Online appendix for the paper

Non-Monotonic Spatial Reasoning with Answer Set Programming Modulo Theories

published in Theory and Practice of Logic Programming

Przemysław Andrzej Wałęga, Carl Schultz, Mehul Bhatt

Spatial Reasoning. www.spatial-reasoning.com
The DesignSpace Group, Germany. www.design-space.org

Universities of: Warsaw (Poland), Münster (Germany), Bremen (Germany)

submitted November 2 2015; revised April 11 2016; accepted June 2 2016

APPENDICES A–H

Appendix A
ASPMT(QS) – Online Prototypical Dissemination

A minimal prototypical implementation of ASPMT(QS) is available online publicly from Docker Hub, a cloud-based registry service for building and shipping applications. The ASPMT(QS) version 1.0 is published at:

https://hub.docker.com/r/spatialreasoning/aspmtqs/

The following are available via Docker Hub:

1. ASPMT(QS). The core system
2. Paper examples. Minimal working examples from the paper (additional programs may be added as the review of this paper progresses)
3. README. Short description and installation instructions

General information about the broader context of this project, related tools, and links to updates / ongoing work etc of declarative spatial reasoning methods are available at:

http://www.spatial-reasoning.com

Appendix B
Proofs

Table B 1: Polynomial encodings of Allen Interval Algebra (IA) relations between intervals t, s (omitting inverses), where t^-, t^+ are the real start- and end-points of interval t, respectively, and $s-, s+$ are the start- and end-points of interval s.

IA Relation	Polynomial Encoding
before	$t^+ < s^-$
meets	$t^+ = s^-$
equal	$t^- = s^- \wedge t^+ = s^+$
overlaps	$t^- < s^- \wedge t^+ > s^- \wedge t^+ < s^+$
starts	$t^- = s^- \wedge t^+ < s^+$
during	$t^- > s^- \wedge t^+ < s^+$
finishes	$t^- > s^- \wedge t^+ = s^+$

Proof

of Proposition 2.

Each Interval Algebra (IA) relation may be described as a set of equations and inequalities between interval endpoints (see Figure 1 in (Allen 1983)), which is a conjunction of polynomial expressions. Let interval t be defined by a start and end point $t^-, t^+ \in \mathbb{R}$ such that $t^- < t^+$. Table B 1 presents the polynomial encodings for Allen relations between two intervals t, s.

Rectangle Algebra (RA) makes use of IA relations in 2 and 3 dimensions (Guesgen 1989) (page 5). Hence, each relation is a conjunction of polynomial expressions. An axis-aligned block A is defined by three intervals A_x, A_y, A_z which represent the projections of the block onto the orthogonal axes x, y, z respectively. An extract of relations are presented in Table B 2. □

Proof

of Proposition 3.

Each Left-Right (LR) relation (Scivos and Nebel 2004) may be described as a set of equations and inequalities between three points p, a, b (Bhatt et al. 2011). Table B 3 presents the encodings between point p and segment with end-points a, b. Point p is projected onto vector v by taking the dot product,

$$(x_p, y_p) \cdot (x_v, y_v) = x_p x_v + y_p y_v.$$

Thus we can project a point p onto a segment (a, b) with $(p - a) \cdot (b - a)$, and we can project the second end point of the segment onto itself with $(b - a) \cdot (b - a)$. These are used to formalise the *behind*, *in between*, and *in front* relations. □

Table B 2: Extract of polynomial encodings of Rectangle Algebra (RA) relations between axis-aligned blocks A, B (omitting inverses), where A_x, A_y, A_z are the intervals of the projection of A onto orthogonal axes x, y, z respectively.

RA Relation	**Polynomial Encoding**
left of	A_x before B_x
below	A_y before B_y
in front	A_z before B_z
meets on left	A_x meets B_x
meets below	A_y meets B_y
meets in front	A_z meets B_z

Table B 3: Polynomial encodings of Left-Right (LR) relations between point p and segment with end-points a, b.

LR Relation	**Polynomial Encoding**
left of	$(x_b - x_a)(y_p - y_a) > (y_b - y_a)(x_p - x_a)$
collinear	$(x_b - x_a)(y_p - y_a) = (y_b - y_a)(x_p - x_a)$
right of	$(x_b - x_a)(y_p - y_a) < (y_b - y_a)(x_p - x_a)$
start	$x_p = x_a \wedge y_p = y_a$
end	$x_p = x_b \wedge y_p = y_b$
coincident	$(p \text{ collinear } a, b) \wedge 0 \leq (p - a) \cdot (b - a) \leq (b - a) \cdot (b - a)$
in between	$(p \text{ collinear } a, b) \wedge 0 < (p - a) \cdot (b - a) < (b - a) \cdot (b - a)$
behind	$(p \text{ collinear } a, b) \wedge 0 > (p - a) \cdot (b - a)$
in front	$(p \text{ collinear } a, b) \wedge (p - a) \cdot (b - a) > (b - a) \cdot (b - a)$

Proof

of Proposition 4.

In order to formalise RCC–5 relations using polynomial constraints, we first formalise relations of a point being inside, outside or on the boundary of a polygon (Bhatt et al. 2011) as presented in Table B 4.

