Artificial Intelligence 238 (2016) 1–10

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

A qualitative spatial representation of string loops as holes

Pedro Cabalar[a,*], Paulo E. Santos[b,*]

[a] *Dept. Computación, University of Corunna, Spain*
[b] *AI and Automation Group (IAAA), Centro Universitário da FEI, São Paulo, Brazil*

A R T I C L E I N F O

Article history:
Received 11 December 2015
Received in revised form 9 April 2016
Accepted 10 May 2016
Available online 13 May 2016

Keywords:
Spatial representation
Problem solving
Reasoning about actions

A B S T R A C T

This research note contains an extension of a previous work by Cabalar and Santos (2011) that formalised several spatial puzzles formed by strings and holes. That approach explicitly ignored some configurations and actions that were irrelevant for the studied puzzles but are physically possible and may become crucial for other spatial reasoning problems. In particular, the previous work did not consider the formation of string loops or the situations where a holed object is partially crossed by another holed object. In this paper, we remove these limitations by treating string loops as dynamic holes that can be created or destroyed by a pair of elementary actions, respectively picking or pulling from strings. We explain how string loops can be recognised in a data structure representing the domain states and define a notation to represent *crossings through string loops*. The resulting formalism is dual in the sense that it also allows understanding any hole as a kind of (sometimes rigid) closed string loop.

1. Introduction

The design of computer programs and machines with commonsense reasoning constitutes an important long-term goal of Artificial Intelligence. In most commonsense reasoning scenarios, the spatial component of the domain plays a fundamental role. People usually reason about spatial entities and their behaviour in their daily lives without apparent effort – it is somehow an embodied (and possibly innate) feature in the human mind. As a simple example, think about all the steps for putting on a pair of trousers and a belt. While children usually learn this process without much difficulty, scenarios like this become a real challenge for computer programs as they must deal with complex geometric figures (e.g. the pants, the zipper), measure-related constraints (such as choosing the right hole in the belt, depending on your waist size) and other object constraints related to rigidness (the belt buckle) versus flexibility (the clothes and the belt).

Research on spatial commonsense reasoning comes from two main sources in the Knowledge Representation (KR) literature. On the one hand, the area of *Reasoning about Actions and Change* comprises a family of logical languages [1–5] for the formalization of an intelligent agent operating in action domains and performing common reasoning tasks such as simulation, planning, temporal explanation or diagnosis. On the other hand, *Qualitative Spatial Reasoning* (QSR) [6,7] aims at the rigorous treatment of qualitative abstractions of spatial entities that constitute the foundations of our commonsense understanding of the external world. Although the combination of QSR and temporal reasoning is not unfrequent in the literature (see for instance [8]), in general QSR approaches have traditionally overlooked a formal treatment of actions as those involved in our previous example or tackled temporal reasoning tasks such as planning, simulation or explanation.

* Corresponding authors.
E-mail addresses: cabalar@udc.es (P. Cabalar), psantos@fei.edu.br (P.E. Santos).

http://dx.doi.org/10.1016/j.artint.2016.05.001
0004-3702/© 2016 Published by Elsevier B.V.

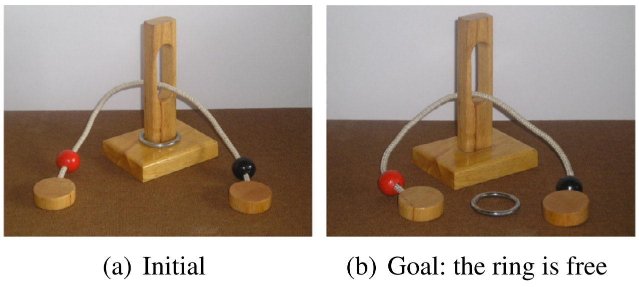


Fig. 1. A spatial puzzle: the Fisherman's Folly.

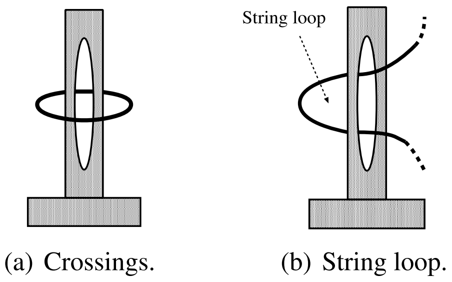


Fig. 2. String loops and hole-hole crossings.

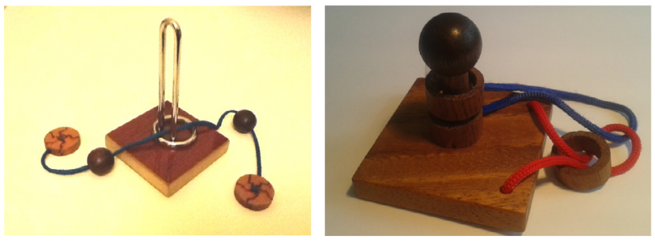
Trying to fill this gap, we have concentrated our efforts in formalizing action domains that involve flexible objects and holes, as they are very common in different scenarios like our trousers example.[1] Our methodology, applied along a series of papers [9–13], has consisted in studying spatial puzzles in a bottom-up fashion, starting from restricted cases and gradually covering new puzzles with more challenging features. Puzzles constitute a good test bed, as they offer a small number of objects requiring a minimum background knowledge about unrelated features, while they keep enough complexity to constitute a challenging problem for KR. Most of these puzzles consist in releasing a rigid ring from an entanglement of strings and other objects.

