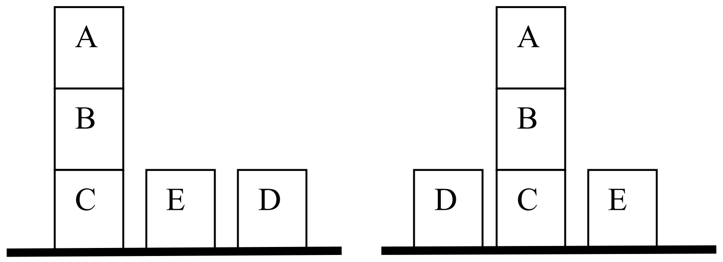
Predicate-Squish (P-Squish) and Block-Stacking: Rough Preliminary Notes

Nils Nilsson

4/3/02

George John proposed an algorithm, SQUISH, for learning teleo-reactive (T-R) programs from teacher-guided action sequences (John, 1994). John's formulation of SQUISH, and subsequent experiments with it (Nilsson, 2000), assumed that the states traversed were described by vectors of sensory inputs. Here, we consider modifications to the algorithm to allow state descriptions consisting of ground atomic formulas in relational calculus. That is, the learning agent represents its world state by a collection, $\mathcal{P}$, of ground atoms. We use examples from the blocks world to explain and illustrate the modified algorithm.

Consider a world consisting of *n* blocks that can be arranged in any configuration such that some blocks are on the tops of others and some might be directly on a table. The predicate forms sensed by an agent, which are used to describe its current state are instances of `On(x,y)`, where `x` can be any block and `y` can be some other block or the table. The blocks are denoted by letters, `A`, `B`, `C`, `D`, . . ., and the table is denoted by `Ta`. So, for example, the configurations shown on the left below would be described by the atoms shown on the right:



```
On(A,B)
On(B,C)
On(C,Ta)
On(D,Ta)
On(E,Ta)
```

(We don't distinguish configurations differing only by horizontal placement of blocks.)

Our blocks world agent has actions described by the action schema `move(x,y)`, which moves a block denoted by `x`, to the place denoted by `y`, where `y` can either be another block or the table. In order to apply `move(x,y)`, the block denoted by `x` cannot have any other block on top of it, and, if `y` denotes a block, it cannot have another block on top of it either. A move action has its intended effect on a block configuration if its conditions are satisfied; otherwise the action has no effect.

The first phase of SQUISH proceeds exactly as John originally described. A teacher guides the agent toward a goal from various initial configurations, and the learning agent keeps track of the states traversed and actions taken in each sequence. The second phase involves combining ("squishing") nodes and then generalizing them. In the case in which there are just two blocks, `A` and `B`, and a goal of having `A` on top of `B`, there are only two non-goal configurations and one goal configuration. The only two possible different sequences of state descriptions that the teacher could use are:

$$\begin{tikzcd}
\boxed{\begin{array}{l}\texttt{On(B,A)}\\ \texttt{On(A,Ta)}\end{array}} \arrow[r, "\texttt{move(B,Ta)}"] & \boxed{\begin{array}{l}\texttt{On(B,Ta)}\\ \texttt{On(A,Ta)}\end{array}} \arrow[r, "\texttt{move(A,B)}"] & \boxed{\begin{array}{l}\texttt{On(A,B)}\\ \texttt{On(B,Ta)}\end{array}} \\
 & \boxed{\begin{array}{l}\texttt{On(B,Ta)}\\ \texttt{On(A,Ta)}\end{array}} \arrow[ur, "\texttt{move(A,B)}"'] &
\end{tikzcd}$$

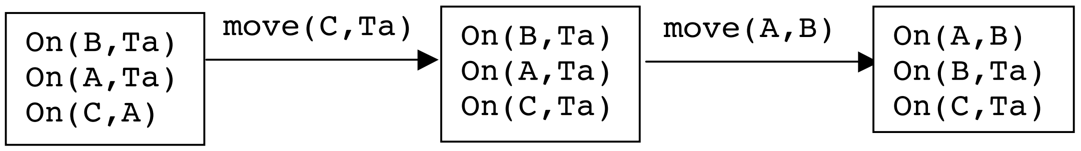
The `SQUISH` rule for combining two sibling state descriptions leading to a parent through the same action specifies that the two vector-valued descriptions be replaced by their union, which can later be generalized to a region (see Nilsson, 2000). Applying this same idea to states described by logical formulas would have us replace the two sibling state descriptions by their disjunction, which could later be generalized. Implementing this combination rule in our example yields the following T-R tree:

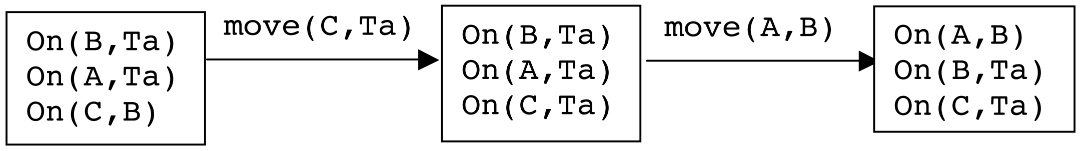
$$\begin{tikzcd}
\boxed{\begin{array}{l}\texttt{On(B,A)}\\ \texttt{On(A,Ta)}\end{array}} \arrow[r, "\texttt{move(B,Ta)}"] & \boxed{\begin{array}{l}\texttt{On(B,Ta)}\\ \texttt{On(A,Ta)}\end{array}} \arrow[r, "\texttt{move(A,B)}"] & \boxed{\texttt{On(A,B)}}
\end{tikzcd}$$

In this tree, we have eliminated the atom `On(B,Ta)` from the goal node since it is not part of the goal description.

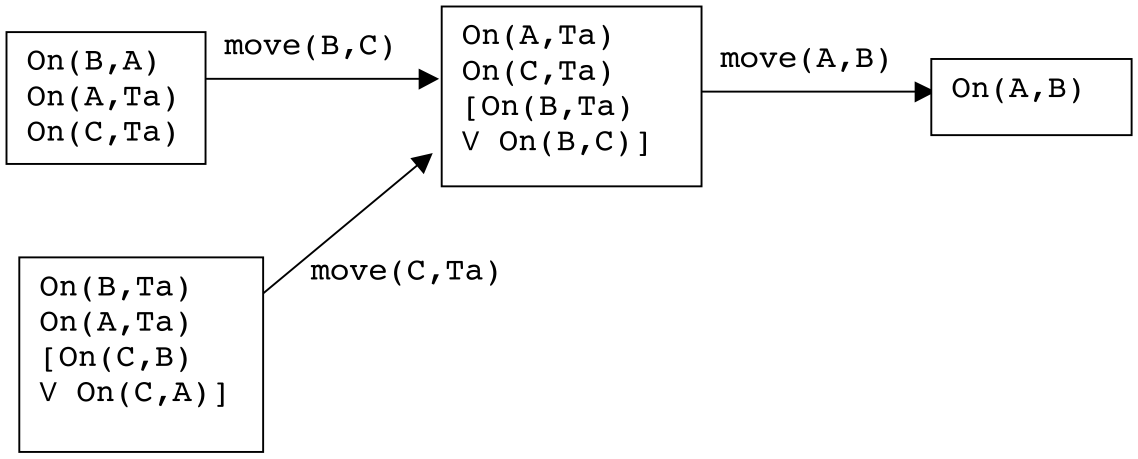
Combining the two state descriptions was easy in this case because both descriptions were identical. Furthermore, no generalization would be needed because the state descriptions used in the T-R tree are the only ones possible with only two blocks. Suppose, however that we add one more block. There are then 13 possible configurations (of which two satisfy the goal condition of having `A` on `B`) and several possible teaching sequences. (A graph showing these 13 configurations appears at the end of this note.) Consider, for example, the following three teaching sequences:

$$\begin{tikzcd}
\boxed{\begin{array}{l}\texttt{On(B,A)}\\ \texttt{On(A,Ta)}\\ \texttt{On(C,Ta)}\end{array}} \arrow[r, "\texttt{move(B,C)}"] & \boxed{\begin{array}{l}\texttt{On(B,C)}\\ \texttt{On(A,Ta)}\\ \texttt{On(C,Ta)}\end{array}} \arrow[r, "\texttt{move(A,B)}"] & \boxed{\begin{array}{l}\texttt{On(A,B)}\\ \texttt{On(B,C)}\\ \texttt{On(C,Ta)}\end{array}}
\end{tikzcd}$$