Each RCC–5 relation may be described by means of relations *part of* $P(a, b)$ and *overlaps* $O(a, b)$. In the domain of convex polygons, $P(a, b)$ is true whenever all vertices of a are in the interior (inside) or on the boundary of b, and $O(a, b)$ is true if there exists a point p that is inside both a and b. Table B 5 presents the encodings and RCC–5 definitions based on the *part of* and *overlaps* relations.[1] Hence, all RCC–5 relations may be described with polynomials, given a finite upper limit on the number of vertices a convex polygon can have. $\square$

[1] An alternative encoding of *overlaps* avoids the additional existentially quantified point due to the hyperplane separation theorem (e.g. see (Schneider 2013) Section 1.3): convex polygons a, b are discrete from each other if there exists a line l such that all vertices of a are *left* or *collinear* to l, and all vertices of b are *right* or *collinear* with l. It is sufficient to check whether some edge of a or some edge of b is such a line of separation (Schultz and Bhatt 2015b) to determine whether a and b are discrete. If a and b are not discrete, then they overlap (i.e. *overlaps* is the negation of *discrete from*).

Table B 4: Polynomial encodings of incidence relations between point p and convex polygon R with vertices $v_1, \ldots, v_n$ (for convenience let $v_{n+1} = v_1$).

Incidence Relation	Polynomial Encoding
inside	$\bigwedge_{i=1}^{n} \Big(p \text{ left of } v_i, v_{i+1}\Big)$
on boundary	$\bigvee_{i=1}^{n} \Big(p \text{ coincident } v_i, v_{i+1}\Big)$
outside	$\bigvee_{i=1}^{n} \Big(p \text{ right of } v_i, v_{i+1}\Big)$

Table B 5: Polynomial encodings of RCC–5 relations (omitting inverses) between convex polygon a with vertices $v_1, \ldots, v_n$ and convex polygon b.

RCC–5 Relations	Polynomial Encoding
part of (P)	$\bigwedge_{i=1}^{n} \Big((v_i \text{ inside } b) \vee (v_i \text{ on boundary } b)\Big)$
overlaps (O)	$\exists p \Big((p \text{ inside } a) \wedge (p \text{ inside } b)\Big)$
equal (EQ)	$(a \text{ part of } b) \wedge (b \text{ part of } a)$
partially overlaps (PO)	$(a \text{ overlaps } b) \wedge \neg(a \text{ part of } b) \wedge \neg(b \text{ part of } a)$
proper part (PP)	$(a \text{ part of } b) \wedge \neg(b \text{ part of } a)$
discrete from (DR)	$\neg(a \text{ overlaps } b)$

Proof

of Proposition 5.

CDC relations are obtained by dividing space with 4 lines into 9 regions. Since halfplanes and their intersections may be described with polynomial expressions, then each of the 9 regions may be encoded with polynomials. A polygon object is in one or more of the 9 cardinal regions by the topological *overlaps* relation between polygons, which can be encoded with polynomials (i.e. by the existence of a shared point) (Bhatt et al. 2011). □

Appendix C $\mathcal{QS}$: Encodings of RCC Relations within ASPMT(QS)

Example of a subset of the encodings for the topological part of $\mathcal{QS}$, namely, RCC-5 relations in a domain of circles:

```
%---------RCC5 circle,circle
%rcc eq - ``circles are equal''
rccEQ(C1,C2)=true <- (x(C1)=X1 & y(C1)=Y1 & r(C1)=R1 & x(C2)=X2 &
y(C2)=Y2 & r(C2)=R2) & (X1=X2 & Y1=Y2 & R1=R2).

rccEQ(C1,C2)=false <- (x(C1)=X1 & y(C1)=Y1 & r(C1)=R1 & x(C2)=X2 &
y(C2)=Y2 & r(C2)=R2) & not (X1=X2 & Y1=Y2 & R1=R2).

%rcc dr - ``circles are discrete''
rccDR(C1,C2)=true <- (x(C1)=X1 & y(C1)=Y1 & r(C1)=R1 & x(C2)=X2 &
y(C2)=Y2 & r(C2)=R2) &
(X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2) >= (R1+R2)*(R1+R2).

rccDR(C1,C2)=false <- (x(C1)=X1 & y(C1)=Y1 & r(C1)=R1 & x(C2)=X2 &
y(C2)=Y2 & r(C2)=R2) &
not (X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2) >= (R1+R2)*(R1+R2).

%rcc pp - ``one circle is a proper part of another''
rccPP(C1,C2)=true <- (x(C1)=X1 & y(C1)=Y1 & r(C1)=R1 & x(C2)=X2 &
y(C2)=Y2 & r(C2)=R2) &
( R1<R2 & (X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2) <= (R1-R2)*(R1-R2) ).

rccPP(C1,C2)=false <- (x(C1)=X1 & y(C1)=Y1 & r(C1)=R1 & x(C2)=X2 &
y(C2)=Y2 & r(C2)=R2) &
not ( R1<R2 & (X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2) <= (R1-R2)*(R1-R2) ).

%rcc ppi - inverse relation of rcc pp
rccPPi(C2,C1)=B <- rccPP(C1,C2)=B.

%rcc po -``circles partially overlap''
rccPO(C1,C2)=true <- (x(C1)=X1 & y(C1)=Y1 & r(C1)=R1 & x(C2)=X2 &
y(C2)=Y2 & r(C2)=R2) &
( (X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2) > (R1-R2)*(R1-R2) &
(X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2) < (R1+R2)*(R1+R2) ).

rccPO(C1,C2)=false <- (x(C1)=X1 & y(C1)=Y1 & r(C1)=R1 & x(C2)=X2 &
y(C2)=Y2 & r(C2)=R2) &
not ( (X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2) > (R1-R2)*(R1-R2) &
(X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2)
< (R1+R2)*(R1+R2) ).
```

Appendix D ASPMT(QS) Encodings for Euclid Constructions

The class of *ruler and compass* problems from Euclid's *Elements* (Heath (ed) 1956) defines constructions of geometric objects using only an idealised ruler and compass: the tools have no markings to measure distances and angles, the compass is collapsable (and so the radius of one circle cannot be transferred directly to another point), and the ruler has infinite length.

