A Graph-Transformational Approach for Proving the Correctness of Reductions between NP-Problems

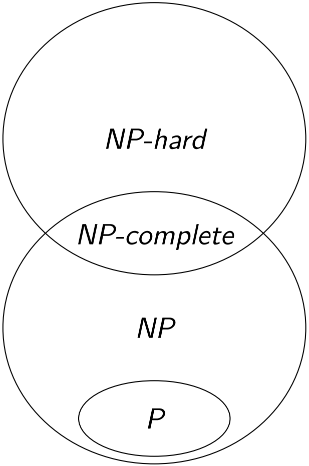
Hans-Jörg Kreowski, Sabine Kuske,
Aaron Lye, and Aljoscha Windhorst

University of Bremen
{kreo,kuske,lye,windhorst}@uni-bremen.de

06.07.2022

13th International Workshop on Graph Computation Models

NP



- $P = NP$ is the 1-Mil-\$-problem of complexity theory. Hypothesis: $P \neq NP$
- P/NP: all decision problems solvable by deterministic/nondeterministic TMs with polynomially bounded computation lengths
- NP contains many theoretically & practically important problems
  – often graph problems
  – often NP-complete
- reductions are key concepts
- NP-complete: maxima w.r.t. reduction

NP and GraTra

- TMs are gratra units[1]
- NP = all decision problems solvable by polynomial gratra systems/units

In this paper: based on gratra modeling of NP problems and their reduction, a proof technique is proposed to prove the correctness of reductions (continuing a former attempt[2])

[1] Kreowski and Kuske. Graph multiset transformation – A new framework for massively parallel computation inspired by DNA computing. *Natural Computing*, 10(2):961–986, 2011.

[2] Ermler, Kuske, Luderer, and von Totth. A graph transformational view on reductions in NP. *Electronic Communications of the EASST*, 61, 2013.

Graph Transformation Units (GTU)

gtu
initial: ⟨graph class expression⟩
rules: ⟨gratra rules (may have NAC)⟩
cond: ⟨control condition⟩
terminal: ⟨graph class expression⟩

Convention: $(I_{gtu}, P_{gtu}, C_{gtu}, T_{gtu})$

We assume:

1. membership for our graph class expressions is decidable in polynomial time
2. control conditions are regular expressions extended by *as long as possible* (!) which allows a stepwise control.

Semantics:
$SEM(gtu) = (SEM(I) \times SEM(T)) \cap \xRightarrow[P,C]{*}$

where $\xRightarrow[P,C]{*}$ is the derivation relation applying P statisfying C.

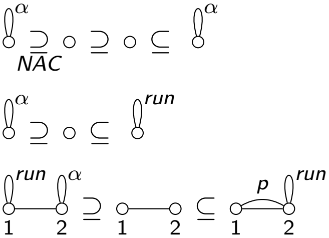
A derivation $G \xRightarrow[P,C]{*} H$ with $G \in SEM(I)$ and $H \in SEM(T)$ is called successful, denoted by $G \xRightarrow[gtu]{*} H$.

Example

hampath●

initial: *standard*

rules:



cond: 1!; 2; 3*

terminal: *forbidden*(α)

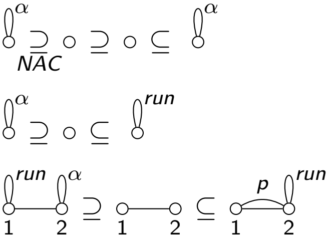
standard = *undirected* ∩ *unlabeled* ∩ *simple* ∩ *loop-free*

Example

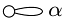
hampath●

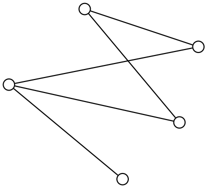
initial: *standard*

rules:



cond: $1!;\ 2;\ 3^*$

terminal: *forbidden*()



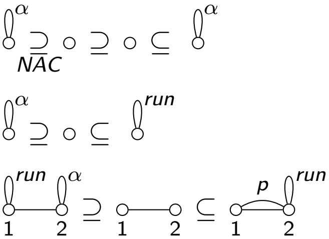
$\mathit{standard} = \mathit{undirected} \cap \mathit{unlabeled} \cap \mathit{simple} \cap \mathit{loop\text{-}free}$

Example

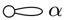
hampath●

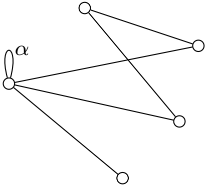
initial: *standard*

rules:



cond: 1!; 2; 3*

terminal: *forbidden*()



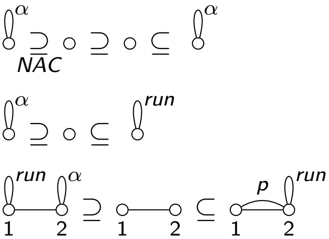
standard = *undirected* ∩ *unlabeled* ∩ *simple* ∩ *loop-free*

Example

hampath●

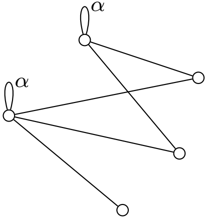
initial: *standard*

rules:



cond: 1!; 2; 3^*

terminal: *forbidden*(α)



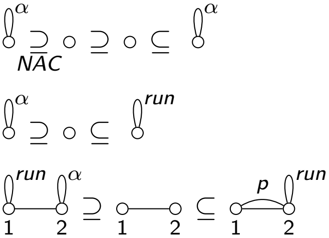
standard = *undirected* ∩ *unlabeled* ∩ *simple* ∩ *loop-free*

Example

hampath●

initial: *standard*

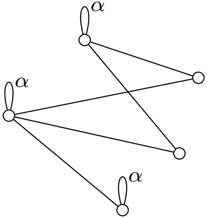
rules:



cond: 1!; 2; 3*

terminal: *forbidden*(α)

standard = *undirected* ∩ *unlabeled* ∩ *simple* ∩ *loop-free*

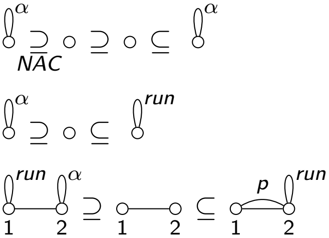


Example

hampath●

initial: *standard*

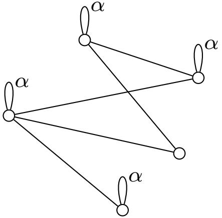
rules:



cond: 1!; 2; 3*

terminal: *forbidden*( )

standard = *undirected* ∩ *unlabeled* ∩ *simple* ∩ *loop-free*

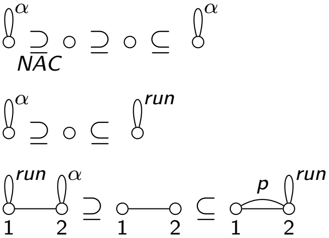


Example

hampath●

initial: *standard*

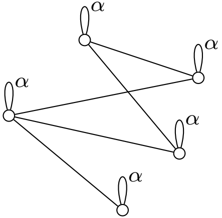
rules:



cond: 1!; 2; 3*

terminal: *forbidden*(α)

standard $=$ *undirected* $\cap$ *unlabeled* $\cap$ *simple* $\cap$ *loop-free*

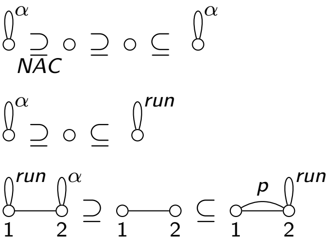


Example

hampath●

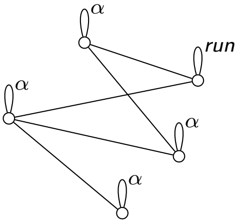
initial: *standard*

rules:



cond: 1!; 2; 3*

terminal: *forbidden*(α)



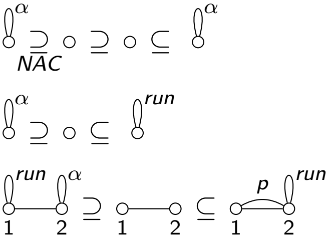
standard = *undirected* ∩ *unlabeled* ∩ *simple* ∩ *loop-free*

Example

hampath●

initial: *standard*

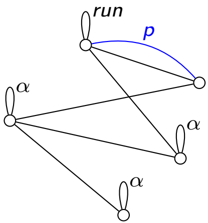
rules:



cond: 1!; 2; 3*

terminal: *forbidden*(α)

standard $=$ *undirected* $\cap$ *unlabeled* $\cap$ *simple* $\cap$ *loop-free*

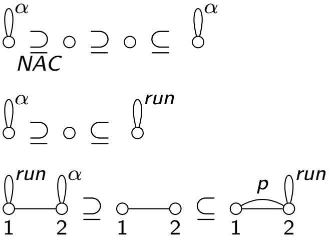


Example

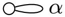
hampath●

initial: *standard*

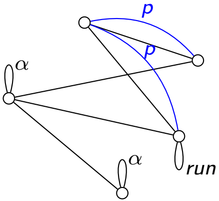
rules:



cond: 1!; 2; 3*

terminal: *forbidden*()

standard = *undirected* ∩ *unlabeled* ∩ *simple* ∩ *loop-free*

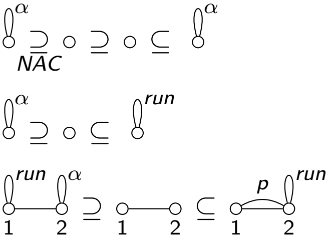


Example

hampath●

initial: *standard*

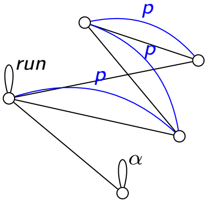
rules:



cond: 1!; 2; 3*

terminal: *forbidden*(α)

standard = *undirected* ∩ *unlabeled* ∩ *simple* ∩ *loop-free*

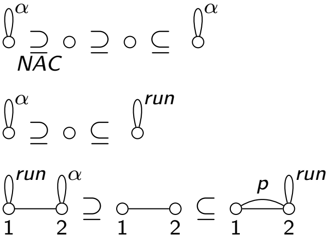


Example

hampath●

initial: *standard*

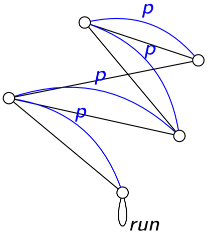
rules:



cond: 1!; 2; 3*

terminal: *forbidden*(α)

standard = *undirected* ∩ *unlabeled* ∩ *simple* ∩ *loop-free*



Modeling NP-problems

The decision problem of *gtu* $DEC(gtu)\colon SEM(I) \to BOOL$ yields

- $TRUE$ for $G \in SEM(I)$ if $(G, H) \in SEM(gtu)$, i.e. there is a *successful* derivation $G \underset{gtu}{\overset{*}{\Longrightarrow}} H$, for some $H \in SEM(T)$
- $FALSE$ otherwise.

