人工知能
第五回講義レポート　課題番号１３

03-150430 ケイ　ウカン

2015/12/07

ユーザーが入力した線画に対して、可能なすべての解釈を出して、解釈が一意である場合はそのまま表示され、多数ある場合はユーザーがキーボードで左右方向キーでそれらの間で切り替えられるようにしている、GUI プログラムを作成しました。使用言語は Javascript で、GUI の作成にあたっては d3 と呼ばれる、情報可視化ライブラリを使っています。zip ファイルを展開したら、lib というフォルダに d3.js が入っているはずです。フォルダ構造はこのままに保たないとプログラムが正常に動作しない可能性があります。

このプログラムにおける、ノードタイプの呼び方とラベリング辞書内容は基本筑波大学の平賀先生の講義ウェブページ ($http://www.slis.tsukuba.ac.jp/\ hiraga.yuzuru.gf/AI/label.shtml$) に基づいています。

プログラムの使用方法について簡単に説明します。$line_drawing.html$ をブラウザで開きます。Firefox については正常な動作が確認できていますが、他のブラウザでは検証を行っていません。

開くと、図 1 のように表示されるはずです。

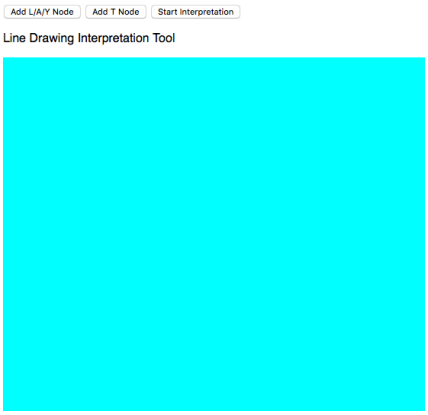

図 1　始めの画面

ノードを追加するときは、ノードタイプによって操作が異なります。T 型以外のノードを追加するときは左上の”Add L/A/Y Node”ボタンをクリックしてから、画面の青い部分に一回クリックします。するとノードが一個表示されます。T 型ノードを追加するには”Add T Node”ボタンを押して、さらにすでに引かれている線の上に一回クリックします。するとノードが追加されますが、場合によって見えにくいことがあります。マウスオーバーすると大きくなるようにしていますので、マウスで探ると見つかるはずです。ボタンが押されていない状態で、二個のノードを前後にクリックすると、その間で線が引かれます。svg 図形に対するクリックイベントはその下にあるキャンバスにも拾われるのでこの少し回りくどい仕様にせざるをえませんでした。線画を入力するとき、この順番を間違えると、プログラムが正常に動作しない場合があります。

線画を入力し終えたら、”Start Interpretation”のボタンを押すと、解釈プログラムが実行されます。

ソースコードについて説明します。

まず、ノード追加の仕組みについて説明します。createNode() と createTNode() の二つの関数はこの役割を担っています。ノードが追加される順に番号が割り当てられます。L/A/Y タイプのノードが追加された場合、その位置座標が nodes と呼ばれる配列のそのノードの番号に対応している要素であるオブジェクトの中に保存され、そのノードと他のノードの間に線が引かれたときはノード情報オブジェクトの中の connected_nodes と呼ばれる配列に接続先のノードの番号が保存されます。線を表す svg 図形には始点と終点の二つのノードの情報が入っている id がつけられます。例えば、ノード i とノード j の間に引かれた線は始点が i で終点が j である場合 i_j、逆の場合 j_i が id としてつけられることになります。さらに、そのリンクは links と呼ばれるオブジェクトにキーとして追加され、リンクタイプ（後述）を代表する値は 0 に初期化されます。一方で、T 型ノードが追加される場合はやや複雑です。クリックされた線の id からその両端のノードの情報を獲得し、互いの接続関係を解除します。すなわち、それぞれのノードの connected_nodes から相手が削除され、links からもそのリンクが削除されます。その後、新しく追加された T 型ノードの connected_nodes に両方のノードが追加され、links がそれと共に更新され、また後のラベリングの便宜上もとの直線に重なる形で新しく線が二本（二つの端点に対して一本ずつ）引かれます。