D.1 Constructing an Equilateral Triangle

Equilateral triangle construction (Proposition 1, Book 1). Given a segment with endpoints p_1, p_2 the task is to construct an equilateral triangle p_1, p_2, p_3. Construct circle c_1 centred on p_1, coincident to p_2. Construct circle c_2 centred on p_2, coincident to p_1. Circles c_1, c_2 intersect at p_3. The claim is that p_1, p_2, p_3 form an equilateral triangle (Figure D 1).

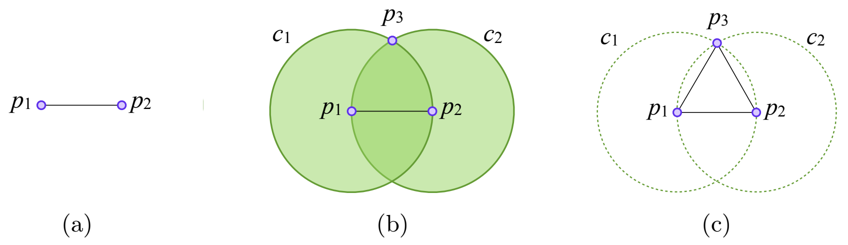


Figure D 1: Ruler compass method for constructing an equilateral triangle given segment p_1, p_2.

```
:- constants
p1   :: point;
p2   :: point;
p3   :: point;
c1   :: circle;
c2   :: circle.

<- coincident(p1,p2).
<- not center(p1,c1).
<- not center(p2,c2).
<- not coincident(p1,c2).
<- not coincident(p2,c1).
<- not coincident(p3,c1).
<- not coincident(p3,c2).
```

To check consistency, i.e., if the constructed triangle is equilateral,add the following line to the code:

```
<- not distanceEQ(p1,p2,p1,p3) | not distanceEQ(p1,p2,p2,p3)
   | not distanceEQ(p1,p3,p2,p3).
```

To check sufficiency, i.e., if it is possible that the triangle is not equilateral add the following line instead:

```
<- distanceEQ(p1,p2,p1,p3) & distanceEQ(p1,p2,p2,p3)
   & distanceEQ(p1,p3,p2,p3).
```

D.2 Bisecting an Angle

Angle Bisector (Proposition 9, Book 1). Given three distinct points p, p_a, p_b such that p_a, p_b are equidistant to p, the task is to bisect the angle formed by the points (about p). Construct circle c centred on p and coincident with p_a and p_b. Construct circles c_a, c_b centred on points p_a, p_b respectively, such that p is coincident with both circles. Circles c_a, c_b intersect at point p_c. The claim is that the segment from p to p_c bisects the angle p_a, p, p_b (Figure D 2).

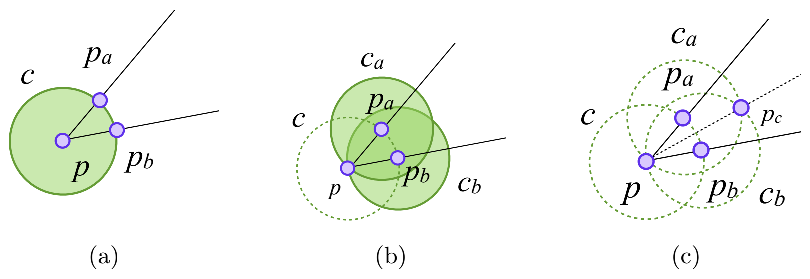


Figure D 2: Ruler and compass method for bisecting the angle p_a, p, p_b.

```
:- constants

p    :: point;
pa   :: point;
pb   :: point;
pc   :: point;
c    :: circle;
ca   :: circle;
cb   :: circle.
```

```
<- coincident(pa,pb).
<- coincident(p,pc).



<- not center(p,c).
<- not center(pa,ca).
<- not center(pb,cb).

<- not coincident(pa,c).
<- not coincident(pb,c).

<- not coincident(p,ca).
<- not coincident(p,cb).

<- not coincident(pc,ca).
<- not coincident(pc,cb).
```

To check consistency, i.e., if it is possible that angles p_c, p, p_a and p_c, p, p_b are equal (equivalently, p_c, p, p_a and p_c, p, p_b are congruent), add the following line to the input program:

```
<- not distanceEQ(p,pa,p,pb) | not distanceEQ(pa,pc,pb,pc).
```

To check sufficiency, i.e., if it is possible that angles p_c, p, p_a and p_c, p, p_b are not equal add the following lines instead:

```
<- car=Xcar & cbr=Xcbr & Xcar!=Xcbr.
<- distanceEQ(p,pa,p,pb) & distanceEQ(pa,pc,pb,pc).
```

D.3 Compass Equivalence Theorem

This theorem establishes that the collapsable property of the idealised compass can in fact be overcome; i.e. a circle's radius can indeed be "copied" to another centre point using only an idealised ruler and compass.

Compass Equivalence Theorem (Proposition 2, Book 1). Given circle c_a centred at point p_a and a distinct point p_b, the task is to construct circle c_b centred on p_b with the same radius as c_a. Construct circle c_1 centred on p_a coincident with p_b. Construct circle c_2 centred on p_b coincident with p_a. Circles c_1, c_2 intersect at point p_c. Construct circle c_3 centred on p_c such that: a point p_d exists with (1) p_d coincident to both c_a and c_3, and (2) p_a lies on the segment p_d, p_3. Construct point p_e such that: (1) p_e is coincident to both c_b and c_3, and (2) p_b lies on the segment p_e, p_3. Finally, construct circle c_b centred on p_b coincident with p_e. The claim is that the radius of c_a equals the radius of c_b (Figure D 3).