Observation

Then $DEC(gtu) \in \mathrm{NP}$ if gtu is polynomial.

gtu is polynomial[3] if there is a polynomial p for each permitted derivation $G \underset{P,C?}{\overset{n}{\Longrightarrow}} H$ such that $n \leq p(size(G))$.

A derivation $G \underset{P}{\overset{*}{\Longrightarrow}} H$ with $G \in SEM(I)$ following the stepwise control is called permitted.

[3] special case compared to Ermler et. al. 2013, due to our assumptions on gtus.

Graph-transformatorial Reductions between NP-Problems

If the NP-problems are given by polynomial graph transformation units gtu, gtu' (called source and target), then a reduction can be modeled as a functional and polynomial graph transformation unit.

Graph-transformatorial Reductions between NP-Problems

If the NP-problems are given by polynomial graph transformation units gtu, gtu' (called source and target), then a reduction can be modeled as a functional and polynomial graph transformation unit.

funct is functional if

1. every permitted derivation can be prolonged into a successful one and
2. for every initial graph G every successful derivation from G yields the same terminal graph (up to isomorphism).

funct is also denoted as
$SEM(funct)\colon I_{funct} \to T_{funct}$.
The resulting graph is denoted by $funct(G)$.

Graph-transformatorial Reductions between NP-Problems

If the NP-problems are given by polynomial graph transformation units gtu, gtu' (called source and target), then a reduction can be modeled as a functional and polynomial graph transformation unit.

funct is functional if

1. every permitted derivation can be prolonged into a successful one and
2. for every initial graph G every successful derivation from G yields the same terminal graph (up to isomorphism).

funct is also denoted as $SEM(funct)\colon I_{funct} \to T_{funct}$.
The resulting graph is denoted by $funct(G)$.

$red = (I_{gtu}, P_{red}, C_{red}, I_{gtu'})$ is a *reduction* of gtu to gtu' if the following correctness condition holds for all $G \in SEM(I_{gtu})$:

$$(G, H) \in SEM(gtu) \text{ for some } H \in SEM(T_{gtu})$$

$$\text{if and only if}$$

$$(red(G), H') \in SEM(gtu') \text{ for some } H' \in SEM(T_{gtu'}).$$

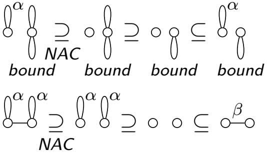
$$\begin{tikzcd}
G \arrow[r, Rightarrow, "*", "red"'] \arrow[d, Rightarrow, "*"', "gtu"] & red(G) \arrow[d, Rightarrow, "*"', "gtu'"] \\
H \arrow[r, rightsquigarrow, shift left, "forward"] \arrow[r, leftsquigarrow, shift right, "backward"'] & H'
\end{tikzcd}$$

Reduction from *clique* to *independent-set*

clique•

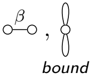
initial: *standard* + *bound*$(\mathbb{N})$

rules:



cond: 1!; 2!

terminal: *forbidden*(



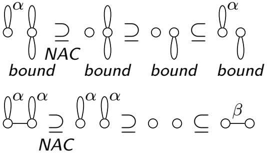
)

Reduction from *clique* to *independent-set*

clique●

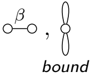
initial: *standard* + *bound*(ℕ)

rules:



cond: 1!; 2!

terminal: *forbidden*(

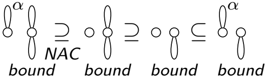


)

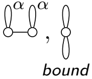
independent-set●

initial: *standard* + *bound*(ℕ)

rules:



terminal: *forbidden*(

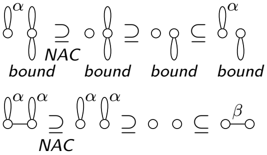


)

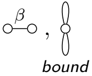
Reduction from *clique* to *independent-set*

clique ●

initial: *standard* + *bound*(ℕ)

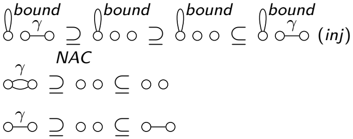
rules: 

cond: 1!; 2!

terminal: *forbidden*()

clique-to-independent-set ●

initial: I_{clique}

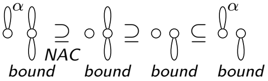
rules: 

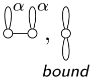
cond: 1!; 2!; 3!

terminal: $I_{independent\text{-}set}$

independent-set ●

initial: *standard* + *bound*(ℕ)

rules: 

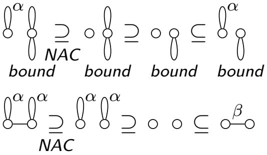
terminal: *forbidden*()

Reduction from *clique* to *independent-set*

clique●

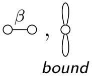
initial: *standard* + *bound*$(\mathbb{N})$

rules:

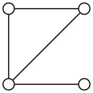


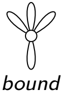
cond: 1!; 2!

terminal: *forbidden*(



)

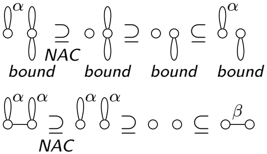




Reduction from *clique* to *independent-set*

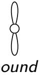
clique●

initial: *standard* + *bound*(ℕ)

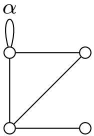
rules: 

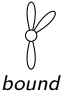
cond: 1!; 2!

terminal: *forbidden*(

, 

)



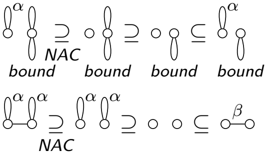


Reduction from *clique* to *independent-set*

clique●

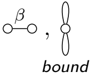
initial: *standard* + *bound*($\mathbb{N}$)

rules:

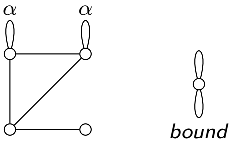


cond: 1!; 2!

terminal: *forbidden*(



)

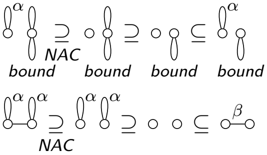


Reduction from *clique* to *independent-set*

clique•

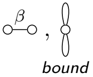
initial: *standard* + *bound*(ℕ)

rules:

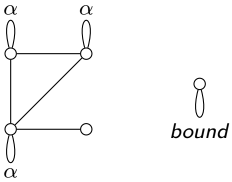


cond: 1!; 2!

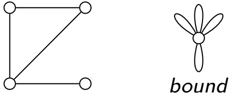
terminal: *forbidden*(



)



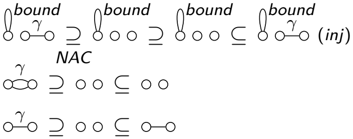
Reduction from *clique* to *independent-set*



clique-to-independent-set●

initial: I_{clique}

rules:



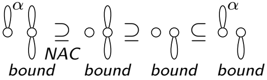
cond: 1!; 2!; 3!

terminal: $I_{independent\text{-}set}$

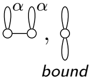
independent-set●

initial: *standard* + *bound*$(\mathbb{N})$

rules:

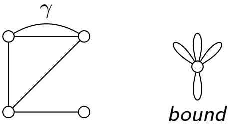


terminal: *forbidden*(



)

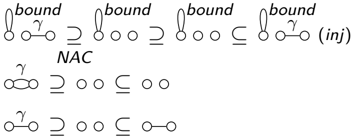
Reduction from *clique* to *independent-set*



clique-to-independent-set●

initial: I_{clique}

rules:



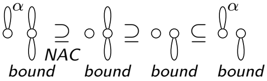
cond: 1!; 2!; 3!

terminal: $I_{\mathit{independent\text{-}set}}$

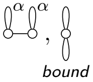
independent-set●

initial: *standard* + *bound*$(\mathbb{N})$

rules:

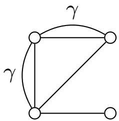


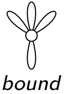
terminal: *forbidden*(



)

Reduction from *clique* to *independent-set*

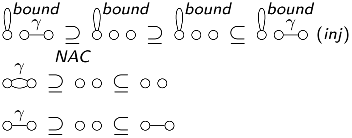




clique-to-independent-set●

initial: I_{clique}

rules:



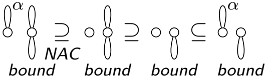
cond: 1!; 2!; 3!

terminal: $I_{independent\text{-}set}$

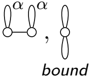
independent-set●

initial: *standard* + *bound*$(\mathbb{N})$

rules:

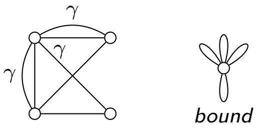


terminal: *forbidden*(



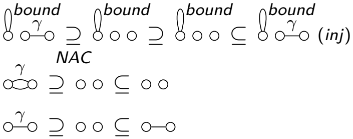
)

Reduction from *clique* to *independent-set*



clique-to-independent-set●

initial: I_{clique}

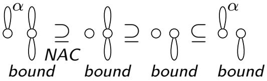
rules: 

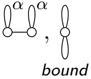
cond: 1!; 2!; 3!

terminal: $I_{independent\text{-}set}$

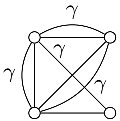
independent-set●

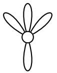
initial: *standard* + *bound*($\mathbb{N}$)

rules: 

terminal: *forbidden*()

Reduction from *clique* to *independent-set*

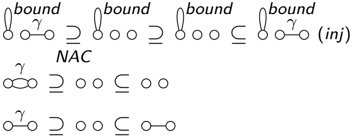



bound

clique-to-independent-set ●

initial: I_{clique}

rules:



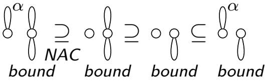
cond: 1!; 2!; 3!

terminal: $I_{independent\text{-}set}$

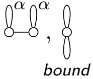
independent-set ●

initial: *standard* + *bound*($\mathbb{N}$)

rules:

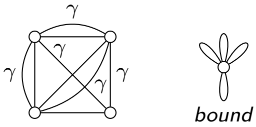


terminal: *forbidden*(



)

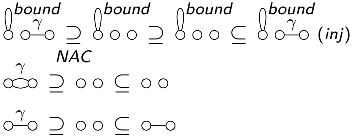
Reduction from *clique* to *independent-set*



clique-to-independent-set ●

initial: I_{clique}

rules:



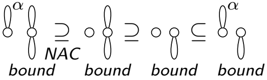
cond: 1!; 2!; 3!

terminal: $I_{independent\text{-}set}$

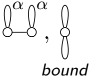
independent-set ●

initial: *standard* + *bound*(ℕ)

rules:

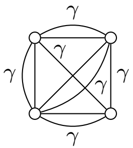


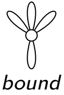
terminal: *forbidden*(



)

Reduction from *clique* to *independent-set*

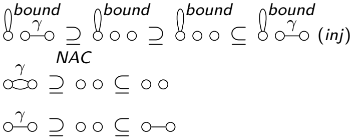




clique-to-independent-set●

initial: I_{clique}

rules:



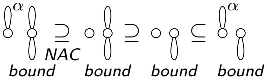
cond: 1!; 2!; 3!

terminal: $I_{independent\text{-}set}$

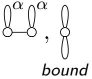
independent-set●

initial: *standard* + *bound*$(\mathbb{N})$

rules:

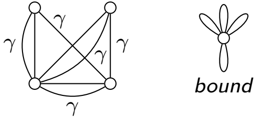


terminal: *forbidden*(



)

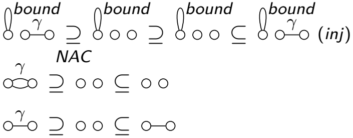
Reduction from *clique* to *independent-set*



clique-to-independent-set●

initial: I_{clique}

rules:



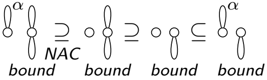
cond: 1!; 2!; 3!

terminal: $I_{independent\text{-}set}$

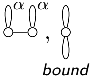
independent-set●

initial: *standard* + *bound*$(\mathbb{N})$

rules:

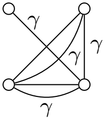


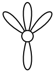
terminal: *forbidden*(



)

Reduction from *clique* to *independent-set*

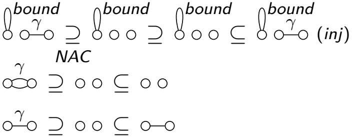




bound

clique-to-independent-set ●

initial: I_{clique}

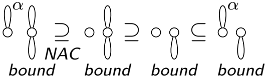
rules: 

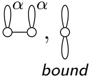
cond: 1!; 2!; 3!

terminal: $I_{independent\text{-}set}$

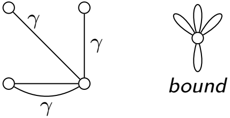
independent-set ●

initial: *standard* + *bound*($\mathbb{N}$)

rules: 

terminal: *forbidden*()

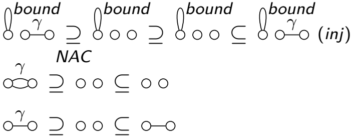
Reduction from *clique* to *independent-set*



clique-to-independent-set●

initial: I_{clique}

rules:



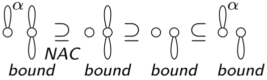
cond: 1!; 2!; 3!

terminal: $I_{independent\text{-}set}$

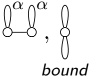
independent-set●

initial: *standard* + *bound*(ℕ)

rules:

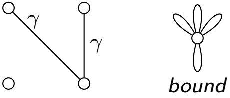


terminal: *forbidden*(



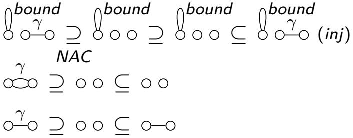
)

Reduction from *clique* to *independent-set*



clique-to-independent-set●

initial: I_{clique}

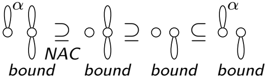
rules: 

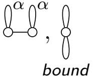
cond: 1!; 2!; 3!

terminal: $I_{independent\text{-}set}$

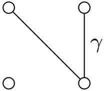
independent-set●

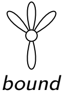
initial: *standard* + *bound*($\mathbb{N}$)

rules: 

terminal: *forbidden*()

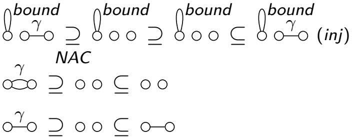
Reduction from *clique* to *independent-set*





clique-to-independent-set●

initial: I_{clique}

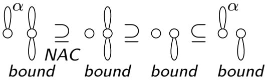
rules: 

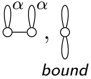
cond: 1!; 2!; 3!

terminal: $I_{independent\text{-}set}$

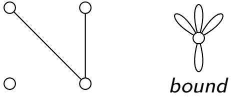
independent-set●

initial: *standard* + *bound*($\mathbb{N}$)

rules: 

terminal: *forbidden*()

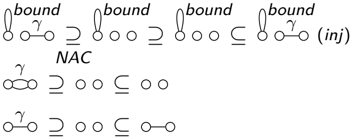
Reduction from *clique* to *independent-set*



clique-to-independent-set●

initial: I_{clique}

rules:



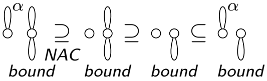
cond: 1!; 2!; 3!

terminal: $I_{independent\text{-}set}$

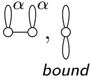
independent-set●

initial: *standard* + *bound*$(\mathbb{N})$

rules:

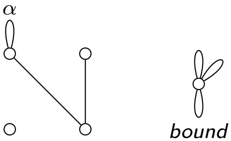


terminal: *forbidden*(



)

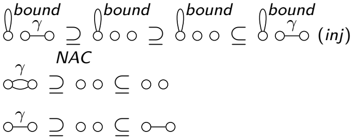
Reduction from *clique* to *independent-set*



clique-to-independent-set●

initial: I_{clique}

rules:



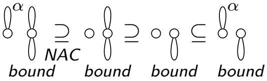
cond: 1!; 2!; 3!

terminal: $I_{independent\text{-}set}$

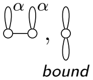
independent-set●

initial: *standard* + *bound*$(\mathbb{N})$

rules:

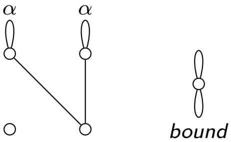


terminal: *forbidden*(



)

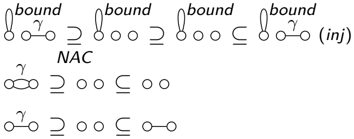
Reduction from *clique* to *independent-set*



clique-to-independent-set●

initial: I_{clique}

rules:



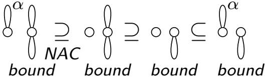
cond: 1!; 2!; 3!

terminal: $I_{independent\text{-}set}$

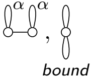
independent-set●

initial: *standard* + *bound*$(\mathbb{N})$

rules:

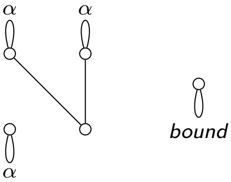


terminal: *forbidden*(



)

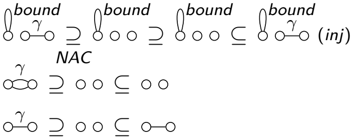
Reduction from *clique* to *independent-set*



clique-to-independent-set●

initial: I_{clique}

rules:



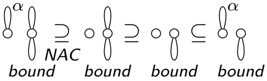
cond: 1!; 2!; 3!

terminal: $I_{independent\text{-}set}$

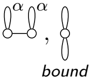
independent-set●

initial: *standard* + *bound*$(\mathbb{N})$

rules:



terminal: *forbidden*(



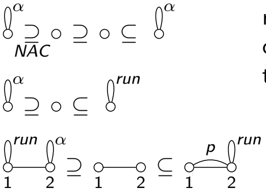
)

Reduction from *hampath* to *2-bounded-spantree*

hampath●

initial: *standard*

rules:



cond: 1!; 2; 3*

terminal: *forbidden*(α)

hampath-to-2-bounded-spantree●

initial: $I_{hampath}$

rules:

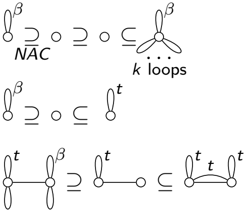
cond: ϵ

terminal: $I_{2\text{-bounded-spantree}}$

k-bounded-spantree●

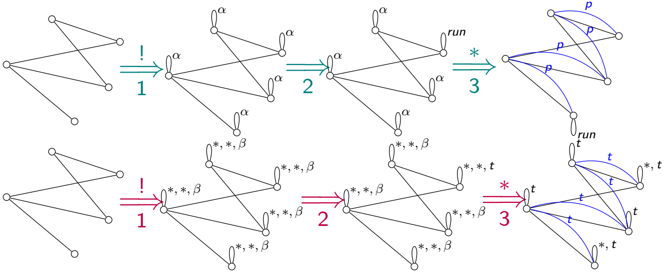
initial: *standard*

rules:



cond: 1!; 2; 3*

terminal: *forbidden*(β)



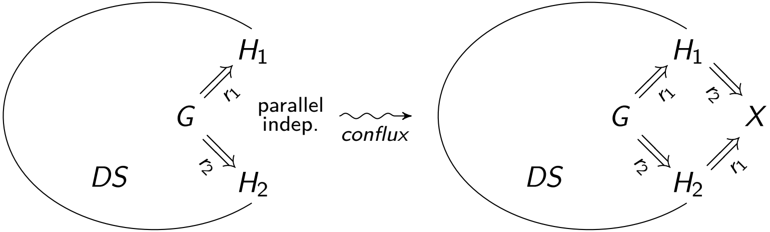
How can we prove the Correctness of the Reduction?

$$\begin{tikzcd}
G \arrow[r, Rightarrow, "*", "red"'] \arrow[d, Rightarrow, "*"', "gtu"] & red(G) & {} \arrow[r, rightsquigarrow, "forward"'] & {} & G \arrow[r, Rightarrow, "*", "red"'] \arrow[d, Rightarrow, "*"', "gtu"] & red(G) \arrow[d, Rightarrow, "*"', "gtu'"] & {} & {} \arrow[l, rightsquigarrow, "backward"] & G \arrow[r, Rightarrow, "*", "red"'] & red(G) \arrow[d, Rightarrow, "*"', "gtu'"] \\
H & & & & H & H' & & & & H'
\end{tikzcd}$$

A derivation structure is a finite unlabeled directed graph DS, such that $V_{DS} \subseteq \mathcal{G}_{\Sigma}$ and every edge $e \in E_{DS}$ is a direct derivation $e = (G \underset{r}{\Longrightarrow} H)$ with $s_{DS}(e) = G$ and $t_{DS}(e) = H$.

