Institute of Computer Science
Knowledge Processing and Information Systems

Titel

Bachelorthesis

Jakob Westphal

28. Juli 2022

Erstgutachter: Prof. Dr. Torsten Schaub
Zweitgutachter: Tobias Stolzmann

Abstract

Abstract in english This is my bachelorthesis

Zusammenfassung

Zusammenfassung auf Deutsch

Acknowledgments

Danksagung!

Contents

1 Introduction

- Answer Set Programming is of growing importance in both academic and industry work (source?) and clingo is a popular solver for this

- Workflow Tools that make working with such programs easier and more comfortable barely exist. Research into these topics is only now getting more traction (source?)

- Testing is a very important part of a conventional software design approach.(source?)

- The topic of code coverage and it's use for conventional programming languages has been shown (source?)

- With this work I want to build on the previous work done on code coverage in answer set programming in this paper by Janhunen et al. [Jan+10]

- To this end I developped a way to efficiently implement the coverage metrics defined in the paper.

- My implementation allows me to compute coverage using answer set programming. It also extends the given metrics to function with almost all existing language construct instead of just for propositional programs.

2 Preliminaries

2.1 Answer Set Programming

- Basic introduction to asp giving all the relevant definitions (with examples + sources (for all/some?))
  - rule, head, (positive/negative)body
  - interpretation that satisfies (positive/negative)body, rule, program
  - Def(a), SuppR(P,I)
  - answer sets, brave/cautious consequence
  - positive atom dependency graph, loops, strongly connected components
  -> adjust these so they work for variables etc. or do this later? (is this needed?/ is it a big adjustment?)

2.2 Testing in Answer Set Programming

- What is input and output of a program?
  - what is a testcase and a testsuite
  - exhaustive test suite for P -> all possible testcases
  - maybe Specification -> the correct (expected) output for every input, what does it mean for a program to "pass/be compliant with" a testcase, when is a program "correct" with respect to a specification (not actually needed for coverage as coverage does not care about specification!)

3 Coverage metrics

- Analogous to concepts of coverage in other (conventional) programming languages (source) I introduce path-like and branch-like coverage metrics according to Janhunen et al. [Jan+10]

3.1 Coverage functions

- general definition of coverage functions (maybe additional source? / compare to what paper did)
  - talk about the difference between all objects and coverable objects
  - trivial/clairvoyant coverage functions maybe not important to discuss?

3.2 Path-like coverage

- general introduction to path like coverage in conventional programming languages (source)
  - use the controlflow graph and cover every possible path through this graph
  - generally the most "complete" coverage metric
  -> these are generally very computationally expensive (exponentially many possible paths)

3.2.1 Program coverage

- Analogous to the conventional path-like coverage I define program coverage
  - Definition + example
  - show that total program coverage means all possible answer sets get produced by the testsuite
  - talk about the problems here? (complexity + not possible for programs with variables as it is necessary to enumerate all possible inputs to find maximum coverage)

3.3 Branch-like coverage

- general introduction to branch-like coverage in conventional programming languages (source)
  - use the controlflow graph and cover every possible branch through this graph
  - less complete but easier to compute, still very potent(?) (source)
  - there are different types of branch like coverage even in conventional programming languages (source), the same is true here

3.3.1 Rule coverage

- Definition + example
  - similar to program coverage in some ways but less complex!
  - some rules may sometimes (or always) not be coverable -> examples -> ties back to beginning of the chapter / thats why coverage is defined on coverable objects
  (- total program coverage implies total rule coverage -> not so relevant but maybe interesting to mention?)

Constraint coverage

(is this an extra section or should this be in the rule coverage section?)
  - constraints are a special type of rule and have to be handled slightly differently because when the body of a constraint is true (=normal rule coverage) this will imply false and therefore not create an answer set / create an unsatisfiable solve call -> we cant check constraints the same way we check other rules
  - solution: (following the suggestion in the paper by Janhunen et al. [Jan+11]) remove the constraint from the program in order to check for its covereage!
  - Definition + examples
  (- maybe reminder that these coverage metrics dont really care about what the output of the program is and whether its according to the specification, therefore removing constraints is okay even though it might completely destroy the functionality of the program)

3.3.2 Loop coverage

- loops play an important role in ASP as seen in (source). Therefore constructing a coverage metric that focuses on them makes sense
  - Definition + example

- generally number of loops in a program in exponential in the number of rules -> expensive to compute! -> introduce 2 more coverage metrics that approximate loop coverage! (one for minimal (singleton) loops and one for maximal loops (strongyl connected components))

(- no real relation to rule coverage (neither implies the other)

- total program coverage implies total loop coverage -> not so relevant but maybe interesting to mention?)

3.3.3 Definition coverage

- 2 ways to introduce definition coverage:

- as a coverage metric for singleton loops (minimal loops) and therefore a special case of loop coverage

- as a representation of the disjunctions in the program (if an atom "a" is defined in multiple rules you could rewrite this as a if B1 v B2 v ...) -> this coverage metric covers these implicit disjunctions -> discuss both but in which order?

- Definition + example

(- this is different to rule coverage! -> total positive rule coverage implies total positive definition coverage, not the other way around and no connection for negative coverage!

- total loop coverage implies total definition coverage

- total program coverage implies total definition coverage)

3.3.4 Component coverage

- strongly connected components are the maximal loops of the program, therefore this is an approximation of loop coverage

- Definition + example

- Definition of negative coverage is different from loop coverage!