Our initial efforts were put in solving the so-called Fisherman's Folly puzzle shown in Fig. 1 using a list-based representation of string crossings. All this work eventually led to an extensive paper [11] describing a complete logical formalization plus a preliminary planner capable of solving a family of related puzzles with similar features.

In [11] some issues were left open. In particular, we did not consider states where a holed object was partially crossing another hole, as in Fig. 2(a), or the formation of string loops as in Fig. 2(b). Both situations were irrelevant for solving the family of puzzles under study but, as it can be imagined, ignoring them may easily suppose a lack of elaboration tolerance for other closely related puzzles. For instance, the variation of Fisherman's Folly shown in Fig. 3(a) is essentially the *same* puzzle with the difference that the holed post has been replaced by a long metallic arc. The latter forms a hole that, in its turn, must cross the ring hole, becoming a case of Fig. 2(a). Although this feature is not essential for solving Fisherman's Folly, there are other puzzles that cannot be solved without removing these restrictions – for instance, in [14] we studied the so-called "*easy-does-it*" puzzle (Fig. 3(b)) which cannot be solved without representing (and acting upon) string loops.

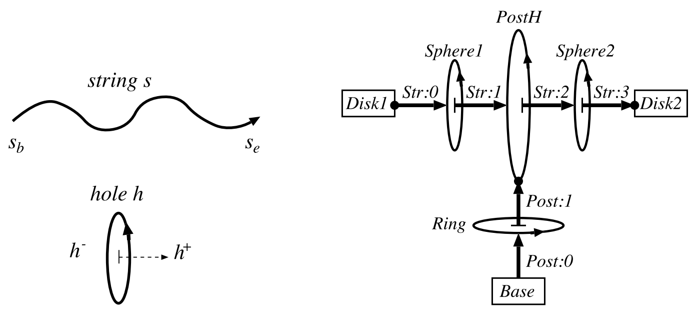
The present paper shows how the recent developments reported in [14] fill up a number of gaps left open in [11] and allow removing the above mentioned limitations by considering the formation of string loops. We describe how to recognize loops in a list of string crossings and define a notation to represent *crossings through string loops*, as they actually behave as regular holes. On top of the previous approach, we identify two basic new actions on strings that may form or destroy loops: (1) *picking* a string segment through a hole, and (2) *pulling* from a string to unwind a loop. The most difficult part of the paper corresponds to the description of the direct and indirect effects of these two actions and, particularly, to the fact that a loop may be inside some larger loop. As a result, an action on a loop may imply inheriting crossings with respect to a larger loop. We also explain how a hole can also be seen as a kind of (sometimes rigid) closed string loop, allowing the representation of problems such as the one in Fig. 2(a). The next section introduces the basis upon which this work was developed.

[1] In fact, most actions in the example involve passing objects through holes: we pass our legs through the trousers sleeves, the button through the buttonhole, the belt through loops in the trousers, the belt tip through the belt buckle, and the buckle bolt through a hole in the belt.



(a) A variation of Fisherman's Folly. (b) Easy does it

Fig. 3. Puzzles.



(a) A string and a hole. (b) Fisherman's Folly initial state

Fig. 4. Examples of diagrams with strings and holes.

2. Describing states: strings, holes and crossings

The basic ontology consists of *strings*, *holes* and *regular objects*. A *string* s is generally understood as a (possibly flexible) long object with two differentiated points called *tips* that are arbitrarily denoted as s_b and s_e to stand for the *beginning* and the *end* of s, respectively. Strings will be graphically represented as (possibly curved) line segments with an arrow head that represents a *direction* from the beginning s_b to the end s_e of the string. A *hole* represents an empty region delimited by a boundary, normally, the body of some object hosting the hole. This work exclusively focuses on "tunnel-shaped" holes, that is, those with two exits. Since we will not represent geometric shapes or measures, we can think about a hole h as a closed surface with two faces, arbitrarily denoted as h^+ and h^-, that represent the hole exits. Given a face f, we write $\overline{f}$ to represent its opposite face, i.e., $\overline{h^+} \overset{def}{=} h^-$ and $\overline{h^-} \overset{def}{=} h^+$. A regular hole h will be represented as an ellipse with an arrow head in its boundary pointing out a *spin*, using the right thumb rule to determine the positive face h^+, as in Fig. 4(a). In the diagrams, the strings crossing a hole h also help to fix its orientation: h^+ is always shown as the "visible" surface, and the string portion in h^- is hidden.

Regular objects (depicted as boxes) do not show any particular spatial feature. They act as points where string tips can be linked to. They also restrict the possible ways in which the string tips can pass through the different holes, something that is relevant for solving a given puzzle.

A system *state* will describe two kinds of basic relations among strings and other objects: *links* and *crossings*. A string tip can be *linked* to a regular object, to a holed object or to another tip.[2] We will represent a link as a thick dot. On the other hand, a string can be passing through several holes at a given situation: each time that a string crosses some hole is called a *crossing*. Note that the same string can be crossing the same hole several times and in different directions although, obviously, at different points in the string. As introduced in [9], we will use a list structure, called *chain*(s), to capture the sequence of hole crossings that each string s traverses from s_b to s_e. Each crossing of s through a hole h is represented by the exit hole face. For instance, if s crosses h from h^- to h^+, we represent the crossing as h^+ in the chain representation. In the diagrams, crossings will be represented as a small dash, perpendicular to the string direction.