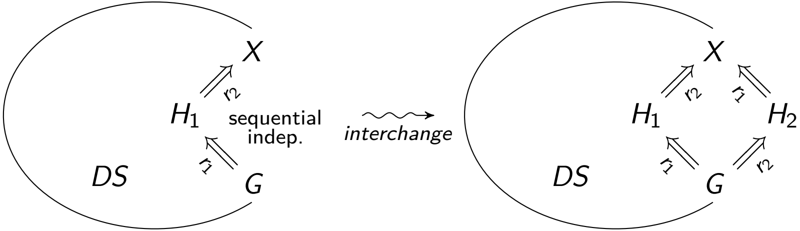
5 operations on derivation structures.
The operations add derivations to given derivation structures so that the results are derivation structures.

conflux



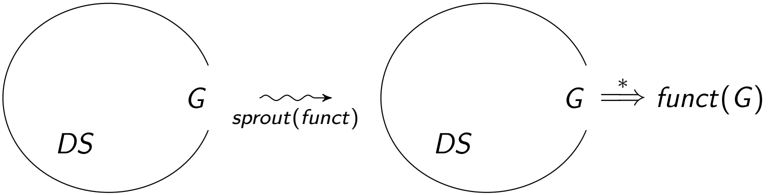
Let $DS \in \mathcal{DS}$ with two parallel independent direct derivations $G \underset{r_i}{\Longrightarrow} H_i$ for $i = 1, 2$ as substructure. Let DS' be the enlargement of DS by the two corresponding direct derivations $d_1 = (H_1 \underset{r_2}{\Longrightarrow} X)$ and $d_2 = (H_2 \underset{r_1}{\Longrightarrow} X)$, i.e., $DS' = DS \cup d_1 \cup d_2$. Then $(DS, DS') \in \mathit{conflux}$.

interchange



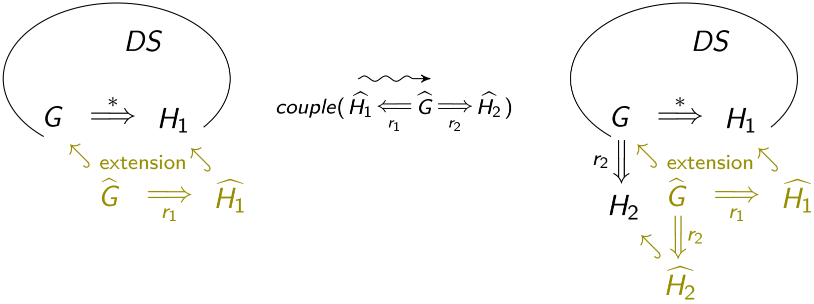
Let $DS \in \mathcal{DS}$ with two sequential independent direct derivations $G \underset{r_1}{\Longrightarrow} H_1 \underset{r_2}{\Longrightarrow} X$ as substructure. Let DS' be the enlargement of DS by the corresponding derivation $d = (G \underset{r_2}{\Longrightarrow} H_2 \underset{r_1}{\Longrightarrow} X)$, i.e., $DS' = DS \cup d$. Then $(DS, DS') \in$ *interchange*.

sprout(*funct*)



Let *funct* be a functional graph transformation unit, $DS \in \mathcal{DS}$ and $G \in V_{DS}$. Let DS' be the enlargement of DS by attaching the derivation $d = (G \overset{*}{\Longrightarrow} funct(G))$ at G, i.e., $DS' = DS \cup d$. Then $(DS, DS') \in sprout(funct)$.

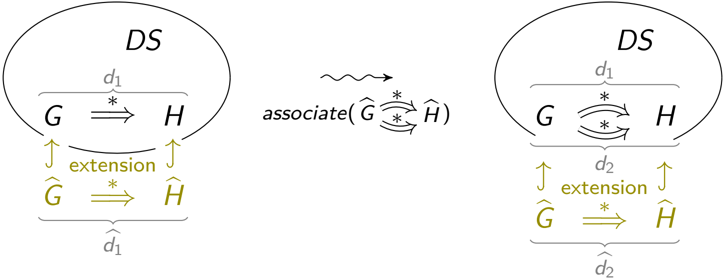
$couple(span)$



DS
$G \overset{*}{\Longrightarrow} H_1$
extension
$\widehat{G} \underset{r_1}{\Longrightarrow} \widehat{H_1}$
$couple(\,\widehat{H_1} \underset{r_1}{\Longleftarrow} \widehat{G} \underset{r_2}{\Longrightarrow} \widehat{H_2}\,)$
DS
$G \overset{*}{\Longrightarrow} H_1$
r_2
H_2
extension
$\widehat{G} \underset{r_1}{\Longrightarrow} \widehat{H_1}$
r_2
$\widehat{H_2}$

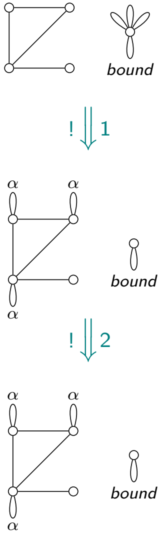
Let *span* be a pair of parallel independent direct derivations $\hat{d}_i = (\hat{G} \underset{r_i}{\Longrightarrow} \hat{H}_i)$ for $i = 1, 2$, $DS \in \mathcal{DS}$, and $G \overset{*}{\Longrightarrow} H_1$ be a substructure of DS extending $\hat{d}_1$. Let DS' be the enlargement of DS by adding one corresponding extension $d_2 = (G \underset{r_2}{\Longrightarrow} H_2)$ of $\hat{d}_2$, i.e., $DS' = DS \cup d_2$, provided it exists. Then $(DS, DS') \in couple(span)$.

associate(*pair*)

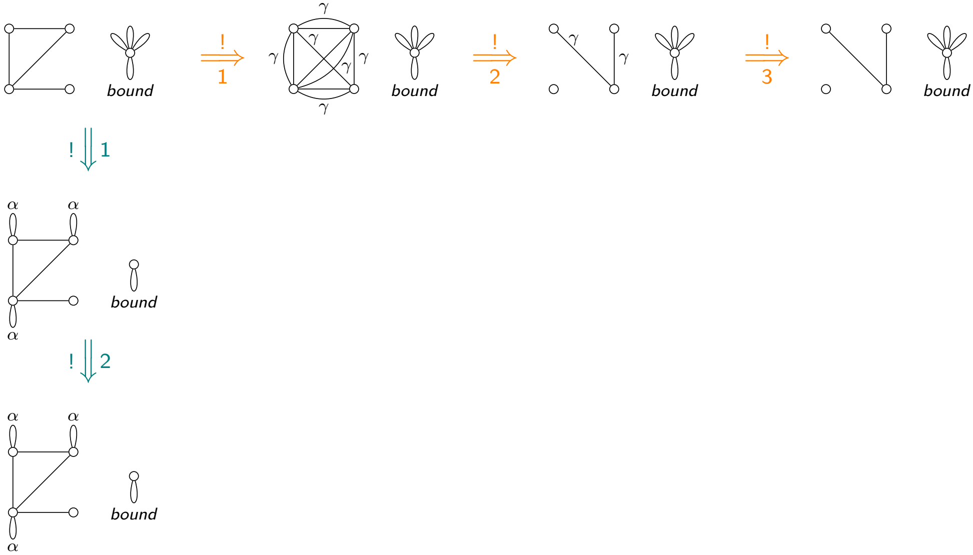


Let *pair* be a pair of derivations $\hat{d}_i = (\hat{G} \overset{*}{\Longrightarrow} \hat{H})$ for $i = 1, 2$, $DS \in \mathcal{DS}$, $d_1 = (G \overset{*}{\Longrightarrow} H)$ be a substructure of DS extending $\hat{d}_1$. Let DS' be the enlargement of DS by adding one extension d_2 of $\hat{d}_2$ from G to H, i.e., $DS' = DS \cup d_2$, provided it exists. Then $(DS, DS') \in \mathit{associate}(\mathit{pair})$.

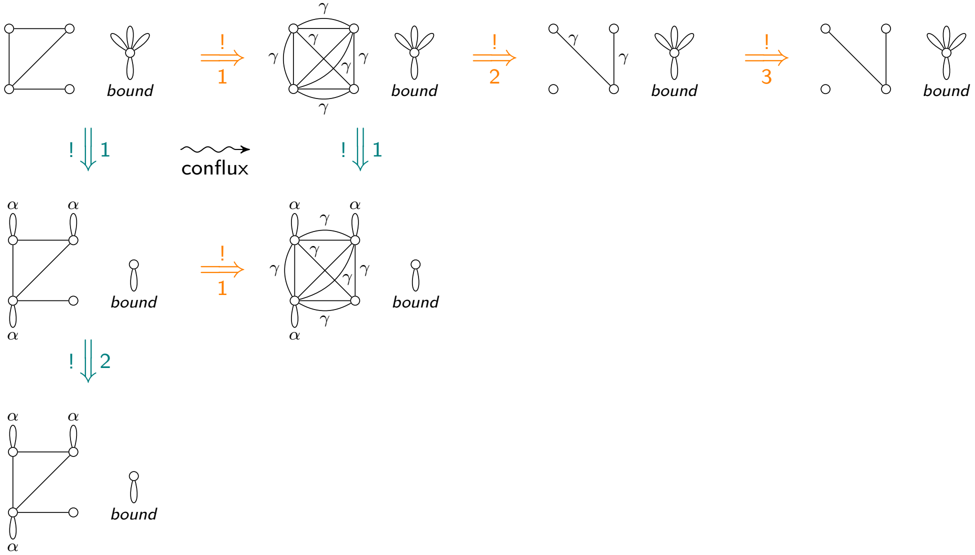
Correctness of the Reduction *clique-to-independent-set*



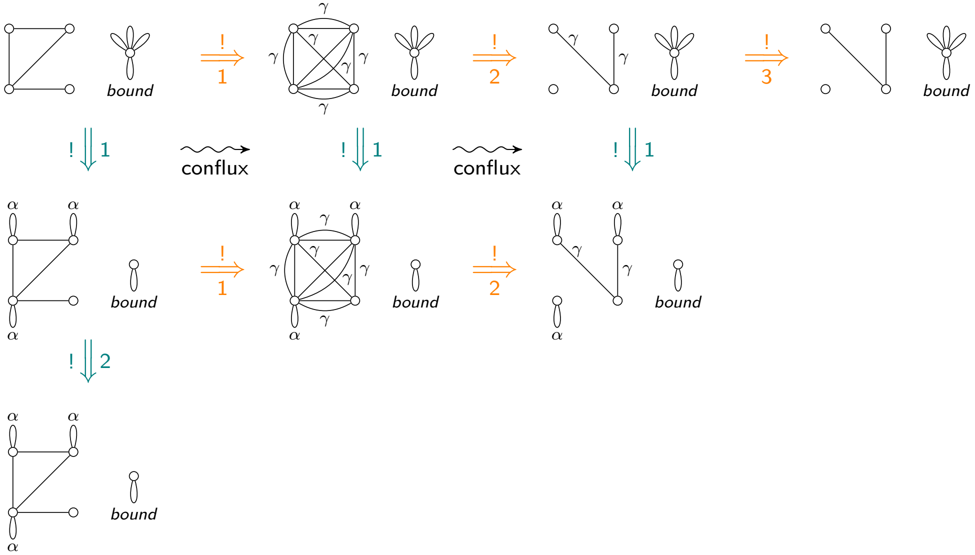
Correctness of the Reduction *clique-to-independent-set*



