $l\sigma$ is unshared
2. $T' \geq T$

# Adequacy Theorem

## Theorem

*Let $\mathcal{R}$ be a TRS and $\mathcal{G}(\mathcal{R})$ the simulating graph rewrite system of $\mathcal{R}$.*
*Let $s$ be a term and $S$ its normal-form sharing term graph representation.*

$$s \xrightarrow{\mathrm{i}}_{\mathcal{R}} t \qquad \Longleftrightarrow \qquad S \overset{\mathrm{i}}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} \cdot \geqslant T$$

*where $T$ is a normal-form sharing term graph representing $t$.*

## Proof Idea

suppose $s = C[l\sigma] \xrightarrow{\mathrm{i}}_{\mathcal{R}} C[r\sigma]$

1. $S \overset{\mathrm{i}}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} T'$ where $T'$ represents $t$
   - subgraph $S \restriction p$ of $S$ representing $l\sigma$ is maximally shared
   - thus $m\colon L \to S \restriction p$ for $L \to R \in \mathcal{G}(\mathcal{R})$ representing $l \to r$
   - redex node $p$ in $S$ representing $l\sigma$ is unshared
2. $T' \geq T$ $\equiv$ all redex nodes in $T'$ unshared

# Adequacy Theorem

## Theorem

*Let $\mathcal{R}$ be a TRS and $\mathcal{G}(\mathcal{R})$ the simulating graph rewrite system of $\mathcal{R}$.*
*Let $s$ be a term and $S$ its normal-form sharing term graph representation.*

$$s \xrightarrow{\mathrm{i}}_{\mathcal{R}} t \qquad \Longleftrightarrow \qquad S \overset{\mathrm{i}}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} \cdot \geqslant T$$

*where $T$ is a normal-form sharing term graph representing $t$.*

## Proof Idea

suppose $s = C[l\sigma] \xrightarrow{\mathrm{i}}_{\mathcal{R}} C[r\sigma]$

1. $S \overset{\mathrm{i}}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} T'$ where $T'$ represents $t$
   - subgraph $S \restriction p$ of $S$ representing $l\sigma$ is maximally shared
   - thus $m\colon L \to S \restriction p$ for $L \to R \in \mathcal{G}(\mathcal{R})$ representing $l \to r$
   - redex node $p$ in $S$ representing $l\sigma$ is unshared
2. $T' \geq T$ $\equiv$ all redex nodes in $T'$ unshared
   - adding $R$ to $S$ does not introduce sharing

□

# Runtime Complexity as Reasonable Cost Model

# Polytime Computability

## Theorem

*Let $\mathcal{R}$ be a confluent and terminating TRS.*
*If $\mathcal{R}$ admits polynomial bounded (innermost) runtime-complexity then the functions computed by $\mathcal{R}$ are polytime-computable.*

# Polytime Computability

## Theorem

*Let $\mathcal{R}$ be a confluent and terminating TRS.*
*If $\mathcal{R}$ admits polynomial bounded (innermost) runtime-complexity then the functions computed by $\mathcal{R}$ are polytime-computable.*

## Proof Idea

consider computation of $f(s_1, \ldots, s_n) = t$

# Polytime Computability

## Theorem

*Let $\mathcal{R}$ be a confluent and terminating TRS.*
*If $\mathcal{R}$ admits polynomial bounded (innermost) runtime-complexity then the functions computed by $\mathcal{R}$ are polytime-computable.*

## Proof Idea

consider computation of $f(s_1, \ldots, s_n) = t$

$$\mathsf{f}(s_1, \ldots, s_n) := s \quad \to_{\mathcal{R}} \quad t_1 \quad \to_{\mathcal{R}} \quad \cdots \quad \to_{\mathcal{R}} \quad t_\ell = t$$

# Polytime Computability

## Theorem

*Let $\mathcal{R}$ be a confluent and terminating TRS.*
*If $\mathcal{R}$ admits polynomial bounded (innermost) runtime-complexity then the functions computed by $\mathcal{R}$ are polytime-computable.*

## Proof Idea

consider computation of $f(s_1, \ldots, s_n) = t$

$$\mathsf{f}(s_1, \ldots, s_n) := s \quad \xrightarrow{\mathsf{i}}_{\mathcal{R}} \quad t_1 \quad \xrightarrow{\mathsf{i}}_{\mathcal{R}} \quad \cdots \quad \xrightarrow{\mathsf{i}}_{\mathcal{R}} \quad t_\ell = t$$

