Explicit Rewriting

François-Régis Sinot

LIX, École Polytechnique, France
Projet Logical: PCRI, CNRS, EP, INRIA, UPS

Workshop on the ρ-calculus, 2005

Ladies and gentlemen. . .

- take a TRS $\mathcal{R} = \{l_i \to r_i\}_i$ on $\Sigma = \{f, \ldots\}$

Ladies and gentlemen. . .

- take a TRS $\mathcal{R} = \{l_i \to r_i\}_i$ on $\Sigma = \{f, \ldots\}$
- we define a new TRS $(\Sigma', \mathcal{R}')$:

Ladies and gentlemen...

- take a TRS $\mathcal{R} = \{l_i \to r_i\}_i$ on $\Sigma = \{f, \ldots\}$
- we define a new TRS $(\Sigma', \mathcal{R}')$:
- $\Sigma' = \Sigma \cup \{\Downarrow, \Uparrow\}$

Ladies and gentlemen...

- take a TRS $\mathcal{R} = \{l_i \to r_i\}_i$ on $\Sigma = \{f, \ldots\}$
- we define a new TRS $(\Sigma', \mathcal{R}')$:
- $\Sigma' = \Sigma \cup \{\Downarrow, \Uparrow\}$
- we write $\Downarrow t$ instead of $\Downarrow(t)$

Ladies and gentlemen. . .

- take a TRS $\mathcal{R} = \{l_i \to r_i\}_i$ on $\Sigma = \{f, \ldots\}$
- we define a new TRS $(\Sigma', \mathcal{R}')$:
- $\Sigma' = \Sigma \cup \{\Downarrow, \Uparrow\}$
- we write $\Downarrow t$ instead of $\Downarrow(t)$
- $\mathcal{R}'$ consists of:

Ladies and gentlemen...

- take a TRS $\mathcal{R} = \{l_i \to r_i\}_i$ on $\Sigma = \{f, \ldots\}$
- we define a new TRS $(\Sigma', \mathcal{R}')$:
- $\Sigma' = \Sigma \cup \{\Downarrow, \Uparrow\}$
- we write $\Downarrow t$ instead of $\Downarrow(t)$
- $\mathcal{R}'$ consists of:
  - $\Downarrow l_i \leadsto \Uparrow r_i$

Ladies and gentlemen...

- take a TRS $\mathcal{R} = \{l_i \to r_i\}_i$ on $\Sigma = \{f, \ldots\}$
- we define a new TRS $(\Sigma', \mathcal{R}')$:
- $\Sigma' = \Sigma \cup \{\Downarrow, \Uparrow\}$
- we write $\Downarrow t$ instead of $\Downarrow(t)$
- $\mathcal{R}'$ consists of:
  - $\Downarrow l_i \rightsquigarrow \Uparrow r_i$
  - $\Downarrow f(x_1, \ldots, x_j, \ldots, x_n) \rightsquigarrow f(x_1, \ldots, \Downarrow x_j, \ldots, x_n)$

Ladies and gentlemen...

- take a TRS $\mathcal{R} = \{l_i \to r_i\}_i$ on $\Sigma = \{f, \ldots\}$
- we define a new TRS $(\Sigma', \mathcal{R}')$:
- $\Sigma' = \Sigma \cup \{\Downarrow, \Uparrow\}$
- we write $\Downarrow t$ instead of $\Downarrow(t)$
- $\mathcal{R}'$ consists of:
  - $\Downarrow l_i \rightsquigarrow \Uparrow r_i$
  - $\Downarrow f(x_1, \ldots, x_j, \ldots, x_n) \rightsquigarrow f(x_1, \ldots, \Downarrow x_j, \ldots, x_n)$
  - $f(x_1, \ldots, \Uparrow x_j, \ldots, x_n) \rightsquigarrow \Uparrow f(x_1, \ldots, x_j, \ldots, x_n)$

Ladies and gentlemen. . .

