PART 3: FORMAL PROPERTIES

BX with Triple Graph Grammars
Formal Notation

compact representation
of a rule application
(aka direct derivation)

even more
compact

compact representation
of derivation

derivation of length \( n > 1 \)

\( G \xrightarrow{p_1} G_1 \xrightarrow{p_2} \cdots \xrightarrow{p_n} G_n \)

\( G \xrightarrow{*} G_n \)
Formal Notation

$G \xrightarrow{p_1} G_1 \xrightarrow{p_2} \cdots \xrightarrow{p_n} G_n$

$G \xrightarrow{*} G_n$

compact representation of derivation

derivation of length $n > 1$

compact notation for derivation with rules of a TGG

$G \xrightarrow{*} G' \in TGG \iff \exists G \xrightarrow{p_1} G_1 \xrightarrow{p_2} \cdots \xrightarrow{p_n} G_n, \forall i \in \{1, \ldots, n\}, p_i \in TGG$
(Transformation) Correctness: Intuition

intuition: \((fpg, bpg)\) should be consistent with its underlying TGG

use rules specified by the user to characterise a subset of “consistent” triples
(Transformation) Correctness: Notation

\[ A \xrightarrow{R} B \]

\[ \text{TGG} \]

\[ C = \{ A \xrightarrow{r} B \mid \exists \emptyset \Rightarrow A \xrightarrow{r} B \in \text{TGG} \} \]

\[ \subseteq R \]

\[ A \xrightarrow{r} B \in C \iff A \xrightarrow{r:C} B \]

\[ C_S = \{ A \mid \exists A \xrightarrow{r:C} B \} \]

\[ A : C_S \xrightarrow{a} A' : C_S \]

use rules specified by the user to characterise a subset of “consistent” triples

just compact notation for consistent triples

this is the set of all consistent source models (consistent target models analogously)

let’s call such a source delta consistent
(Transformation) Correctness: Laws

A $\xrightarrow{R} B$

TGG

for a consistent triple

and a consistent source delta

$fpg$ is correct if it produces a consistent triple

$A' : C_S$

$B' : C_T$

$a$

$b$

$r : C$

$r' : C$

$fpg$

$+\$

$bpg$

$fpg$
(Transformation) Completeness

intuition: \((fpg, bpg)\) should be defined for any consistent delta

formally, completeness just means that \((fpg, bpg)\) must be total on all consistent source/target models and consistent source/target deltas

remember: the supplied TGG only covers a small subset of all deltas!
Consistent source delta \( \equiv \) Arrow connecting points on the surface

Synchronizer always finds a "path" along the surface for any consistent source delta

Completeness
Stability

don’t do anything for the “idle” delta

sounds trivial, but it rules out “batch mode” TGG tools
Stability

A batch forward transformation only takes the current source model as input and extends it to a consistent triple after target changes that do not affect consistency.

Stability is violated.

Anthony Anjorin: An Introduction To Triple Graph Grammars
Entering a concrete brand doesn’t affect consistency but cannot be retained by a batch transformation.
Other laws

1. Hippocraticness
2. (Weak) Undoability
3. (Weak) Invertibility
4. Functional Behaviour
5. Domain Correctness
6. Domain Completeness
7. Local Completeness
8. ...
Other laws

1. Hippocraticness
2. (Weak) Undoability
3. (Weak) Invertibility
4. Functional Behaviour
5. Domain Correctness
6. Domain Completeness
7. Local Completeness
8. ...

Diskin et al. calls such SDLs “well-behaved”
Other laws

1. Hippocraticness
2. (Weak) Undoability
3. (Weak) Invertibility
4. Functional Behaviour
5. Domain Correctness
6. Domain Completeness
7. Local Completeness
8. …

In general TGG-based synchronisation does not obey any of these laws …

… but suitable restrictions can be posed to determine adequate subclasses of TGGs

TGGs offer a “playground” for exploring formal properties and how to guarantee them (statically or dynamically)
PART 4: FROM TGGS TO SDLS

BX with Triple Graph Grammars
What do TGG tools do?