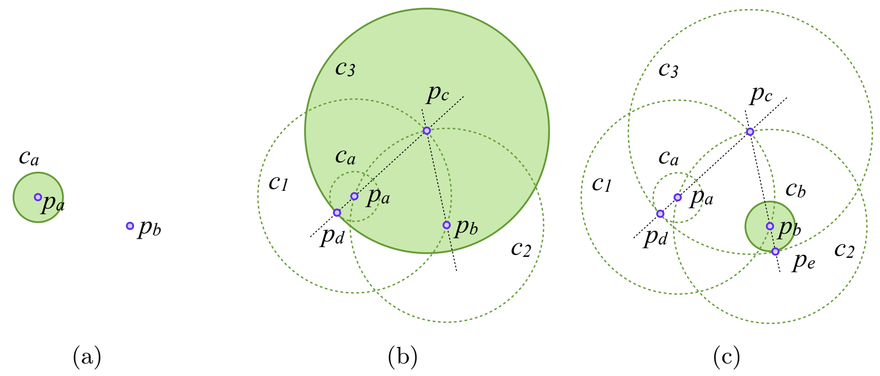


Figure D 3: Ruler and compass method for compass equivalence theorem, transferring the radius of c_a to construct a new circle c_b centred on p_b with the same radius.

```
:- constants
pa   :: point;
pb   :: point;
pc   :: point;
pd   :: point;
pe   :: point;
ca   :: circle;
cb   :: circle;
c1   :: circle;
c2   :: circle;
c3   :: circle.

<- coincident(pa,pb).

<- not center(pa,ca).
<- not center(pa,c1).
<- not center(pb,cb).
<- not center(pb,c2).
<- not center(pc,c3).

<- not coincident(pa,c2).
<- not coincident(pb,c1).
<- not coincident(pc,c1).
<- not coincident(pc,c2).

<- not inside_seg(pa,pc,pd).
<- not inside_seg(pb,pc,pe).

<- not coincident(pd,ca).
<- not coincident(pe,cb).
<- not coincident(pd,c3).
<- not coincident(pe,c3).
```

To check consistency, i.e., if the constructed circles have the same radius add the following line to the input program:

```
<- not car=cbr.
```

To check sufficiency, i.e., if it is possible that the circles have various radius, add the following lines instead:

```
%try to satisfy car!=cbr
<- car=cbr.
```

Appendix E RCC Composition with ASPMT(QS)

ASPMT(QS) is able to compute composition tables for qualitative calculi, e.g., for Region Connection Calculus. To check what may be a relation between circles c_1 and c_3, while c_1 partially overlaps c_2 and c_2 is a proper part of c_3 use the following input program:

```
:- constants

c1  :: circle;
c2  :: circle;
c3  :: circle.

<- not rccPO(c1,c2).
<- not rccPP(c2,c3).
```

In order to check if it is possible that c_1 partially overlaps c_3 add the following line:

```
<- not rccPO(c1,c3).
```

In order to check if it is possible that c_1 is a proper part of c_3 add the following line instead:

```
<- not rccPP(c1,c3).
```

Both of the above programs are satisfiable. However, if we state that there is any other relation between c_1 and c_3, then the program will be unsatisfied. For example, try to add a constraint that c_1 is equal to c_3 in order to obtain inconsistency:

```
<- not rccEQ(c1,c3).
```

Appendix F Encodings for Ramification Problem Examples

The ramification problem examples should be run with flag “–p”, i.e., with explicit encodings of spatial relations in the input file:

aspmtqs -p "input file"

