Client-side Transitive Closure Operation

CS511 Term Project, Spring 2011
Luis Carrasco and Gordon Towne
http://oop.lcarrasco.com/ (project description website)
http://oop.lcarrasco.com/project/ (project implementation)
https://github.com/luiscarrascob/CS511---JS-Closure (repository)
May 2011

Overview

In this project, a web utility was developed that allows for the transitive closure of a number of logical formulas to be computed over a given universe. The utility supplies the user with a text-based interface with which to input (1) the atoms that make up the universe, (2) initial assumed relationships between those atoms and (3) conditional formulas that assert additional relationships over the universe of atoms. The text input is used to generate a hypergraph data structure, where each atom is a labeled node, and the relationships between those nodes are represented by labeled hyperedges. The hypergraph is created containing all of the initial nodes and hyperedges specified by the user. This hypergraph supports a closure operation, where the additional hyperedges are added over the existing nodes that satisfy the formulas specified by the user. Using this scheme, it is possible to verify formulas of propositional logic containing the operators for implication, conjunction and disjunction.

System Specification

Syntax

The text interface provided to the user employs the following syntax:

Introduction of atoms

intro A,B,C.

This statement introduces atoms A,B, and C. Nodes with corresponding labels are added to the hypergraph.

Assumption of relationships

assume "P"(A,B) "P"(B,C) "Q"(A,B)

This statement introduces relationships P and Q over the already introduced atoms A, B, and C.

Conditional formulas

for all x,y,z. "P"(x,y) and "P"(y,z) implies "Q"(x,z) and "P"(x,z)

This statement asserts a conditional property over the relationships P and Q on the nodes introduced in the hypergraph. It expresses that for all assignments of existing nodes to the variables x, y, and z, if the assigned nodes satisfy the properties P(x,y) and P(y,z) with respect to the existing hyperedges, additional hyperedges should be added that encode the relationships Q(x,z) and P(x,z)

In this syntax, all introductions must be made first, followed by all assumptions, and then all conditional formulas.

System Components

User Interface and Input Parsing

Before the user interface is initialized the for the first time, a color array is loaded into memory and shuffled that will be later used to assign colors to edges.

As the user types, the parse method is called every time that a "keyUp" event is occurs. The parser passes linearly over the input text and produces three HTML strings **(1)** to the first formula's box, **(2)** a box for the variable, and **(3)** the second formulas box. These three strings are constructed simultaneously. The running time of this algorithm is $O(n)$ where n is the number of characters in the input string. An overview of the construction of these three strings follows.

(1) The algorithm works by looking for keywords (*intro*, *assume*, *for all*, *and*, *or*, *implies*) and delimiters (*""*, *( )*). When a keyword or delimiter is found, it takes appropriate action. Specifically,

- when a keyword is found *""* and *""* tags are added around it with the relevant class name
- when an opening quotation mark or or parenthesis is found it adds a closing tag with the relevant class name
- when a *"for all", "assume", or "var"* keyword is found, it wraps the entire line in a *div* tag

(2) To construct this string only the lines starting with the keyword "*intro"* are considered. When the keyword is encountered all the content between it and the following period is taken and then explode this string by the "," character and add tags around each substring.

(3) The procedure to build this string is outlined as follows

- When a for all, or assume keywords, it wraps the whole line in a div tag
- If it is *"for all"*: get all the comma delimited strings until the next period and wrap all those elements inside *""* and *""* tags with the class of "a-node". Set the style of this to be invisible.
- When an opening quotation is found it adds an opening *""* tag with the relevant class name
- if it's an opening quotation then it is the start of a new edge add a "*"*
- Check the color array to see if an edge of this type has been added before, if yes then retrieve that registered color, if not then get a new color
- If it is an opening parenthesis then it is the start of a new set of nodes then add a *""*
- Skip to the closing parenthesis, and for the "," delimited strings in between add them with a *""* and *""* around it
- Close the "*"*
- Continue
- When an *"and"*, *"or"*, *"implies"* keyword is found, then wrap it with the appropriate *""* and *"/"* tags.

For example, consider the following translation of a conditional formula into HTML.