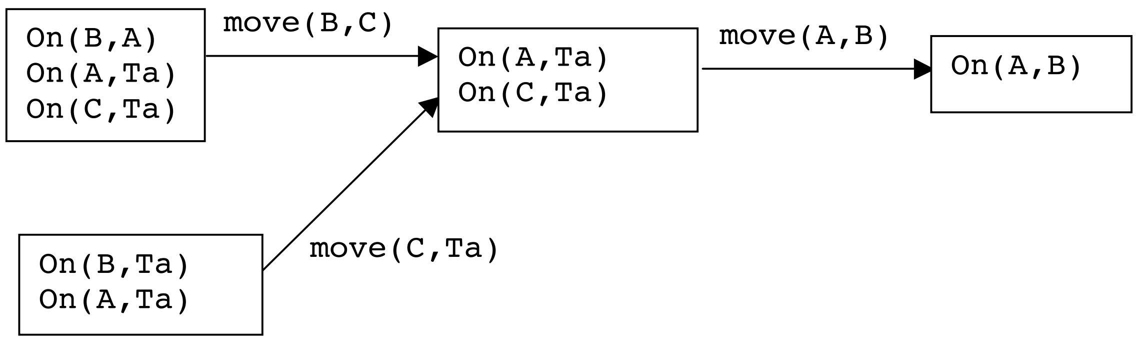
Applying the combination rule (recursively from the goal and again assuming the goal state is described simply by `On(A,B)`) to these three sequences yields the following T-R program:



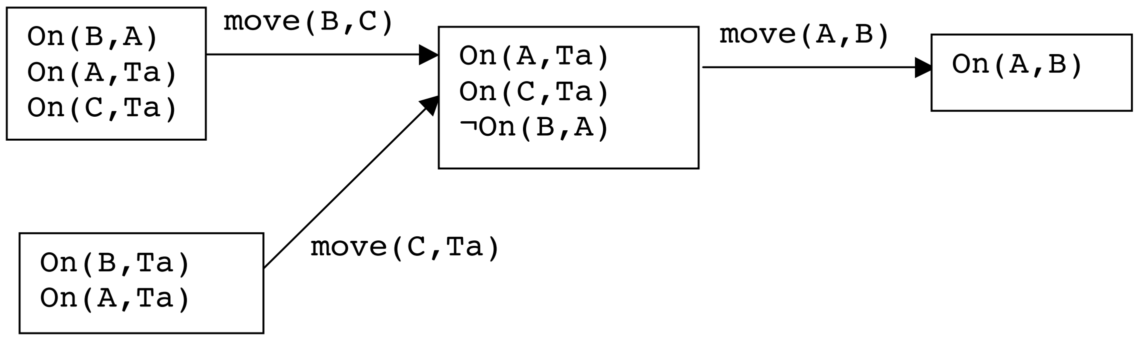
Here, we have disjunctions in two of the state descriptions. Even though this T-R tree has been developed from inadequate teaching experience, we are interested in the problem of how to generalize nodes so that they will correctly capture more state descriptions than those from which they were derived.

Our first suggestion is to over-generalize by eliminating the disjunctions entirely (keeping only the atoms common to both state descriptions) and then to correct this over-generalization by successively adding negated atoms as required by additional experience with the developing T-R tree. (Negations will be handled by “negation as failure”; that is, an atom is assumed to be false if it cannot be proved true.)