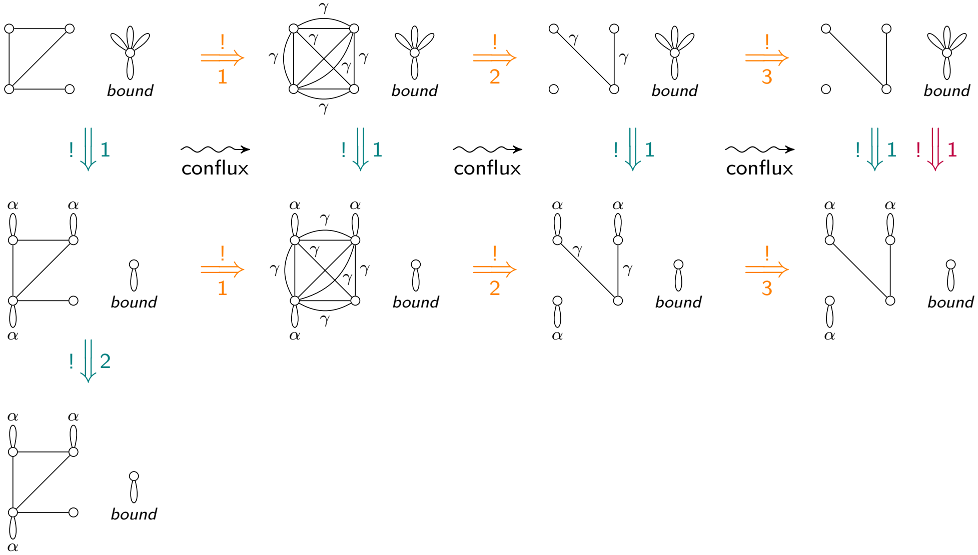
Correctness of the Reduction *clique-to-independent-set*



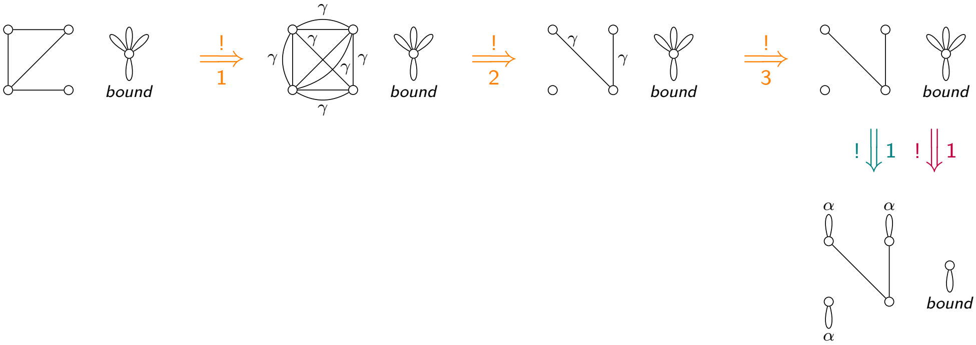
Correctness of the Reduction *clique-to-independent-set*



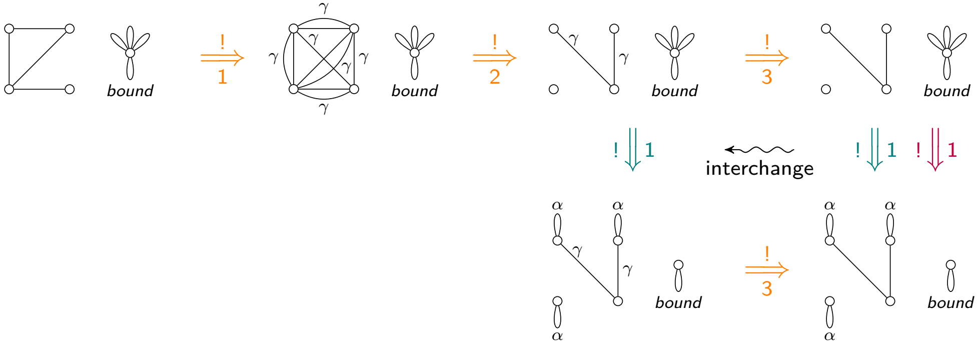
Correctness of the Reduction *clique-to-independent-set*



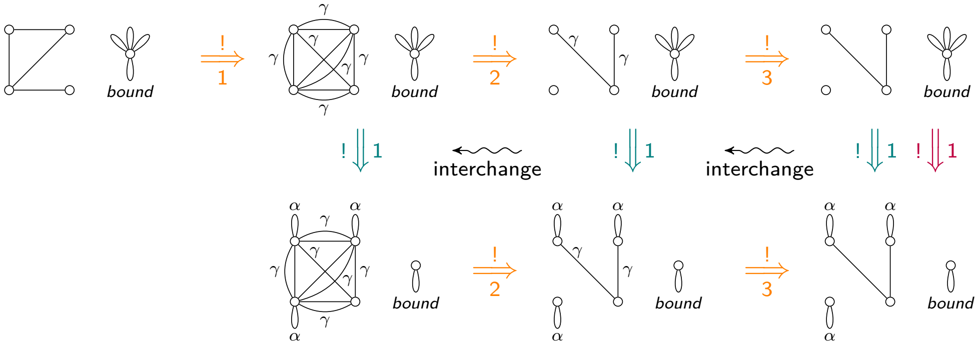
Correctness of the Reduction *clique-to-independent-set*



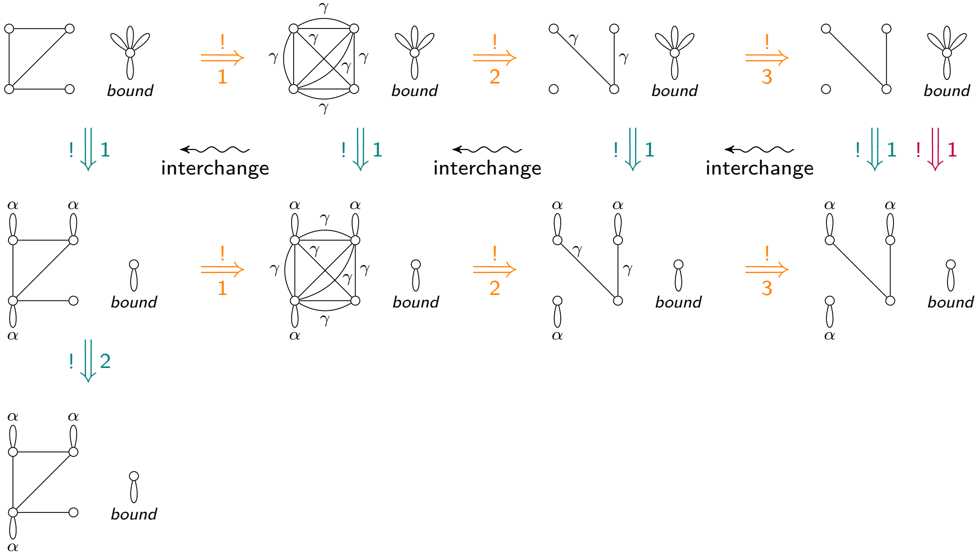
Correctness of the Reduction *clique-to-independent-set*



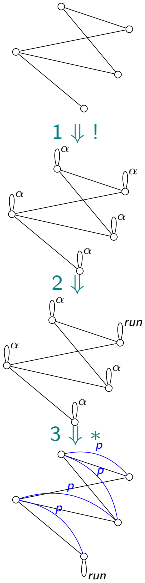
Correctness of the Reduction *clique-to-independent-set*



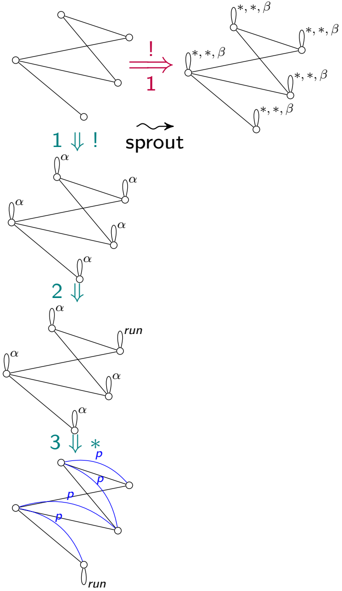
Correctness of the Reduction *clique-to-independent-set*



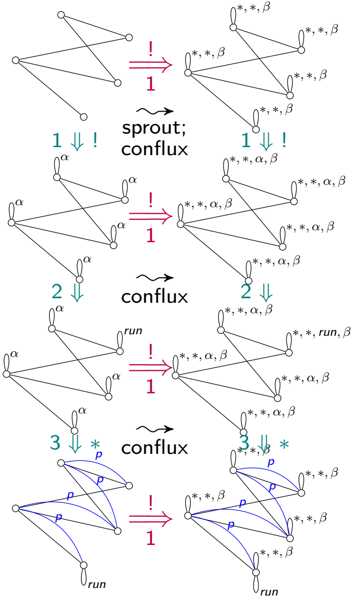
Correctness of the Reduction *hampath-to-2-bounded-spantree*



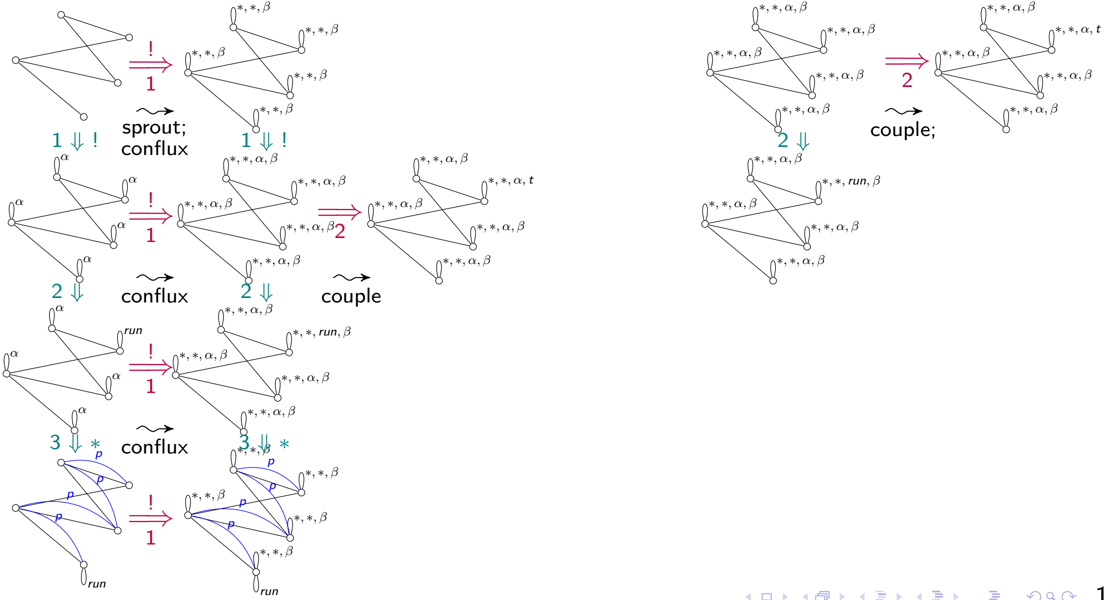
Correctness of the Reduction *hampath-to-2-bounded-spantree*



