Erfüllbarkeit von verallgemeinerten Graphbedingungen

Lara Stoltenow

Erfüllbarkeit von verallgemeinerten Graphbedingungen mit SAT-Solvern

Lara Stoltenow

Motivation / Ziel

- Bedingungen über Graphen formulieren

  (Ausdrucksmächtigkeit ungefähr wie First Order Logic:

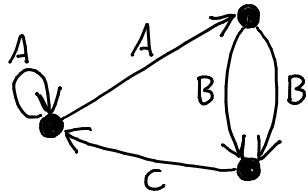
  "es gibt Teilgraph ...", "immer wenn ... dann ...")

- Automatisiert auf Erfüllbarkeit prüfen und

  - Modell oder
  - Unerfüllbarkeitsbeweis

  ausgeben.

Graphen

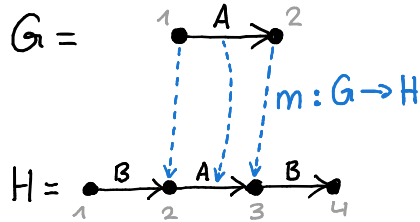


Dieser Talk: gerichtete Graphen mit beschrifteten Mehrfachkanten

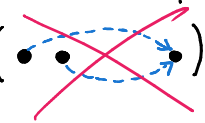
(ist aber alles auch auf Hyperkanten, Knotenlabels usw. anwendbar)

Graphmorphismen

- Strukturerhaltende Abbildung zwischen zwei Graphen
- z.B. um ein Subgraph-Vorkommen von G in H zu beschreiben



- Alternative Interpretation: $g: I \to G$ ist ein Graph G mit Interface I
- Dieser Talk: Meistens injektiv



- Notation: $[1 \to 2] \to [\bullet \to 1 \to 2 \to \bullet]$
  (neue Elemente grün, Rest wird nach Form/Position identifiziert)

Graphbedingungen

$\forall f.A$, $\exists f.A$ für einen Graphmorphismus f und Kindbedingung A
$A \wedge B$, $A \vee B$, $true$, $false$
(im Wesentlichen First Order Logic)

"es gibt irgendwo im Graphen ...." $\exists \emptyset \to [\bullet \xrightarrow{A} \bullet].\ true$

"immer wenn es ... gibt, dann gibt es in der direkten Umgebung auch..." $\forall \emptyset \to [1 \xrightarrow{A} 2].\ (\exists [1 \xrightarrow{A} 2] \to [1 \xrightarrow{A} 2 \xrightarrow{B} 3].true \vee \exists [1 \xrightarrow{A} 2] \to [0 \xrightarrow{B} 1 \xrightarrow{A} 2].true)$

"es gibt keine antiparallelen Kanten" $\forall \emptyset \to [1 \underset{A}{\overset{A}{\rightleftarrows}} 2].\ false$

(Negation: wie in FOL: $\neg\exists \ldots \equiv \forall\neg \ldots$)

(Un-)Erfüllbarkeit prüfen

- Gegeben: Graphbedingung,

  gesucht: Modell oder Unerfüllbarkeitsbeweis

- $\forall / \exists / \wedge / \vee / \neg$, im Wesentlichen First Order Logic → unentscheidbar 😕

(Un-)Erfüllbarkeit prüfen

- Gegeben: Graphbedingung,

  gesucht: Modell oder Unerfüllbarkeitsbeweis

- $\forall / \exists / \wedge / \vee / \neg$, im Wesentlichen First Order Logic → unentscheidbar 😕

- gibts eigentlich schon Tools für (Prover9/Mace, z3, ...)

  - braucht aber umständliche Kodierung Graph $\rightleftarrows$ SMT-Formel

  - schwer auf andere Kategorien verallgemeinerbar

- Unser Ansatz: direkt auf Graphen arbeiten

  - Operationen der Kategorie leichter implementierbar als die Kodierung

(Un-)Erfüllbarkeit prüfen

- Gegeben: Graphbedingung,

  gesucht: Modell oder Unerfüllbarkeitsbeweis

- $\forall / \exists / \wedge / \vee / \neg$, im Wesentlichen First Order Logic → unentscheidbar 😕

- gibts eigentlich schon Tools für [illegible] ein Tableau-basiertes Verfahren von uns für [illegible]

  - braucht aber umständliche Kodierung Graph $\rightleftharpoons$ SMT-Formel

  - schwer auf andere Kategorien verallgemeinerbar

(Un-)Erfüllbarkeit prüfen

- Gegeben: Graphbedingung,

  gesucht: Modell oder Unerfüllbarkeitsbeweis

- $\forall / \exists / \wedge / \vee / \neg$, im Wesentlichen First Order Logic → unentscheidbar 😕

- gibts eigentlich schon Tools für Theorem Proving, z.B. Z3?

  ein Tableau-basiertes Verfahren von uns für

  - braucht aber umständliche Kodierung Graph ⇄ SMT-Formel

  - schwer auf andere Kategorien verallgemeinerbar

  - wir haben die Korrektheit zwar bewiesen, aber
    woher wissen wir, dass es auch wirklich funktioniert?