# Polytime Computability

## Theorem

*Let $\mathcal{R}$ be a confluent and terminating TRS.*
*If $\mathcal{R}$ admits polynomial bounded (innermost) runtime-complexity then the functions computed by $\mathcal{R}$ are polytime-computable.*

## Proof Idea

consider computation of $f(s_1, \ldots, s_n) = t$

$$S \overset{\mathsf{i}}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} \cdot \geqslant T_1 \overset{\mathsf{i}}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} \cdot \geqslant \cdots \overset{\mathsf{i}}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} \cdot \geqslant T_\ell = T$$

- due to the adequacy theorem
  - $S$ normal-form sharing graph representation of $s = \mathsf{f}(s_1, \ldots, s_n)$
  - $T$ normal-form sharing graph representation of result $t$

# Polytime Computability

## Theorem

*Let $\mathcal{R}$ be a confluent and terminating TRS.*
*If $\mathcal{R}$ admits polynomial bounded (innermost) runtime-complexity then the functions computed by $\mathcal{R}$ are polytime-computable.*

## Proof Idea

consider computation of $f(s_1, \ldots, s_n) = t$

$$S \overset{\mathsf{i}}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} \cdot \geqslant T_1 \overset{\mathsf{i}}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} \cdot \geqslant \cdots \overset{\mathsf{i}}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} \cdot \geqslant T_\ell = T$$

1. length $\ell$ bounded polynomially in the size of $s$ $\qquad \ell \leqslant p(|s|)$

# Polytime Computability

## Theorem

*Let $\mathcal{R}$ be a confluent and terminating TRS.*
*If $\mathcal{R}$ admits polynomial bounded (innermost) runtime-complexity then the functions computed by $\mathcal{R}$ are polytime-computable.*

## Proof Idea

consider computation of $f(s_1, \ldots, s_n) = t$

$$S \overset{\text{i}}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} \cdot \geqslant T_1 \overset{\text{i}}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} \cdot \geqslant \cdots \overset{\text{i}}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} \cdot \geqslant T_\ell = T$$

1. length $\ell$ bounded polynomially in the size of $s$ $\quad \ell \leqslant p(|s|)$
2. $T_i \overset{\text{i}}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} \cdot \geqslant T_{i+1}$ polytime-computable (in the size of $s$) $\quad q(|s|)$

# Polytime Computability

## Theorem

*Let $\mathcal{R}$ be a confluent and terminating TRS.*
*If $\mathcal{R}$ admits polynomial bounded (innermost) runtime-complexity then the functions computed by $\mathcal{R}$ are polytime-computable.*

## Proof Idea

consider computation of $f(s_1, \ldots, s_n) = t$

$$S \overset{\text{i}}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} \cdot \geqslant T_1 \overset{\text{i}}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} \cdot \geqslant \cdots \overset{\text{i}}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} \cdot \geqslant T_\ell = T$$

1. length $\ell$ bounded polynomially in the size of $s$ $\quad \ell \leqslant p(|s|)$
2. $T_i \overset{\text{i}}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} \cdot \geqslant T_{i+1}$ polytime-computable (in the size of $s$) $\quad q(|s|)$
3. overall, $S$ reducible to $T$ in time $p(|s|) * q(|s|)$

# Polytime Computability

## Theorem

*Let $\mathcal{R}$ be a confluent and terminating TRS.*
*If $\mathcal{R}$ admits polynomial bounded (innermost) runtime-complexity then the functions computed by $\mathcal{R}$ are polytime-computable.*

## Proof Idea

consider computation of $f(s_1, \ldots, s_n) = t$

$$S \overset{i}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} \cdot \geqslant T_1 \overset{i}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} \cdot \geqslant \cdots \overset{i}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} \cdot \geqslant T_\ell = T$$

1. length $\ell$ bounded polynomially in the size of $s$ $\quad \ell \leqslant p(|s|)$
2. $T_i \overset{i}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} \cdot \geqslant T_{i+1}$ polytime-computable (in the size of $s$) $\quad q(|s|)$
   - size of $T_i$ polynomially bounded in the size of $S$ $\quad |T_i| \leqslant c * i$
3. overall, $S$ reducible to $T$ in time $p(|s|) * q(|s|)$

# Polytime Computability

## Theorem

*Let $\mathcal{R}$ be a confluent and terminating TRS.*
*If $\mathcal{R}$ admits polynomial bounded (innermost) runtime-complexity then the functions computed by $\mathcal{R}$ are polytime-computable.*