A TGG tool does some "magic" and produces a symmetric delta lens!
1. All arrows are **monomorphic** (injective) (my head hurts otherwise)

2. Colours indicate deletion (red) and creation (green)
Running Example

Anthony Anjorin: An Introduction To Triple Graph Grammars
Running Example

19

:Hospital
name = "Springfield General Hospital"

:Doctor
name = "Nick Riviera"

:Aspirin
Brand = Ascriptin

:Patient
name = "Lisa"

:Ibuprofen

:HospitalTo DosagePlan

:Hospital

:MedicationTo Dosage

:Medication

:DosagePlan

:Dosage

MediSoft ← R MediSupply

+
What do TGG tools do?

1. (Re-)Alignment:

**basic idea:** realise correct fpg in three steps

```
A      r:C      A'
     ^        ^
    /        /  fAlgn
   a        r':C
     ↓        ↓
    B       B'
```

```
A      r:C      B
     ^        ^
    /        /  fAlgn
   a        id
     ↓        ↓
    A'       B
```

- This does not change the target model.
- Note that the result is not necessarily consistent!
- \( f\text{Algn} \) recomputes correspondence links.
Running Example: Re-Alignment

Anthony Anjorin: An Introduction To Triple Graph Grammars
What do TGG tools do?

1. (Re-)Alignment:

2. Rollback:

re-establish consistency by deleting elements
Running Example: Rollback

Anthony Anjorin: An Introduction To Triple Graph Grammars
What do TGG tools do?

1. (Re-)Alignment:

2. Rollback:

3. (Re-)Translation:

starting from a consistent triple

apply rules of the TGG until the final source model is obtained
Running Example: Re-Translation

\[ \hat{r} : C \]

\[ a' \]

\[ \hat{r} \]

\[ b' \]

\[ A' \]

\[ r' : C \]

\[ A \]

\[ B' \]

\[ B \]

\[ fAdd \]
What do TGG tools do?

1. (Re-)Alignment:

2. Rollback:

3. (Re-)Translation:
Some remarks on implementation

**easy:** TGG tools represent correspondence links explicitly so can just delete “dangling” links (formally a pullback)

Arrows in this diagram are monic typed graph morphisms (and not deltas!)
Some remarks on implementation

**hard:** requires a complete remarking of all elements (very inefficient), most TGG tools employ some kind of optimisation technique.
Some remarks on implementation

**easy**: just apply TGG rules wherever they match (typically quite efficient)

**but**: requires backtracking in general, so most TGG tools pose some (rather technical) restrictions.
A TGG tool that actually inspects the delta to be propagated is trivially stable.

So incremental TGG tools are stable, batch TGG tools are not.
Proving correctness

**hard**: show that **Del** (whatever strategy is applied) always produces a consistent intermediate result.

**easy**: in each step, a TGG rule is applied, so if translation succeeds, the result is consistent by definition.
Proving completeness

**hard**: show that Del (whatever strategy is applied) always produces a consistent intermediate result.

**easy**: if backtracking strategy is taken (but very inefficient)

**hard**: show that translation process succeeds without backtracking.
Some closing remarks on TGGs and “least change”

... as a rollback to a state from which a delta to the final state can be derived from the TGG... completeness is achieved by interpreting every delta that is not already explicitly fixed by the TGG...
Some closing remarks on TGGs and “least change”

the result is not always what is “expected”
in the worst case, this can result in a complete rollback and re-translation (just as bad as batch, and less efficient!)
Some closing remarks on TGGs and “least change”

having this as a last resort (fallback) is often nice …

… but there is currently no elegant way to exercise fine granular control over the behaviour of TGG-based synchronisers
Running Example: Least Change?

would be helpful to be able to “optimise” the default algorithm in a problem-specific manner (and remain correct, complete, …)
TGG Research Challenge (one of many! see [1])

TGG-based synchroniser development is not “smooth” :)
TGG Research Challenge (one of many!)

Implementation Effort

Control

TGGs

Solver-Based

Putback-Based

Get-Based

Combinator-Based
1. Install VirtualBox from www.virtualbox.org
2. Download this VM: https://db.tt/gYgQMSHsZ
3. Open VM, start Eclipse with shortcut on desktop
4. Choose workspaces/task3 as your workspace
5. Follow instructions from https://db.tt/GJ8fyeVm