Correctness of the Reduction *hampath-to-2-bounded-spantree*



Correctness of the Reduction *hampath-to-2-bounded-spantree*



sprout; conflux

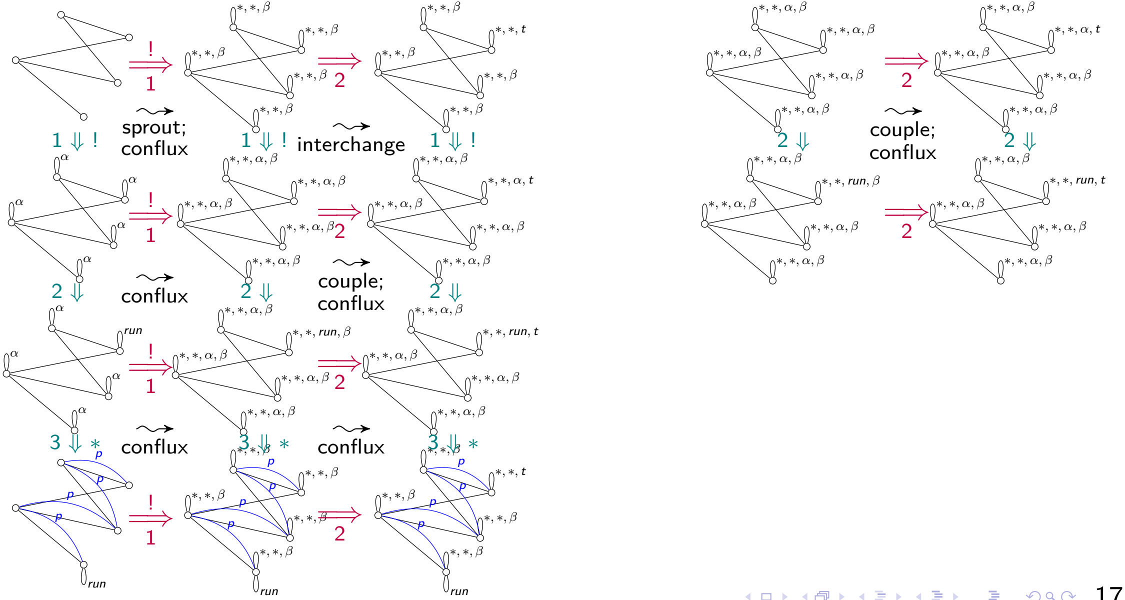
conflux

conflux

couple

couple;

Correctness of the Reduction *hampath-to-2-bounded-spantree*



sprout; conflux

interchange

conflux

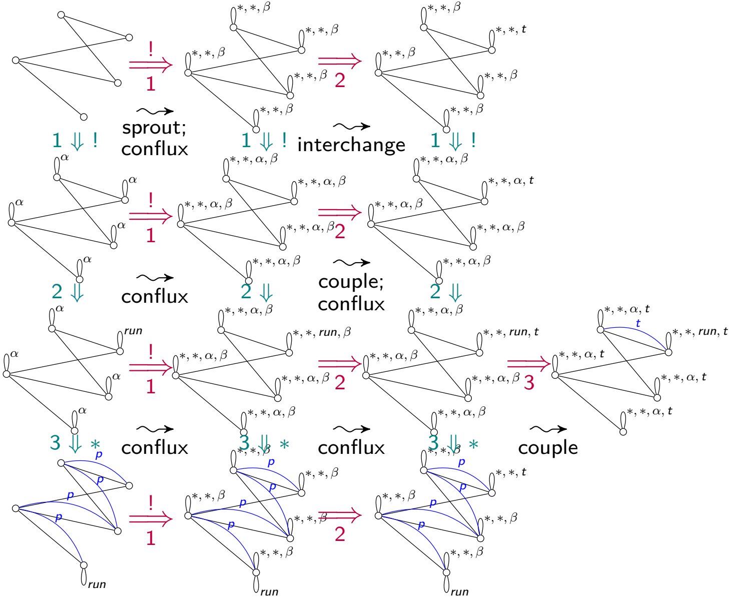
couple; conflux

conflux

conflux

couple; conflux

Correctness of the Reduction *hampath-to-2-bounded-spantree*



sprout; conflux

interchange

conflux

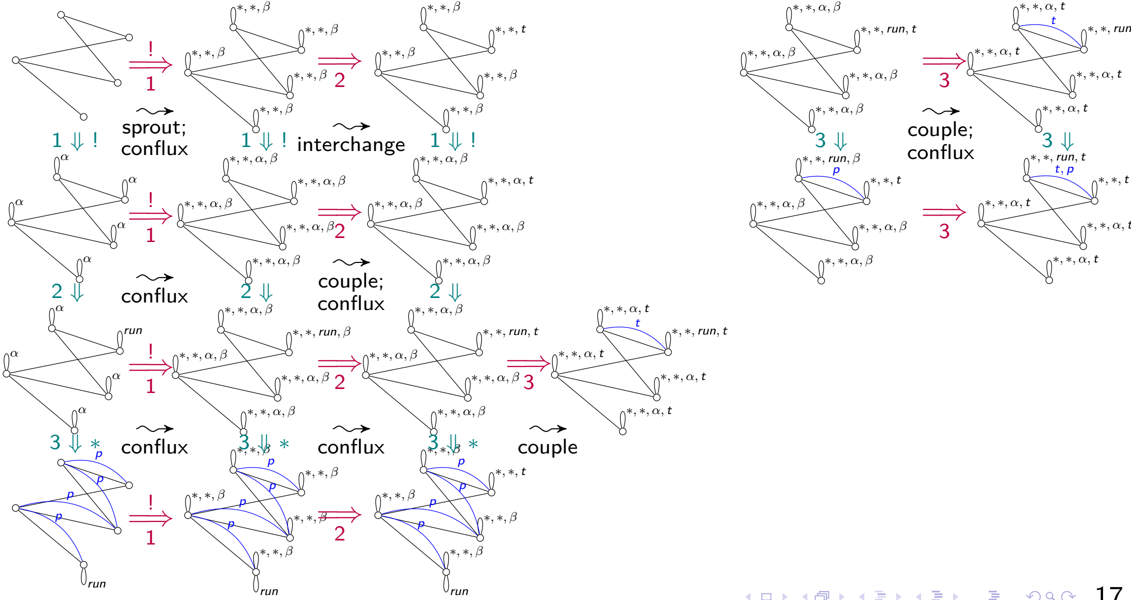
couple; conflux

conflux

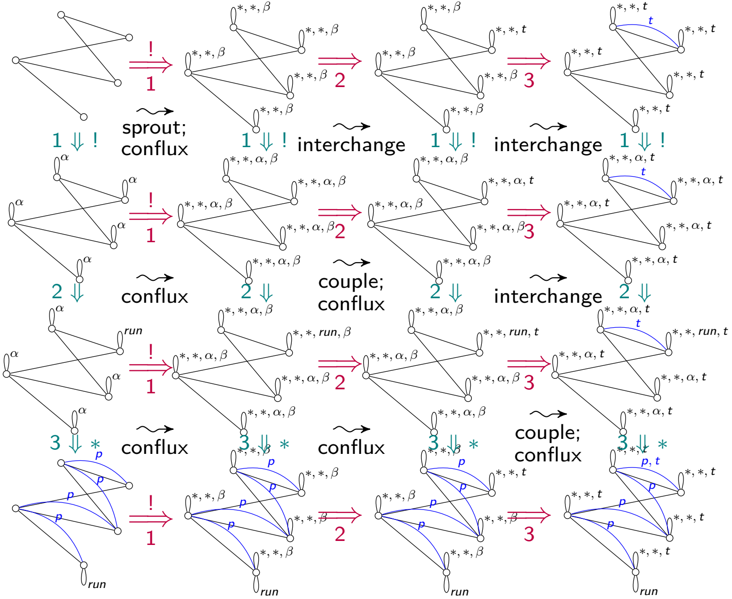
conflux

couple

Correctness of the Reduction *hampath-to-2-bounded-spantree*



Correctness of the Reduction *hampath-to-2-bounded-spantree*



sprout; conflux

interchange

interchange

conflux

couple; conflux

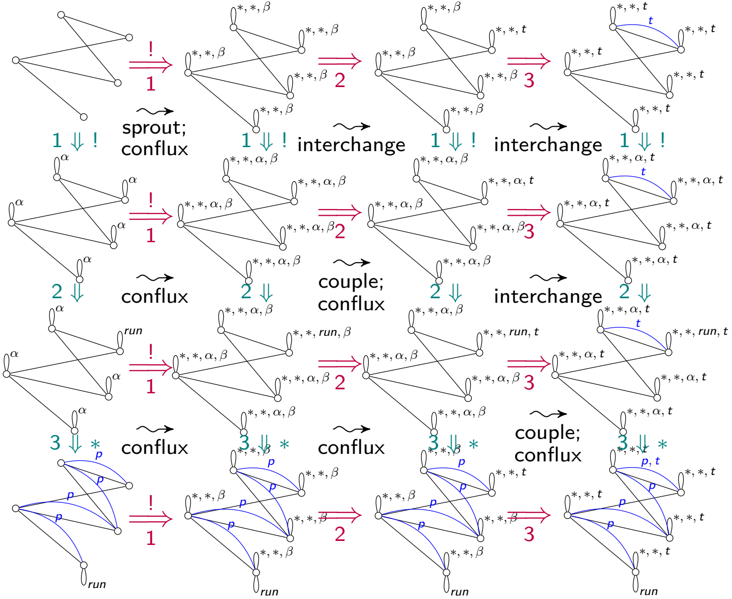
interchange

conflux

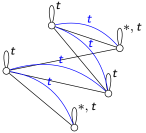
conflux

couple; conflux

Correctness of the Reduction *hampath-to-2-bounded-spantree*


$\overset{!}{\Longrightarrow}_1$
$\Longrightarrow_2$
$\Longrightarrow_3$
$1 \Downarrow !$
$2 \Downarrow$
$3 \Downarrow *$
$\rightsquigarrow$ sprout; conflux
$\rightsquigarrow$ interchange
$\rightsquigarrow$ interchange
$\rightsquigarrow$ conflux
$\rightsquigarrow$ couple; conflux
$\rightsquigarrow$ interchange
$\rightsquigarrow$ conflux
$\rightsquigarrow$ conflux
$\rightsquigarrow$ couple; conflux
$*, *, \beta$
$*, *, t$
$*, *, \alpha, \beta$
$*, *, \alpha, t$
$*, *, run, \beta$
$*, *, run, t$
α
run
p
p, t
t

iteration yields



Proof procedure

$$\begin{tikzcd}
G \arrow[r, Rightarrow, "*", "red"'] \arrow[d, Rightarrow, "*"', "gtu"] & red(G) \arrow[d, Rightarrow, "*"', "gtu'"] \\
H \arrow[r, squiggly, shift left, "forward"] & H' \arrow[l, squiggly, shift left, "backward"']
\end{tikzcd}$$

