Verifying Graph Transformation Systems with Description Logics

Jon H. Brenas[1], Rachid Echahed[2] and Martin Strecker[3]

[1]UTHSC - ORNL, University of Memphis, Tennessee, USA

[2]CNRS and Université Grenoble Alpes, Grenoble, France

[3]Université de Toulouse, IRIT, Toulouse, France

June 25th, 2018

Partial Correctness à la Hoare of Graph and Model Transformation Systems

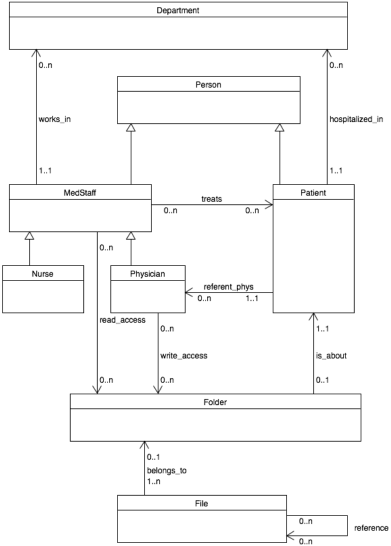
To be proven: $\{Pre(input)\}$ *Program* $\{Post(output)\}$

- Program is a graph or model transformation system
- input and output are graphs or models
- *Pre* and *Post* are description logic (DL) formulas over the inputs and the outputs

Outline

Models/Graphs

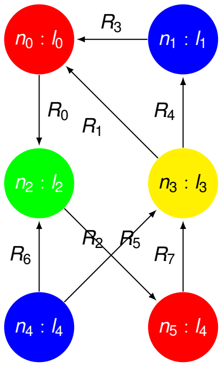
- Different kinds of nodes and edges



Logically Decorated Graphs

Let $\mathcal{L}$ be a set of formulas, a logically decorated graph G is a tuple $(N, E, \lambda_N, \lambda_E, s, t)$ where:

- N is a set of *nodes*,
- E is a set of *edges*,
- $\lambda_N : N \to 2^{\mathcal{L}}$ is a *node labeling function*,
- $\lambda_E : E \to \mathcal{L}$ is an *edge labeling function*
- source and target functions: $s : E \to N$ and $t : E \to N$



In this talk, the set $\mathcal{L}$ consists of description logic formulas.

Outline

Why considering Description Logics (DLs)?

- DLs constitute a formal basis of knowledge representation languages.
- DLs provide logical basis for ontologies.
  (E.g., the web ontology language OWL is based on DLs)
- Reasoning problems for DLs are decidable (in general)

DL Syntax

a DL syntax allows one to define:

- Concept names, which are equivalent to classical first-order logic unary predicates,
- Role names, which are equivalent to binary predicates and
- Individuals, which are equivalent to classical constants.

There are various DLs in the literature, they mainly differ by the logical operators they offer to construct concept and role expressions or axioms.

DL syntax: Concepts and roles

Let $\mathcal{C}_0$ (resp. $\mathcal{R}_0$ and $\mathcal{O}$) be a set of atomic concepts (resp. atomic roles and nominals).
Let $c_0 \in \mathcal{C}_0$, $r_0 \in \mathcal{R}_0$, $o \in \mathcal{O}$, and n an integer.

The set of concepts C and roles R are defined by:

$C := \top \mid c_0 \mid \exists R.C \mid \neg C \mid C \vee C$
$\quad \mid o$ (nominals, $\mathcal{O}$)
$\quad \mid \exists R.Self$ (self loops, $\mathcal{S}elf$)
$\quad \mid (< \ n\ R\ C)$ (counting quantifiers, $\mathcal{Q}$)

$R := r_0$
$\quad \mid U$ (universal role, $\mathcal{U}$)
$\quad \mid R^-$(inverse role, $\mathcal{I}$)

Examples of DL logics: $\mathcal{ALC}$, $\mathcal{ALCUO}$, $\mathcal{ALCUI}$, $\ldots$