(Un-)Erfüllbarkeit prüfen

- Gegeben: Graphbedingung,

  gesucht: Modell oder Unerfüllbarkeitsbeweis

- $\forall / \exists / \wedge / \vee / \neg$, im Wesentlichen First Order Logic → unentscheidbar 😕

- gibts eigentlich schon Tools für Theorem Proving, z.B. Z3?

  ein Tableau-basiertes Verfahren von uns für

  - braucht aber umständliche Kodierung Graph ⇄ SMT-Formel

  - schwer auf andere Kategorien verallgemeinerbar ???

  - wir haben die Korrektheit zwar bewiesen, aber
    woher wissen wir, dass es auch wirklich funktioniert?

Related work

CARL VON OSSIETZKY universität OLDENBURG

Fakultät II – Informatik, Wirtschafts- und Rechtswissenschaften
Department für Informatik

Development of Correct Graph Transformation Systems

Dissertation zur Erlangung des Grades eines
Doktors der Naturwissenschaften

vorgelegt von

Dipl.-Inform. Karl-Heinz Pennemann

Oldenburg, 18. Mai 2009

v.a. Section 3.3

UNIVERSITÄT DUISBURG-ESSEN
FAKULTÄT FÜR INGENIEURWISSENSCHAFTEN
ABTEILUNG INFORMATIK UND ANGEWANDTE KOGNITIONSWISSENSCHAFT

Masterarbeit

Erfüllbarkeit und Modellsuche für Graphbedingungen mit Hilfe von prädikatenlogischen Solvern

Marleen Matjeka
Matrikelnummer:

UNIVERSITÄT DUISBURG ESSEN
Offen im Denken

Abteilung Informatik und Angewandte Kognitionswissenschaft
Fakultät für Ingenieurwissenschaften
Universität Duisburg-Essen

Prädikatenlogische Formeln für Graphbedingungen

Universum = Graphelemente (Knoten oder Kanten)

Prädikate:

$inc(e, s, t)$ = e ist eine Kante von s nach t : 

$lab_A(e)$ = Kante e hat Label A : 

Abkürzungen:

$node(n) \quad :\Leftrightarrow \quad \neg \exists s,t : inc(n,s,t)$

Prädikatenlogische Formeln für Graphbedingungen

Universum = Graphelemente (Knoten oder Kanten)

Prädikate $inc(e,s,t)$ = , $lab_A(e)$, $(node(n))$

Zusatzbedingungen (frei nach Pennemann):

- Knoten sind keine Kanten:

$$\forall n: node(n) \rightarrow \left(\neg lab_A(n) \wedge \neg lab_B(n) \ldots \wedge \neg \exists s,t: inc(n,s,t)\right)$$

- Nicht-Knoten sind Kanten...

$$\forall e: \neg node(e) \rightarrow ($$

mit eindeutigem Label

$$\left((\underline{lab_A}(e) \wedge \neg lab_B(e) \wedge \ldots) \vee (\underline{lab_B}(e) \wedge \neg lab_A(e) \wedge \ldots) \right)$$

und eindeutigem Start/Ziel

$$\wedge \forall s_1,s_2,t_1,t_2: inc(e,s_1,t_1) \wedge inc(s_2,t_2) \rightarrow s_1=s_2 \wedge t_1=t_2$$

$)$

Prädikatenlogische Formeln für Graphbedingungen

Universum = Graphelemente (Knoten oder Kanten)

Prädikate $inc(e,s,t)$ =  , $lab_A(e)$, $(node(n))$

Zusatzbedingungen (frei nach Pennemann):

- Knoten sind keine Kanten:

$$\forall n: node(n) \rightarrow \left(\neg lab_A(n) \wedge \neg lab_B(n) \ldots \wedge \neg \exists s,t: inc(n,s,t)\right)$$

- Nicht-Knoten sind Kanten... $\forall e: \neg node(e) \rightarrow ($

mit eindeutigem Label $\left((\underline{lab_A}(e) \wedge \neg lab_B(e) \wedge \ldots) \vee (\underline{lab_B}(e) \wedge \neg lab_A(e) \wedge \ldots) \right)$

und eindeutigem Start/Ziel $\wedge \forall s_1,s_2,t_1,t_2: \underline{inc(e,s_1,t_1) \wedge inc(s_2,t_2)} \rightarrow s_1=s_2 \wedge t_1=t_2$

Funktionssymbole??

)

Übersetzen von Bedingungen in Formeln

Überwiegend 1:1, und Graphmorphismen wie folgt:

Bei $\exists m.A$ führt Morphismus m einige Elemente neu ein und diese fügen wir dann der Formel hinzu.