- because of this different definition positive/negative component coverage of a specific component implies positive/negative loop coverage for all subset loops of the component (would not be the case for negative coverage if definition is different)

(- total positive loop coverage implies total positive component coverage, however this is not true for negative coverage because of the different definition (see above)

- total program coverage implies total component coverage)

3.4 Complexity

4 Implementation

(maybe include a short history of what i tried but didnt work?)

4.1 General approach

- explain general approach of itroducing labels to the program in order to compute the coverage using ASP
  - why does this work?
  - why does this not change the program?

4.2 Adjustments to original definitions / Implementing coverage for further program classes

- what has to change when trying to use this with variables etc.?
  - why does it still work?
  - why is this correct / comparable to what was done without variables?

4.3 What works / what doesnt work

- overview and explanation of all the existing language constructs in ASP/clingo and why they work / dont work

5 Outlook

- what remains to be done?
  - what can be added?

6 Conclusion

7 Test

1. Bla.
   (Bla.)

2. Bla. (Bla.)

Chapter 1 Referenz zu Kapitel. Section 7.1 Referenz zu Unterkapitel. Figure 7.1 Referenz zu Bild. Listing 7.1 Referenz zu Listing Janhunen et al. [Jan+10] Zitat mit Namen der Autoren [Jan+10] Zitat nur mit Abkürzung Answer Set Programming (ASP) Link zu Abkürzungen *symbol grounding problem* Randkommentar [1] Fussnote Symbol

7.1 Perzepte und Symbole

Bla.

$$\text{match}(\sigma, \gamma) \Leftrightarrow \forall p \in \sigma \; \exists \phi \in \text{feat}(\gamma) : \; g(p, \phi, \gamma(\phi))$$

Proof. Zu jeder Teilmenge $M \subseteq A = \{a, b, c\}$ ist $P^M = P$. Die Teilmengen $\emptyset$, $\{a\}$, $\{c\}$, $\{a, b\}$ und $\{b, c\}$ sind keine Modelle von P^M. $\{a, c\}$, $\{a, b, c\}$ und $\{b\}$ sind Modelle von P^M. $\{a, b, c\}$ ist kein minimales Modell von P^M, da $\{b\} \subseteq \{a, b, c\}$. Da $\{a, c\} \nsubseteq \{b\}$ und $\{b\} \nsubseteq \{a, c\}$, sind beide Modelle minimal und damit stabile Modelle von P. □

[1] Bla

M	P_1^M	$Cn(P_1^M)$
$\emptyset$	$\{a \leftarrow a, b \leftarrow\}$	$\{b\}$
$\{a\}$	$\{a \leftarrow a\}$	$\emptyset$
$\{b\}$	$\{a \leftarrow a, b \leftarrow\}$	$\{b\}$
$\{a, b\}$	$\{a \leftarrow a\}$	$\emptyset$

Table 7.1: $P_1 = \{a \leftarrow a, b \leftarrow naf a\}$ hat ein stabiles Modell $\{b\}$.

Figure 7.1: Ein Kamerabild mit eingezeichneten Perzepten.

```
symbol(cup_1; cup_2; cup_3; spoon; diningtable).

is_on(
    cup_1, diningtable;
    cup_2, diningtable;
    cup_3, diningtable
).

is_inside_of(spoon, cup_3).

contains(
    cup_1, coffee;
    cup_2, coffee;
    cup_3, hot_chocolate
).
```

Listing 7.1: Eine symbolische Beschreibung der Objekte in bla.

`#show p(X,Y) : q(X).`
Test für `#show p(X)` in einer Zeile.

$$
\begin{aligned}
X &= \{\text{cup}_1, \text{cup}_2\} \\
\Pi &= \{\pi_1, \pi_2, \pi_3\} \\
\Phi &= \{\text{coffee}, \text{tea}, \text{hot}, \text{cold}\} \\
T &= \{t_1, t_2\} \\
\beta(\text{cup}_1, t_1) &= \{\text{coffee}\} \\
\beta(\text{cup}_2, t_1) &= \emptyset \\
\beta(\pi_1, t_1) &= \{\text{coffee}\} \\
\beta(\pi_2, t_1) &= \{\text{tea}, \text{cold}\} \\
\beta(\pi_3, t_2) &= \{\text{tea}\}
\end{aligned}
$$

Abbreviations

ASP Answer Set Programming

List of Figures

List of Tables

Listings

Bibliography

[Jan+10] Tomi Janhunen et al. “On testing answer-set programs”. In: *ECAI 2010*. IOS Press, 2010, pp. 951–956 (cit. on pp. 1, 5, 15).

[Jan+11] Tomi Janhunen et al. “Random vs. Structure-Based Testing of Answer-Set Programs: An Experimental Comparison”. In: *Logic Programming and Nonmonotonic Reasoning*. Ed. by James P. Delgrande and Wolfgang Faber. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 242–247. ISBN: 978-3-642-20895-9 (cit. on p. 6).

Declaration

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne Hilfe Dritter und ohne Zuhilfenahme anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe. Die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen sind als solche kenntlich gemacht.

Die „Richtlinie zur Sicherung guter wissenschaftlicher Praxis für Studierende an der Universität Potsdam (Plagiatsrichtlinie) - Vom 20. Oktober 2010“, im Internet unter http://uni-potsdam.de/ambek/ambek2011/1/Seite7.pdf, habe ich zur Kenntnis genommen.

Berlin, 28. Juli 2022

Jakob Westphal