As an example of all these elements, consider the schematic representation of the Fisherman's Folly puzzle in Fig. 4(b). The diagram shows two long objects (*Str* and *Post*), three regular objects (*Base*, *Disk*1 and *Disk*2) and four holed objects (*PostH*, *Sphere*1, *Sphere*2 and *Ring*). The list of crossings for the string *Str* corresponds to $chain(Str) = [Sphere1^+, PostH^+, Sphere2^+]$.

[2] For simplicity, we do not consider linking tips from different strings, but this could also be perfectly possible to be defined within the formalism presented.

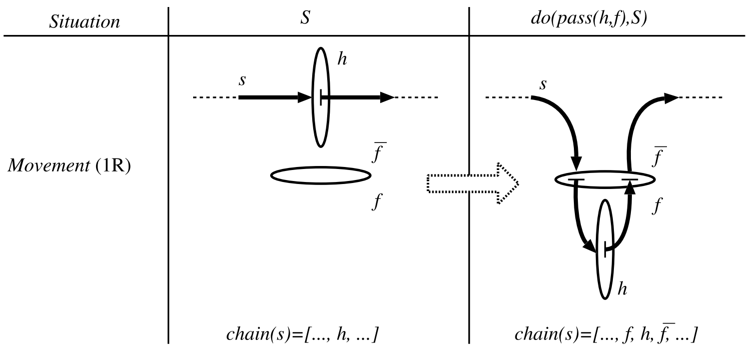


Fig. 5. Effects of passing a holed object through another hole [11].

Suppose we have a list for a string s of the form $chain(s) = [x_1, \ldots, x_n]$ containing n crossings. Then, we can consider a division of the string into $n+1$ (string) *segments*, we represent as $s{:}i$ for $i = 0, \ldots, n$ so that crossing x_j is preceded by segment $s{:}(j-1)$ and followed by segment $s{:}(j+1)$. For instance, Fig. 4(b) shows four string segments,[3] $Str{:}0$ to $Str{:}3$, for string Str.

3. Passing objects through holes

The system dynamics is described in terms of transitions between states caused by the execution of actions. In [11], we considered an elementary action $pass(o, f)$ for passing an object o through some hole h toward one of its faces $f \in \{h^+, h^-\}$. The executability of this action was limited by the specification of *constraints*. For that purpose, we defined a (static) predicate $CannotPass(o, h, s)$ meaning that object o cannot pass through hole h when the latter is being crossed by the set of strings s. For instance, some constraints in the Fisherman's Folly are that the post base cannot pass through the ring, i.e. $CannotPass(Base, Ring, \emptyset)$, that a sphere $x \in \{Sphere1, Sphere2\}$ cannot pass through the post hole, that is, $CannotPass(x, PostH, \emptyset)$ or that it cannot pass through the ring when the latter is crossed by the post $CannotPass(x, Ring, \{Post\})$.

The execution of $pass(o, f)$ could affect the string crossings in the cases where o was a string tip or a holed object. When o is a string tip, the movement is very simple: depending on the direction, it just adds or removes the last crossing in the string chain. When o was a holed object, on the contrary, the effects were more complex, especially if o was, in its turn, crossed by other strings. Fig. 5 shows the result of passing the holed object h toward a ring face f when the string s is crossing h – this was encoded as movement (1R). A second movement not displayed, (1L), performed the opposite movement undoing (1R). Note that we assumed that these movements had to be *complete*, that is, we disregarded any intermediate state where the moved ring was *partially overlapping* the crossed ring since, although physically possible, these states were cases of Fig. 2(a). A second observation is that, in the resulting state of (1R), string s has formed a new loop (as in Fig. 2(b)) because of being pulled through ring p. The situation where that loop is crossed by a new string could not be represented before, but will be part of the generalisation we introduce in this paper.

Another feature from [11] was the extension of action *pass* to sets of objects linked altogether. Thus, for instance, since Str_e and $Disk2$ are linked in the Fisherman's Folly initial state, we can execute $pass(\{Str_e, Disk2\}, PostH^-)$ meaning that both $pass(Str_e, PostH^-)$ and $pass(Disk2, PostH^-)$ are simultaneously performed. The rest of the movements are shown in Fig. 6. Note that state S_5 has reached the goal since, at this point, the ring hole $Ring$ does not occur in any list, i.e., it is not crossed by any long object. In Fig. 6 $do(A, S)$ denotes the resulting situation after performing action A on situation S.

