Formalization of Graph Search Algorithms and Its Applications.

Conference Paper · September 1998

DOI: 10.1007/BFb0055153 · Source: DBLP

5 authors, including:

Masami Hagiya
The University of Tokyo
207 PUBLICATIONS **1,834** CITATIONS
SEE PROFILE

Shin-ya Nishizaki
Tokyo Institute of Technology
26 PUBLICATIONS **127** CITATIONS
SEE PROFILE

Formalization of Graph Search Algorithms and Its Applications

Mitsuharu Yamamoto[1], Koichi Takahashi[2], Masami Hagiya[3], Shin-ya Nishizaki[4], and Tetsuo Tamai[5]

[1] Faculty of Science, Chiba University
[2] Electrotechnical Laboratory
[3] Graduate School of Science, The University of Tokyo
[4] Graduate School of Information Science and Engineering, Tokyo Institute of Technology
[5] Graduate School of Arts and Sciences, The University of Tokyo

Abstract. This paper describes a formalization of a class of fixed-point problems on graphs and its applications. This class captures several well-known graph theoretical problems such as those of shortest path type and for data flow analysis. An abstract solution algorithm of the fixed-point problem is formalized and its correctness is proved using a theorem proving system. Moreover, the validity of the A* algorithm, considered as a specialized version of the abstract algorithm, is proved by extending the proof of the latter. The insights we obtained through these formalizations are described. We also discuss the extension of this approach to the verification of model checking algorithms.

1 Introduction

There are two approaches to the verification of computer systems. One is to verify the correctness of a system using an interactive proof checker such as HOL[8], NQTHM[2], or PVS[14] and so on. This approach is of wide use but has a high cost with respect to human efforts. The other is to ensure correctness by an exhaustive search of the state space with a model checker, without human interaction. This approach is restricted to the specific properties of computer systems, but is highly practicable for finite-state systems with a limited number of states.

The use of the former approach in software verification has shifted from the verification of concrete programs (codes) to that of designs and algorithms. One reason is that the verification of programs is of little importance when considering the reliability of an entire software system, while verification requires very precise inferences.

This study focuses on a graph search algorithm. Although search algorithms usually aim at finding solutions, they are sometimes used to ensure that "no solution exists under the given conditions". When the aim is to find solutions, one can confirm whether the computed solutions satisfy the given conditions, regardless of the correctness of the search algorithm. On the other hand, it is

essential to state the correctness of the algorithm when one is trying to rigidly assert the absence of solutions.

Model checking is nothing but to show the absence of solutions (e.g., deadlock states) using some search algorithms. In general, the algorithms and software used for model checking are elaborately tuned, with many optimized strategies, in order to improve efficiency. Hence, it becomes non-trivial to ensure the correctness of such algorithms and softwares used in a real world.

In order to deal with several graph search algorithms uniformly, based on the work by Kildall [11], Tamai [15] formulated a class of graph search problems and then characterized two kinds of solutions: fixed-point solutions and a path solutions. This formulation captures several well-known problems of graph searches such as shortest path problems and data flow analyses. Tamai also characterized an abstract algorithm that computes a fixed-point solution, and informally showed its correctness.

In this study, we formalize Tamai's formulation with the HOL system. The two kinds of solutions and the abstract algorithm are formalized and the correctness of the algorithm is proved. Moreover, we obtain a formal proof of the A* algorithm, an optimized algorithm for solving a class of shortest path problems, by extending the proof of the abstract algorithm.

Since a formalization is usually hard work, we generally try to reduce the load by simplifying or making things more abstract. This sometimes leads to better abstraction or the more general settings of problems. For example, we managed to weaken the premises of the abstract algorithm, and thereby obtained insights on the correctness of the A* algorithm. This algorithm is characterized by the question of how to select a vertex from a pool of unprocessed vertices. However, through our formalization we found that this selection does not affect the correctness of the algorithm. This result can then be applied to the correctness of some variants of the A* algorithm, e.g., one that selects a vertex with certain heuristics in order to avoid falling into an infinite loop.

The usefulness of formalizations by proof checkers has been criticized mainly because of their high cost. However, if the verification of basic (abstract) algorithms can be used to derive the correctness of variants of the algorithm with relatively low cost, then we believe that the original verification is worth the costs of a great deal of human interaction.

This paper is organized as follows. Section 2 briefly reviews the formulation of problems on which our work based. This section also outlines the HOL system by which the formal proof is written. The actual formalization is described in Sect. 3 and Sect. 4, and is divided into the verification of the abstract algorithm and that of the A* algorithm. Section 5 reveals the insights that we obtained from this work. This paper ends with related work, concluding remarks and our probable future direction in Sect. 6 and Sect. 7.