The Growth scenario:

```
:- sorts
step; astep;
point; circle.

:- objects
0..1                 ::  step;
0..0                 ::  astep;
a,b,c                ::  circle.

:- constants
x(circle,step)               ::  real[0..100];
y(circle,step)               ::  real[0..100];
r(circle,step)               ::  real[0..100];
rccEQ(circle,circle,step)    ::  boolean;
rccDC(circle,circle,step)    ::  boolean;
rccEC(circle,circle,step)    ::  boolean;
rccPP(circle,circle,step)    ::  boolean;
grow(circle,astep)           ::  boolean.

:- variables
C, C1, C2                  ::  circle;
S                          ::  step;
AS                         ::  astep.

%------Growing
{grow(C,AS)=false}.

{x(C,AS+1)=X} <- x(C,AS)=X.
{y(C,AS+1)=X} <- y(C,AS)=X.
{r(C,AS+1)=X} <- r(C,AS)=X.

{r(C,AS+1)=X} <- grow(C,AS)=true.
<- grow(C,AS)=true & r(C,AS)=R1 & r(C,AS+1)=R2 & R2<=R1.
```

```
%-----------Initial state------------------------------
{x(C,0)=X}.
{y(C,0)=X}.
{r(C,0)=X}.
rccPP(a,b,0)=true.
rccEC(b,c,0)=true.

grow(a,0)=true.

%------------Goal state --------------------------------
rccEQ(a,b,1)=true.

%rcc eq
rccEQ(C1,C2,S)=true <- (x(C1,S)=X1 & y(C1,S)=Y1 & r(C1,S)=R1
& x(C2,S)=X2 & y(C2,S)=Y2 & r(C2,S)=R2) & (X1=X2 & Y1=Y2 & R1=R2).

rccEQ(C1,C2,S)=false <- (x(C1,S)=X1 & y(C1,S)=Y1 & r(C1,S)=R1
& x(C2,S)=X2 & y(C2,S)=Y2 & r(C2,S)=R2)
& not (X1=X2 & Y1=Y2 & R1=R2).

%rcc pp
rccPP(C1,C2,S)=true <- (x(C1,S)=X1 & y(C1,S)=Y1 & r(C1,S)=R1
& x(C2,S)=X2 & y(C2,S)=Y2 & r(C2,S)=R2) &
( R1<R2 & (X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2) <= (R1-R2)*(R1-R2) ).

rccPP(C1,C2,S)=false <- (x(C1,S)=X1 & y(C1,S)=Y1 & r(C1,S)=R1
& x(C2,S)=X2 & y(C2,S)=Y2 & r(C2,S)=R2)
& not (R1<R2 & (X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2) <= (R1-R2)*(R1-R2)).

%rcc ec
rccEC(C1,C2,S)=true <- (x(C1,S)=X1 & y(C1,S)=Y1 & r(C1,S)=R1
 & x(C2,S)=X2 & y(C2,S)=Y2 & r(C2,S)=R2) &
(X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2) = (R1+R2)*(R1+R2).

rccEC(C1,C2,S)=false <- (x(C1,S)=X1 & y(C1,S)=Y1 & r(C1,S)=R1
& x(C2,S)=X2 & y(C2,S)=Y2 & r(C2,S)=R2) &
not (X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2) = (R1+R2)*(R1+R2).

%rcc dc
rccDC(C1,C2,S)=true <- (x(C1,S)=X1 & y(C1,S)=Y1 & r(C1,S)=R1
& x(C2,S)=X2 & y(C2,S)=Y2 & r(C2,S)=R2)
& (X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2) > (R1+R2)*(R1+R2).

rccDC(C1,C2,S)=false <- (x(C1,S)=X1 & y(C1,S)=Y1 & r(C1,S)=R1
& x(C2,S)=X2 & y(C2,S)=Y2 & r(C2,S)=R2) &
not (X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2) > (R1+R2)*(R1+R2).
```

Encoding for the Motion example:

```
:- sorts
step; astep;
point; circle.

:- objects
0..1                 ::  step;
0..0                 ::  astep;
a,b,c                ::  circle.

:- constants
x(circle,step)               ::  real[0..100];
y(circle,step)               ::  real[0..100];
r(circle,step)               ::  real[0..100];
rccDC(circle,circle,step)    ::  boolean;
rccEC(circle,circle,step)    ::  boolean;
rccTPP(circle,circle,step)   ::  boolean;
rccNTPP(circle,circle,step)  ::  boolean;
move(circle,astep)           ::  boolean.

:- variables
C, C1, C2                  ::  circle;
S                          ::  step;
AS                         ::  astep.

%------Moving
{move(C,AS)=false}.

{x(C,AS+1)=X} <- x(C,AS)=X.
{y(C,AS+1)=X} <- y(C,AS)=X.
{r(C,AS+1)=X} <- r(C,AS)=X.

{x(C,AS+1)=X} <- move(C,AS)=true.
{y(C,AS+1)=X} <- move(C,AS)=true.
<- move(C,AS)=true & x(C,AS)=X1 & y(C,AS)=Y1 & x(C,AS+1)=X2
& y(C,AS+1)=Y2 & X1=X2 & Y1=Y2.
```

```
%-----------Initial state------------------------------
x(a,0)=0.
y(a,0)=0.
x(c,0)=10.
y(c,0)=0.
{x(C,0)=X}.
{y(C,0)=X}.
{r(C,0)=X}.
rccNTPP(a,b,0)=true.
rccEC(b,c,0)=true.
move(a,0)=true.
x(a,1)=1.
y(a,1)=0.
%------------Goal state --------------------------------
rccTPP(a,b,1)=true.
rccEC(a,c,1)=false.
rccDC(a,c,1)=false.
%rcc ec
rccEC(C1,C2,S)=true <- (x(C1,S)=X1 & y(C1,S)=Y1 & r(C1,S)=R1
& x(C2,S)=X2 & y(C2,S)=Y2 & r(C2,S)=R2)
& (X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2) = (R1+R2)*(R1+R2).

rccEC(C1,C2,S)=false <- (x(C1,S)=X1 & y(C1,S)=Y1 & r(C1,S)=R1
& x(C2,S)=X2 & y(C2,S)=Y2 & r(C2,S)=R2)
& not (X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2) = (R1+R2)*(R1+R2).
%rcc dc
rccDC(C1,C2,S)=true <- (x(C1,S)=X1 & y(C1,S)=Y1 & r(C1,S)=R1
& x(C2,S)=X2 & y(C2,S)=Y2 & r(C2,S)=R2)
& (X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2) > (R1+R2)*(R1+R2).

rccDC(C1,C2,S)=false <- (x(C1,S)=X1 & y(C1,S)=Y1 & r(C1,S)=R1
& x(C2,S)=X2 & y(C2,S)=Y2 & r(C2,S)=R2)
& not (X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2) > (R1+R2)*(R1+R2).
%rcc tpp
rccTPP(C1,C2,S)=true <- (x(C1,S)=X1 & y(C1,S)=Y1 & r(C1,S)=R1
& x(C2,S)=X2 & y(C2,S)=Y2 & r(C2,S)=R2)
& ( R1<R2 & (X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2) = (R1-R2)*(R1-R2)).

rccTPP(C1,C2,S)=false <- (x(C1,S)=X1 & y(C1,S)=Y1 & r(C1,S)=R1
& x(C2,S)=X2 & y(C2,S)=Y2 & r(C2,S)=R2)
& not ( R1<R2 & (X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2) = (R1-R2)*(R1-R2)).
%rcc ntpp
rccNTPP(C1,C2,S)=true <- (x(C1,S)=X1 & y(C1,S)=Y1 & r(C1,S)=R1
& x(C2,S)=X2 & y(C2,S)=Y2 & r(C2,S)=R2)
& ( R1<R2 & (X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2) < (R1-R2)*(R1-R2) ).

rccNTPP(C1,C2,S)=false <- (x(C1,S)=X1 & y(C1,S)=Y1 & r(C1,S)=R1
& x(C2,S)=X2 & y(C2,S)=Y2 & r(C2,S)=R2)
& not ( R1<R2 & (X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2) < (R1-R2)*(R1-R2) ).
```