4. String loops as holes

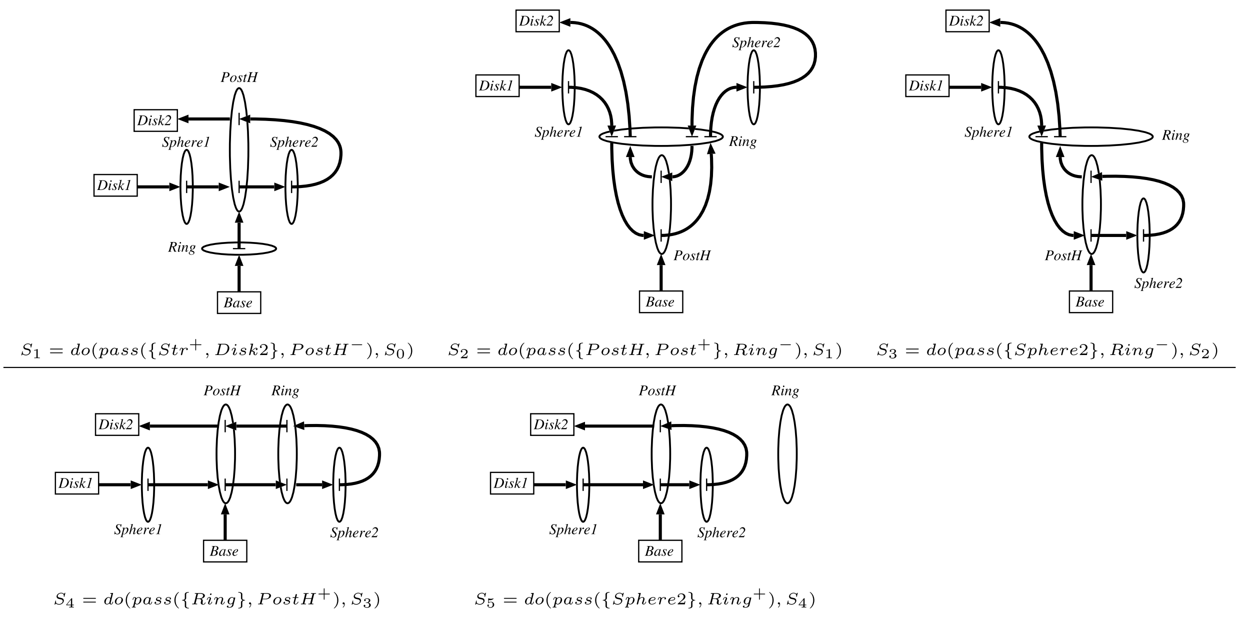
Although [11] did not consider loops as formal objects, the truth is that they can be easily detected in any structure $chain(s)$ by recognizing two (possibly non-consecutive) crossings through the same hole with the general pattern:

$$chain(s) = [\ldots, \underbrace{f, \ldots, \overline{f}}_{loop}, \ldots],$$

where $f, \overline{f}$ are the two faces of a same hole. For example, state S_1 in Fig. 6 has a loop formed by the interaction of Str and the hole $PostH$ that can be directly seen in $chain(Str)$ as:

[3] We place an arrow head in each segment to remind the general direction of the string.

$State$	$chain(Post)$	$chain(Str)$
S_0	$[Ring^+]$	$[Sphere1^+, PostH^+, Sphere2^+]$
S_1	$[Ring^+]$	$[Sphere1^+, PostH^+, Sphere2^+, PostH^-]$
S_2	$[\,]$	$[Sphere1^+, Ring^-, PostH^+, Ring^+, Sphere2^+,$ $Ring^-, PostH^-, Ring^+]$
S_3	$[\,]$	$[Sphere1^+, Ring^-, PostH^+, Sphere2^+, PostH^-, Ring^+]$
S_4	$[\,]$	$[Sphere1^+, PostH^+, Ring^-, Sphere2^+, Ring^+, PostH^-]$
S_5	$[\,]$	$[Sphere1^+, PostH^+, Sphere2^+, PostH^-]$



$S_1 = do(pass(\{Str^+, Disk2\}, PostH^-), S_0)$ $S_2 = do(pass(\{PostH, Post^+\}, Ring^-), S_1)$ $S_3 = do(pass(\{Sphere2\}, Ring^-), S_2)$

$S_4 = do(pass(\{Ring\}, PostH^+), S_3)$ $S_5 = do(pass(\{Sphere2\}, Ring^+), S_4)$

Fig. 6. A formal solution for the Fisherman's puzzle and its graphical representation [11].

$$chain(Str) = [Sphere1^+, \underbrace{PostH^+, Sphere2^+, PostH^-}_{loop}]. \tag{1}$$

An interesting observation is that loops formed in this way also constitute a new kind of "hole" on which we can apply the same actions we use for normal or *permanent* holes (passing objects through them, picking or pulling other strings, etc.) but that differ from the latter in that they are *temporary*, that is, they can be created or destroyed depending on the operations that are performed on their host strings. Several types of loop-holes can be considered depending on how they are formed. Fig. 7 shows four general types. Types 1 and 2 are loops formed by the interaction of a string and a hole, represented here as a ring. Type 1 are loops where the string passes twice through the same hole, but in opposite directions. This kind of loops can be formed by "pulling" a segment of the string through the hole, and will actually constitute the main focus of this paper. Loops of type 2 correspond to cases where the string passes twice through the same hole but in the same direction. This situation can only be achieved by a *sewing*-like sequence of actions passing the string tip through the different crossings. Type 3 constitutes the case where both string tips are linked together or linked to the same object. As we will see, this can be represented as a particular subcase of type 1. Finally, type 4 corresponds to "virtual" loops formed by a string crossing or superposing itself (the figure shows just one possibility, but more cases can be built using different strings). In these cases, the loop is more conceptual than physical, since the string crossings do not constitute real joints, but still, most practical knot handbooks actually describe and manipulate these loops as regular holes. This classification just considers loops generated by a string. Complex loops can be formed by the interaction of different objects that sometimes behave as a string such as, for instance, a chain formed by different links, a train consisting of linked wagons, etc. This issue is left for future investigations.

In order to represent loops of Type 1 and crossings through them, we introduce the following notation. Take some $chain(s) = [x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n]$ with $0 \leq i < j \leq n$ and assume that $x_j = \overline{x_i}$, that is, x_i and x_j are opposite hole faces. Then, by $l(s, x_i, [i, j])$ we denote the loop comprising the set of segments $\{(s{:}i), \ldots, (s{:}j-1)\}$. As an example, the loop in Formula (1) can be represented as $l(Str, PostH^+, [2, 4])$ and comprises the segments $Str{:}2$ and $Str{:}3$ (respectively the third and fourth segments, since we count from 0) that can be seen in state S_1 of Fig. 6.