Let $gtu = (I_{gtu}, P_{gtu}, C_{gtu}, T_{gtu})$ and $gtu' = (I_{gtu'}, P_{gtu'}, C_{gtu'}, T_{gtu'})$
be two polynomial graph transformation units.
Let $red = (I_{gtu}, \overline{P_1} \cup \overline{P_2}, \overline{C}, I_{gtu'})$
with $\overline{P_1} \cap \overline{P_2} = \emptyset$
be a functional graph transformation unit
s.t. each of its successful derivations has the form $G \overset{*}{\underset{\overline{P_1}}{\Longrightarrow}} AS_0 \overset{*}{\underset{\overline{P_2}}{\Longrightarrow}} red(G)$.

Let P_{gtu} and $\overline{P_1}$, $\overline{P_2}$ and $P_{gtu'}$, as well as P_{gtu} and $P_{gtu'}$ be disjoint and independent.
Then the correctness of *red* may be proved as follows:

Algorithm Forward

(f1) $\mathit{conflux}$!

(f2) repeat wrt $C_{gtu'}$:

If next rule to be applied is the first one of a functional section of a permitted derivation given by funct

then 1. apply $\mathit{sprout}(\mathit{funct})$ to AS
2. $\mathit{funct}(AS)$ is the new AS
3. $\mathit{conflux}$!

else 1. $\mathit{couple}(\mathit{span})$ for some given span
2. $(\mathit{conflux}|\mathit{interchange})$!

(f3) 1. $\mathit{conflux}$!
2. cont. derivation wrt $C_{gtu'}$

(f4) Check $H' \in SEM(T_{gtu'})$

$$\begin{tikzcd}
G \arrow[r, Rightarrow, "*", "\overline{P_1}"'] \arrow[d, Rightarrow, "*"', "P_{gtu}"] & AS_0 \arrow[r, Rightarrow, "*", "\overline{P_2}"'] & red(G) \\
H & &
\end{tikzcd}$$

Algorithm Forward

```
(f1) $conflux!$
(f2) repeat wrt $C_{gtu'}$:
  If next rule to be applied is the first one of
     a functional section of a permitted
     derivation given by $funct$
  then 1. apply $sprout(funct)$ to $AS$
       2. $funct(AS)$ is the new $AS$
       3. $conflux!$
  else 1. $couple(span)$ for some
          given $span$
       2. $(conflux|interchange)!$
(f3) 1. $conflux!$
     2. cont. derivation wrt $C_{gtu'}$
(f4) Check $H' \in SEM(T_{gtu'})$
```

$$\begin{tikzcd}
G \arrow[r, Rightarrow, "*", "\overline{P_1}"'] \arrow[d, Rightarrow, "*"', "P_{gtu}"] & AS_0 \arrow[r, Rightarrow, "*", "\overline{P_2}"'] \arrow[d, Rightarrow, "*"', "P_{gtu}"] & red(G) \\
H \arrow[r, Rightarrow, "*", "\overline{P_1}"'] & H_0 &
\end{tikzcd}$$

Algorithm Forward

```
(f1)  $conflux$!
(f2)  repeat wrt $C_{gtu'}$:
        If  next rule to be applied is the first one of
            a functional section of a permitted
            derivation given by $funct$
      then  1. apply $sprout(funct)$ to $AS$
            2. $funct(AS)$ is the new $AS$
            3. $conflux$!
      else  1. $couple(span)$ for some
               given $span$
            2. $(conflux|interchange)$!
(f3)  1. $conflux$!
      2. cont. derivation wrt $C_{gtu'}$
(f4)  Check $H' \in SEM(T_{gtu'})$
```

$$\begin{tikzcd}
G \arrow[r, Rightarrow, "*", "\overline{P_1}"'] \arrow[d, Rightarrow, "*"', "P_{gtu}"] & AS_0 \arrow[r, Rightarrow, "*", "\overline{P_2}"'] \arrow[d, Rightarrow, "*"', "P_{gtu}"] \arrow[dr, Rightarrow, "*"', "P_{gtu'}"] & red(G) \\
H \arrow[r, Rightarrow, "*", "\overline{P_1}"'] & H_0 & funct(AS)
\end{tikzcd}$$

Algorithm Forward

```
(f1)  $conflux$!
(f2)  repeat wrt $C_{gtu'}$:
        If  next rule to be applied is the first one of
            a functional section of a permitted
            derivation given by $funct$
      then  1. apply $sprout(funct)$ to $AS$
            2. $funct(AS)$ is the new $AS$
            3. $conflux$!
      else  1. $couple(span)$ for some
               given $span$
            2. $(conflux|interchange)$!
(f3)  1. $conflux$!
      2. cont. derivation wrt $C_{gtu'}$
(f4)  Check $H' \in SEM(T_{gtu'})$
```

$$\begin{tikzcd}
G \arrow[r, Rightarrow, "*"] \arrow[r, Rightarrow, "\overline{P_1}"', phantom] \arrow[dd, Rightarrow, "*"'] \arrow[dd, Rightarrow, "P_{gtu}", phantom] & AS_0 \arrow[r, Rightarrow, "*"] \arrow[r, Rightarrow, "\overline{P_2}"', phantom] \arrow[dd, Rightarrow, "*"'] \arrow[dd, Rightarrow, "P_{gtu}", phantom] \arrow[dr, Rightarrow, "P_{gtu'}"] \arrow[dr, Rightarrow, "*"', phantom] & red(G) \\
 & & funct(AS_0) = AS_1 \\
H \arrow[r, Rightarrow, "*"] \arrow[r, Rightarrow, "\overline{P_1}"', phantom] & H_0 &
\end{tikzcd}$$

Algorithm Forward

```
(f1) $conflux$!
(f2) repeat wrt $C_{gtu'}$:
  If next rule to be applied is the first one of
     a functional section of a permitted
     derivation given by $funct$
  then 1. apply $sprout(funct)$ to $AS$
       2. $funct(AS)$ is the new $AS$
       3. $conflux$!
  else 1. $couple(span)$ for some
          given $span$
       2. $(conflux|interchange)$!
(f3) 1. $conflux$!
     2. cont. derivation wrt $C_{gtu'}$
(f4) Check $H' \in SEM(T_{gtu'})$
```

$$\begin{tikzcd}
G \arrow[r, Rightarrow, "*", "\overline{P_1}"'] \arrow[dd, Rightarrow, "*"', "P_{gtu}"] & AS_0 \arrow[r, Rightarrow, "*", "\overline{P_2}"'] \arrow[dd, Rightarrow, "*"', "P_{gtu}"] \arrow[dr, Rightarrow, "*"', "P_{gtu'}"] & red(G) \\
 & & AS_1 \arrow[dd, Rightarrow, "*"', "P_{gtu}"] \\
H \arrow[r, Rightarrow, "*", "\overline{P_1}"'] & H_0 \arrow[dr, Rightarrow, "P_{gtu'}", "*"'] & \\
 & & H_1
\end{tikzcd}$$

Algorithm Forward

```
(f1) $conflux!$
(f2) repeat wrt $C_{gtu'}$:
  If   next rule to be applied is the first one of
       a functional section of a permitted
       derivation given by $funct$
  then 1. apply $sprout(funct)$ to $AS$
       2. $funct(AS)$ is the new $AS$
       3. $conflux!$
  else 1. $couple(span)$ for some
          given $span$
       2. $(conflux|interchange)!$
(f3) 1. $conflux!$
     2. cont. derivation wrt $C_{gtu'}$
(f4) Check $H' \in SEM(T_{gtu'})$
```

$$\begin{tikzcd}
G \arrow[r, Rightarrow, "*", "\overline{P_1}"'] \arrow[dd, Rightarrow, "*"', "P_{gtu}"] & AS_0 \arrow[r, Rightarrow, "*", "\overline{P_2}"'] \arrow[d, Rightarrow, "P_{gtu}"', "*"] \arrow[dr, Rightarrow, "P_{gtu'}"] & red(G) \\
 & X \arrow[d, Rightarrow, "P_{gtu}"', "*"] & Y = AS_1 \\
H \arrow[r, Rightarrow, "*", "\overline{P_1}"'] & H_0 &
\end{tikzcd}$$

Algorithm Forward

(f1) *conflux*!

(f2) repeat wrt $C_{gtu'}$:

If next rule to be applied is the first one of a functional section of a permitted derivation given by *funct*

then 1. apply *sprout*(*funct*) to *AS*
2. *funct*(*AS*) is the new *AS*
3. *conflux*!

else 1. *couple*(*span*) for some given *span*
2. (*conflux*|*interchange*)!