Appendix G Encodings for Geometric Reasoning and the Frame Problem

The frame problem examples should be run with flag "–p", i.e., with explicit encodings of spatial relations in the input file:

aspmtqs -p "input file"

The Attachment I scenario:

```
:- sorts
step; astep;
point; circle.

:- objects
0..1                 ::  step;
0..0                 ::  astep;
car,trailer,garage   ::  circle.

:- constants
x(circle,step)                ::  real[0..100];
y(circle,step)                ::  real[0..100];
r(circle,step)                ::  real[0..100];
rccPP(circle,circle,step)     ::  boolean;
rccEC(circle,circle,step)     ::  boolean;
rccDC(circle,circle,step)     ::  boolean;
move(circle,astep)            ::  boolean;
attach(circle,circle,astep)   ::  boolean;
attached(circle,circle,step)  ::  boolean.

:- variables
C, C1, C2                    ::  circle;
S                            ::  step;
AS                           ::  astep;
B                            ::  boolean.

%------Actions
% move and attach are external actions
{move(C,AS)=B}.
{attach(C1,C2,AS)=B}.
% cannot attach and move in same step
<- attach(C1,C2,AS)=true & move(C1,AS)=true.
<- attach(C1,C2,AS)=true & move(C2,AS)=true.
```

```
%-----Attaching
% only car can attach trailer
attach(C1,C2,AS)=false <- C1!=car | C2!=trailer.
% nothing can attach itself
attach(C1,C2,AS)=false <- C1=C2.
% objects can attach only when are rccEC
<- attach(C1,C2,S)=true & rccEC(C1,C2,S)=false.
% nothing is attached with itself
attached(C1,C2,S)=false <- C1=C2.
% attached is symmetric
<- attached(C1,C2,S)=B & not attached(C2,C1,S)=B.
% attachement don't change
{attached(C1,C2,AS+1)=B} <- attached(C1,C2,AS)=B.
% attach makes objects attached
attached(C1,C2,AS+1)=true <- attached(C1,C2,AS)=false
& attach(C1,C2,AS)=true.
% cannot attach already attached objects
<- attach(C1,C2,S)=true & attached(C1,C2,S)=true.
% attached objects are rccEC
<- attached(C1,C2,S)=true & rccEC(C1,C2,S)=false.
%-----Moving
% garage and trailer cannot move
move(C,AS)=false <- C=garage | C=trailer.
{x(C,S+1)=X} <- x(C,S)=X.
{y(C,S+1)=X} <- y(C,S)=X.
{r(C,S+1)=X} <- r(C,S)=X.
{x(C,S+1)=X} <- move(C,S)=true.
{y(C,S+1)=X} <- move(C,S)=true.
{x(C2,S+1)=X} <- attached(C1,C2,S)=true & move(C1,S)=true.
{y(C2,S+1)=X} <- attached(C1,C2,S)=true & move(C1,S)=true.
%-----Geometry
<- r(C,S)=X & X<=0.
% car must be rccPP or rccDC with garage
<- rccPP(car,garage,S)=false & rccDC(car,garage,S)=false.
% trailer must be rccPP or rccDC with garage
<- rccPP(trailer,garage,S)=false & rccDC(trailer,garage,S)=false.
%-----------Initial state-------------------------------
{x(C,0)=X}.
{y(C,0)=X}.
{r(C,0)=X}.
x(car,0)=0.
y(car,0)=0.
x(garage,0)=10.
y(garage,0)=10.
{attached(C1,C2,0)=false}.
attached(car,trailer,0)=true.
attached(trailer,car,0)=true.
rccDC(car,garage,0)=true.
rccDC(trailer,garage,0)=true.
```

```
%-------------Goal state ---------------------------------
rccPP(car,garage,1)=true.

%rcc pp
rccPP(C1,C2,S)=true <- (x(C1,S)=X1 & y(C1,S)=Y1 & r(C1,S)=R1
& x(C2,S)=X2 & y(C2,S)=Y2 & r(C2,S)=R2)
& ( R1<R2 & (X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2) <= (R1-R2)*(R1-R2) ).

rccPP(C1,C2,S)=false <- (x(C1,S)=X1 & y(C1,S)=Y1 & r(C1,S)=R1
& x(C2,S)=X2 & y(C2,S)=Y2 & r(C2,S)=R2)
& not (R1<R2 & (X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2) <= (R1-R2)*(R1-R2)).

%rcc ec
rccEC(C1,C2,S)=true <- (x(C1,S)=X1 & y(C1,S)=Y1 & r(C1,S)=R1
& x(C2,S)=X2 & y(C2,S)=Y2 & r(C2,S)=R2)
& (X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2) = (R1+R2)*(R1+R2).

rccEC(C1,C2,S)=false <- (x(C1,S)=X1 & y(C1,S)=Y1 & r(C1,S)=R1
& x(C2,S)=X2 & y(C2,S)=Y2 & r(C2,S)=R2)
& not  (X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2) = (R1+R2)*(R1+R2).

%rcc dc
rccDC(C1,C2,S)=true <- (x(C1,S)=X1 & y(C1,S)=Y1 & r(C1,S)=R1
& x(C2,S)=X2 & y(C2,S)=Y2 & r(C2,S)=R2)
& (X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2) > (R1+R2)*(R1+R2).

rccDC(C1,C2,S)=false <- (x(C1,S)=X1 & y(C1,S)=Y1 & r(C1,S)=R1
& x(C2,S)=X2 & y(C2,S)=Y2 & r(C2,S)=R2)
& not (X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2) > (R1+R2)*(R1+R2).
```