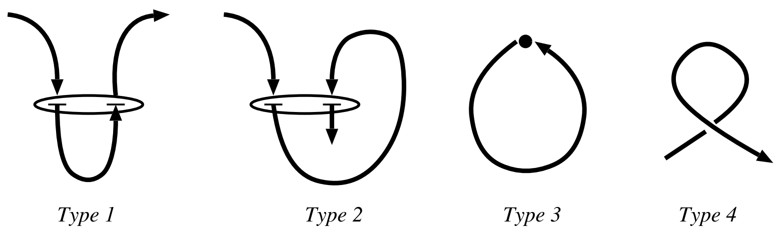


Fig. 7. Effects of passing a holed object through another hole.

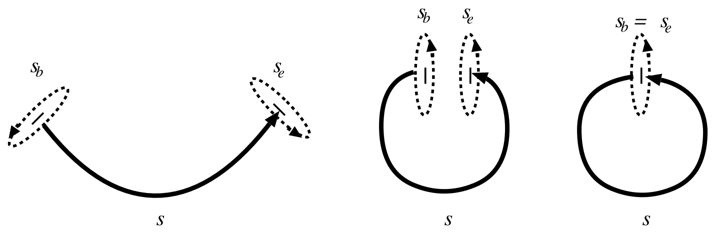


Fig. 8. When both tips are joined, the whole string forms a loop.

The previous notation can also be used to represent loops of Type 3. To this end, we consider that tips s_b and s_e in a string s actually define a pair of *virtual holes* that constitute the initial and final crossings in $chain(Str)$. We take the criterion that s_b is always crossed toward its negative face, and s_e toward its positive one. For instance, Formula (1) can be represented instead as:

$$chain(Str) = [Str_b^-, Sphere1^+, PostH^+, Sphere2^+, PostH^-, Str_e^+].$$

For coherence, we assume now that the first crossing Str_b^- has index 0 since there is no previous segment and, similarly, Str_e^+ has index $n+1$, which coincides with the number of segments in the string. A loop $l(s, x_i, [i, i+1])$ consisting of a single segment $s{:}i$ receives the name of *single loop*.

To illustrate how a loop of Type 3 is formed, consider the leftmost diagram in Fig. 8 where string s has both tips free with the corresponding $chain(s) = [s_b^-, s_e^+]$. If both tips are moved toward each other until they are linked together (or they are linked to the same object) we reach the situation in the rightmost diagram where holes s_b and s_e become *the same*, i.e., $s_b = s_e$ and we use s_b to denote both. In this case, the string has $chain(s) = [s_b^-, s_b^+]$ and ends up forming a (closed) single loop $l(s, s_b^-, [0, 1]) = \{(s{:}0)\}$. Note how the diagram for this loop is essentially identical to the one we use for a holed object. When a closed string R forming a single loop $chain(R) = [R_b^-, R_b^+]$ is persistent, i.e., it cannot be made open by any domain action, it is named as *ring* and its related loop $l(R, R_b^-, [0, 1])$ is abbreviated as the string name R by a slight abuse of notation. In the Fisherman's Folly, $Ring, PostH, Disk1$ and $Disk2$, are examples of rings.

As a more elaborated example of loop of Type 3, note how we can represent now the variation of the Fisherman's Folly in Fig. 3(a) where the post is replaced by a metallic arc. Fig. 9 shows the diagrammatic representation from which we can derive $chain(Post) = [Post_b^-, Ring^+, Ring^-, Post_b^+]$ since in this case $Post_b = Post_e$ as both tips are linked to the $Base$. As a result, we get two loops in the post: the main loop, $l(Post, Post_b^-, [0, 3]) = \{(Post{:}0), (Post{:}1), (Post{:}2)\}$, formed by all the $Post$ segments, and an inner loop $l(Post, Ring^+, [1, 2]) = \{(Post{:}1)\}$. This example illustrates another interesting feature, since $l(Post, Ring^+, [1, 2]) \subset l(Post, Post_b^-, [0, 3])$, that is, a loop may be included in another.

The inclusion of loops inside larger loops is well illustrated by state S_2 in Fig. 6. At that state, $chain(Str)$ has the form[4]:

$$[Sphere1^+, \underbrace{\underbrace{Ring^-, PostH^+, Ring^+}_{l(Str,Ring^-,[2,4])}, \overbrace{Ring^+, Sphere2^+, Ring^-}^{l(Str,Ring^+,[4,6])}, \underbrace{Ring^-, PostH^-, Ring^+}_{l(Str,Ring^-,[6,8])}}_{l(Str,Ring^-,[2,8])}]$$

forming four loops that share endpoints and, moreover, three of them are included in the larger (outermost) one $l(Str, Ring^-, [2, 8])$.

[4] This chain omits the initial and final crossings Str_b^- and Str_e^+ for brevity.

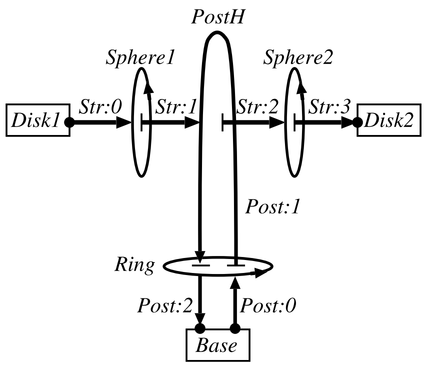


Fig. 9. Initial state for Fisherman's Folly variation in Fig. 3(a).