Examples of properties

Examples of some requirements about the organization of a hospital:

- All patients of a pediatrician are children:
  First-order formula:
  $\forall x, y. Pediatrician(x) \wedge Has\_patient(x, y) \Rightarrow Child(y)$
  DL formula ($\mathcal{ALCU}$): $\forall U. Pediatrician \Rightarrow \forall Has\_patient. Child$

Examples of properties

Examples of some requirements about the organization of a hospital:

- All patients of a pediatrician are children:
  First-order formula:
  $\forall x, y. Pediatrician(x) \wedge Has\_patient(x, y) \Rightarrow Child(y)$
  DL formula ($\mathcal{ALCU}$): $\forall U. Pediatrician \Rightarrow \forall Has\_patient.Child$
- Dr. Smith is a pediatrician:
  First-order formula: $\exists x. Dr.Smith = x \wedge Pediatrician(x)$
  DL formula ($\mathcal{ALCUO}$): $\exists U. Dr.Smith \wedge Pediatrician$

Examples of properties

Examples of some requirements about the organization of a hospital:

- All patients of a pediatrician are children:
  First-order formula:
  $\forall x, y. Pediatrician(x) \wedge Has\_patient(x, y) \Rightarrow Child(y)$
  DL formula ($\mathcal{ALCU}$): $\forall U. Pediatrician \Rightarrow \forall Has\_patient. Child$
- Dr. Smith is a pediatrician:
  First-order formula: $\exists x. Dr.Smith = x \wedge Pediatrician(x)$
  DL formula ($\mathcal{ALCUO}$): $\exists U. Dr.Smith \wedge Pediatrician$
- All patients are a doctor's patients:
  First-order formula:
  $\forall x, y. Patient(x) \Rightarrow Has\_patient(y, x) \wedge Doctor(y)$
  DL formula ($\mathcal{ALCUI}$):$\forall U. Patient \Rightarrow \exists Has\_patient^{-}. Doctor$

Examples of properties (Continued)

Examples of some requirements about the organization of a hospital:

1. An operation can only be associated with one operating room:
   First-order formula:
   $\forall x, y, z. Operation(x) \wedge Scheduled\_in(x, y) \wedge Scheduled\_in(x, z) \wedge Operation\_room(y) \wedge Operation\_room(z) \Rightarrow y = z$
   DL formula ($\mathcal{ALCUQ}$):
   $\forall U.Operation \Rightarrow (< 2 Scheduled\_in.Operation\_room)$

Examples of properties (Continued)

Examples of some requirements about the organization of a hospital:

1. An operation can only be associated with one operating room:
   First-order formula:
   $\forall x, y, z. Operation(x) \wedge Scheduled\_in(x, y) \wedge Scheduled\_in(x, z) \wedge Operation\_room(y) \wedge Operation\_room(z) \Rightarrow y = z$
   DL formula ($\mathcal{ALCUQ}$):
   $\forall U. Operation \Rightarrow (< 2 Scheduled\_in. Operation\_room)$
2. A doctor can not be his/her own patient:
   First-order formula: $\forall x. Doctor(x) \Rightarrow \neg Has\_patient(x, x)$
   DL formula ($\mathcal{ALCUQ}$): $\forall U. Doctor \Rightarrow \neg \exists Has\_patient. SELF$

Outline

Graph Transformation

- There are several ways to transform graphs:
  - Imperative Programs
  - Rule-Based Programs
  - Knowledge-Base updates
  - Non-classical Logics
  - ...

Graph Transformation

- There are several ways to transform graphs:
  - Imperative Programs
  - Rule-Based Programs
    - Algebraic/Categorial approaches (DPO, SPO, SqPO, AGREE)
    - Algorithmic approaches
  - Knowledge-Base updates
  - Non-classical Logics
  - ...