$\forall \emptyset \to [\bullet_1]. \exists [①] \to [① \xrightarrow{A} ②]. \text{true}$

Übersetzen von Bedingungen in Formeln

Überwiegend 1:1, und Graphmorphismen wie folgt:

Bei $\exists m.A$ führt Morphismus m einige Elemente neu ein und diese fügen wir dann der Formel hinzu.

$\forall \emptyset \to [\bullet_1].\ \exists [\textcircled{1}] \to [\textcircled{1} \xrightarrow{A} \bullet_2].\ true$

(

$\forall n_1.\ node(n_1) \to$

$\exists n_2, e.\ (node(n_2) \wedge n_1 \neq n_2 \wedge \neg node(e) \wedge lab_A(e)$

$\wedge\ inc(e, n_1, n_2) \wedge true\ )$

Übersetzen von Bedingungen in Formeln

$\forall \varphi \rightarrow [\bullet_1]. \exists [\textcircled{1}] \rightarrow [\textcircled{1} \xrightarrow{A} \bullet_2]. \mathit{true}$

Übersetzen von Bedingungen in Formeln

$\forall \bullet \rightarrow [\bullet]. \exists [\circ] \rightarrow [\circ \xrightarrow{A} \circ]. true$

```
%% GF_FirstEx_AllSucc_mace.txt
formulas(assumptions).

% Knoten sind keine Kanten
all n (node(n) -> ( - labA(n) & - labB(n) & all s all t -inc(n,s,t) )).

% Nicht-Knoten sind Kanten,
all e (- node(e) -> (
        % haben ein Label (aber eindeutig)
        ( (labA(e) & - labB(e)) | (labB(e) & - labA(e)) )
        % haben Start und Zielknoten
        & exists s exists t (inc(e,s,t) & node(s) & node(t))
        % und haben eindeutige Start und Zielknoten
        & all s1 all s2 all t1 all t2 ((inc(e,s1,t1) & inc(e,s2,t2)) -> (s1=s2 & t1=t2))
)).


all n1 node(n1) -> exists n2 exists e (node(n2) & n1 != n2 & -node(e) & labA(e) & inc(e,n1,n2) & $T).

end_of_list.
```

Übersetzen von Bedingungen in Formeln

$\forall \phi \rightarrow [1].\exists [0] \rightarrow [0 \xrightarrow{A} 2].\ true$

```
0 [2 jobs] 23:24 penma@lf261lara:/wh/Documents/LADR1007B-win% ./bin/mace4.exe  -
n 2  -m 1  -f GF_FirstEx_AllSucc_mace.txt | ./bin/interpformat.exe cooked | egre
p -v '^-'

=== Mace4 starting on domain size 2. ===

------ process 22156 exit (max_models) ------
% number = 1
% seconds = 0

% Interpretation of size 2

n1 = 0.

c1 = 0.

c2 = 0.

c3 = 0.

f1(0) = 1.
f1(1) = 0.

f2(0) = 1.
f2(1) = 0.


  labB(0).

  node(1).

  inc(0,1,1).
```

Übersetzen von Bedingungen in Formeln

$\forall \phi \rightarrow [1].\exists [0] \rightarrow [0 \xrightarrow{A} 2].\ true$

```
0 [2 jobs] 23:24 penma@lf261lara:/wh/Documents/LADR1007B-win% ./bin/mace4.exe  -
n 2  -m 1  -f GF_FirstEx_AllSucc_mace.txt | ./bin/interpformat.exe cooked | egre
p -v '^-'

=== Mace4 starting on domain size 2. ===

------ process 22156 exit (max_models) ------
% number = 1
% seconds = 0

% Interpretation of size 2

n1 = 0.

c1 = 0.

c2 = 0.

c3 = 0.

f1(0) = 1.
f1(1) = 0.

f2(0) = 1.
f2(1) = 0.


  labB(0).

  node(1).

  inc(0,1,1).
```

???

Übersetzen von Bedingungen in Formeln

$\forall \varphi \rightarrow [\bullet_1].\ \exists [\circ_1] \rightarrow [\circ_1 \xrightarrow{A} \circ_2].\ true$

```
%% GF_FirstEx_AllSucc_mace.txt
formulas(assumptions).

% Knoten sind keine Kanten
all n (node(n) -> ( - labA(n) & - labB(n) & all s all t -inc(n,s,t) )).

% Nicht-Knoten sind Kanten,
all e (- node(e) -> (
        % haben ein Label (aber eindeutig)
        ( (labA(e) & - labB(e)) | (labB(e) & - labA(e)) )
        % haben Start und Zielknoten
        & exists s exists t (inc(e,s,t) & node(s) & node(t))
        % und haben eindeutige Start und Zielknoten
        & all s1 all s2 all t1 all t2 ((inc(e,s1,t1) & inc(e,s2,t2)) -> (s1=s2 & t1=t2))
)).


all n1 (node(n1) -> exists n2 exists e (node(n2) & n1 != n2 & -node(e) & labA(e) & inc(e,n1,n2) & $T)).

end_of_list.
```

