1/18

Equivalence and Independence in Controlled Graph-Rewriting Processes

ICGT@STAF 2018, Toulouse

<u>Géza Kulcsár</u>¹, Andrea Corradini², Malte Lochau¹

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

geza.kulcsar@es.tu-darmstadt.de

June 26, 2018

• Graph-rewriting systems are inherently non-deterministic

3

2/18

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

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

(日) (同) (三) (三)

3

2/18

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

(日) (同) (三) (三)

3

2/18

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

イロト 不同下 イヨト イヨト

3

2/18

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

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

- Graph-rewriting systems are inherently non-deterministic
- When designing algorithms, need to add control structures:
 - sequentialization
 - conditionals
 - iteration and recursion
 - parallelism
- → Controlled graph rewriting [Bunke, Schürr, Habel & Plump, Plump & Steinert, Kreowski et al.; PROGRES, Fujaba, eMoflon]
- However, mostly I/O semantics so far, not considering:

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

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

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

イロト 不得下 イヨト イヨト 二日

3/18

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

イロン イロン イヨン イヨン 三日

3/18

- *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)

- *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

- *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

- *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

4/18

Graph-Rewriting Actions

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

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

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

- 32

5/18

Unmarked Processes

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

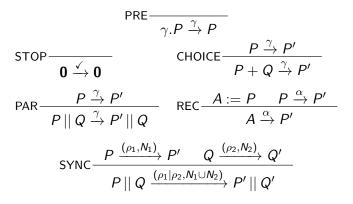
Definition (Unmarked Process Term Syntax)

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

where γ ranges over Act and A := P.

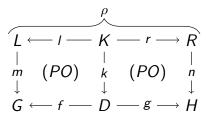
Unmarked Transition System (UTS)

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



Rule Applications



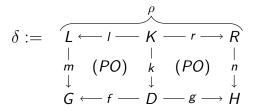


- A linear derivation from G_0 is a sequence of rule applications $G_0 \xrightarrow{p_1 @ m_1} \dots \xrightarrow{p_n @ m_n} G_n$ with no parallel rules
- A parallel derivation from G_0 is a sequence of rule applications $G_0 \xrightarrow{\rho_1 @ m_1} \dots \xrightarrow{\rho_n @ m_q} G_n$ with (potentially)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

Marked Transition System (MTS)

MARK
$$\xrightarrow{P \xrightarrow{(\rho,N)} P'} G \xrightarrow{\rho@m} H \quad \forall p \in N : G \xrightarrow{p}$$

 $(P,G) \xrightarrow{(\rho,\delta,N)} M(P',H)$



$$\mathsf{STOP} \xrightarrow{P \xrightarrow{\checkmark} P'} (P, G) \xrightarrow{\checkmark} M(P', G)$$

8/18

Equivalence of Graph-Rewriting Processes

- A trace of a state P is a sequence of transition labels starting from it, e.g., (ρ₁, δ₁, N₁)(ρ₂, δ₂, N₂)...(ρ_n, δ_n, N_n)
- P and Q are trace equivalent (P ≃^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*.

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

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*.

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

• 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 *R* from *G*.

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

• 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:

$$Q_{\mathcal{R}} = \mathbf{0} + \left(\left(\sum_{\boldsymbol{p} \in \mathcal{R}} \boldsymbol{p}.\mathbf{0} + \varepsilon.\mathbf{0} \right) || Q_{\mathcal{R}} \right)$$

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

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

- 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 $GP = \{p_1, \dots, p_n\}$ is an elementary graph program, then $\llbracket GP \rrbracket := \sum_{i=1}^n p_i . \mathbf{0}.$
 - $\llbracket GP_1; GP_2 \rrbracket := \llbracket GP_1 \rrbracket \operatorname{sp} \llbracket GP_2 \rrbracket.$
 - $\llbracket GP \downarrow \rrbracket := A_{GP\downarrow} \in \mathcal{K}$ where $A_{GP\downarrow} := \llbracket GP \rrbracket \ ; A_{GP\downarrow} + \widehat{\llbracket GP \rrbracket}$.

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

- 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 $GP = \{p_1, \dots, p_n\}$ is an elementary graph program, then $\llbracket GP \rrbracket := \sum_{i=1}^n p_i . \mathbf{0}.$
 - $\llbracket GP_1; GP_2 \rrbracket := \llbracket GP_1 \rrbracket \operatorname{\mathfrak{g}} \llbracket GP_2 \rrbracket.$
 - $\llbracket GP \downarrow \rrbracket := A_{GP\downarrow} \in \mathcal{K}$ where $A_{GP\downarrow} := \llbracket GP \rrbracket ; A_{GP\downarrow} + \widehat{\llbracket GP \rrbracket}$.

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

- [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 .

$$MARK \xrightarrow{P \xrightarrow{(\rho,N)} P'} G \xrightarrow{\rho@m} H \quad \forall p \in N : G \xrightarrow{p} (P, [G]) \xrightarrow{(\rho,[\delta],N)} (P', [H])$$

イロト 不得下 イヨト イヨト 二日

Abstraction Preserves Bisimilarity and Trace Equivalence

Proposition

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

- $(P,G) \simeq^{BS}_{M} (Q,H)$ implies $(P,[G]) \simeq^{BS}_{A} (Q,[H])$
- $(P,G) \simeq_{M}^{T} (Q,H)$ implies $(P,[G]) \simeq_{A}^{T} (Q,[H])$

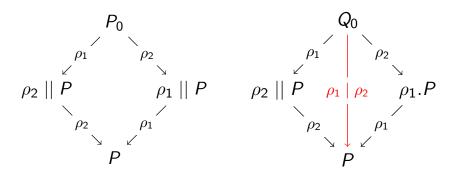
Abstract Bisimilarity is a Congruence

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

 $\begin{aligned} (P+R,[G]) \simeq^{BS}_{A} (Q+R,[G]), \\ (P \parallel R,[G]) \simeq^{BS}_{A} (Q \parallel R,[G]), \text{and} \\ (\gamma.P,[G]) \simeq^{BS}_{A} (\gamma.Q,[G]) \text{ for any } \gamma \in Act \end{aligned}$

Parallel Independence and Bisimulation

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



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

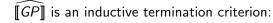
Conclusion and Ongoing Work

- 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*

イロト 不得下 イヨト イヨト 二日

18/18

Backup: Encoding Termination Criterion



- $\widehat{\llbracket GP \rrbracket} := (\varepsilon, \{p_1, \dots, p_n\}).\mathbf{0}$ if $GP = \{p_1, \dots, p_n\}$ is an elementary program;
- $\llbracket \widehat{GP_1}; \widehat{GP_2} \rrbracket := \llbracket \widehat{\llbracket GP_1} \rrbracket + \llbracket GP_1 \rrbracket \operatorname{\mathfrak{s}} \widehat{\llbracket GP_2} \rrbracket;$
- $\llbracket GP \downarrow \rrbracket := (p, \{p\}).0$, where p is any rule.