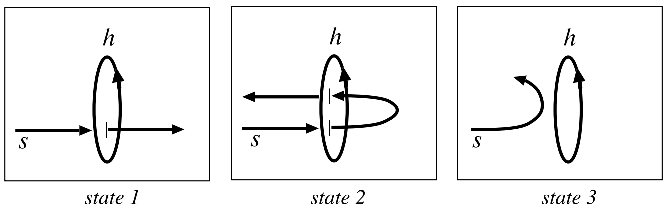


Fig. 10. Undoing passing a string tip forms an intermediate state with a loop.

5. Actions on loops

The introduction of loops in the representation of states implies a reconsideration of the set of actions from [11] that may be relevant for causing state transitions. One first obvious way to build a state containing loops is by starting with a free string s and successively passing its end tip s_e through all the involved holes in the order represented by $chain(s)$. This is what we may informally call "sewing." For this purpose, action $pass(s_e, f)$ from [11] (described above) does not require any modification: each time a new hole is crossed, it includes a new crossing in $chain(s)$. However, in [11], actions $pass(s_e, f)$ or $pass(s_b, f)$ could also remove a crossing from $chain(s)$ and this effect must be disregarded now. The effects of these actions in [11] were *symmetric* in the sense that passing a string tip toward one hole face f and then backwards to $\overline{f}$ returns the chain to its original state. For instance, imagine a string s and a hole h initially unrelated, as in Fig. 4(a). If $pass(s_e, h^+)$ is executed, we would get $chain(s) = [s_b^-, h^+, s_e^+]$ depicted as state 1 in Fig. 10. If the next movement is $pass(s_e, h^-)$ the effect of that action, according to [11], was completely "pulling" back from the string tip s_e leaving $chain(s) = [s_b^-, s_e^+]$ free again, as in state 3 in Fig. 10, i.e., the original situation in Fig. 4(a). In this paper, we consider instead that the result of passing a string tip through a hole *always creates a new crossing*. As a result, in the example of Fig. 10, executing $pass(s_e, h^-)$ on state 1 will actually lead to state 2, where we have $chain(s) = [s_b^-, h^+, h^-, s_e^+]$ forming the single loop $l(s, h^+, [1, 2])$, disregarded in [11] as an irrelevant intermediate step. Once this restriction is removed, however, we must consider a pair of new actions. For instance, to allow the transition from state 2 to state 3, *pulling* from a string segment to undo a single loop is required. Moreover, in some situations we can also require the opposite movement, that is, passing from state 3 to state 2 by just *picking* the string without actually moving its tips. For instance, it may be the case that the string tips are linked to some object that cannot pass through the hole, so we cannot create the loop in state 2 by performing $pass(s_e, h^+)$ and then $pass(s_e, h^-)$ as before, but we can still pick the string to h^+ to form the loop.

To sum up, we define two new actions, $pick(s{:}i, f)$ and $pull(s{:}i, f)$, that respectively allow picking or pulling some string segment $s{:}i$ toward a hole face f. The direct effects of these two actions are represented in Fig. 11, which shows the effect of picking from left to right and the effect of pulling in the opposite direction. As we can see, Fig. 11 is quite similar to Fig. 5 with the only difference that now the string can be picked or pulled without the necessity of having a holed object h to be passed through the target hole.

We assume that the action $pick(s{:}i, f)$ can always be executed on a string, regardless the origin and target of segment $x : i$, and that it always creates a new single loop[5] of the form $[\dots, f, \overline{f}, \dots]$ in $chain(s)$. As happens with passing a tip, the action *pick* is not symmetric: if we execute $pick(s{:}i, f)$ on the left state of Fig. 11 and then we pick again toward the opposite face, $pick(s{:}i + 1, \overline{f})$ we do not return to the original state, but we obtain a nested loop of the form $chain(s) = [s_b^-, \overline{f}, f, \overline{f}, f, s_e^+]$ instead.

[5] We are assuming an ideal string that can be arbitrarily stretched. In the real world, the number of loops may be limited by the relative sizes (of the loop and the host string, the string length and its stretchability).

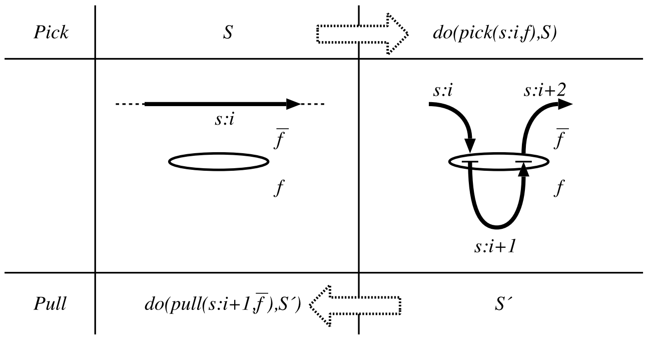


Fig. 11. Picking and pulling a loop.

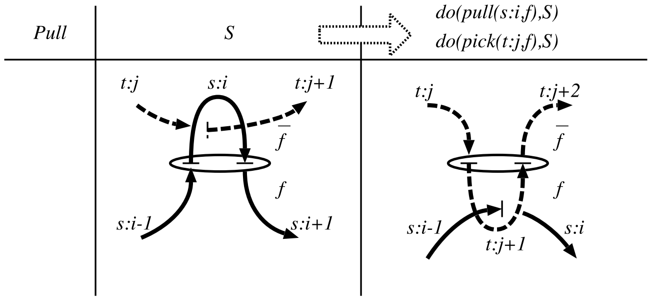


Fig. 12. Pulling loop in $s{:}i$ causes picking string t as indirect effect.