Übersetzen von Bedingungen in Formeln

$\forall \phi \rightarrow [\,1\,].\ \exists\, [\,0\,] \rightarrow [\,0 \xrightarrow{A} 2\,].\ true$

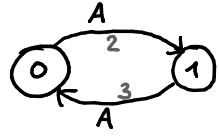
```
0 [2 jobs] 23:28 penma@lf261lara:/wh/Documents/LADR1007B-win% ./bin/mace4.
exe  -n 2  -m 1  -f GF_FirstEx_AllSucc_mace.txt | ./bin/interpformat.exe c
ooked | egrep -v '^-'

=== Mace4 starting on domain size 2. ===

=== Mace4 starting on domain size 3. ===

=== Mace4 starting on domain size 4. ===

------ process 20452 exit (max_models) ------
% number = 1
% seconds = 0

% Interpretation of size 4

f1(0) = 0.
f1(1) = 0.
f1(2) = 0.
f1(3) = 1.

f2(0) = 0.
f2(1) = 0.
f2(2) = 1.
f2(3) = 0.

f3(0) = 1.
f3(1) = 0.
f3(2) = 0.
f3(3) = 0.

f4(0) = 2.
f4(1) = 3.
f4(2) = 0.
f4(3) = 0.

  labA(2).
  labA(3).


  node(0).
  node(1).

  inc(2,0,1).
  inc(3,1,0).
```

???

Übersetzen von Bedingungen in Formeln

$\forall \phi \rightarrow [1].\ \exists [0] \rightarrow [0 \xrightarrow{A} 2].\ true$

```
0 [2 jobs] 23:29 penma@lf261lara:/wh/Documents/LADR1007B-win% ./bin/mace4.
exe  -n 2  -m 1  -f GF_FirstEx_AllSucc_mace.txt | ./bin/interpformat.exe c
ooked | egrep -v '^-' | egrep -v '^f' | cat -s

=== Mace4 starting on domain size 2. ===

=== Mace4 starting on domain size 3. ===

=== Mace4 starting on domain size 4. ===

------ process 1376 exit (max_models) ------
% number = 1
% seconds = 0

% Interpretation of size 4

  labA(2).
  labA(3).

  node(0).
  node(1).

  inc(2,0,1).
  inc(3,1,0).
```



Issues

- Laufzeit!
- Zwei Tools parallel:

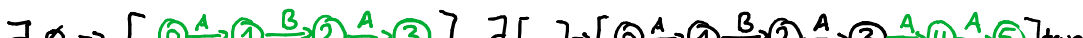
  Prover9 für Unerfüllbarkeitsbeweise

  Mace4 für endliche Modelle.

- Kodierung umständlich und fehleranfällig, weil Elemente von Hand getypt werden müssen

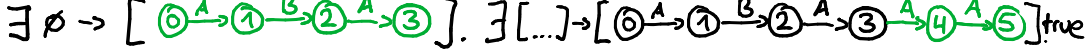
- Macht (mehr oder weniger) Bruteforce mit aufsteigender Länge

Issues




```
%% GF_BigExists_mace.txt
...
exists n0 exists n1 exists n2 exists n3 exists e1 exists e2 exists e3 (
        n0 != n1 & n0 != n2 & n0 != n3 &
        n1 != n2 & n1 != n3 & n2 != n3 &
        e1 != e2 & e1 != e3 & e2 != e3 &
        node(n0) & node(n1) & node(n2) & node(n3) &
        -node(e1) & -node(e2) & -node(e3) &
        inc(e1,n0,n1) & labA(e1) &
        inc(e2,n1,n2) & labB(e2) &
        inc(e3,n2,n3) & labA(e3) &
        exists n4 exists n5 exists e4 exists e5 (
                n0 != n4 & n0 != n5 & n1 != n4 & n1 != n5 &
                n2 != n4 & n2 != n5 & n3 != n4 & n3 != n5 & n4 != n5 &
                e1 != e4 & e1 != e5 & e2 != e4 & e2 != e5 &
                e3 != n4 & n3 != n5 & n4 != n5 &
                inc(e4,n3,n4) & labA(e4) &
                inc(e5,n4,n5) & labA(e5)
        )
).
```

Issues

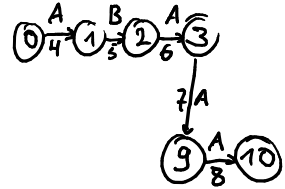



```
2 [1 job] 23:43 penma@lf261lara:/wh/Documents/LADR1007B-win% ./bin/mace4.exe  -n 2  -m 1
  -f GF_BigExists_mace.txt | ./bin/interpformat.exe cooked | egrep -v '^-' | egrep -v '^
f' | cat -s

=== Mace4 starting on domain size 2. ===

=== Mace4 starting on domain size 3. ===

=== Mace4 starting on domain size 4. ===

=== Mace4 starting on domain size 5. ===

=== Mace4 starting on domain size 6. ===

=== Mace4 starting on domain size 7. ===

=== Mace4 starting on domain size 8. ===

=== Mace4 starting on domain size 9. ===

=== Mace4 starting on domain size 10. ===

------ process 17956 exit (exhausted) ------
```