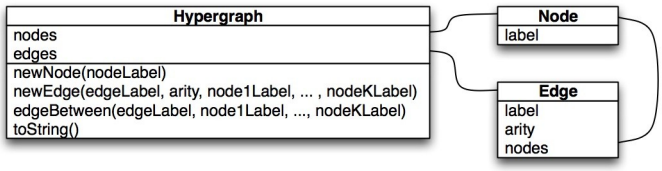
for all i,j,k,l,m,n. "P"(i,j,k) and "P"(l,m,n) implies "Q"(i,l)

HTML Representaion

```
<div class="wff-a assert">
	<span class="edge ">
		<a>P</a>
		<span class="nodes" style="background-color:#75B4FF;">
			<span class="node">i</span>
			<span class="node">j</span>
			<span class="node">k</span>
		</span>
	</span>
	<span class="and">a</span>
	<span class="edge ">
		<a>P</a>
		<span class="nodes" style="background-color:#75B4FF;">
			<span class="node">l</span>
			<span class="node">m</span>
			<span class="node">n</span>
		</span>
	</span>
	<div class="arrow-right"></div>
	<span class="edge span-implies">
		<a>Q</a>
		<span class="nodes" style="background-color:#D9C400;">
			<span class="node">i</span>
			<span class="node">l</span>
		</span>
	</span>
</div>
```

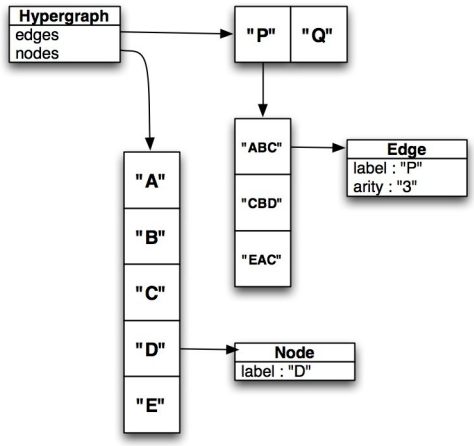
Hypergraph

The hypergraph data structure is represented at top level by Hypergraph Javascript object, which contains associative two arrays for Nodes and Edges respectively. The associative array for nodes is keyed on the label for each node. Thus, operations on the Hypergraph involving the insertion, or query for existence of nodes can be accomplished in *O(1)* time.



Since there are likely multiple edges in the graph with the same label, a hyperedge is uniquely specified by its label, and the ordered labels of the nodes that it adjoins. The Hypergraph data structure's "edges" property references an associative array keyed on edge labels, which stores associative arrays of edges keyed on a collection of the ordered labels of its adjoining edges. For example, the instance of the Hypergraph illustrated in the following diagram encodes a hypergraph over the nodes A, B, C, D, and

E. The graph contains two classes of edges, those labeled "P", and those labeled "Q". Of these, edges three instances of edges labeled "P" exist: connecting nodes A,B,C; C,B,D; and E,A,C respectively. In this structure, edges between a set of existing nodes can be inserted and looked up in *O(a)* time, where *a* is the arity of the hyperedge.



The Hypergraph class contains the following methods:

- ***newNode(nodeLabel)*** - takes a node label and adds a correspondingly labeled node to the graph. This method returns true if the node is successfully added, and false otherwise. In particular, this method will return false if there is already a node in the graph with the specified label.
- ***newEdge(edgeLabel, arity, node1Label, ... , nodeKLabel)*** - takes a hyperedge label, the arity of the hyperedge, and the labels of the set of nodes to adjoin, and adds a corresponding edge to the graph if possible. This method will return 0 on success, -1 if an inconsistency was found with the arity argument, or -2 if one of the nodeLabel arguments.
- ***edgeBetween(edgeLabel, node1Label, ... , nodeKLabel)*** - takes an edge label and a set of node labels and returns true iff there exists an edge in the hypergraph between the nodes in the order specified.

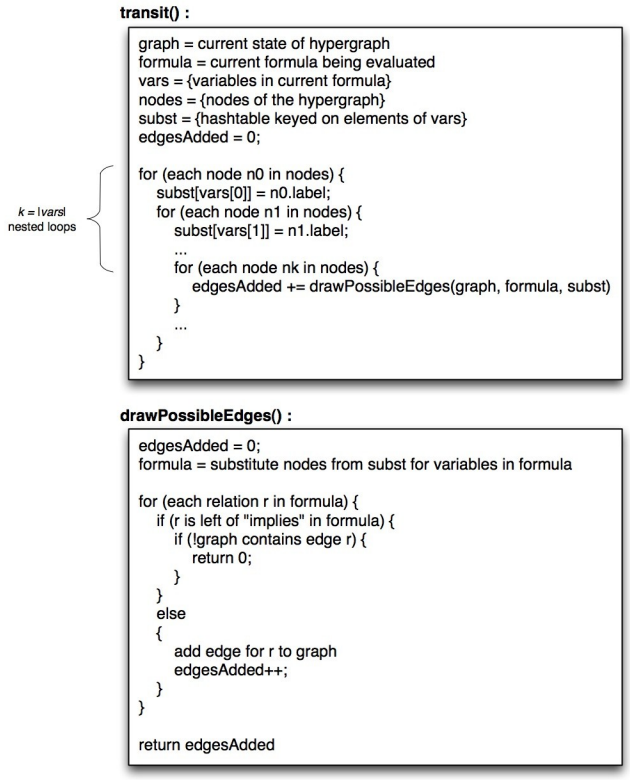
Closure Operation

A closure operation was also implemented over the Hypergraph data structure. Specifically, an algorithm was written that takes a set of conditional formulas in the syntax specified above, and for each formula and valid substitution mapping nodes to variables tests whether each relationships to the left of the "implies" holds in the hypergraph. If all of the relationships hold for a given formula and substitution, then edges are added to the graph such that the relationships to the right of "implies" are satisfied with respect to that substitution.