Graph Transformation: Considered Rules

The considered Graph Rewriting rules are of the form $L \rightarrow R$ where:

- L is a graph
- R is a sequence of elementary actions

Some Elementary Actions

Let $\mathcal{C}_0$ (resp. $\mathcal{R}_0$) be a set of node (resp. edge) labels. An *elementary action*, say *a*, may be of the following forms:

- a *node addition* $add_N(i)$ (resp. *node deletion* $del_N(i)$)

Some Elementary Actions

Let $\mathcal{C}_0$ (resp. $\mathcal{R}_0$) be a set of node (resp. edge) labels. An *elementary action*, say a, may be of the following forms:

- a *node addition* $add_N(i)$ (resp. *node deletion* $del_N(i)$)
- a *node label addition* $add_C(i, c)$ (resp. *node label deletion* $del_C(i, c)$) where i is a node and c is a label in $\mathcal{C}_0$.

Some Elementary Actions

Let $\mathcal{C}_0$ (resp. $\mathcal{R}_0$) be a set of node (resp. edge) labels. An *elementary action*, say a, may be of the following forms:

- a *node addition* $add_N(i)$ (resp. *node deletion* $del_N(i)$)
- a *node label addition* $add_C(i, c)$ (resp. *node label deletion* $del_C(i, c)$) where i is a node and c is a label in $\mathcal{C}_0$.
- an *edge addition* $add_E(e, i, j, r)$ (resp. *edge deletion* $del_E(e, i, j, r)$) where e is an edge, i and j are nodes and r is an edge label in $\mathcal{R}_0$.

Some Elementary Actions

Let $\mathcal{C}_0$ (resp. $\mathcal{R}_0$) be a set of node (resp. edge) labels. An *elementary action*, say a, may be of the following forms:

- a *node addition* $add_N(i)$ (resp. *node deletion* $del_N(i)$)
- a *node label addition* $add_C(i, c)$ (resp. *node label deletion* $del_C(i, c)$) where i is a node and c is a label in $\mathcal{C}_0$.
- an *edge addition* $add_E(e, i, j, r)$ (resp. *edge deletion* $del_E(e, i, j, r)$) where e is an edge, i and j are nodes and r is an edge label in $\mathcal{R}_0$.
- a *global edge redirection* $i \gg j$ where i and j are nodes. It redirects all incoming edges of i towards j.

Some Elementary Actions

Let $\mathcal{C}_0$ (resp. $\mathcal{R}_0$) be a set of node (resp. edge) labels. An *elementary action*, say a, may be of the following forms:

- a *node addition* $add_N(i)$ (resp. *node deletion* $del_N(i)$)
- a *node label addition* $add_C(i, c)$ (resp. *node label deletion* $del_C(i, c)$) where i is a node and c is a label in $\mathcal{C}_0$.
- an *edge addition* $add_E(e, i, j, r)$ (resp. *edge deletion* $del_E(e, i, j, r)$) where e is an edge, i and j are nodes and r is an edge label in $\mathcal{R}_0$.
- a *global edge redirection* $i \gg j$ where i and j are nodes. It redirects all incoming edges of i towards j.
- a *merge action* $mrg(i, j)$ where i and j are nodes.

Some Elementary Actions

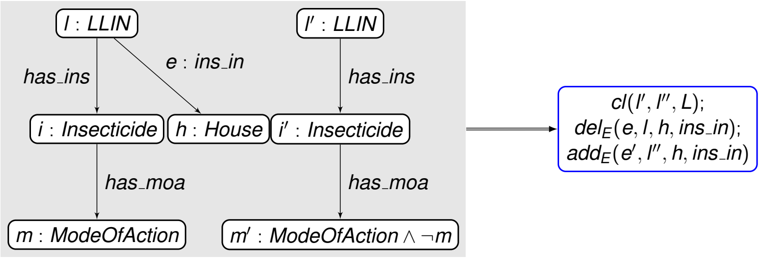
Let $\mathcal{C}_0$ (resp. $\mathcal{R}_0$) be a set of node (resp. edge) labels. An *elementary action*, say a, may be of the following forms:

- a *node addition* $add_N(i)$ (resp. *node deletion* $del_N(i)$)
- a *node label addition* $add_C(i, c)$ (resp. *node label deletion* $del_C(i, c)$) where i is a node and c is a label in $\mathcal{C}_0$.
- an *edge addition* $add_E(e, i, j, r)$ (resp. *edge deletion* $del_E(e, i, j, r)$) where e is an edge, i and j are nodes and r is an edge label in $\mathcal{R}_0$.
- a *global edge redirection* $i \gg j$ where i and j are nodes. It redirects all incoming edges of i towards j.
- a *merge action* $mrg(i, j)$ where i and j are nodes.
- a *clone action* $cl(i, j, L_{in}, L_{out}, L_{l\_in}, L_{l\_out}, L_{l\_loop})$ where i and j are nodes and L_{in}, L_{out}, L_{l_in}, L_{l_out} and L_{l_loop} are subsets of $\mathcal{R}_0$. It clones a node i by creating a new node j and connects j to the rest of a host graph according to different information given in the parameters $L_{in}, L_{out}, L_{l\_in}, L_{l\_out}, L_{l\_loop}$.

Graph Rewrite Systems: Example

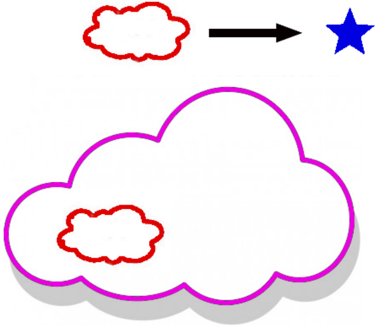
ρ_0:

ρ_1:



Match

- To be able to apply rules, we need to define when they can be applied.



Match

Definition: Match

A *match* h between a lhs L and a graph G is a pair of functions $h = (h^N, h^E)$, with $h^N : N^L \to N^G$ and $h^E : E^L \to E^G$ such that:

1. $\forall e \in E^L, s^G(h^E(e)) = h^N(s^L(e))$
2. $\forall e \in E^L, t^G(h^E(e)) = h^N(t^L(e))$
3. $\forall n \in N^L, \forall c \in \lambda_N^L(n), h^N(n) \models c$
4. $\forall e \in E^L, \lambda_E^G(h^E(e)) = \lambda_E^L(e)$

Remark: The third condition says that for every node, n, of the lhs, the node to which it is associated, $h(n)$, in G has to satisfy every concept in $\lambda_N^L(n)$. This condition clearly expresses additional negative and positive conditions which are added to the "structural" pattern matching.

Rewrite Step and Rewrite Derivation

Definition: Rewrite step

Let $\rho = L \to R$ be a rule and G and G' be two graphs.
G rewrites into G' using rule ρ, noted $G \to_\rho G'$ iff:

- There exists a match h from the left-hand side L to G, and
- $G \rightsquigarrow_{h(R)} G'$. I.e., G' is the result of performing $h(R)$ on G

Definition: Rewrite derivation

Let $\mathcal{R}$ be graph transformation system and G and G' be two graphs.
A *rewrite derivation* from G to G', noted $G \to_{\mathcal{R}} G'$, is a sequence $G \to_{\rho_0} G_1 \to_{\rho_1} \dots \to_{\rho_n} G'$ such that $\forall i.\rho_i \in \mathcal{R}$.

Strategies

- A strategy is a word of the following language defined by $s ::=$
  - ρ (application of a rule)
  - $s; s$ (sequential composition of strategies)
  - $s \oplus s$ (non-deterministic choice between two strategies)
  - s^* (iteration as long as possible of a strategy)
  - $\ldots$

Strategies

- A strategy is a word of the following language defined by $s ::=$
  - ρ (application of a rule)
  - $s; s$ (sequential composition of strategies)
  - $s \oplus s$ (non-deterministic choice between two strategies)
  - s^* (iteration as long as possible of a strategy)
  - $\ldots$
- Example: Strategy $strat = s_0; s_1^*; s_2$ performs once the sub-strategy s_0, iterates as much as possible sub-strategy s_1, before performing once sub-strategy s_2.