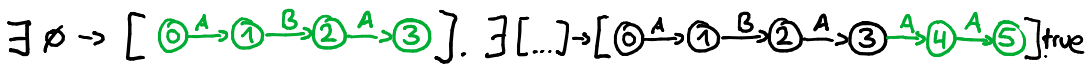
Issues

$\exists \phi \rightarrow [\ 0 \xrightarrow{A} 1 \xrightarrow{B} 2 \xrightarrow{A} 3\ ].\ \exists [\ldots] \rightarrow [\ 0 \xrightarrow{A} 1 \xrightarrow{B} 2 \xrightarrow{A} 3 \xrightarrow{A} 4 \xrightarrow{A} 5\ ]$ true

```
0 [1 job] 23:44 penma@lf261lara:/wh/Documents/LADR1007B-win% ./bin/mace4.exe  -n 10 -N 1
5  -m 1  -f GF_BigExists_mace.txt | ./bin/interpformat.exe cooked | egrep -v '^-' | egre
p -v '^[cf]' | cat -s

=== Mace4 starting on domain size 10. ===

=== Mace4 starting on domain size 11. ===

------ process 18752 exit (max_models) ------
% number = 1
% seconds = 0

% Interpretation of size 11

  labA(4).
  labA(6).
  labA(7).
  labA(8).

  labB(5).

  node(0).
  node(1).
  node(2).
  node(3).
  node(9).
  node(10).

  inc(4,0,1).
  inc(5,1,2).
  inc(6,2,3).
  inc(7,3,9).
  inc(8,9,10).
```

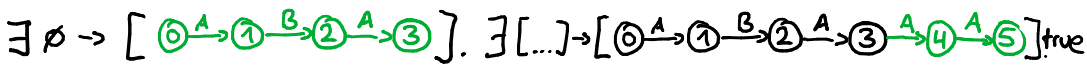


z3




```
%% GF_BigExists_z3_sorts.txt
...
(assert
        (exists ((n0 Node) (n1 Node) (n2 Node) (n3 Node) (e1 Edge) (e2 Edge) (e3 Edge)) (and
                (distinct n0 n1 n2 n3)
                (distinct e1 e2 e3)
                (inc e1 n0 n1) (labA e1)
                (inc e2 n1 n2) (labB e2)
                (inc e3 n2 n3) (labA e3)
                (exists ((n4 Node) (n5 Node) (e4 Edge) (e5 Edge)) (and
                        (distinct n0 n1 n2 n3 n4 n5)
                        (distinct e1 e2 e3 e4 e5)
                        (inc e4 n3 n4) (labA e4)
                        (inc e5 n4 n5) (labA e5)
                ))
        ))
)
```

z3




```
%% GF_BigExists_z3_sorts.txt
...
(assert
        (exists ((n0 Node) (n1 Node) (n2 Node) (n3 Node) (e1 Edge) (e2 Edge) (e3 Edge)) (and
                (distinct n0 n1 n2 n3)
                (distinct e1 e2 e3)
                (inc e1 n0 n1) (labA e1)
                (inc e2 n1 n2) (labB e2)
                (inc e3 n2 n3) (labA e3)
                (exists ((n4 Node) (n5 Node) (e4 Edge) (e5 Edge)) (and
                        (distinct n0 n1 n2 n3 n4 n5)
                        (distinct e1 e2 e3 e4 e5)
                        (inc e4 n3 n4) (labA e4)
                        (inc e5 n4 n5) (labA e5)
                ))
        ))
)
```