The closure operation is implemented by repeatedly performing a transit operation on the hypergraph that adds as many nodes as possible relative to the current state of hypergraph. The process of adding hyperedges to the graph means that potentially more relationships are satisfied after each call to the transit operation than before. Thus, in order to perform closure, the transit operation must be performed

repeatedly until no more hyperedges can be added. In this implementation, the closure algorithm terminates when a call to transit results in no more edges being added to the graph.

Psuedocode for the transit algorithm is given in the following diagram:



Application to Propositional Logic

Representation of Logical Formulas

A potential application of this utility is the verification of formulas of propositional logic. Using the syntax provided, hypergraphs can be created that express a formula of propositional logic. Specifically, hypergraphs can be created that represent formulas containing the operations of implication, conjunction and disjunction.

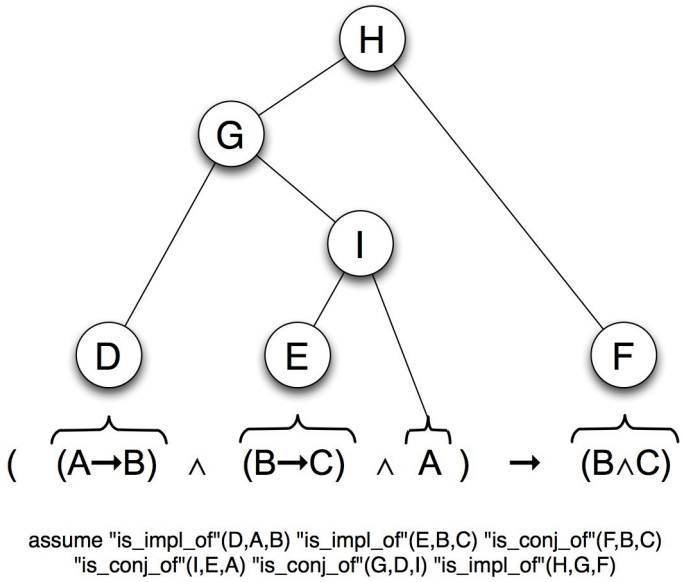
The following is a simple (potentially automatable) heuristic for translating general formulas of this form into the syntax used by this utility. Given a general formula of propositional logic in this form, fully parenthesize the formula according to rules for operator binding. Then, for each pair of parentheses, apply a label that is fresh with respect to the atoms of the formula. Each of these labels will be added to the hypergraph as a node. Next, express the relationship of the subformula represented by this node to the other subformulas using the following three relations.

- ***"is_conj_of"(X,A,B)*** - node X represents a subformula that is the conjunction of the

subformulas represented by nodes A and B.

- ***"is_disj_of"(X,A,B)*** - node X represents a subformula that is the disjunction of the subformulas represented by nodes A and B.
- "***is_impl_of"(X,A,B)*** - node X represents the subformula "A implies B"

This process is demonstrated in the following example.



Here, the initial formula over atoms *A, B* and *C,* are augmented with additional nodes *D,E,F,G,H,* and *I* to represent the component subformulas. The assumption statement corresponding to the structure resulting from this parse tree is given in the diagram.

Representation of Proof Rules of Propositional Logic

Once a logical formula is expressed as an instance of a hypergraph, it is possible to represent the proof rules of propositional logic as conditional formulas over this hypergraph. Specifically, a number of proof rules have been established that allow allow specific valuations of *"true"* and *"false"* to be applied to the nodes of the hypergraph, given an initial valuation of a sufficient subset of the nodes.

For example, the proof rule *modus tolens* can be expressed in this syntax in the following way

for all a,b,x. "is_impl_of"(x,a,b) and "true"(a) and "true"(x) implies "true"(b)

This states, that for all subformulas of the original formula. If subformula *x* represents the implication of subformulas *a* and *b*, and both the premise and implication are true, then the conclusion is also true. Correspondingly, the proof rule *modus ponens* can be expressed as follows

for all a,b,x. "is_impl_of"(x,a,b) and "false"(a) and "true"(x) implies "false"(b)

Given an adequate set of such proof rules and a consistent valuation of initial *"true"* and *"false"* relations to nodes in the hypergraph, computing the closure algorithm will result in a valuation of "true" or "false" to every node in the graph. The valuation of the node representing the top level of the parse tree (*H* in the example diagramed above) can then be checked to evaluate the overall formula.