Strategies

- A strategy is a word of the following language defined by $s ::=$
  - ρ (application of a rule)
  - $s; s$ (sequential composition of strategies)
  - $s \oplus s$ (non-deterministic choice between two strategies)
  - s^* (iteration as long as possible of a strategy)
  - $\ldots$
- Example: Strategy $strat = s_0; s_1^*; s_2$ performs once the sub-strategy s_0, iterates as much as possible sub-strategy s_1, before performing once sub-strategy s_2.
- A derivation $G \rightarrow_{\rho_0} G_1 \rightarrow_{\rho_1} \ldots \rightarrow_{\rho_n} G'$ is controlled by a strategy *strat* iff the word $\rho_0 \rho_1 \ldots \rho_n$ belongs to the language defined by strategy *strat*.

Outline

Specification and Correctness

Definition: Specification

A *specification spec* is a triple $(Pre, strat, Post)$ where:

- *Pre* is a DL formula called the *precondition*
- *strat* is a strategy with respect to a graph transformation system $\mathcal{R}$
- *Post* is a DL formula called the *postcondition*.

Definition: Correctness

A specification $spec = (Pre, strat, Post)$ is said to be *correct* iff:

- for all graphs G,
- for all graphs G' such that $G \rightarrow_{strat} G'$
- if $G \models Pre$ then $G' \models Post$

Floyd-Hoare Logics

- Let $\mathcal{R}$ be a graph transformation system
- Let *strat* be a strategy and $\rho_0 \ldots \rho_{n-1}\rho_n$ an element of *strat*
- Let *Pre* and *Post* be two DL formulas
- Aim: Prove that specification $spec = (Pre, strat, Post)$ is correct

Pre

ρ_0;

...

ρ_{n-1};

ρ_n;

Post

Floyd-Hoare Logics

- Let $\mathcal{R}$ be a graph transformation system
- Let *strat* be a strategy and $\rho_0 \ldots \rho_{n-1}\rho_n$ an element of *strat*
- Let *Pre* and *Post* be two DL formulas
- Aim: Prove that specification $spec = (Pre, strat, Post)$ is correct

Pre
a_0;
...

a_{m-1};

a_m;
Post

Floyd-Hoare Logics

- Let $\mathcal{R}$ be a graph transformation system
- Let *strat* be a strategy and $\rho_0 \ldots \rho_{n-1}\rho_n$ an element of *strat*
- Let *Pre* and *Post* be two DL formulas
- Aim: Prove that specification $spec = (Pre, strat, Post)$ is correct

Pre

a_0;

...

a_{m-1};

$Post[a_m]$

a_m;

Post

Floyd-Hoare Logics

- Let $\mathcal{R}$ be a graph transformation system
- Let *strat* be a strategy and $\rho_0 \ldots \rho_{n-1}\rho_n$ an element of *strat*
- Let *Pre* and *Post* be two DL formulas
- Aim: Prove that specification $spec = (Pre, strat, Post)$ is correct

Pre

a_0;

...

$Post[a_m][a_{m-1}]$

a_{m-1};

$Post[a_m]$

a_m;

Post

Floyd-Hoare Logics

- Let $\mathcal{R}$ be a graph transformation system
- Let *strat* be a strategy and $\rho_0 \ldots \rho_{n-1}\rho_n$ an element of *strat*
- Let *Pre* and *Post* be two DL formulas
- Aim: Prove that specification $spec = (Pre, strat, Post)$ is correct

$Pre \Rightarrow Post[a_m][a_{m-1}]...[a_0]$
a_0;
...
$Post[a_m][a_{m-1}]$
a_{m-1};
$Post[a_m]$
a_m;
Post

Substitutions

Definition: Substitution

A *substitution*, written $[a]$, is associated to each elementary action a, such that for all graphs G and DL formulas ϕ, $(G \models \phi[a]) \Leftrightarrow (G' \models \phi)$ where G' is obtained from G after application of action a,i.e., $G \rightsquigarrow_a G'$.

$$\begin{array}{lll} G & \rightsquigarrow_a & G' \\ \phi[a] & & \phi \end{array}$$

Generating Weakest Preconditions

We define $wp(a, Q)$ the weakest precondition for an elementary action a and a formula Q.