このように、線画の入力が終わった時点ですべてのノードに関してその座標と接続されているノードが nodes 配列の一つの要素として保存されていて、すべてのリンクが links オブジェクトの中に保存されています。

”Start Interpretation”がクリックされると、interpret() 関数が実行される。この関数はさらに以下のような機能を実行します。まず、各ノードが A、T、Y、L のどのタイプに属されるかを判断する。T ノードは追加時にすでにその情報を保存してあって、L 型ノードは接続されているノードの数が二個という点で特徴的なのですぐに判断できます。一方で、Y 型と A 型の判断については三つのリンクによってなされる三つの角の間で、どれか二つの角度の和が三つ目に一致するかどうかを見て、一致するものがあれば、A 型だとわかり、そうでない場合は Y 型だと判断できます、ただし、浮動小数点数である以上、完全に一致することはないので、差が 0.001 ラジアン以下の場合は等しいと判断する。

すべてのノードのタイプが判明されたら、今度は reorder_nodes() 関数によって各ノードの connected_nodes 配列の中に保存されているノードの順番を、平賀先生の講義ページに記載されている辞書に一致するようにします。Y 型に関しては辞書にあるものが完全対称であるか、または回転によって互いに一致するようにすることができるため、特にこのプロセスは不要だが、他のノードタイプに関しては二次元平面上での回転操作などを通して辞書に載ってあるラベリングパタンの線の配置と同じようにして、ノードの保存順番が違う場合はそれらのインデックスを交換する操作を行います。

このプロセスが終われば、解を実際に求める関数 find_solutions() に入ります。この関数の内部でさらにまず一つ目の解を求めることを試みる find_first_solution() が呼び出され、一つも解が見つからない場合はその旨を伝えるメッセージが表示され、そうでない場合は次の解を求める関数 find_next_solution() が繰り返して呼び出され、解が見つからなくなると終了とする。

find_first_solution() は nodes 配列の先頭から見ていき、既存の制約、すなわちすでにされたラベリングのもとで、ラベリングができるかを見て、できなければバックトラックを実行する。ラベリングできるかどうかの判断について少し詳しく説明する。線にはそれぞれ 1 、 2 、 3 、または 4 のどれかのラベルがつけられていて、 1 は始点から終点（これらは線の id を決めるときに決めた順番に一致している）に向かう矢印、 2 は終点から始点に向かう矢印を表していて、 3 はプラス（凸）、 4 はマイナス（凹）を表してます。それぞれのノードでラベリングの実行可能性について判断するとき、まずは 1 か 2 がつけられている線についてその方向（自分に向かっているのか、または自分から接続先に向かうか）をチェックします。この手順を経て初めて、 1 と 2 の区別ができて、矛盾が生じているかどうかがわかります。一方で、 3 と 4 の判断は比較的に簡単です。まとめると、それぞれのノードを通るリンクがまだラベリングされていない場合、またはラベリングされているがその制約条件を満たすような辞書エントリが存在する場合、ラベリングが可能であると判断する。これを実現したのが update_links() 関数である。一つ目の引数が i である場合、i 番目のノードについてリンクの更新ができるかどうかをチェックして、できる場合はその更新を実行して 1 を返し、できない場合は更新せず 0 を返す仕様となっています。二つ目の引数が 0 のとき、制約を考慮する必要があるのに対して、1 のとき制約を考慮しなくて済むことを表しています。これはバックトラッキングのとき 0 番目までたどったとき次の解を探さないといけないのに「制約」のせいで更新できないことを防ぐためです。

一つのノードでリンクが更新されたら、その時点でのリンク情報は「履歴」として progress_storage 配列に store_progress の呼び出しによって保存され、バックトラック関数から参照できるようにしています。i 番目のノード更新後の状態に戻す操作は recover_links(i) 関数によって実現されます。

バックトラッキングを担当している関数は backtrack(i) であり、i 番目のノードで解が見つからず、その前のノードラベルを見直す操作をします。前のノードラベルを、i 番目のノードで解が見つかるまで増やし、それが辞書上そのノードタイプで可能な最大な数字となったら再帰的にバックトラックし、解を見つける試みをする仕様です。