2 Preliminary

2.1 Graph Algorithm

This section describes a brief overview of Tamai's paper [15], on which our formalization is based. Some parts of his formulation are omitted or changed from the original ones so as to make them more abstract or to facilitate the formalization. We will mention such changes in Sect. 5.1.

First we formulate the general setting of a search problem, consisting a (possibly cyclic) directed graph, a semi-lattice, and edge functions. Then we explain two kinds of solutions of a problem: fixed-point solutions and path solutions. After showing some examples of the problem, we explain an abstract algorithm which computes a fixed-point solution of the problem.

General Setting Let $G = (V, E)$ be a directed graph; let V be a set of vertices and E a set of edges. The starting vertex and the ending vertex of an edge $e \in E$ is expressed as $h(e)$ and $t(e)$, respectively. A set of incoming edges $\mathrm{in}(v)$ and outgoing edges $\mathrm{out}(v)$ of a vertex $v \in V$ is defined ordinarily.

$$\mathrm{in}(v) = \{e \in E \mid t(e) = v\}, \qquad \mathrm{out}(v) = \{e \in E \mid h(e) = v\}.$$

We assume that there is a distinguished vertex $s \in V$ such that $\mathrm{in}(s) = \emptyset$. Afterwards this vertex is treated as a starting point in a search problem.

A path $p = (v, [e_1, \ldots, e_n])$ $(n \geq 0)$ in the graph G is a (possibly empty) finite sequence of edges equipped with an entry vertex such that adjacent edges share the same end point, i.e. $v = h(e_1)$ and $t(e_i) = h(e_{i+1})$ for $i = 1, \ldots, n-1$. Note that a path $p = (v, [e_1, \ldots, e_n])$ starts from a vertex v and ends with a vertex w, where $w = t(e_n)$ when $n > 0$ and $w = v$ otherwise. A vertex w is said to be "reachable from v" when there is a path that starts from v and ends with w.

Let L be a semi-lattice, i.e., an underlying set L equipped with a meet operation $\wedge$ that satisfies the following conditions[1]:

1. $\forall x, y \in L.\, x \wedge y = y \wedge x$;
2. $\forall x, y, z \in L.\, x \wedge (y \wedge z) = (x \wedge y) \wedge z$;
3. $\forall x.\, x \wedge x = x$.

The partial order relation $\leq$ on L is defined as:

$$\forall x, y \in L.\, x \leq y \iff x \wedge y = x.$$

We further assume the existence of a top element $\top$ in L, i.e., $\forall x \in L.\, x \wedge \top = x$.

In a given problem, each edge $e \in E$ is associated with a monotonic function f_e from L to L, where $f_e(x) = \top$ only if $x = \top$. This function is naturally extended to a function f_p for a path p as $f_p = f_{e_n} \circ \cdots \circ f_{e_1}$ for $p = (v, [e_1, \ldots, e_n])$.

[1] In Tamai's paper [15], a (full) lattice is used instead of a semi-lattice. However we use only semi-lattices since the join operation $\vee$ is not used in this formalization.

The problem we are to formalize is formulated as follows. Given a graph $G = (V, E)$, a lattice L, a distinguished vertex $s \in V$ and an initial value $b_s \in L \setminus \{\top\}$ for the vertex s, and a function assignment for each edge $f_e \in L^L$ for $e \in E$, then compute an assignment of an element $x_v \in L$ for each vertex $v \in L$ that satisfies a certain condition. Such a condition determines a kind of solution of the problem, and two kinds of solutions are explained below.

Fixed-Point Solution An assignment $x_v \in L$ (for $v \in V$) is a *fixed-point solution* if it satisfies the following equations.

$$\begin{cases} x_s = b_s \\ x_v = \bigwedge_{t(e)=v \text{ and } h(e) \text{ is reachable from } s} f_e(x_{h(e)}) \quad (v \neq s) \end{cases}$$

The second equation is slightly inaccurate and misleading. It should be read as "the right hand side exists and equals to x_v" here.

Path Solution An assignment $x_v \in L$ (for $v \in V$) is a *path solution* if it satisfies:

$$x_v = \bigwedge_{p \in \mathrm{path}(s,v)} f_p(b_s)$$

where $\mathrm{path}(s, v)$ is a set of paths from vertex s to vertex v. Again, the above equation means "the right hand side exists and equals to x_v."

Problem Examples Several concrete problems can be characterized as special cases of the general setting formulated above. We show some examples here.