many-sorted logic ♡

z3

$\exists \rho \rightarrow [\, 0 \xrightarrow{A} 1 \xrightarrow{B} 2 \xrightarrow{A} 3 \,]. \; \exists [\ldots] \rightarrow [\, 0 \xrightarrow{A} 1 \xrightarrow{B} 2 \xrightarrow{A} 3 \xrightarrow{A} 4 \xrightarrow{A} 5 \,] \text{ true}$

```
0 [1 job] 23:54 penma@lf261lara:/wh/Documents/LADR1007B-
.exe   -smt2 -T:3   "$(wslpath -w GF_BigExists_z3_sorts.
sat
(
  ;; universe for Edge:
  ;;   Edge!val!0 Edge!val!1 Edge!val!2 Edge!val!3 Edge!
  ;; -----------
  ;; definitions for universe elements:
  (declare-fun Edge!val!0 () Edge)
  (declare-fun Edge!val!1 () Edge)
  (declare-fun Edge!val!2 () Edge)
  (declare-fun Edge!val!3 () Edge)
  (declare-fun Edge!val!4 () Edge)
  ;; cardinality constraint:
  (forall ((x Edge))
          (or (= x Edge!val!0)
              (= x Edge!val!1)
              (= x Edge!val!2)
              (= x Edge!val!3)
              (= x Edge!val!4)))
  ;; -----------
  ;; universe for Node:
  ;;   Node!val!3 Node!val!0 Node!val!1 Node!val!4 Node!
  ;; -----------
  ;; definitions for universe elements:
  (declare-fun Node!val!3 () Node)
  (declare-fun Node!val!0 () Node)
```

```
;; -----------
(define-fun inc ((x!0 Edge) (x!1 Node) (x!2 Node)) Bool
  (or (and (= x!0 Edge!val!4)
           (not (= x!0 Edge!val!0))
           (not (= x!0 Edge!val!1))
           (= x!1 Node!val!4)
           (not (= x!1 Node!val!5))
           (not (= x!1 Node!val!0))
           (not (= x!1 Node!val!3))
           (not (= x!1 Node!val!1))
           (= x!2 Node!val!5)
           (not (= x!2 Node!val!0))
           (not (= x!2 Node!val!3))
           (not (= x!2 Node!val!1)))
      (and (= x!0 Edge!val!0)
           (not (= x!0 Edge!val!1))
           (= x!1 Node!val!1)
           (not (= x!2 Node!val!4))
           (not (= x!2 Node!val!5))
           (not (= x!2 Node!val!0))
           (not (= x!2 Node!val!3))
           (not (= x!2 Node!val!1)))
      (and (= x!0 Edge!val!3)
           (not (= x!0 Edge!val!4))
           (not (= x!0 Edge!val!0))
           (not (= x!0 Edge!val!1))
           (= x!1 Node!val!3)
           (not (= x!1 Node!val!1))
           (= x!2 Node!val!4)
           (not (= x!2 Node!val!5))
           (not (= x!2 Node!val!0))
           (not (= x!2 Node!val!3))
           (not (= x!2 Node!val!1)))
      (and (= x!0 Edge!val!1)
           (not (= x!1 Node!val!4))
```

z3

(Formel ist genau so aus dem Testgenerator rausgefallen)

```
%% GF_testMS30-2.txt
...
(assert (and
        (not (exists ((x0 Node)) (and (not (exists ((x1 Node)) (and (and
        (distinct x1 x0)
        (exists ((z0 Edge)) (and (and (inc z0 x0 x1 ) (labA z0 )) true)))))))))
        (exists ((x2 Node)) (and true))
))

(check-sat)
(get-model)
0 [1 job] 0:02 penma@lf261lara:/wh/Documents/LADR1007B-win% /wh/Documents/z3-4.12.2-
x64-win/bin/z3.exe   -smt2 -T:3   "$(wslpath -w GF_testMS30-2.txt )"
```

z3

(Formel ist genau so aus dem Testgenerator rausgefallen)

```
%% GF_testMS30-2.txt
...
(assert (and
        (not (exists ((x0 Node)) (and (not (exists ((x1 Node)) (and (and
        (distinct x1 x0)
        (exists ((z0 Edge)) (and (and (inc z0 x0 x1 ) (labA z0 )) true)))))))))
        (exists ((x2 Node)) (and true))
))

(check-sat)
(get-model)
0 [1 job] 0:02 penma@lf261lara:/wh/Documents/LADR1007B-win% /wh/Documents/z3-4.12.2-
x64-win/bin/z3.exe    -smt2 -T:3    "$(wslpath -w GF_testMS30-2.txt )"
timeout
```

z3

(Formel ist genau so aus dem Testgenerator rausgefallen)

```
%% GF_testMS30-2.txt
...
(assert (and
        (not (exists ((x0 Node)) (and (not (exists ((x1 Node)) (and (and
        (distinct x1 x0)
        (exists ((z0 Edge)) (and (and (inc z0 x0 x1 ) (labA z0 )) true)))))))))
        (exists ((x2 Node)) (and true))
))

(check-sat)
(get-model)
0 [1 job] 0:02 penma@lf261lara:/wh/Documents/LADR1007B-win% /wh/Documents/z3-4.12.2-
x64-win/bin/z3.exe   -smt2 -T:3   "$(wslpath -w GF_testMS30-2.txt )"
timeout
```

z3

(Formel ist genau so aus dem Testgenerator rausgefallen)

```
%% GF_testMS30-2-pp.txt
...
(assert (and
        (not (exists ((x0 Node)) (not (exists ((x1 Node)) (and (and
        (distinct x1 x0)
        (exists ((z0 Edge)) (and (inc z0 x0 x1 ) (labA z0 ) true))))))))
        (exists ((x2 Node)) true)
))

(check-sat)
(get-model)
0 [2 jobs] 0:09 penma@lf261lara:/wh/Documents/LADR1007B-win% /wh/Documents/z3-4.12.2
-x64-win/bin/z3.exe   -smt2 -T:3   "$(wslpath -w GF_testMS30-2-pp.txt )"
sat
(
  ;; universe for Edge:
  ;;   Edge!val!0 Edge!val!1 Edge!val!6 Edge!val!2 Edge!val!3 Edge!val!7 Edge!val!5
Edge!val!4
  ;; -----------
```