The Attachment II scenario:

```
:- sorts
step; astep;
point; circle.

:- objects
0..2                :: step;
0..1                :: astep;
car,trailer,garage  :: circle.
```

```
:- constants
x(circle,step)                    ::  real[0..100];
y(circle,step)                    ::  real[0..100];
r(circle,step)                    ::  real[0..100];
rccPP(circle,circle,step)         ::  boolean;
rccEC(circle,circle,step)         ::  boolean;
rccDC(circle,circle,step)         ::  boolean;
move(circle,astep)                ::  boolean;
disattach(circle,circle,astep)    ::  boolean;
attached(circle,circle,step)      ::  boolean.

:- variables
C, C1, C2       ::  circle;
S               ::  step;
AS              ::  astep;
B               ::  boolean.

%------Actions
% move and attach/disattach are external actions
{move(C,AS)=B}.
{disattach(C1,C2,AS)=B}.

% cannot disattach and move in same step
<- disattach(C1,C2,AS)=true & move(C1,AS)=true.
<- disattach(C1,C2,AS)=true & move(C2,AS)=true.

%-----Attaching/Disattaching and Attach
% only car can attach/disattach trailer
disattach(C1,C2,AS)=false <- C1!=car | C2!=trailer.
% nothing can attach/disattach itself
disattach(C1,C2,AS)=false <- C1=C2.

% nothing is attached with itself
attached(C1,C2,S)=false <- C1=C2.
% attached is symmetric
<- attached(C1,C2,S)=B & not attached(C2,C1,S)=B.
% attachement don't change
{attached(C1,C2,AS+1)=B} <- attached(C1,C2,AS)=B.
% disattach makes objects not attached
attached(C1,C2,AS+1)=false <- attached(C1,C2,AS)=true
& disattach(C1,C2,AS)=true.
attached(C2,C1,AS+1)=false <- attached(C1,C2,AS)=true
& disattach(C1,C2,AS)=true.
% cannot disattach not attached objects
<- disattach(C1,C2,AS)=true & attached(C1,C2,AS)=false.
% attached objects are rccEC
<- attached(C1,C2,S)=true & rccEC(C1,C2,S)=false.
```

```
%-----Moving
% garage and trailer cannot move
move(C,AS)=false <- C=garage | C=trailer.

{x(C,S+1)=X} <- x(C,S)=X.
{y(C,S+1)=X} <- y(C,S)=X.
{r(C,S+1)=X} <- r(C,S)=X.

{x(C,S+1)=X} <- move(C,S)=true.
{y(C,S+1)=X} <- move(C,S)=true.

{x(C2,S+1)=X} <- attached(C1,C2,S)=true & move(C1,S)=true.
{y(C2,S+1)=X} <- attached(C1,C2,S)=true & move(C1,S)=true.

%-----Geometry
<- r(C,S)=X & X<=0.

% car must be rccPP or rccDC with garage
<- rccPP(car,garage,S)=false & rccDC(car,garage,S)=false.
% trailer must be rccPP or rccDC with garage
<- rccPP(trailer,garage,S)=false & rccDC(trailer,garage,S)=false.
%-----------Initial state------------------------------
{x(C,0)=X}.
{y(C,0)=X}.
{r(C,0)=X}.
r(car,0)=1.
x(car,0)=10.
y(car,0)=10.
r(trailer,0)=1.
x(trailer,0)=10.
y(trailer,0)=12.
r(garage,0)=9.
x(garage,0)=1.
y(garage,0)=1.
disattach(car,trailer,0)=false.

{attached(C1,C2,0)=false}.
attached(car,trailer,0)=true.
attached(trailer,car,0)=true.

rccDC(car,garage,0)=true.
rccDC(trailer,garage,0)=true.
%------------Goal state --------------------------------
rccPP(car,garage,2)=true.

rccPP(trailer,garage,2)=true.
```