As we can see in Fig. 11, if we want to undo a single loop, we must actually use the action *pull*. The execution of $pull(s{:}i, f)$ is only possible if $chain(s) = [x_1, \ldots, x_i, x_{i+1}, \ldots, x_n]$ has a single loop at the i-th position, that is, if $x_{i+1} = f$ then $x_i = \overline{f}$. The direct effect of this action is just removing the pair of crossings x_i, x_{i+1} from the list. The main complication, however, is that *pull* may also imply two types of *indirect effects*. First, when the pulled loop was crossed by other strings, these end up picked in the same direction. As an example, Fig. 12 shows how $pull(s{:}i, f)$ picks string t (dashed) that was crossing the removed loop $s{:}i$. In the resulting state, string t ends up forming a new single loop $t{:}j+1$, which is now crossed by s. This operation takes place for any string crossing the removed loop. That is, if we had n strings crossing $s{:}i$ we would get n new single loops crossed by s. The ordering in which those crossings occur in $chain(s)$ is arbitrary, understanding the action outcome as non-deterministic.

A second kind of indirect effect associated with the action *pull* has to do with the inclusion of a single loop inside a larger one. When two loops L and L' satisfy $L \subseteq L'$ (as sets of segments), any string that crosses L is also crossing L'. In the *chain* structure, we just represent the crossing through the smallest loop L, but when the latter is unwinded by some *pull* action, the removed crossing through L must be replaced by a crossing through the next loop L' in the inclusion hierarchy.

To illustrate the second kind of indirect effect associated to *pull*, consider again Fig. 12 and imagine that strings s and t respectively represent the *Post* and *Str* at Fig. 9 and that f is the hole face $Ring^-$. The movement $pull(s{:}i, f)$ corresponds in this context to $pull(Post{:}1, Ring^-)$, i.e., pulling the post downwards or, if preferred, sliding the ring upwards the post. The initial state of $chain(Str)$ in Fig. 9 is:

$$[Str_b^-, Sphere1^+, \underline{l(Post, Ring, [1,2])^+}, Sphere2^+, Str_e^+]$$

and, according to Fig. 12, *Str* will be picked toward $Ring^-$. So, in principle, we should just replace the crossing through the pulled loop $l(Post, Ring, [1,2])^+$ by the new pair of crossings $Ring^-, Ring^+$ in the list above. However, as we saw before, $l(Post, Ring, [1,2])$ was actually part of a larger loop $l(Post, Str_b^-, [0,3])$ since both ends of the post are linked to the *Base*. As a result, the crossing through $l(Post, Ring, [1,2])^+$ must be actually replaced by $l(Post, Ring, [0,1])^+$ in the resulting state, where we have readjusted the loop interval $[0,3]$ to $[0,1]$ since crossing 1 and 2 in $chain(Post)$ are removed. In this way, the resulting state for $chain(Str)$ corresponds to:

$$[Str_b^-, Sphere1^+, \underline{Ring^-, l(Post, Ring, [0,1])^+, Ring^+}, Sphere2^+, Str_e^+].$$

To conclude this section, it is worth mentioning that the previous actions in [11], movement (1R) in Fig. 5 and its inverse (1L), that allowed passing holed objects through holes, can be seen now as macro actions consisting of sequences of elementary pickings and pullings. For instance, Fig. 13 shows how movement (1R) in Fig. 5 can be achieved by first picking a segment of ring h toward f^- and then pulling from the rest of h, which will drag string s as an indirect pick.

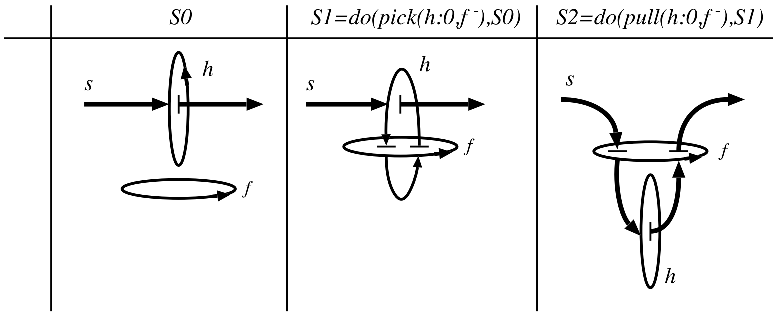


Fig. 13. Movement (1R) in Fig. 5 decomposed as a *pick* followed by a *pull*.

6. Related work

To the best of our knowledge, the current AI literature presents no other work aiming at the representation and reasoning about domains containing flexible objects and holes. However, a few related references from the standpoint of philosophy are worth mentioning. A finer ontology of holes is described in [15,16], whereas a general first-order theory of holes and spatial inclusion is created within an interplay of ontological, mereological, topological and morphological concepts. Similarly, [17] describes an investigation of compositionality, lexical and normative elements present in natural knots and suggests a research agenda for the investigation of the structure underlying the human ability to make knots. Research on the topological structure of knots (known as Knot Theory) [18], although interested in mathematical knots (whose ends are tied together), is somewhat related to the work presented in this paper in the sense that the Reidemeister moves can be implemented as actions to be applied on knots (as shown in [19]). Although rigorously well-defined, both philosophical and topological approaches to theories about flexible or immaterial objects are defined on a level of abstraction that makes them unfeasible to be applied on the automated solution of puzzles.