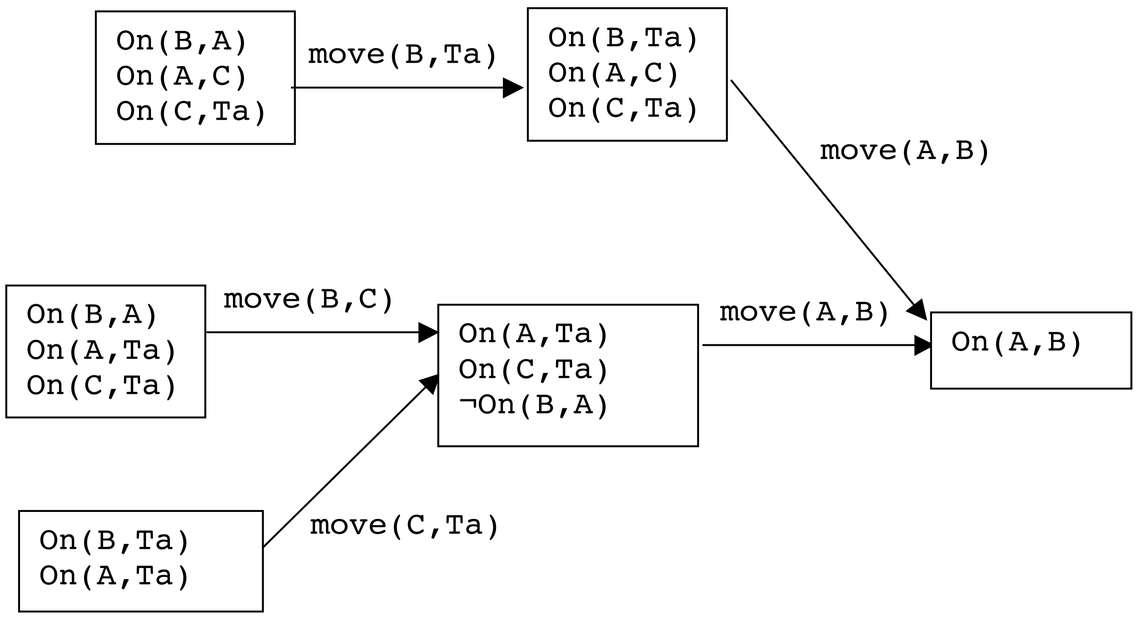
The generalized T-R tree would then be the following:



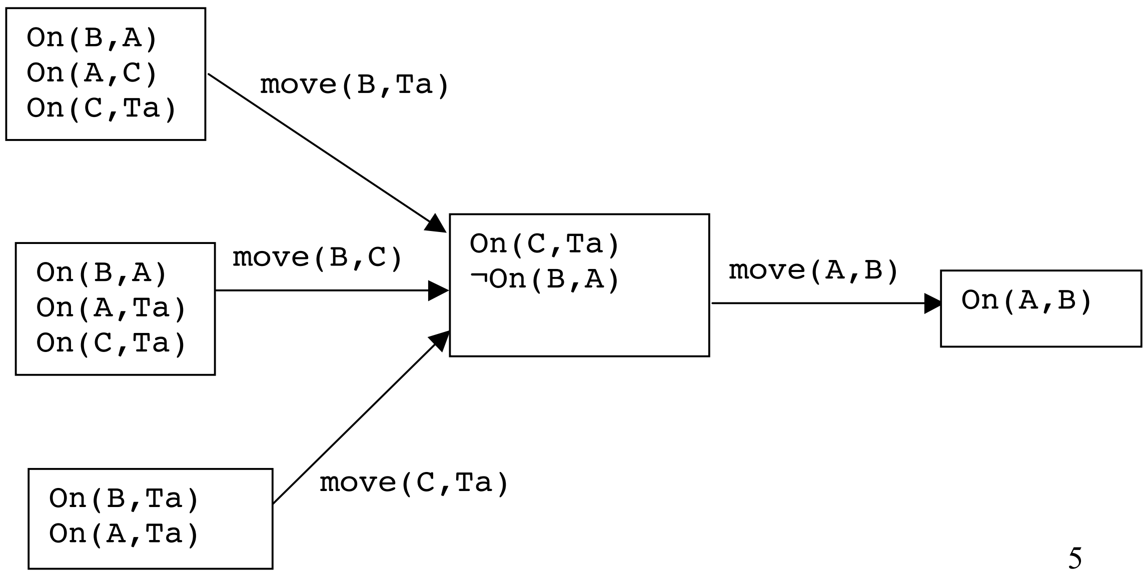
We illustrate the process of adding any necessary negated atoms by an example in which the above tree is used to control actions---calling on a teacher for guidance only when the tree fails us. Consider the configuration described by the conjunction of `On(B,A)`, `On(A,Ta)`, and `On(C,Ta)`. This configuration is “captured” by the node whose action is `move(A,B)`, but that action cannot be applied. Evidently it is the extra conjunct, `On(B,A)`, in the state description that prevents the action. Suppose we add the negation of this conjunct to the node to prevent it from capturing this description. This addition produces the following T-R tree:



This tree correctly handles 7 of the 13 possible configurations. The other six are not captured by the tree at all. Therefore, if one of them is encountered, the system must appeal to the teacher. Suppose the next situation encountered is described by one of these six, namely the conjunction of `On(B,A)`, `On(A,C)`, and `On(C,Ta)`. No node in the tree matches this description. Suppose the teacher applies the action `move(B,Ta)`, which produces the configuration described by `On(B,Ta)`, `On(A,C)`, and `On(C,Ta)`. Again, no node matches this description. Suppose the teacher next applies the action `move(A,B)`, producing the configuration described by `On(B,Ta)`, `On(A,B)`, and `On(C,Ta)`. This description matches the goal. We graft this teaching sequence onto the T-R tree and root it at the goal to produce the following tree:

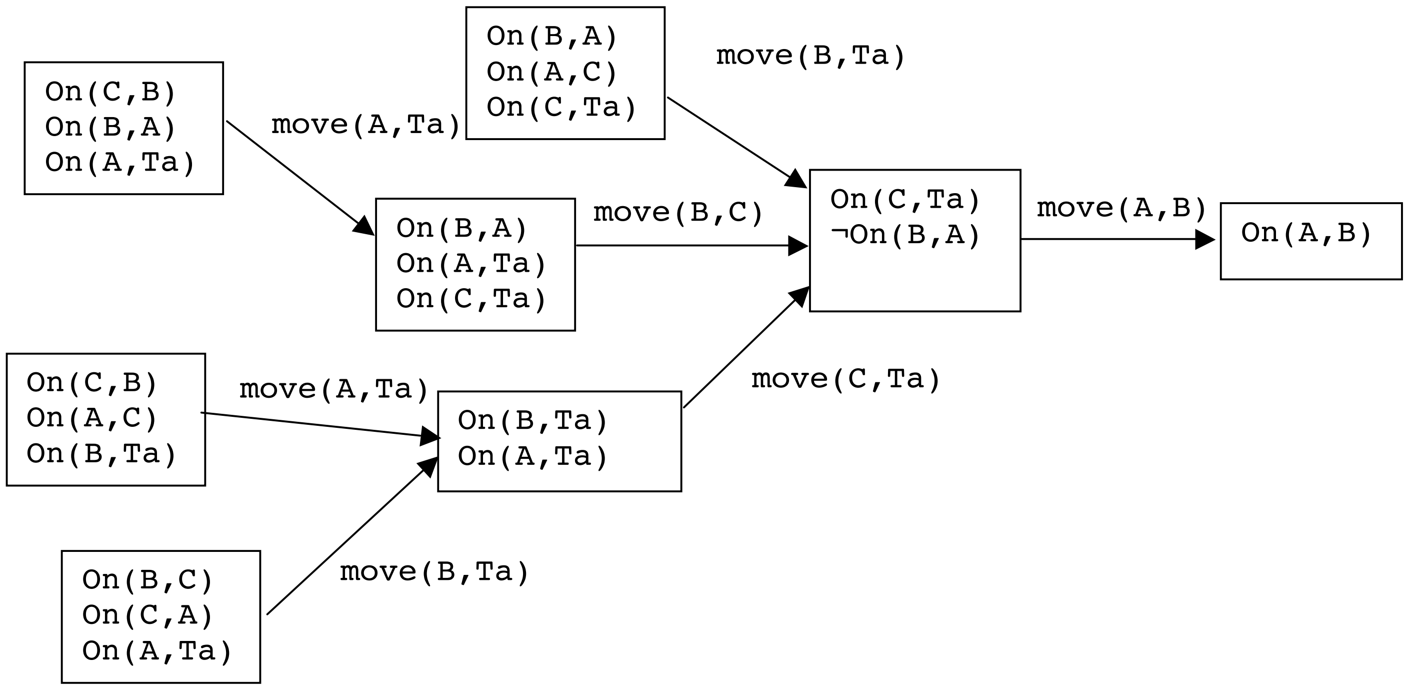


Here again, we can effect a combination. Eliminating the disjunction (but keeping the important negation) yields the tree:



One way to justify our keeping of ¬`On(B,A)` is to note that in one of the nodes participating in the combination, namely `On(B,Ta)`, `On(A,C)`, `On(C,Ta)`, we could have added ¬`On(B,A)` because it is implied by the existence of `On(B,Ta)`. With this addition, `On(B,Ta)` occurs in both nodes to be combined and thus would be retained.

Now there are just three configurations not captured by the tree. Teacher-guided moves from these configurations lead to the following final T-R tree:



Here are some additional matters not yet investigated:

1) Converting a learned T-R program to one in which constants are replaced by variables
2) Other ways of generalizing (and particularizing) a node
3) Inventing new predicates
4) Converting a learned T-R program to one that is applicable to situations with different numbers of blocks than the situation encountered in teaching experiences
5) Producing recursive programs
6) Converting common action sequences into macro-operators

REFERENCES

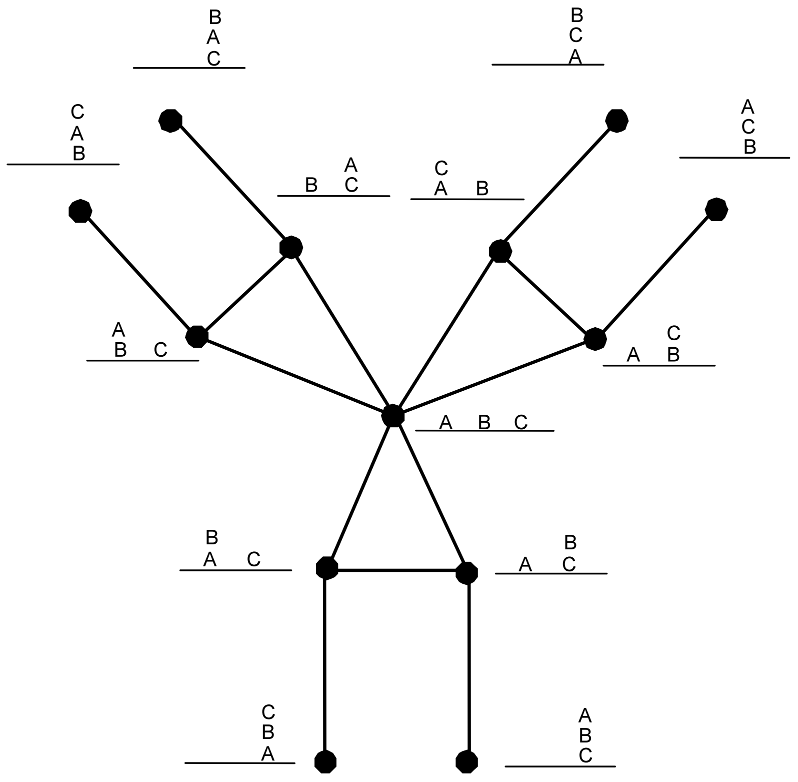
John 1994

John, G., "SQUISH: A Preprocessing Method for Supervised Learning of T-R Trees From Solution Paths," Draft Memo, Stanford Computer Science Dept., November 22, 1994.

Nilsson 2000

Nilsson, N., "Learning Strategies for Mid-Level Robot Control: Some Preliminary Considerations and Experiments," May, 2000 (draft research note, available on the web at: http://www.robotics.stanford.edu/users/nilsson/trweb/learningcontrol.ps)

The 13 configurations of three blocks linked by actions:


B A C
B C A
C A B
A C B
A B C
C A B
A B C
C A B
A B C
B A C
B A C
C B A
A B C