Table of Contents

Table of Contents

Motivation: Formal Verification of Programs With Aggregates

- Aggregates are widely used ASP constructs
- They intuitively represent functions on sets

Example: Paths in a graph

$$cost(a, b, 3).\quad cost(b, c, 7).\quad cost(c, a, 1).$$
$$path(a, b).\quad path(b, c).\quad path(c, a).$$
$$expensive \;\text{:-}\; \#sum\{C, X, Y : path(X, Y), cost(X, Y, C)\} \geq 5.$$

Motivation: Formal Veri cation of Non-ground Programs

[redacted]

Grounding replaces variables with constants from the program signature

p(X) :- q(X; Y):

might be replaced by rules

p(1) :- q(1; 1):
p(1) :- q(1; 2):
p(2) :- q(2; 1):
: : :

Motivation: Formal Verification of Non-ground Programs

Disadvantages of grounding

1. Reasoning about the two-step ground and solve procedure is cumbersome
2. Inseparability of problem class and instance

Automatic Verification

1. First-order theorem provers can help verify the adherence of a first-order theory to a specification
2. We would like to translate ASP programs with aggregates into first-order theories

Context

De ning Aggregate Semantics

- The semantics of aggregates are traditionally captured via groundin
- Our goal is to characterize aggregates using the language of classic logic

sum( )

We wish to express that sum is a function on a set of tuples:

sum( ) is the numeral corresponding to the sum of the weights of all tuples in , if contains nitely many tuples with non-zero weights; and 0 otherwise

Example: $sum(\{\langle 2; a\rangle; \langle 3; b\rangle; \langle c; d\rangle\}) = 5$

Table of Contents

Example

$e1 = Y : r(X;Y;Z)=X$
$e2 = Y;Z : r(X;Y;Z)$

$q(X) \wedge sum(set_{e1}(X))$ $(X))$ $e1($