1. **Reachability on graphs** Let L be a lattice $\{\bot, \top\}$ with $\bot \leq \top$, f_e be an identity function for all $e \in E$, and b_s be $\bot$. Then the solution x_v for $v \in V$ is $\bot$ if v is reachable from s. Moreover, $x_v = \top$ if we consider the path solution x_v and v is unreachable from s.
2. **Shortest path problem** Let L be a set of real numbers $\mathbb{R}$ with an additional top element, equipped with the total order on $\mathbb{R}$. When distance d_e is associated with each edge e, define $f_e = \lambda x.\, x + d_e$. If $b_s = 0$, the solution x_v coincides with the distance of the shortest path from s to v.
3. **Finite State Automaton** Consider a finite state automaton whose edge e is labeled by an element a_e of alphabet Σ. Let L be set of all the subsets of Σ^*, equipped with superset ordering "$\supseteq$", and b_s be a singleton set consisting of an empty string ϵ, where s is the initial state, and define $f_e = \lambda x.\, \{wa_e \mid w \in x\}$. Then the path solution x_v corresponds to a set of strings accepted at a final state v.
4. **Dataflow analysis** Let us explain by taking the reachable definition problem as an example. Let L be a set of all the subsets of value definitions in a program, equipped with the subset ordering "$\subseteq$". The execution flows of the

program form a directed graph as usual. A function assigned to each edge e is

$$f_e(x) = (x \cap K_e^c) \cup G_e$$

where K_e^c is a complement of a set of killed definitions and G_e is a set of generated definitions by a program fragment corresponding to the edge e. Then the path solution gives a vertex a set of definitions that is reachable at the execution point corresponding to the vertex. More examples can be found in [11].

Iterative Solution Algorithm An abstract algorithm (Algorithm P) that computes fixed-point solution is given in Fig. 1

Fig. 1. Algorithm P

For each $v \in V$, $x_v \leftarrow \top$
$x_s \leftarrow b_s$
$S \leftarrow \{s\}$
While $S \neq \emptyset$ do
    Take and remove an arbitrary vertex v from S.
    For each $e \in \mathrm{out}(v)$ do
        If $x_{t(e)} \not\preceq f_e(x_v)$ then
            $x_{t(e)} \leftarrow x_{t(e)} \wedge f_e(x_v)$
            Add $t(e)$ to S.

The correctness of the algorithm is shown in [15].

2.2 A* Algorithm

As we mentioned, a class of graph problems in Sect. 2.1 subsumes shortest path problems. Here we explain the A* algorithm [13], one of the algorithms for solving shortest path problems using heuristic information to reduce search space, as an extension of the iterative solution algorithm(Fig. 1).

In the A* algorithm, heuristic information is given as estimation of distance $\mathrm{h}(v)$ from each vertex v to the fixed goal vertex. The algorithm can be formulated by replacing the following line in Fig. 1

Take and remove an arbitrary vertex v from S.

with

Take and remove a vertex v from S such that
$x_v + \mathrm{h}(v) \leq x_{v'} + \mathrm{h}(v')$ for $v' \in S$.
If v is the goal vertex, the algorithm stops.

(Recall that $x_v \in L = \mathbb{R} \cup \{\top\}$.)

If this estimation $\mathrm{h}(v)$ gives a lower bound of distance from v to the goal, the above algorithm computes one of the shortest paths if the algorithm terminates and such a path exists.

2.3 HOL

Our formalization is done by the HOL system, a tactical theorem prover based on Church's simple theory of types[6].

Some algebraic structures are used in the formalization: directed graphs and semi-lattices. We used the abstract theory library [16] that is bundled with the standard HOL90 distribution. With this library, one can make a predicate characterizing an algebraic structure more or less implicit, and obtain facilities of instantiating the abstract structure to a concrete one. Although instantiation of theorems is not used in this formalization, it will be useful when we try to apply this formalization to concrete problems.

We here introduce notations used in the later sections. By attaching the turnstile mark $\vdash$, we will specify that its following sentence is a theorem. The mark $\vdash_{def}$ is used for definitions and constant specifications similarly. We use Greek letters, $\alpha, \beta, \gamma, \ldots$, as type variables, **bold** font letters for type constants and sans serif font letters for term constants. Inside a definition or a constant specification, the defined term constant is written by an underlined symbol. Conditional expressions are denoted as "`if` A `then` B `else` C" where B and C must be of the same type, and A must be a boolean, whereas such expressions are written as "$A \Rightarrow B \mid C$" in the HOL system. Sometimes we put comments "*(** here is comments **)*" between term expressions for readability.

3 Formalization of the Abstract Algorithm

3.1 Overall Structure

Our formalization is comprised of five parts: partial order, semi-lattice, directed graph, fixed-point problem, and A* algorithm. Each of which corresponds to a theory, and as a whole they construct a hierarchical structure illustrated in Fig. 2.

The rest of this section describes the actual formalization for each part.

3.2 Partial Order and Semi-Lattice

Theory of partial order is comprised of a part of the CPO (Complete Partial Orders) theory by Camilleri and Prasetya [3] and some auxiliary theorems. Most of the operations on semi-lattices is directly imported from those of CPOs in the library.

