Equivalence and Independence in Controlled Graph-Rewriting Processes

ICGT@STAF 2018, Toulouse

Géza Kulcsár[1], Andrea Corradini[2], Malte Lochau[1]

[1]Real-Time Systems Lab, TU Darmstadt
[2]Department of Computer Science, University of Pisa

geza.kulcsar@es.tu-darmstadt.de

June 26, 2018

Motivation

- Graph-rewriting systems are inherently non-deterministic

Motivation

- Graph-rewriting systems are inherently non-deterministic
- When designing algorithms, need to add control structures:

Motivation

- Graph-rewriting systems are inherently non-deterministic
- When designing algorithms, need to add control structures:
    - sequentialization

Motivation

- Graph-rewriting systems are inherently non-deterministic
- When designing algorithms, need to add control structures:
  - sequentialization
  - conditionals

Motivation

- Graph-rewriting systems are inherently non-deterministic
- When designing algorithms, need to add control structures:
  - sequentialization
  - conditionals
  - iteration and recursion

Motivation

- Graph-rewriting systems are inherently non-deterministic
- When designing algorithms, need to add control structures:
  - sequentialization
  - conditionals
  - iteration and recursion
  - parallelism

Motivation

- Graph-rewriting systems are inherently non-deterministic
- When designing algorithms, need to add control structures:
  - sequentialization
  - conditionals
  - iteration and recursion
  - parallelism
- $\Rightarrow$ *Controlled* graph rewriting [Bunke, Schürr, Habel & Plump, Plump & Steinert, Kreowski et al.; PROGRES, Fujaba, eMoflon]

Motivation

- Graph-rewriting systems are inherently non-deterministic
- When designing algorithms, need to add control structures:
    - sequentialization
    - conditionals
    - iteration and recursion
    - parallelism
- $\Rightarrow$ *Controlled* graph rewriting [Bunke, Schürr, Habel & Plump, Plump & Steinert, Kreowski et al.; PROGRES, Fujaba, eMoflon]

- However, mostly I/O semantics so far, not considering:

Motivation

- Graph-rewriting systems are inherently non-deterministic
- When designing algorithms, need to add control structures:
  - sequentialization
  - conditionals
  - iteration and recursion
  - parallelism
- $\Rightarrow$ *Controlled* graph rewriting [Bunke, Schürr, Habel & Plump, Plump & Steinert, Kreowski et al.; PROGRES, Fujaba, eMoflon]

- However, mostly I/O semantics so far, not considering:
  - Concurrency in graph-rewriting algorithms

Motivation

- Graph-rewriting systems are inherently non-deterministic
- When designing algorithms, need to add control structures:
  - sequentialization
  - conditionals
  - iteration and recursion
  - parallelism
- $\Rightarrow$ *Controlled* graph rewriting [Bunke, Schürr, Habel & Plump, Plump & Steinert, Kreowski et al.; PROGRES, Fujaba, eMoflon]

- However, mostly I/O semantics so far, not considering:
  - Concurrency in graph-rewriting algorithms
  - Reactive (non-terminating) specifications

In this paper...

- *Process-algebraic* (CCS-like) *syntax and semantics* for controlled graph rewriting

In this paper...

- *Process-algebraic* (CCS-like) *syntax and semantics* for controlled graph rewriting
- *Transition systems* both for control processes and for their executions on graphs

In this paper...

- *Process-algebraic* (CCS-like) *syntax and semantics* for controlled graph rewriting
- *Transition systems* both for control processes and for their executions on graphs
- *Expressiveness* of our control language (encoding *graph programs* by Habel & Plump)

In this paper...

- *Process-algebraic* (CCS-like) *syntax and semantics* for controlled graph rewriting
- *Transition systems* both for control processes and for their executions on graphs
- *Expressiveness* of our control language (encoding *graph programs* by Habel & Plump)
- Handling of *parallel rules and derivations* by *synchronization*

In this paper...

- *Process-algebraic* (CCS-like) *syntax and semantics* for controlled graph rewriting
- *Transition systems* both for control processes and for their executions on graphs
- *Expressiveness* of our control language (encoding *graph programs* by Habel & Plump)
- Handling of *parallel rules and derivations* by *synchronization*
- An *abstract* semantics, with graphs up to isomorphism

In this paper...