解が見つかったら、この解を表すリンクのラベルが results 配列の一つの要素として、オブジェクトの形で保存されます。解が見つからなくなった場合、解の数が表示され、さらに方向キーを押すことで解の間で表示を切り替えることができます。

次に、幾つかの実行例を示します。

まず、立方体を表す線画について実行してみたら、 4 つの結果が現れますが、そのうち三つは回転によって重なるものですので、ユニークな解は二つで、それらを図 2 と図 3 に示します。

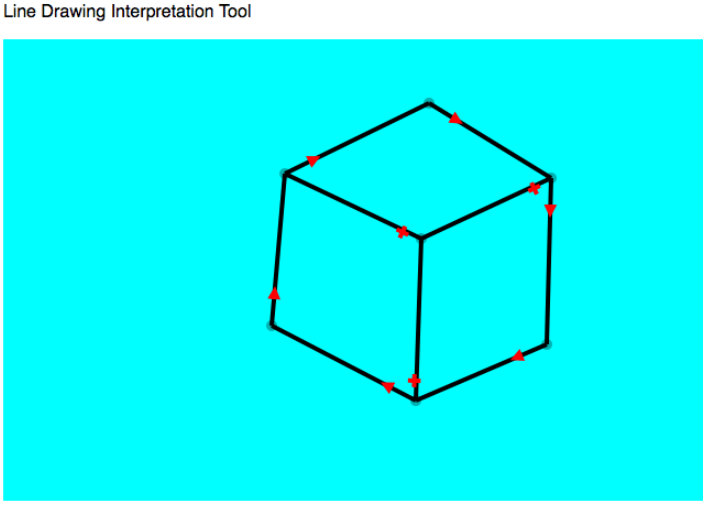

図 2　立方体解釈例 1

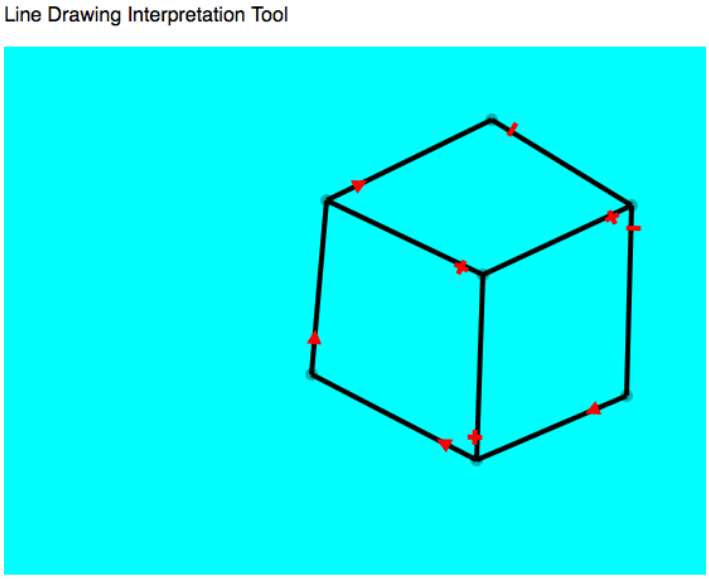

図３　立方体解釈例２

これらは立方体が壁に接しているか、または浮いているかという二つの状態に対応しています。

また、教科書図 3.24 で実行してみたら、図４に示される結果を代表とし、７つの異なる結果が出てきます。そのすべてを示すことを控えます。

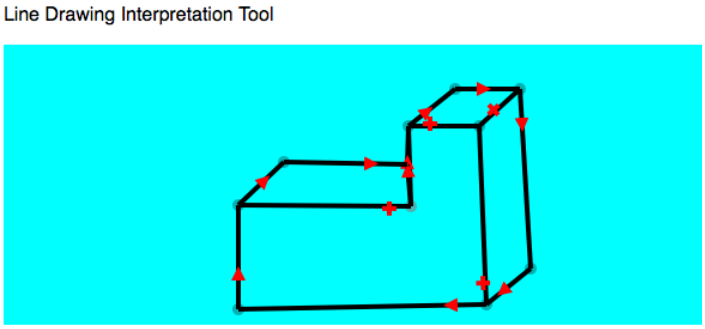

