Supplemental material for: Revisiting the Evaluation of Theory of Mind through Question Answering

Matthew Le **Y-Lan Boureau** **Maximilian Nickel**
Facebook AI Research
New York, NY
{mattle,ylan,maxn}@fb.com

1 Generating the ToMi dataset

The following three pseudo-code algorithms describe how ToMi is created. Alg. 1 describes how a balanced dataset is produced, assuming access to a story generator, by generating stories with rejection until there are the same number for each of the three types considered (true belief, false belief, second order false belief). The oracle that determines the type of the story is obtained by keeping track of all characters's beliefs as the story is being generated. The next two algorithms describe the story generator. Alg. 2 shows how the core story body is created. Alg. 3 shows how random distractor statements are introduced, with both the comings and goings of a distractor agent, and irrelevant statements about containers and objects.

Algorithm 1: Produces balanced set of stories among the three target types (true belief, false belief, second order false belief), with a story generator (generateStory) and an oracle that knows the type of a story (typeStory)

```
Data: target number of stories of each type N
Result: set of stories S of size 3 * N
n[tb], n[fb], n[sofb] ← (0, 0, 0) ;
S ← {};
while min(n[tb], n[fb], n[sofb]) < N do
    newStory ← generateStory;
    ts ← typeStory(newStory);
    if n[ts] < N then
        n[ts] += 1;
        S ← S+ newStory;
    end
end
```

Algorithm 2: Core story generation

```
Data: world with locations, containers, objects, agents; oracle tracking where things are
Result: story s
s ← "";
(l, l_distr) ← two random locations with no replacement;
(o, c1, c2) ← random object in location l, container of o, container in l ≠ c1 ;
(a1, a2) ← two random agents with no replacement;
(Sally, Anne) ← shuffle (a1, a2) ;
(a1, a2) ← shuffle(a1, a2);
s += "a1 enters l";
s += "a2 enters l";
s += "o is in c1";
nLocChanges ← randomChoice(1, 2);
if nLocChanges == 1 then
    s1 ← "Anne moves o from c1 to c2";
    s2 ← "Sally exits l";
    (s1, s2) ← shuffle(s1, s2);
    s += s1;
    s += s2;
    return s                                        // Single exit
end
// More than one agent move;
(act1, act2, act3) ← shuffle(move_obj, change_loc, change_loc);
if act1 == move_obj then
    s += "Anne moves o from c1 to c2";
else
    s += "Sally exits l";
end
if act2 == move_obj then
    s += "Anne moves o from c1 to c2";
else if Sally in l then
    s += "Sally exits l";
else
    s += "Sally enters randomChoice(l, l_distr)"
end
if act3 == move_obj then
    s += "Anne moves o from c1 to c2";
else
    if flipCoin == tail then
        s += "Anne exits l";
    end
    s += "Sally enters randomChoice(l, l_distr)"
end
```

Algorithm 3: Distractor agent entries and distractor statements

Data: main story s with location l, distractor location l_{distr}, agents (a_1, a_2); set of world containers C, set of world objects O; distractor agent a_3

Result: updated story s

$nDistractorEntrances \leftarrow$ randomChoice$(0, 1, 2)$;
$insertionIndices \leftarrow$ random choice of $nDistractorEntrances$ from 1 to length(s);
if $nDistractorEntrances > 0$ **then**
  $s_1 \leftarrow$ "a_3 enters randomChoice(l, l_{distr})";
  $s \leftarrow$ insert s_1 in s at min index of $insertionIndices$;
end
if $nDistractorEntrances > 1$ **then**
  $s_2 \leftarrow$ "a_3 exits a_3's current location";
  $s \leftarrow$ insert s_2 in s at max index of $insertionIndices$;
end
$nDistractorStatements \leftarrow$ randomChoice$(0, 1, 2)$;
$insertionIndices \leftarrow$ random choice of $nDistractorStatements$ from 1 to length(s);
for $i \leftarrow 1$ **to** $nDistractorStatements$ **do**
  $a \leftarrow$ randomChoice(a_1, a_2, a_3);
  $thing \leftarrow$ randomChoice$(C \cup O)$;
  $s_1 \leftarrow$ "a $(likes|dislikes|loves|hates)$ the $thing$";
  $s \leftarrow$ insert s in s at i-th index of $insertionIndices$;
end