A set of the proof rules established for implication, conjunction and disjunction are provided in the Appendix.

Performance

The computational cost of the algorithms developed are dominated by the cost of performing the closure operation on the graph. To evaluate the computational complexity of the closure algorithm, let us first examine the complexity of the transit algorithm.

In each call to transit, for each formula, the set of all valid substitutions are computed over the variables specified in that formula. Then, with respect to a given substitution, every relationship specified in the formula is evaluated at most once. As stated above, evaluating a relationship in this sense corresponds to querying for the existence of a hyperedge in the hypergraph. Thus, a single call to transit will result in $O(|N|^k fp)$ complexity, where

- N is the set of nodes in the hypergraph
- k is an upper bound on the number of variables in each formula
- f is the number of formulas asserted
- p is an upper bound on the number of relationships specified in each formula.

An evaluation of the number of times transit must be called to perform closure in the general case is more complex. In the worst case, the closure algorithm will run until the hypergraph is fully connected, which would result in an intractable computation for all but the smallest graphs. However, in many practical instances the algorithm terminates within reasonable bounds.

Consider, for example, the evaluation of formulas of propositional logic as described above. In this instance, the graph contains a single node for each parenthesized subformula of the original formula. That is, the graph contains as many nodes as does the parse tree of the formula. Note that the arity of each of the conditional formulas *"is_conj_of"()*, *"is_disj_of"*, and *"is_impl_of"* is exactly three. Thus k in the above statement of complexity is always equal to three. Likewise, f and p are both constants, as they are determined by the finite set of proof rules for propositional logic. Therefore, each call to transit in the context of propositional logic evaluation is $O(|N|^3)$

Performing closure in the context of propositional logic represents assigning a true or false valuation to each of the subformula nodes. On each successive call to transit, at least one edge is added. If this were not the case, the algorithm would terminate. To perform closure, each of the $|N|$ nodes receives a single valuation, so the transit operation must be performed at most $|N|$ times. The overall time complexity of the closure algorithm is $O(|N|^4)$

Resources

Several libraries were used for certain components of the user interface for this utility. Specifically,

- jQuery - used for UI functionality and text input parsing
- Yahoo YUI - used for UI and cross browser uniformity
- InfoVis - used to draw a graphical representation of the hypergraph <http://thejit.org/>

Appendix

Full set of proof rules for implication, conjunction, and disjunction

for all a,b,x. "is_impl_of"(x,a,b) and "true"(a) and "true"(x) implies "true"(b)
for all a,b,x. "is_impl_of"(x,a,b) and "false"(a) and "true"(x) implies "false"(b)
for all a,b,x. "is_impl_of"(x,a,b) and "true"(b) implies "true"(x)
for all a,b,x. "is_impl_of"(x,a,b) and "true"(a) and "false"(b) implies "false"(x)
for all a,b,x. "is_impl_of"(x,a,b) and "true"(a) and "false"(x) implies "false"(b)
for all a,b,x. "is_impl_of"(x,a,b) and "false"(a) and "false"(b) implies "true"(x)
for all a,b,x. "is_impl_of"(x,a,b) and "false"(a) and "true"(b) implies "true"(x)
for all a,b,x. "is_conj_of"(x,a,b) and "true"(a) and "true"(b) implies "true"(x)
for all a,b,x. "is_conj_of"(x,a,b) and "true"(a) and "true"(x) implies "true"(b)
for all a,b,x. "is_conj_of"(x,a,b) and "true"(b) and "true"(x) implies "true"(a)
for all a,b,x. "is_conj_of"(x,a,b) and "false"(a) implies "false"(x)
for all a,b,x. "is_conj_of"(x,a,b) and "false"(b) implies "false"(x)
for all a,b,x. "is_conj_of"(x,a,b) and "true"(a) and "false"(x) implies "false"(b)
for all a,b,x. "is_conj_of"(x,a,b) and "true"(b) and "false"(x) implies "false"(a)
for all a,b,x. "is_disj_of"(x,a,b) and "true"(a) implies "true"(x)
for all a,b,x. "is_disj_of"(x,a,b) and "true"(b) implies "true"(x)
for all a,b,x. "is_disj_of"(x,a,b) and "false"(a) and "false"(b) implies "false"(x)
for all a,b,x. "is_disj_of"(x,a,b) and "false"(a) and "true"(x) implies "true"(b)
for all a,b,x. "is_disj_of"(x,a,b) and "false"(b) and "true"(x) implies "true"(a)