- $wp(a,\ Q) = Q[a]$

Generating Weakest Preconditions

We define $wp(a, Q)$ the weakest precondition for an elementary action a and a formula Q.

- $wp(a,\ Q) = Q[a]$ How to handle substitutions?

Floyd-Hoare Logics: a classical example
The assignment instruction (action)

Weakest precondition: $wp(Post, [x := X + 1]) \equiv x > 5[x := X + 1]$

Action: $x := x + 1$;

Post: $Post \equiv x > 5$

Floyd-Hoare Logics: a classical example
The assignment instruction (action)

$wp(Post, [x := X + 1]) \equiv x > 5[x := X + 1] \equiv x > 4$

Action: $x := x + 1$;

Post: $Post \equiv x > 5$

Floyd-Hoare Logics: a basic case

$wp(Post, Add_E(e, a, b, R)) \equiv \exists U.(a \wedge (> 5R.\top))[Add_E(e, a, b, R)]$

Action: $Add_E(e, a, b, R)$;

Post: $\exists U.(a \wedge (> 5R.\top))$

Floyd-Hoare Logics: a basic case

$$wp(Post, Add_E(e,a,b,R)) \equiv \exists U.(a \wedge (> 5R.\top))[Add_E(e,a,b,R)] \equiv$$
$$(\exists U.(a \wedge \exists R.b) => \exists U.(a \wedge (> 5R.\top)))$$
$$\wedge(\exists U.(a \wedge \forall R.\neg b) => \exists U.(a \wedge (> 4R.\top)))$$

Action: $Add_E(e,a,b,R)$;

Post: $\exists U.(a \wedge (> 5R.\top))$

Closure Under Substitutions

Definition: Closure Under Substitution

A logic $\mathcal{L}$ is said to be *closed under substitution* iff for every formula $\phi \in \mathcal{L}$, every substitution $[a]$, $\phi[a] \in \mathcal{L}$.

DLs and Closure Under Substitutions

Theorem: The description logics $\mathcal{ALCUO}$, $\mathcal{ALCUOI}$, $\mathcal{ALCQUOI}$, $\mathcal{ALCUOS}elf$, $\mathcal{ALCUOIS}elf$, and $\mathcal{ALCQUOIS}elf$ are closed under substitutions.

Theorem: The description logics $\mathcal{ALCQUO}$ and $\mathcal{ALCQUOS}elf$ are not closed under substitutions.

Generating Weakest Preconditions (continued)

We define $wp(strat, Q)$ the weakest precondition for a strategy $strat$ and a formula Q.

- $wp(s_0; s_1,\ Q) = wp(s_0, wp(s_1,\ Q))$
- $wp(s_0 \oplus s_1,\ Q) = wp(s_0, Q) \wedge wp(s_1, Q)$
- $wp(\rho,\ Q) = App(\rho) \Rightarrow Q[a_n]...[a_0]$ where ρ's right-hand side is $a_0; ...; a_n$

Generating Weakest Preconditions

We define $wp(strat, Q)$ the weakest precondition for a strategy *strat* and a formula Q.

- $wp(\rho,\ Q) = App(\rho) \Rightarrow Q[a_n]...[a_0]$

Definition: Application Condition

Given a rule ρ, the *application condition* $App(\rho)$ is a formula such that a graph $G \models App(\rho)$ iff there exists a match between the left-hand side of ρ and G

Generating Weakest Preconditions

We define $wp(strat, Q)$ the weakest precondition for a strategy $strat$ and a formula Q.