- take a TRS $\mathcal{R} = \{l_i \to r_i\}_i$ on $\Sigma = \{f, \ldots\}$
- we define a new TRS $(\Sigma', \mathcal{R}')$:
- $\Sigma' = \Sigma \cup \{\Downarrow, \Uparrow\}$
- we write $\Downarrow t$ instead of $\Downarrow(t)$
- $\mathcal{R}'$ consists of:
  - $\Downarrow l_i \rightsquigarrow \Uparrow r_i$
  - $\Downarrow f(x_1, \ldots, x_j, \ldots, x_n) \rightsquigarrow f(x_1, \ldots, \Downarrow x_j, \ldots, x_n)$
  - $f(x_1, \ldots, \Uparrow x_j, \ldots, x_n) \rightsquigarrow \Uparrow f(x_1, \ldots, x_j, \ldots, x_n)$
- that's it !

So what ?

- t always in normal form; start from $\Downarrow t$

So what ?

- t always in normal form; start from $\Downarrow t$
- always exactly one occurrence of $\Downarrow$ or $\Uparrow$

So what ?

- t always in normal form; start from $\Downarrow t$
- always exactly one occurrence of $\Downarrow$ or $\Uparrow$
- reduction always happens at this unique occurrence

So what ?

- t always in normal form; start from $\Downarrow t$
- always exactly one occurrence of $\Downarrow$ or $\Uparrow$
- reduction always happens at this unique occurrence
- so rewriting is **simpler**

So what ?

- t always in normal form; start from $\Downarrow t$
- always exactly one occurrence of $\Downarrow$ or $\Uparrow$
- reduction always happens at this unique occurrence
- so rewriting is **simpler**
- it is also **equivalent** and **more explicit**:
  prop.: $t \to u \iff \Downarrow t \leadsto^* \Uparrow u$

Multi-step reduction

- add the rule *(restart)*: $\Uparrow x \leadsto \Downarrow x$

Multi-step reduction

- add the rule *(restart)*: $\Uparrow x \rightsquigarrow \Downarrow x$
- **prop.**: $t \longrightarrow^* u \iff \Uparrow t \rightsquigarrow^* \Uparrow u$

Multi-step reduction

- add the rule *(restart)*: $\Uparrow x \leadsto \Downarrow x$
- **prop.**: $t \to^* u \iff \Uparrow t \leadsto^* \Uparrow u$
- more precisely:
  prop.: $t \to^n u \iff \Uparrow t \leadsto^* \Uparrow u$ using n times *(restart)*

Example 1: length of reductions

- $\mathcal{R} = \{f(a) \to b, a \to c\}$

Example 1: length of reductions

- $\mathcal{R} = \{f(a) \rightarrow b, a \rightarrow c\}$
- $f(a) \rightarrow b$ (one step)

Example 1: length of reductions

- $\mathcal{R} = \{f(a) \to b, a \to c\}$
- $f(a) \to b$ (one step)
- $\Downarrow f(a) \leadsto \Uparrow b$

Example 1: length of reductions

- $\mathcal{R} = \{f(a) \to b, a \to c\}$
- $f(a) \to b$ (one step)
- $\Downarrow f(a) \rightsquigarrow \Uparrow b$
- $f(a) \to f(c)$ (one step)

Example 1: length of reductions

- $\mathcal{R} = \{f(a) \to b, a \to c\}$
- $f(a) \to b$ (one step)
- $\Downarrow f(a) \leadsto \Uparrow b$
- $f(a) \to f(c)$ (one step)
- $\Downarrow f(a) \leadsto f(\Downarrow a) \leadsto f(\Uparrow c) \leadsto \Uparrow f(c)$

Example 2: confluence ?

- $\mathcal{R} = \{f(a) \to b\}$ confluent

Example 2: confluence ?

- $\mathcal{R} = \{f(a) \to b\}$ confluent
- $f(a) \to b$ only possible reduction

Example 2: confluence ?

- $\mathcal{R} = \{f(a) \to b\}$ confluent
- $f(a) \to b$ only possible reduction
- $\Downarrow f(a) \leadsto \Uparrow b$

Example 2: confluence ?