- *Process-algebraic* (CCS-like) *syntax and semantics* for controlled graph rewriting
- *Transition systems* both for control processes and for their executions on graphs
- *Expressiveness* of our control language (encoding *graph programs* by Habel & Plump)
- Handling of *parallel rules and derivations* by *synchronization*
- An *abstract* semantics, with graphs up to isomorphism
- *Equivalence* and *congruence* notions of CCS are reflected in our setting

Graph-Rewriting Actions

- DPO rules: $p : (L \leftarrow K \rightarrow R)$
- ...Parallel composition of DPO rules:

$$p_1|p_2 : \quad (\, L_1 + L_2 \quad \leftarrow \quad K_1 + K_2 \quad \rightarrow \quad R_1 + R_2 \,)$$

- $\mathcal{R}$ is the set of rule names, $\mathcal{R}^*$ the set of parallel rule names ranged over by ρ
- *Actions* are pairs $(\rho, N) \in Act$ where $\rho \in \mathcal{R}^*$ and $N \subseteq \mathcal{R}$ is a set of *non-applicability conditions*

Unmarked Processes

- Unmarked processes specify control over rule applications, having a CCS-like syntax

Definition (Unmarked Process Term Syntax)

$$P, Q ::= \mathbf{0} \;\mid\; \gamma.P \;\mid\; A \;\mid\; P + Q \;\mid\; P \,||\, Q$$

where γ *ranges over* Act *and* $A := P$.

Unmarked Transition System (UTS)

- The semantics of unmarked processes is an LTS where states are processes and transitions are labeled with actions

$$\text{PRE}\ \frac{}{\gamma.P \xrightarrow{\gamma} P}$$

$$\text{STOP}\ \frac{}{\mathbf{0} \xrightarrow{\checkmark} \mathbf{0}} \qquad \text{CHOICE}\ \frac{P \xrightarrow{\gamma} P'}{P + Q \xrightarrow{\gamma} P'}$$

$$\text{PAR}\ \frac{P \xrightarrow{\gamma} P'}{P \,||\, Q \xrightarrow{\gamma} P' \,||\, Q} \qquad \text{REC}\ \frac{A := P \quad P \xrightarrow{\alpha} P'}{A \xrightarrow{\alpha} P'}$$

$$\text{SYNC}\ \frac{P \xrightarrow{(\rho_1, N_1)} P' \quad Q \xrightarrow{(\rho_2, N_2)} Q'}{P \,||\, Q \xrightarrow{(\rho_1 | \rho_2, N_1 \cup N_2)} P' \,||\, Q'}$$

Rule Applications

- $G \overset{\rho @ m}{\Longrightarrow} H$

$$
\begin{tikzcd}
L \arrow[rr, phantom, shift left=6, "\overbrace{\hphantom{L \longleftarrow K \longrightarrow R}}^{\rho}"] \arrow[d, "m"'] \arrow[r, phantom, "(PO)" below=12pt] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "k"] & R \arrow[d, "n"] \\
G & D \arrow[l, "f"] \arrow[r, "g"'] \arrow[u, phantom, ""] & H
\end{tikzcd}
$$

$$
\begin{tikzcd}
L \arrow[d, "m"'] \arrow[dr, phantom, "(PO)"] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "k"] \arrow[dr, phantom, "(PO)"] & R \arrow[d, "n"] \\
G & D \arrow[l, "f"] \arrow[r, "g"'] & H
\end{tikzcd}
$$

- A *linear derivation* from G_0 is a sequence of rule applications $G_0 \overset{p_1 @ m_1}{\Longrightarrow} \dots \overset{p_n @ m_n}{\Longrightarrow} G_n$ with no parallel rules
- A *parallel derivation* from G_0 is a sequence of rule applications $G_0 \overset{\rho_1 @ m_1}{\Longrightarrow} \dots \overset{\rho_n @ m_n}{\Longrightarrow} G_n$ with (potentially) parallel rules $\rho_i = p_{i1} | \dots | p_{ik}$

Marked Transition System (MTS)

$$\text{MARK}\,\frac{P \xrightarrow{(\rho,N)} P' \quad G \xRightarrow{\rho @ m} H \quad \forall p \in N : G \not\xRightarrow{p}}{(P,G) \xrightarrow{(\rho,\delta,N)}_M (P',H)}$$

$$\delta := \begin{tikzcd} \overbrace{L & K \arrow[l, "l"'] \arrow[r, "r"] & R}^{\rho} \\ G \arrow[from=u, "m"'] \arrow[ur, phantom, "(PO)"] & D \arrow[l, "f"] \arrow[r, "g"'] \arrow[from=u, "k"] \arrow[ur, phantom, "(PO)"] & H \arrow[from=u, "n"] \end{tikzcd}$$

$$\text{STOP}\,\frac{P \xrightarrow{\checkmark} P'}{(P,G) \xrightarrow{\checkmark}_M (P',G)}$$

Equivalence of Graph-Rewriting Processes

- A *trace* of a state P is a sequence of transition labels starting from it, e.g., $(\rho_1, \delta_1, N_1)(\rho_2, \delta_2, N_2)\ldots(\rho_n, \delta_n, N_n)$
- P and Q are trace equivalent ($P \simeq^T Q$) if they have the same set of traces
- P and Q are *bisimilar* ($P \simeq^{BS} Q$) if for each $P \xrightarrow{\alpha}_M P'$, there is $Q \xrightarrow{\alpha}_M Q'$ such that $P' \simeq^{BS} Q'$ (and vice versa)