In [14] we presented a solution to the Easy-does-it puzzle (Fig. 3(b)) that demanded the mathematical formalisation of string loops. That paper also introduced a proof of correctness of our proposed formalism with respect to Reidemeister moves in knot theory. The present paper complements the work reported in [14] putting it in a more general context, discussing how it can be used to solve some of the issues left open in [11].

7. Concluding remarks

This paper discussed a solution to the challenging problem of formally describing a particular characteristic of flexible objects such as strings: their capacity of making loops that can be used (and reasoned about) as holes in spatial reasoning processes. This solution resolves two issues left open in our previous work (reported in [11]), namely, the representation of states where a holed object partially crosses another hole and the creation of string loops. In this paper, we have described the identification of string loops in lists of string crossings, together with the actions related to the creation and unwinding of string loops. In possession of these actions, the framework in [11] can now be used to reason about spatial puzzles where the manipulation of loops is an essential part of the solution.

Future research shall be conducted mainly in two fronts: the consideration of actions related to winding (and unwinding) knots and the deployment of these ideas in real application domains. The latter may include tasks such as *autonomous needle steering* or the actual manipulation of (and reasoning about) real world objects by a humanoid robot, such as the *Darpa Robotics Challenge*[6] that has as one of its goals the implementation of a humanoid robot with the "*ability to manipulate and use a diverse assortment of tools designed for humans*".

Acknowledgements

We wish to thank the reviewers of both this paper and [11] for pointing out some of the problems treated in the current paper. Paulo E. Santos acknowledges financial support from FAPESP grant 2012/ 04089-3, and CNPq "bolsa de produtividade em pesquisa" grant 307093/2014-0. Pedro Cabalar was partially supported by Spanish MINECO project TIN2013-42149-P. This work was developed while Paulo E. Santos was visiting the University of Corunna sponsored by the "Bolsa Ibero-Americana para Jovens Professores e Pesquisadores – Santander Universidades".

References

[1] J. McCarthy, P. Hayes, Some philosophical problems from the standpoint of artificial intelligence, Mach. Intel. J. 4 (1969) 463–512.

[6] http://www.theroboticschallenge.org/overview, accessed in Nov. 2015.

[2] R. Kowalski, M. Sergot, A logic-based calculus of events, New Gener. Comput. 4 (1986) 67–95.
[3] M. Thielscher, Introduction to the fluent calculus, Electron. Transact. Artif. Intel. 2 (3–4) (1998) 179–192.
[4] P. Doherty, J. Gustafsson, L. Karlsson, J. Kvarnström, (TAL) temporal action logics: language specification and tutorial, Electron. Transact. Artif. Intel. 2 (3–4) (1998) 273–306.
[5] M. Gelfond, V. Lifschitz, Action languages, Electron. Transact. Artif. Intel. 2 (3–4) (1998) 193–210.
[6] A.G. Cohn, J. Renz, Qualitative spatial representation and reasoning, in: F. van Hermelen, V. Lifschitz, B. Porter (Eds.), Handbook of Knowledge Representation, Elsevier, 2008, pp. 551–596.
[7] G. Ligozat, Qualitative Spatial and Temporal Reasoning, John Wiley & Sons, 2013.
[8] B. Bennett, A.G. Cohn, F. Wolter, M. Zakharyaschev, Multi-dimensional modal logic as a framework for spatio-temporal reasoning, Appl. Intell. 17 (2002) 239–251.
[9] P. Cabalar, P. Santos, Strings and holes: an exercise on spatial reasoning, in: Proc. of the 10th Ibero-American Artificial Intelligence Conference, IBERAMIA'06, in: Lecture Notes in Artificial Intelligence, vol. 4140, Springer, Ribeirão Preto, Brazil, 2006, pp. 419–429.
[10] P.E. Santos, P. Cabalar, The space within Fisherman's Folly: playing with a puzzle in mereotopology, Spat. Cogn. Comput. 8 (1–2) (2008) 47–64.
[11] P. Cabalar, P.E. Santos, Formalising the Fisherman's Folly puzzle, Artif. Intell. 175 (1) (2011) 346–377.
[12] P.E. Santos, P. Cabalar, An investigation of actions, change, space within a hole-loop dichotomy, in: Proc. of the 11th Intl. Symp. on Logical Formalizations of Commonsense Reasoning, Commonsense'13, Ayia Napa, Cyprus, 2013.
[13] P.E. Santos, P. Cabalar, An investigation of actions, change, space, in: Proc. of the 23rd International Conference on Automated Planning and Scheduling, ICAPS 2013, Rome, Italy, 2013.
[14] P.E. Santos, P. Cabalar, Framing holes within a loop hierarchy, Spat. Cogn. Comput. 16 (2016) 54–95.
[15] R. Casati, A.C. Varzi, Parts and Places, MIT Press, 1999.
[16] A.C. Varzi, Reasoning about space: the hole story, Log. Log. Philos. 4 (1996) 3–39.
[17] R. Casati, Knowledge of knots: shapes in action, in: Shapes 2.0: The Shapes of Things, in: CEUR Workshop Proceedings, vol. 1007, 2013, http://ceur-ws.org/Vol-1007.
[18] K. Reidemeister, Knot Theory, BCS Associates, 1983.
[19] J. Takamatsu, T. Morita, K. Ogawara, H. Kimura, K. Ikeuchi, Representation for knot-tying tasks, IEEE Trans. Robot. 22 (1) (2006) 65–78.