- $\mathcal{R} = \{f(a) \rightarrow b\}$ confluent
- $f(a) \rightarrow b$ only possible reduction
- $\Downarrow f(a) \rightsquigarrow \Uparrow b$
- but $\Downarrow f(a) \rightsquigarrow f(\Downarrow a) \not\rightsquigarrow$

Example 2: confluence ?

- $\mathcal{R} = \{f(a) \to b\}$ confluent
- $f(a) \to b$ only possible reduction
- $\Downarrow f(a) \rightsquigarrow \Uparrow b$
- but $\Downarrow f(a) \rightsquigarrow f(\Downarrow a) \not\to$
- preservation of confluence ?

Confluence

- **def.[x-confluence]**: if $\Uparrow t \rightsquigarrow^* \Uparrow u$ and $\Uparrow t \rightsquigarrow^* \Uparrow v$, then there exists w such that $\Uparrow u \rightsquigarrow^* \Uparrow w$ and $\Uparrow v \rightsquigarrow^* \Uparrow w$

Confluence

- **def.[x-confluence]**: if $\Uparrow t \leadsto^* \Uparrow u$ and $\Uparrow t \leadsto^* \Uparrow v$, then there exists w such that $\Uparrow u \leadsto^* \Uparrow w$ and $\Uparrow v \leadsto^* \Uparrow w$
- **prop.**: $\rightarrow$ confluent $\iff$ $\leadsto$ x-confluent

Termination

- **def.[x-terminating]**: no infinite sequence $\Uparrow t_1 \rightsquigarrow^* \Uparrow t_2 \rightsquigarrow^* \ldots \rightsquigarrow^* \Uparrow t_n \rightsquigarrow^* \ldots$

Termination

- **def.[x-terminating]**: no infinite sequence
  $\Uparrow t_1 \leadsto^* \Uparrow t_2 \leadsto^* \ldots \leadsto^* \Uparrow t_n \leadsto^* \ldots$
- **prop.**: x-terminating $\equiv$ terminating

Termination

- **def.[x-terminating]**: no infinite sequence
  $\Uparrow t_1 \rightsquigarrow^* \Uparrow t_2 \rightsquigarrow^* \ldots \rightsquigarrow^* \Uparrow t_n \rightsquigarrow^* \ldots$
- **prop.**: x-terminating $\equiv$ terminating
- **prop.**: $\rightarrow$ terminating $\iff$ $\rightsquigarrow$ terminating

Embedding strategies

- easy: just remove some rules

Embedding strategies

- easy: just remove some rules
- example: "frozen operator"

  $\Sigma = \{\texttt{skip}, \texttt{done}, ;\}$

  $\mathcal{R} = \{\texttt{skip} \rightarrow \texttt{done},$

  $\qquad \texttt{done};x \rightarrow x\}$

  strategy: never reduce the right subterm of ;

Embedding strategies

- easy: just remove some rules
- example: "frozen operator"
  $\Sigma = \{\texttt{skip}, \texttt{done}, ;\}$
  $\mathcal{R} = \{\texttt{skip} \to \texttt{done},$
  $\qquad \texttt{done};x \to x\}$
  strategy: never reduce the right subterm of ;
- solution: remove rule $\Downarrow(x;y) \rightsquigarrow x;\Downarrow y$

Related work

- CPS transformations

Related work

- CPS transformations
- ρ-calculus

Related work

- CPS transformations
- ρ-calculus
- rewriting logic

Conclusion

- the reduction process is made **explicit** at the **syntactic** level

Conclusion

- the reduction process is made **explicit** at the **syntactic** level
- the explicit TRS is simpler and equivalent

Conclusion

- the reduction process is made **explicit** at the **syntactic** level
- the explicit TRS is simpler and equivalent
- thus a **compilation technique** of TRSs into TRSs

Conclusion

- the reduction process is made **explicit** at the **syntactic** level
- the explicit TRS is simpler and equivalent
- thus a **compilation technique** of TRSs into TRSs
- wide spectrum of application of the idea of explicit token