Proposition

For any unmarked processes P, Q and graph G,

- $P \simeq^{BS} Q$ *implies* $(P, G) \simeq^{BS}_M (Q, G)$
- $P \simeq^{T} Q$ *implies* $(P, G) \simeq^{T}_M (Q, G)$

Correspondence between Traces and Derivations

- Each successful trace of a marked process (P, G) uniquely identifies an *underlying parallel derivation* of $\mathcal{R}$ from G.

$$(\rho_1, \delta_1, N_1)(\rho_2, \delta_2, N_2) \ldots (\rho_n, \delta_n, N_n)\checkmark$$

Correspondence between Traces and Derivations

- Each successful trace of a marked process (P, G) uniquely identifies an *underlying parallel derivation* of $\mathcal{R}$ from G.

$$(\rho_1, \delta_1, N_1)(\rho_2, \delta_2, N_2) \ldots (\rho_n, \delta_n, N_n)\checkmark$$

- We can construct a process

$$P_{\mathcal{R}} = \mathbf{0} + \sum_{p \in \mathcal{R}} p.P_{\mathcal{R}}$$

such that $(P_{\mathcal{R}}, G)$ has as successful trace each *linear* derivation starting from a graph G...

Correspondence between Traces and Derivations

- Each successful trace of a marked process (P, G) uniquely identifies an *underlying parallel derivation* of $\mathcal{R}$ from G.

$$(\rho_1, \delta_1, N_1)(\rho_2, \delta_2, N_2) \ldots (\rho_n, \delta_n, N_n)\checkmark$$

- We can construct a process

$$P_{\mathcal{R}} = \mathbf{0} + \sum_{p \in \mathcal{R}} p.P_{\mathcal{R}}$$

  such that $(P_{\mathcal{R}}, G)$ has as successful trace each *linear* derivation starting from a graph G...

- ...and we can do the same for *parallel* derivations:

$$Q_{\mathcal{R}} = \mathbf{0} + ((\sum_{p \in \mathcal{R}} p.\mathbf{0} + \varepsilon.\mathbf{0}) \mid\mid Q_{\mathcal{R}})$$

Expressiveness: Encoding Graph Programs

- Reference: a minimal control language, *graph programs*, being computationally complete [Habel & Plump 2001]

Expressiveness: Encoding Graph Programs

- Reference: a minimal control language, *graph programs*, being computationally complete [Habel & Plump 2001]
  - choice, sequential composition, as-long-as-possible iteration

Expressiveness: Encoding Graph Programs

- Reference: a minimal control language, *graph programs*, being computationally complete [Habel & Plump 2001]
  - choice, sequential composition, as-long-as-possible iteration
- Graph programs can be encoded into unmarked processes, even without using parallel composition:
  - If $GP = \{p_1, \ldots, p_n\}$ is an elementary graph program, then $[\![GP]\!] := \sum_{i=1}^{n} p_i.\mathbf{0}$.
  - $[\![GP_1; GP_2]\!] := [\![GP_1]\!] \fatsemi [\![GP_2]\!]$.
  - $[\![GP\downarrow]\!] := A_{GP\downarrow} \in \mathcal{K}$ where $A_{GP\downarrow} := [\![GP]\!] \fatsemi A_{GP\downarrow} + \widehat{[\![GP]\!]}$.

where $P \fatsemi Q := P[Q/\mathbf{0}]$ (syntactic substitution)

Expressiveness: Encoding Graph Programs

- Reference: a minimal control language, *graph programs*, being computationally complete [Habel & Plump 2001]
  - choice, sequential composition, as-long-as-possible iteration
- Graph programs can be encoded into unmarked processes, even without using parallel composition:
  - If $GP = \{p_1, \ldots, p_n\}$ is an elementary graph program, then $[\![GP]\!] := \sum_{i=1}^{n} p_i.\mathbf{0}$.
  - $[\![GP_1; GP_2]\!] := [\![GP_1]\!] \fatsemi [\![GP_2]\!]$.
  - $[\![GP\downarrow]\!] := A_{GP\downarrow} \in \mathcal{K}$ where $A_{GP\downarrow} := [\![GP]\!] \fatsemi A_{GP\downarrow} + \widehat{[\![GP]\!]}$.

  where $P \fatsemi Q := P[Q/\mathbf{0}]$ (syntactic substitution)
- $\widehat{[\![GP]\!]}$ is a process representing the termination criterion for GP iteration
  - defined inductively using non-applicability conditions
  - successful exactly if $[\![GP]\!]$ fails