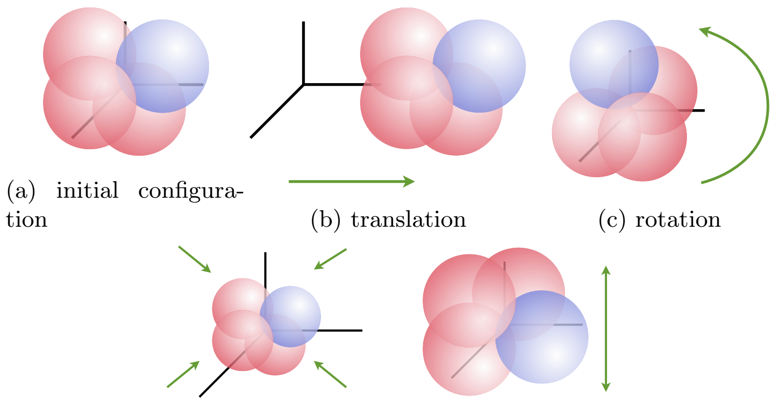
```
%rcc pp
rccPP(C1,C2,S)=true <- (x(C1,S)=X1 & y(C1,S)=Y1 & r(C1,S)=R1
& x(C2,S)=X2 & y(C2,S)=Y2 & r(C2,S)=R2)
& ( R1<R2 & (X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2) <= (R1-R2)*(R1-R2) ).

rccPP(C1,C2,S)=false <- (x(C1,S)=X1 & y(C1,S)=Y1 & r(C1,S)=R1
& x(C2,S)=X2 & y(C2,S)=Y2 & r(C2,S)=R2)
& not (R1<R2 & (X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2) <= (R1-R2)*(R1-R2)).

%rcc ec
rccEC(C1,C2,S)=true <- (x(C1,S)=X1 & y(C1,S)=Y1 & r(C1,S)=R1
& x(C2,S)=X2 & y(C2,S)=Y2 & r(C2,S)=R2)
& (X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2) = (R1+R2)*(R1+R2).

rccEC(C1,C2,S)=false <- (x(C1,S)=X1 & y(C1,S)=Y1 & r(C1,S)=R1
& x(C2,S)=X2 & y(C2,S)=Y2 & r(C2,S)=R2)
& not  (X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2) = (R1+R2)*(R1+R2).

%rcc dc
rccDC(C1,C2,S)=true <- (x(C1,S)=X1 & y(C1,S)=Y1 & r(C1,S)=R1
& x(C2,S)=X2 & y(C2,S)=Y2 & r(C2,S)=R2)
& (X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2) > (R1+R2)*(R1+R2).

rccDC(C1,C2,S)=false <- (x(C1,S)=X1 & y(C1,S)=Y1 & r(C1,S)=R1
& x(C2,S)=X2 & y(C2,S)=Y2 & r(C2,S)=R2)
& not (X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2) > (R1+R2)*(R1+R2).
```

Appendix H Optimisations for Spatial Reasoning in ASPMT(QS)

While computational performance is not our focus here, we describe our future work in integrating spatial optimisations that greatly expand the horizon of problems that can be solved by ASPMT(QS).

The computational complexity of solving general systems of polynomial constraints is highly prohibitive. Specifically, the complexity of Quantifier Elimination by Cylindrical Algebraic Decomposition (Collins 1975) is double exponential in the number of variables in the polynomial constraints, $O(2^{2^n})$ (Arnon et al. 1984). Thus, even small spatial problems become intractable in practice without utilising more efficient polynomial constraint encodings that exploit the structural properties of qualitative spatial domains.

In (Schultz and Bhatt 2015b) we present one powerful optimisation referred to as *spatial symmetry pruning*. The concept is as follows: certain qualitative relations are preserved by certain transformations (on the embedding space). For example, the topological connectivity of a configuration of spheres is not altered if the spheres are translated to some other position as illustrated in Figure H 1 (or rotated, reflected, uniformly scaled).



(a) initial configuration (b) translation (c) rotation (d) uniform scaling (e) reflection

Figure H 1: Topological relations between four spheres maintained after various affine transformations.

We can exploit such properties by spatial symmetry pruning. Transformations that preserve the qualitative relationships in a given scenario can be "traded" for degrees of freedom of the objects in the problem. The effect is eliminating real quantifiers from the polynomial constraints without loss of generality. Given the drastic computational complexity of solving polynomial constraints, eliminating even a few variables from the underlying polynomial constraints greatly increases both runtime performance, and the range of problems that can be solved in a practical amount of time.

Moreover, spatial problems can often be decomposed into sub-problems that can be solved independently. Spatial symmetry pruning can be reapplied within each sub-problem, see (Schultz and Bhatt 2015b) Section 3.5 for further details. Thus, we are building knowledge about space and spatial properties of objects into the spatial solver at a declarative level, in a modular, extensible, systematic manner, that has a significant impact on performance.

For example, consider the equilateral triangle construction problem in D.1. Without any symmetry pruning, the solving time for the sufficiency task is rather long, approximately 40 seconds (on a MacBook Pro Intel Core i7). The relations used in the problem are incidence and distance between points and circles, which are preserved by translation, rotation, reflection, and uniform scaling. By consulting the available pruning cases for this selection of transformations (see Table 2, (Schultz and Bhatt 2015b)), we determine that the position of two points can be replaced by any real value without loss of generality; that is, we eliminate four quantified variables from the problem $(x_{p_1}, y_{p_1}, x_{p_2}, y_{p_2})$.

The performance gain is drastic: the problem now takes approximately 0.1 seconds to solve, i.e. two orders of magnitude faster. Note that, as this is a sufficiency task, the correct solution is *unsatisfiable*.

In this example we have manually employed the optimisation pruning case from (Schultz and Bhatt 2015b). One key topic of our future work is automatically applying such optimisations within ASPMT(QS).

```
:- constants
p1   :: point;
p2   :: point;
p3   :: point;
c1   :: circle;
c2   :: circle.

<- coincident(p1,p2).
<- not center(p1,c1).
<- not center(p2,c2).
<- not coincident(p1,c2).
<- not coincident(p2,c1).
<- not coincident(p3,c1).
<- not coincident(p3,c2).

<- distanceEQ(p1,p2,p1,p3) & distanceEQ(p1,p2,p2,p3)
& distanceEQ(p1,p3,p2,p3).

%% employ an optimisation pruning case from
%% (Schultz and Bhatt 2015b) by fixing the position
%% of two points without loss of generality:

p1x=0.
p1y=0.
p2x=10.
p2y=0.
```