z3

(Formel ist genau so aus dem Testgenerator rausgefallen)

```
%% GF_testMS30-2-pp.txt
...
(assert (and
        (not (exists ((x0 Node)) (not (exists ((x1 Node)) (and (and
        (distinct x1 x0)
        (exists ((z0 Edge)) (and (inc z0 x0 x1 ) (labA z0 ) true))))))))
        (exists ((x2 Node)) true)
))

(check-sat)
(get-model)
0 [2 jobs] 0:09 penma@lf261lara:/wh/Documents/LADR1007B-win% /wh/Documents/z3-4.12.2
-x64-win/bin/z3.exe    -smt2 -T:3    "$(wslpath -w GF_testMS30-2-pp.txt )"
sat
(
  ;; universe for Edge:
  ;;   Edge!val!0 Edge!val!1 Edge!val!6 Edge!val!2 Edge!val!3 Edge!val!7 Edge!val!5
Edge!val!4
  ;; -----------
```

z3

(Formel ist genau so aus dem Testgenerator rausgefallen)

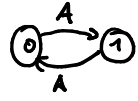
```
%% GF_testMS30-2-ppp.txt
...
(assert (and
        (forall ((x0 Node)) (exists ((x1 Node)) (and (and
        (distinct x1 x0)
        (exists ((z0 Edge)) (and (inc z0 x0 x1 ) (labA z0 ) true))))))
        (exists ((x2 Node)) true)
))

(check-sat)
(get-model)
0 [2 jobs] 0:10 penma@lf261lara:/wh/Documents/LADR1007B-win% /wh/Documents/z3-4.12.2
-x64-win/bin/z3.exe    -smt2 -T:3    "$(wslpath -w GF_testMS30-2-ppp.txt )"
timeout
```

z3

(Formel ist genau so aus dem Testgenerator rausgefallen)

```
%% GF_testMS30-2-ppp.txt
...
(assert (and
        (forall ((x0 Node)) (exists ((x1 Node)) (and (and
        (distinct x1 x0)
        (exists ((z0 Edge)) (and (inc z0 x0 x1 ) (labA z0 ) true))))))
        (exists ((x2 Node)) true)
))

(check-sat)
(get-model)
0 [2 jobs] 0:10 penma@lf261lara:/wh/Documents/LADR1007B-win% /wh/Documents/z3-4.12.2
-x64-win/bin/z3.exe    -smt2 -T:3    "$(wslpath -w GF_testMS30-2-ppp.txt )"
timeout
```

$$\forall\, \varnothing \to [\,①\,].\ \exists\, [\,①\,] \to [\,① \xrightarrow{A} ②\,].\ \text{true}$$



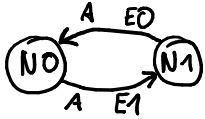
4

Z3 4.x uses two main engines for handling quantifiers: EMatching and MBQI (Model Based Quantifier Instantiation). The EMatching engine is only effective for unsatisfiable instances. That is, it will never be able to show that a formula (containing quantifiers) is satisfiable. On the other hand, the MBQI can do that. Actually, it can decide many useful fragments. The Z3 guide (section Quantifiers) describes some of these fragments. That being said, Z3 does not have finite model finder for first-order logic (like Paradox). This is a useful feature, and we may include it in the future. The example in your message, can be solved using Z3. You can try it here.

Regarding Patterns, they are "hints" for the EMatching engine. Since the EMatching engine can't show that a problem is satisfiable, they will not really help. For satisfiable instances, we can add patterns because we don't want the EMatching engine to get in the way of the MBQI engine by generating too many instances; or we want to eagerly prune the search space by asserting simple instances of the quantifier. We may also disable the EMatching engine using the option `(set-option :ematching false)`.

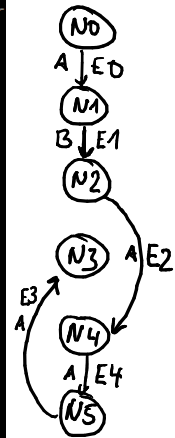
cvc5

```
0 [2 jobs] 0:21 penma@lf261lara:/wh/Documents/LADR1007B-win% /wh/Downloads/cvc5-2023
-07-26-x86_64-win64-production.exe  --tlimit=5000 --lang smt2.6 --finite-model-find
  "$(wslpath -w GF_testMS30-2.txt  )" --model-u-print=decl-sort-and-fun --produce-pr
oofs 2>/dev/null | tail -n +3 | sed 's/(as \(@[^ ]\+\) [^)]\+)/\1/g'
sat
(
; cardinality of Node is 2
(declare-sort Node 0)
(declare-fun @Node_0 () Node)
(declare-fun @Node_1 () Node)
; cardinality of Edge is 2
(declare-sort Edge 0)
(declare-fun @Edge_0 () Edge)
(declare-fun @Edge_1 () Edge)
(define-fun inc (($x1 Edge) ($x2 Node) ($x3 Node)) Bool (or (and (= @Edge_0 $x1) (=
@Node_1 $x2) (= @Node_0 $x3)) (and (= @Edge_1 $x1) (= @Node_0 $x2) (= @Node_1 $x3)))
)
(define-fun labA (($x1 Edge)) Bool true)
(define-fun labB (($x1 Edge)) Bool false)
)
```