## Proof Idea

consider computation of $f(s_1, \ldots, s_n) = t$

$$S \overset{\text{i}}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} \cdot \geqslant T_1 \overset{\text{i}}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} \cdot \geqslant \cdots \overset{\text{i}}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} \cdot \geqslant T_\ell = T$$

1. length $\ell$ bounded polynomially in the size of $s$ $\quad \ell \leqslant p(|s|)$
2. $T_i \overset{\text{i}}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} \cdot \geqslant T_{i+1}$ polytime-computable (in the size of $s$) $\quad q(|s|)$
   - size of $T_i$ polynomially bounded in the size of $S$ $\quad |T_i| \leqslant c * p(|S|)$
3. overall, $S$ reducible to $T$ in time $p(|s|) * q(|s|)$

# Polytime Computability

## Theorem

*Let $\mathcal{R}$ be a confluent and terminating TRS.*
*If $\mathcal{R}$ admits polynomial bounded (innermost) runtime-complexity then the functions computed by $\mathcal{R}$ are polytime-computable.*

## Proof Idea

consider computation of $f(s_1, \ldots, s_n) = t$

$$S \overset{\mathsf{i}}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} \cdot \geqslant T_1 \overset{\mathsf{i}}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} \cdot \geqslant \cdots \overset{\mathsf{i}}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} \cdot \geqslant T_\ell = T$$

1. length $\ell$ bounded polynomially in the size of $s$ $\quad \ell \leqslant p(|s|)$
2. $T_i \overset{\mathsf{i}}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} \cdot \geqslant T_{i+1}$ polytime-computable (in the size of $s$) $\quad q(|s|)$
   - size of $T_i$ polynomially bounded in the size of $S$ $\quad |T_i| \leqslant c * p(|s|)$
3. overall, $S$ reducible to $T$ in time $p(|s|) * q(|s|)$

# Polytime Computability

## Theorem

*Let $\mathcal{R}$ be a confluent and terminating TRS.*
*If $\mathcal{R}$ admits polynomial bounded (innermost) runtime-complexity then the functions computed by $\mathcal{R}$ are polytime-computable.*

## Proof Idea

consider computation of $f(s_1, \ldots, s_n) = t$

$$S \overset{i}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} \cdot \geqslant T_1 \overset{i}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} \cdot \geqslant \cdots \overset{i}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} \cdot \geqslant T_\ell = T$$

1. length $\ell$ bounded polynomially in the size of $s$ $\quad \ell \leqslant p(|s|)$
2. $T_i \overset{i}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} \cdot \geqslant T_{i+1}$ polytime-computable (in the size of $s$) $\quad q(|s|)$
   - size of $T_i$ polynomially bounded in the size of $S$ $\quad |T_i| \leqslant c * p(|s|)$
   - $T_i \overset{i}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} \cdot \geqslant T_{i+1}$ computable in time polynomial in the size of $T_i$
3. overall, $S$ reducible to $T$ in time $p(|s|) * q(|s|)$

# Conclusion and Future Work

# Conclusion and Future Work

runtime-complexity is a reasonable cost model for rewriting

- polynomial (innermost) runtime-complexity induces polytime computability
- extensible to non-deterministic computation

# Conclusion and Future Work

## runtime-complexity is a reasonable cost model for rewriting

- polynomial (innermost) runtime-complexity induces polytime computability
- extensible to non-deterministic computation

## Future Work

- extension to full-rewriting

# Conclusion and Future Work

## runtime-complexity is a reasonable cost model for rewriting

- polynomial (innermost) runtime-complexity induces polytime computability
- extensible to non-deterministic computation

## Current Work

- extension to full-rewriting

# Conclusion and Future Work

## runtime-complexity is a reasonable cost model for rewriting

- polynomial (innermost) runtime-complexity induces polytime computability
- extensible to non-deterministic computation

## Current Work

- extension to full-rewriting

$$s \to_{\mathcal{R}} t \qquad \Longleftrightarrow \qquad S \leq S_1 \geq S_2 \Rightarrow_{\mathcal{G}(\mathcal{R})} T$$

well known

# Conclusion and Future Work

## runtime-complexity is a reasonable cost model for rewriting

- polynomial (innermost) runtime-complexity induces polytime computability
- extensible to non-deterministic computation

## Current Work

- extension to full-rewriting

$$s \to_{\mathcal{R}} t \quad \Longleftrightarrow \quad S \leq S_1 \geq S_2 \Rightarrow_{\mathcal{G}(\mathcal{R})} T$$

well known

### Difficulty

- efficiency