(f3) 1. *conflux*!
2. cont. derivation wrt $C_{gtu'}$

(f4) Check $H' \in SEM(T_{gtu'})$

$$\begin{tikzcd}
G \arrow[r, Rightarrow, "*", "\overline{P_1}"'] \arrow[dd, Rightarrow, "*"', "P_{gtu}"] & AS_0 \arrow[r, Rightarrow, "*", "\overline{P_2}"'] \arrow[d, Rightarrow, "P_{gtu}"', "*"] \arrow[dr, Rightarrow, "P_{gtu'}"] & red(G) \\
 & X \arrow[d, Rightarrow, "P_{gtu}"', "*"] & Y = AS_1 \arrow[dd, Rightarrow, "*"', "P_{gtu}"] \\
H \arrow[r, Rightarrow, "*", "\overline{P_1}"'] & H_0 \arrow[dr, Rightarrow, "P_{gtu'}"] & \\
 & & H_1
\end{tikzcd}$$

Algorithm Forward

```
(f1) $conflux!$
(f2) repeat wrt $C_{gtu'}$:
     If   next rule to be applied is the first one of
          a functional section of a permitted
          derivation given by $funct$
     then 1. apply $sprout(funct)$ to $AS$
          2. $funct(AS)$ is the new $AS$
          3. $conflux!$
     else 1. $couple(span)$ for some
             given $span$
          2. $(conflux|interchange)!$
(f3) 1. $conflux!$
     2. cont. derivation wrt $C_{gtu'}$
(f4) Check $H' \in SEM(T_{gtu'})$
```

$$\begin{tikzcd}
G \arrow[r, Rightarrow, "*", "\overline{P_1}"'] \arrow[dd, Rightarrow, "*"', "P_{gtu}"] & AS_0 \arrow[r, Rightarrow, "*", "\overline{P_2}"'] \arrow[dd, Rightarrow, "*"', "P_{gtu}"] \arrow[dr, Rightarrow, "P_{gtu'}", "*"'] & red(G) & \\
 & & AS_1 \arrow[dd, Rightarrow, "*"', "P_{gtu}"] \arrow[dr, Rightarrow, "P_{gtu'}", "*"'] & \\
H \arrow[r, Rightarrow, "*", "\overline{P_1}"'] & H_0 \arrow[dr, Rightarrow, "P_{gtu'}", "*"'] & & AS_2 \arrow[dd, Rightarrow, "*"', "P_{gtu}"] \\
 & & H_1 \arrow[dr, Rightarrow, "P_{gtu'}", "*"'] & \\
 & & & H_2
\end{tikzcd}$$

Algorithm Forward

(f1) *conflux*!

(f2) repeat wrt $C_{gtu'}$:

If next rule to be applied is the first one of a functional section of a permitted derivation given by *funct*

then 1. apply $sprout(funct)$ to AS

2. $funct(AS)$ is the new AS

3. *conflux*!

else 1. $couple(span)$ for some given *span*

2. $(conflux|interchange)!$

(f3) 1. *conflux*!

2. cont. derivation wrt $C_{gtu'}$

(f4) Check $H' \in SEM(T_{gtu'})$

$$\begin{tikzcd}
G \arrow[r, Rightarrow, "*", "\overline{P_1}"'] \arrow[dd, Rightarrow, "*"', "P_{gtu}"] & AS_0 \arrow[r, Rightarrow, "*", "\overline{P_2}"'] \arrow[dd, Rightarrow, "*"', "P_{gtu}"] \arrow[dr, Rightarrow, "*"', "P_{gtu'}"] & red(G) \\
 & & AS_m \arrow[dd, Rightarrow, "*"', "P_{gtu}"] \\
H \arrow[r, Rightarrow, "*", "\overline{P_1}"'] & H_0 \arrow[dr, Rightarrow, "P_{gtu'}", "*"'] & \\
 & & H_m
\end{tikzcd}$$

Algorithm Forward

(f1) *conflux*!

(f2) repeat wrt $C_{gtu'}$:

  If next rule to be applied is the first one of a functional section of a permitted derivation given by *funct*

then 1. apply $sprout(funct)$ to AS

  2. $funct(AS)$ is the new AS

  3. *conflux*!

else 1. $couple(span)$ for some given *span*

  2. $(conflux|interchange)!$

(f3) 1. *conflux*!

  2. cont. derivation wrt $C_{gtu'}$

(f4) Check $H' \in SEM(T_{gtu'})$

$$\begin{tikzcd}
G \arrow[r, Rightarrow, "*", "\overline{P_1}"'] \arrow[dd, Rightarrow, "*"', "P_{gtu}"] & AS_0 \arrow[rr, Rightarrow, "*", "\overline{P_2}"'] \arrow[dd, Rightarrow, "*"', "P_{gtu}"] \arrow[dr, Rightarrow, "*"', "P_{gtu'}"] & & red(G) \arrow[dr, Rightarrow, "*"', "P_{gtu'}"] & \\
& & AS_m \arrow[rr, Rightarrow, "*", "\overline{P_2}"'] \arrow[dd, Rightarrow, "*"', "P_{gtu}"] & & \widetilde{red}(G) \\
H \arrow[r, Rightarrow, "*", "\overline{P_1}"'] & H_0 \arrow[dr, Rightarrow, "P_{gtu'}", "*"'] & & & \\
& & H_m & &
\end{tikzcd}$$

Algorithm Forward

```
(f1)  conflux!
(f2)  repeat wrt $C_{gtu'}$:
  If  next rule to be applied is the first one of
      a functional section of a permitted
      derivation given by funct
then  1. apply sprout(funct) to AS
      2. funct(AS) is the new AS
      3. conflux!
else  1. couple(span) for some
         given span
      2. (conflux|interchange)!
(f3)  1. conflux!
      2. cont. derivation wrt $C_{gtu'}$
(f4)  Check $H' \in SEM(T_{gtu'})$
```

$$\begin{tikzcd}
G \arrow[r, Rightarrow, "*", "\overline{P_1}"'] \arrow[dd, Rightarrow, "*"', "P_{gtu}"] & AS_0 \arrow[r, Rightarrow, "*", "\overline{P_2}"'] \arrow[dd, Rightarrow, "*"', "P_{gtu}"] \arrow[dr, Rightarrow, "*"', "P_{gtu'}"] & red(G) \arrow[dr, Rightarrow, "*"', "P_{gtu'}"] & \\
 & & AS_m \arrow[r, Rightarrow, "*", "\overline{P_2}"'] \arrow[dd, Rightarrow, "*"', "P_{gtu}"] & \widetilde{red}(G) \arrow[dd, Rightarrow, "*"', "P_{gtu'}"] \\
H \arrow[r, Rightarrow, "*", "\overline{P_1}"'] & H_0 \arrow[dr, Rightarrow, "P_{gtu'}", "*"'] & & \\
 & & H_m & H'
\end{tikzcd}$$

Algorithm Forward

(f1) *conflux*!

(f2) repeat wrt $C_{gtu'}$:

If next rule to be applied is the first one of a functional section of a permitted derivation given by *funct*

then
1. apply *sprout*(*funct*) to *AS*
2. *funct*(*AS*) is the new *AS*
3. *conflux*!

else
1. *couple*(*span*) for some given *span*
2. (*conflux*|*interchange*)!

(f3)
1. *conflux*!
2. cont. derivation wrt $C_{gtu'}$

(f4) Check $H' \in SEM(T_{gtu'})$

$$\begin{tikzcd}
G \arrow[r, Rightarrow, "*", "\overline{P_1}"'] \arrow[dd, Rightarrow, "*"', "P_{gtu}"] & AS_0 \arrow[r, Rightarrow, "*", "\overline{P_2}"'] \arrow[dd, Rightarrow, "*"', "P_{gtu}"] \arrow[dr, Rightarrow, "*"', "P_{gtu'}"] & red(G) \arrow[dr, Rightarrow, "*"', "P_{gtu'}"] & \\
 & & AS_m \arrow[r, Rightarrow, "*", "\overline{P_2}"'] \arrow[dd, Rightarrow, "*"', "P_{gtu}"] & \widetilde{red}(G) \arrow[dd, Rightarrow, "*"', "P_{gtu'}"] \\
H \arrow[r, Rightarrow, "*", "\overline{P_1}"'] & H_0 \arrow[dr, Rightarrow, "*"', "P_{gtu'}"] & & \\
 & & H_m & H'
\end{tikzcd}$$

Algorithm Forward

```
(f1) $conflux$!
(f2) repeat wrt $C_{gtu'}$:
       If   next rule to be applied is the first one of
            a functional section of a permitted
            derivation given by $funct$
       then 1. apply $sprout(funct)$ to $AS$
            2. $funct(AS)$ is the new $AS$
            3. $conflux$!
       else 1. $couple(span)$ for some
               given $span$
            2. $(conflux|interchange)$!
(f3) 1. $conflux$!
     2. cont. derivation wrt $C_{gtu'}$
(f4) Check $H' \in SEM(T_{gtu'})$
```

Algorithm Backward

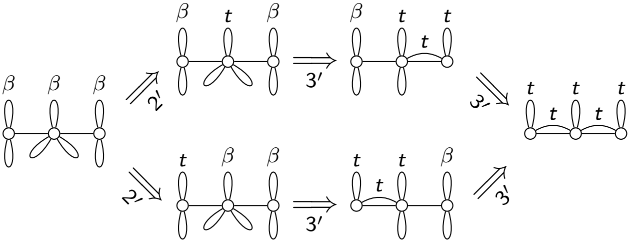
```
(b1) $interchange$!
(b2) repeat wrt $C_{gtu}$:
       If   next rule to be applied is the first one of
            a functional section of a permitted
            derivation given by $funct$
       then 1. apply $sprout(funct)$ to $AS$
            2. $funct(AS)$ is the new $AS$
            3. $conflux$!
       else 1. $couple(span)$ for some
               given $span$ (converse)
            2. $(conflux|interchange)$!
(b3) 1. $interchange$!
     2. cont. derivation wrt $C_{gtu}$
(b4) Check $H \in SEM(T_{gtu})$
```

Preprocessing

A preprocessing may be necessary before the backward part of the proof procedure can work. The reason is that a graph may have several successful derivations, but only some of them may be suitable for forward and backward processing. In such a case, one may apply *associate*(*pair*) for appropriate pairs of derivations together with *interchange*.

Example

association pair in spantree



Conclusion

- We have made a proposal for a *constructive approach* for proving correctness of reductions between NP problems by graph-transformational means.
- The approach is an attempt in the early stage of development and needs further investigation, to shed more light on its usefulness.

Future Work

1. The proof procedure has turned out to be suitable in several cases, but it seems to fail in other cases, e.g. reduction from the vertex cover problem to the Hamiltonian cycle problem.
2. To cover more cases or to simplify proofs, one may look for further operations. Candidates are operations like *conflux* and *interchange* that are based on independence including parallelization, sequentialization, and shift.
3. Another possibility is to generalize the coupling by considering spans that consist of derivations rather than direct derivations.
4. The preprocessing needs more attention.
5. Is there any chance of tool support for such correctness proofs?
6. A reduction between NP problems is a kind of model transformation. Do results of the theory of model transformations and their correctness proofs like triple graph grammars apply to reductions?

Future Work

1. The proof procedure has turned out to be suitable in several cases, but it seems to fail in other cases, e.g. reduction from the vertex cover problem to the Hamiltonian cycle problem.
2. To cover more cases or to simplify proofs, one may look for further operations. Candidates are operations like *conflux* and *interchange* that are based on independence including parallelization, sequentialization, and shift.
3. Another possibility is to generalize the coupling by considering spans that consist of derivations rather than direct derivations.
4. The preprocessing needs more attention.
5. Is there any chance of tool support for such correctness proofs?
6. A reduction between NP problems is a kind of model transformation. Do results of the theory of model transformations and their correctness proofs like triple graph grammars apply to reductions?

Thank you for your attention