cvc5

```
0 [2 jobs] 0:24 penma@lf261lara:/wh/Documents/LADR1007B-win% /wh/Downloads/cvc5-2023
-07-26-x86_64-win64-production.exe  --tlimit=5000 --lang smt2.6 --finite-model-find
  "$(wslpath -w GF_BigExists_z3_sorts.txt  )" --model-u-print=decl-sort-and-fun --pr
oduce-proofs 2>/dev/null | tail -n +3 | sed 's/(as \(@[^ ]\+\) [^)]\+)/\1/g'
sat
(
; cardinality of Node is 6
(declare-sort Node 0)
(declare-fun @Node_0 () Node)
(declare-fun @Node_1 () Node)
(declare-fun @Node_2 () Node)
(declare-fun @Node_3 () Node)
(declare-fun @Node_4 () Node)
(declare-fun @Node_5 () Node)
; cardinality of Edge is 5
(declare-sort Edge 0)
(declare-fun @Edge_0 () Edge)
(declare-fun @Edge_1 () Edge)
(declare-fun @Edge_2 () Edge)
(declare-fun @Edge_3 () Edge)
(declare-fun @Edge_4 () Edge)
(define-fun inc (($x1 Edge) ($x2 Node) ($x3 Node)) Bool (or (and (= @Edge_1 $x1) (=
@Node_1 $x2) (= @Node_2 $x3)) (and (= @Edge_2 $x1) (= @Node_2 $x2) (= @Node_4 $x3))
(and (= @Edge_4 $x1) (= @Node_4 $x2) (= @Node_5 $x3)) (and (= @Edge_3 $x1) (= @Node_
5 $x2) (= @Node_3 $x3)) (and (= @Edge_0 $x1) (= @Node_0 $x2) (= @Node_1 $x3))))
(define-fun labA (($x1 Edge)) Bool (or (= @Edge_0 $x1) (= @Edge_2 $x1) (= @Edge_4 $x
1) (= @Edge_3 $x1)))
(define-fun labB (($x1 Edge)) Bool (= @Edge_1 $x1))
)
```



Weiteres

- Parsen von diesem Output macht wirklich keinen Spaß
- cvc5 hat auch Bindings für verschiedene Programmiersprachen
  - leider etwas cursed und mäßig gut dokumentiert
- z3 hat auch Bindings
  - auch cursed und unterdokumentiert

Weiteres

$\forall\, \varnothing \to [①].\ \mathrm{false}$

Weiteres

$\forall \varnothing \to [①].\ \mathrm{false}$

```
%% GF_AllFalse.txt
...
(assert (forall ((n Node)) false))

(check-sat)
(get-model)
0 [3 jobs] 0:37 penma@lf261lara:/wh/Documents/LADR1007B-win% /wh/Downloads/cvc5-2023
-07-26-x86_64-win64-production.exe  --tlimit=5000 --lang smt2.6 --finite-model-find
  "$(wslpath -w GF_AllFalse.txt  )" --model-u-print=decl-sort-and-fun --produce-proo
fs 2>/dev/null  | tail -n +3
unsat
(error "Cannot get model unless after a SAT or UNKNOWN response.")
```

```
0 [3 jobs] 0:37 penma@lf261lara:/wh/Documents/LADR1007B-win% /wh/Documents/z3-4.12.2
-x64-win/bin/z3.exe   -smt2 -T:3   "$(wslpath -w GF_AllFalse.txt )"
unsat
(error "line 33 column 10: model is not available")
```

Weiteres

- Parsen von diesem Output macht wirklich keinen Spaß
- cvc5 hat auch Bindings für verschiedene Programmiersprachen
  - leider etwas cursed und mäßig gut dokumentiert
- z3 hat auch Bindings
  - auch cursed und unterdokumentiert
- Leere Universen können nicht gefunden werden
  - separat testen, ob Modell?
  - $\emptyset$, [①], [① ②], [① ② ③], ...

Future work

- Übersetzung weiter testen ... funktioniert schon ☺
- Vergleichen mit Tableau-Verfahren
- Kombinierte Algorithmen bauen

  ~ wenn SAT-Solver nicht weiterkommt, dann Tableau-Schritt machen und nochmal versuchen

  vergrößert die Formel zwar ... macht sie aber weniger komplex(?)

  + Tableauschritt garantiert Fortschritt (bei injektiven Graphmorphismen)

Fragen?

Vielen Dank für eure Aufmerksamkeit! ☺