- $wp(a,\ Q) = Q[a]$
- $wp(\epsilon,\ Q) = Q$
- $wp(a; \alpha,\ Q) = wp(a, wp(\alpha, Q))$
- $wp(s_0; s_1,\ Q) = wp(s_0, wp(s_1,\ Q))$
- $wp(s_0 \oplus s_1,\ Q) = wp(s_0, Q) \wedge wp(s_1, Q)$
- $wp(\rho,\ Q) = App(\rho) \Rightarrow Q[a_n]...[a_0]$

Generating Weakest Preconditions

$wp(strat, Q)$ computes the weakest precondition for a strategy *strat* and a formula Q.

- $wp(a,\ Q) = Q[a]$
- $wp(\epsilon,\ Q) = Q$
- $wp(a; \alpha,\ Q) = wp(a, wp(\alpha, Q))$
- $wp(s_0; s_1,\ Q) = wp(s_0, wp(s_1,\ Q))$
- $wp(s_0 \oplus s_1,\ Q) = wp(s_0, Q) \wedge wp(s_1, Q)$
- $wp(\rho,\ Q) = App(\rho) \Rightarrow Q[a_n]...[a_0]$
- $wp(s^*,\ Q) = inv_s$

Verification Conditions

- $vc(\rho,\ Q) = \top$
- $vc(s_0; s_1,\ Q) = vc(s_0, wp(s_1,\ Q)) \wedge vc(s_1, Q)$
- $vc(s_0 \oplus s_1,\ Q) = vc(s_0, Q) \wedge vc(s_1, Q)$
- $vc(s^*,\ Q) =$
  $(inv_s \wedge \neg App(s) \Rightarrow Q) \wedge (inv_s \wedge App(s) \Rightarrow wp(s, inv_s)) \wedge vc(s, inv_s)$

Soundness of the verification

Definition: Correctness formula

Let $spec = (Pre, strat, Post)$ be a specification. We call *correctness formula* the formula $correct(spec) = (Pre \Rightarrow wp(strat, Post)) \wedge vc(strat, Post)$.

Theorem:

If $correct(spec)$ is valid, then for all graphs G, G' such that $G \rightarrow_{strat} G'$, $G \models Pre$ implies $G' \models Post$.

Decidability of the verification

Theorem:
Let $spec = (Pre, strat, Post)$ be a specification using one of the following DL logics $\mathcal{ALCUO}$, $\mathcal{ALCUOI}$, $\mathcal{ALCQUOI}$, $\mathcal{ALCUOS}elf$, $\mathcal{ALCUOIS}elf$, and $\mathcal{ALCQUOIS}elf$. Then, the correctness of $spec$ is decidable.

Conclusion

- We identified several DL logics that can be used for the verification of graph/model transformation systems (those closed under substitutions)

Conclusion

- We identified several DL logics that can be used for the verification of graph/model transformation systems (those closed under substitutions)
- We identified DL logics which are not closed under substitutions and thus cannot be involved in the computation of weakest preconditions

Conclusion

- We identified several DL logics that can be used for the verification of graph/model transformation systems (those closed under substitutions)
- We identified DL logics which are not closed under substitutions and thus cannot be involved in the computation of weakest preconditions
- The considered graph transformation systems are featuring actions such as node cloning and merging, in addition to classical node and edge addition and deletion.

Conclusion

- We identified several DL logics that can be used for the verification of graph/model transformation systems (those closed under substitutions)
- We identified DL logics which are not closed under substitutions and thus cannot be involved in the computation of weakest preconditions
- The considered graph transformation systems are featuring actions such as node cloning and merging, in addition to classical node and edge addition and deletion.
- The considered graphs/models are assumed to be labeled by by concepts and roles of the considered DL logics.

Conclusion

- We identified several DL logics that can be used for the verification of graph/model transformation systems (those closed under substitutions)
- We identified DL logics which are not closed under substitutions and thus cannot be involved in the computation of weakest preconditions
- The considered graph transformation systems are featuring actions such as node cloning and merging, in addition to classical node and edge addition and deletion.
- The considered graphs/models are assumed to be labeled by by concepts and roles of the considered DL logics.
- Future work:
  - An implementation with connections to SMT solvers
  - Allow the use of data as labels in addition to logical formulas
  - Devise other decidable logics