Abstract MTS

- This "concrete" MTS definition is infinitely branching

Definition (Abstract Marked Transition System)

$[G]$ denotes the isomorphism class of graph G and $[\delta]$ the isomorphism class of DPO diagram δ.

$$\textit{MARK}\ \frac{P \xrightarrow{(\rho,N)} P' \qquad G \xRightarrow{\rho @ m} H \qquad \forall p \in N : G \not\xRightarrow{p}}{(P,[G]) \xrightarrow{(\rho,[\delta],N)}_A (P',[H])}$$

Abstraction Preserves Bisimilarity and Trace Equivalence

Proposition

For any unmarked processes P, Q and graphs G, H,

- $(P, G) \simeq_M^{BS} (Q, H)$ *implies* $(P, [G]) \simeq_A^{BS} (Q, [H])$
- $(P, G) \simeq_M^{T} (Q, H)$ *implies* $(P, [G]) \simeq_A^{T} (Q, [H])$

Abstract Bisimilarity is a Congruence

Given $P, Q, R \in \mathcal{P}$ with $P \simeq^{BS} Q$.

$$(P + R, [G]) \simeq_A^{BS} (Q + R, [G]),$$
$$(P \,||\, R, [G]) \simeq_A^{BS} (Q \,||\, R, [G]), \text{ and}$$
$$(\gamma.P, [G]) \simeq_A^{BS} (\gamma.Q, [G]) \text{ for any } \gamma \in Act$$

Parallel Independence and Bisimulation

Let $P_0 := \rho_1.(\rho_2 \,||\, P) + \rho_2.(\rho_1 \,||\, P)$ and $Q_0 := \rho_1.P \,||\, \rho_2.\mathbf{0}$. There exist parallel independent applications $\rho_1 @ m_1, \rho_2 @ m_2$ on G, if and only if $(P_0, [G]) \not\simeq_{\mathcal{D}}^{BS} (Q_0, [G])$.

$$\begin{tikzcd}
& P_0 \arrow[dl, "\rho_1"'] \arrow[dr, "\rho_2"] & \\
\rho_2 \,||\, P \arrow[dr, "\rho_2"'] & & \rho_1 \,||\, P \arrow[dl, "\rho_1"] \\
& P &
\end{tikzcd}
\qquad
\begin{tikzcd}
& Q_0 \arrow[dl, "\rho_1"'] \arrow[dr, "\rho_2"] \arrow[dd, red, "\rho_1 \mid \rho_2"] & \\
\rho_2 \,||\, P \arrow[dr, "\rho_2"'] & & \rho_1.P \arrow[dl, "\rho_1"] \\
& P &
\end{tikzcd}$$

Related Work

- **Concurrent Semantics**
  - Formal notion of (concurrent) graph processes (Corradini et al. 1996, Baldan et al. 1999)
- **Semantics of control:**
  - A denotational input/output semantics for controlled graph-rewriting processes (Schürr 1996)
  - Composable graph transformation units (Kreowski et al. 2008)
  - *Graph Programs*, a graph programming language with an operational semantics and results regarding computational completeness (Plump and Habel 2001, Plump and Steinert 2009)
  - Tool support: Henshin, PROGRES, eMoflon, ...

Conclusion and Ongoing Work

- We proposed to extend graph rewriting by a process-algebraic control layer and obtained (preserved) results from CCS theory

Conclusion and Ongoing Work

- We proposed to extend graph rewriting by a process-algebraic control layer and obtained (preserved) results from CCS theory
- The concrete MTS has too much information (and is too strict)
  - $\Rightarrow$ an *abstract interpretation* semantics to obtain equivalences weaker then isomorphism

Conclusion and Ongoing Work

- We proposed to extend graph rewriting by a process-algebraic control layer and obtained (preserved) results from CCS theory
- The concrete MTS has too much information (and is too strict)
  - $\Rightarrow$ an *abstract interpretation* semantics to obtain equivalences weaker then isomorphism
- The abstract MTS cannot capture the truly concurrent semantics of graph rewriting
  - $\Rightarrow$ capture independence in an *asynchronous transition system*

Backup: Encoding Termination Criterion

$\widehat{[\![GP]\!]}$ is an inductive termination criterion:

- $\widehat{[\![GP]\!]} := (\varepsilon, \{p_1, \ldots, p_n\}).\mathbf{0}$ if $GP = \{p_1, \ldots, p_n\}$ is an elementary program;
- $\widehat{[\![GP_1; GP_2]\!]} := \widehat{[\![GP_1]\!]} + [\![GP_1]\!] \mathbin{;} \widehat{[\![GP_2]\!]}$;
- $\widehat{[\![GP \downarrow]\!]} := (p, \{p\}).\mathbf{0}$, where p is any rule.