図４　教科書図 3.24 による実行結果

さらに、不可能立体二つについて実行してみました。まずは図５に示される図形で、結果が存在しないという表示をプログラムから返されている様子は図６で確認できます。

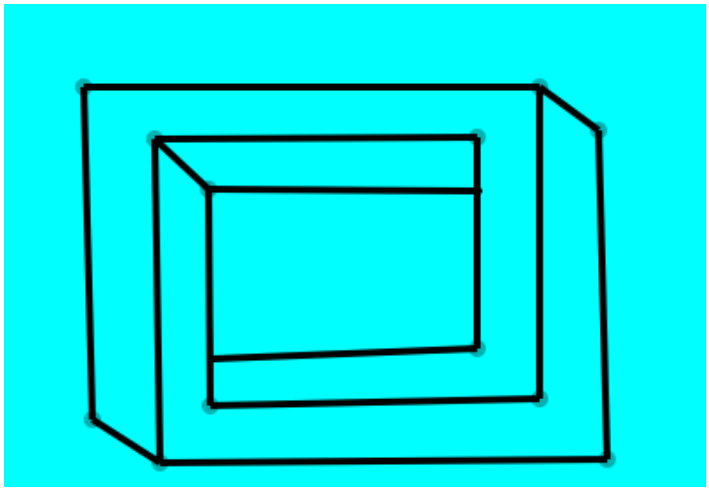

図５　不可能立体１

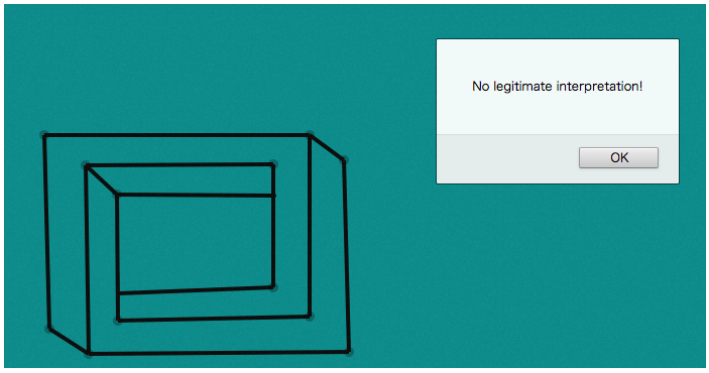

図 6　不可能立体 1 の実行結果

また、もう一つの不可能立体として、教科書図 3.25 でも実験しました。その結果を図 7 に示します。

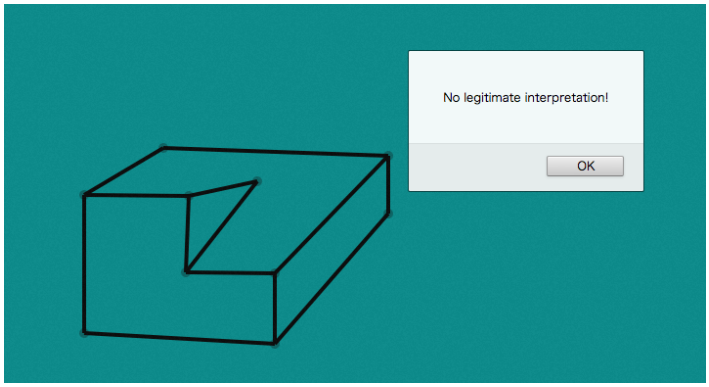

図 7　不可能立体 2（教科書図 3.25）の実行結果

一方で、ネッカーキューブは仕様上、４つのノードと接続する T 型ノードを作る必要性が生じ、一つのノードは最高 3 つのノードにつながっているという前提に反するため、実験していないです。予想としては最低二種類の解釈ができるため、前部と後部の区別がつなかくなるのではないかと思います。