- $\mathsf{Cap}\, r\, x\, y$: the greatest lower bound of x and y wrt. r;

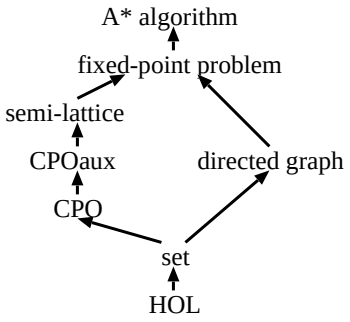

Fig. 2. Overall Structure of Our Formalization

- $\mathsf{Top}\, r$: the greatest element in type A according to a relation r on A;
- $\mathsf{Mono}\, r\, s\, f$: f is a monotonic function wrt. r and s,
  i.e., $\forall x\, y.\, r\, x\, y \Longrightarrow s\, (f\, x)(f\, y)$;
- $\mathsf{Distrib}\, r\, s\, f$: f is a distributive function wrt. r and s,
  i.e., $\forall x\, y.\, f\, ((\mathsf{Cap}\, r)\, x\, y) = (\mathsf{Cap}\, s)\, (f\, x)\, (f\, y)$.

where r and s are partial order relations. All of these but the last one are from the CPO library. Since a partial order relation is the only algebraic structure in CPO, almost all operations on CPO take partial order relations as arguments.

Using the abstract theory library, a semi-lattice is defined as an abstract entity with the following theory obligation (i.e., predicate specifying the conditions the abstract operations should satisfy).

$$\begin{aligned}\vdash_{def}\ & \forall(l : \alpha\, \mathbf{semilattice}).\ \underline{\mathsf{semilattice\_oblig}}\, l \Longleftrightarrow \\ & (\forall x\, y.\ \mathsf{meet}\, l\, x\, y = \mathsf{meet}\, l\, y\, x) \wedge \\ & (\forall x\, y\, z.\, \mathsf{meet}\, l\, x\, (\mathsf{meet}\, l\, y\, z) = \mathsf{meet}\, l\, (\mathsf{meet}\, l\, x\, y)\, z) \wedge \\ & (\forall x.\ \mathsf{meet}\, l\, x\, x = x)\end{aligned}$$

An element l of the type $\alpha\, \mathbf{semilattice}$, which is automatically defined by the abstract theory library, represents a semi-lattice whose underlying set is α. The meet operation "$\wedge$" of a semi-lattice l is denoted by $\mathsf{meet}\, l$, which is the only abstract operation of a semi-lattice. Note that the type of elements x, y, z of a lattice $l : \alpha\, \mathbf{semilattice}$ is α.

With the help of the library, we can make theory obligation predicates implicit. As a result, the statement

$$\forall(l : \alpha\, \mathbf{semilattice}).\ P\ l$$

actually means

$$\forall(l : \alpha\, \mathbf{semilattice}).\ \mathsf{semilattice\_oblig}\, l \Longrightarrow P\ l.$$

In the rest of this paper, we omit theory obligation predicates for brevity.

Although meet corresponds to the "$\wedge$" operation and it determines a semi-lattice, this operation is not so frequently used. Instead the partial order relation LEQ, derived from "$\wedge$" as follows, is used throughout the formalization, since almost all operations in CPO library takes partial orders as arguments as we mentioned above, and we extensively use the CPO library.

$$\vdash_{def} \underline{\mathsf{LEQ}}\, l\, x\, y \iff (\mathsf{meet}\, l\, x\, y = x)$$

As a result, operations on a semi-lattice are denoted less compactly as "$\mathsf{Top}(\mathsf{LEQ}\, l)$" and "$\mathsf{Cap}(\mathsf{LEQ}\, l)\, x\, y$".

On the other hand, the assumption on the existence of the top element in a lattice l is denoted as "$\mathsf{HAS\_Top}\, l$".

3.3 Directed Graph

Our basic definition of directed graphs follows Wong's work[17]. The most significant difference is that a directed graph is defined as an abstract entity as in the case of semi-lattices.

$$\vdash_{def} \forall g.\ \underline{\mathsf{graph\_oblig}}\, g \iff (\forall e.\, e \in \mathsf{ES}\, g \Longrightarrow \mathsf{es}\, e \in \mathsf{VS}\, g \wedge \mathsf{ed}\, e \in \mathsf{VS}\, g)$$

Here is a summary of important operations on directed graphs.

- $\mathsf{VS}\, g$: set of vertices of a graph g;
- $\mathsf{ES}\, g$: set of edges of a graph g;
- $\mathsf{es}\, e$: starting vertex of an edge e (formerly denoted as $h(e)$);
- $\mathsf{ed}\, e$: ending vertex of an edge e (formerly denoted as $t(e)$);
- $\mathsf{INCIDENT\_FROM}\, g\, v$: set of outgoing edges from a vertex v in a graph g (formerly denoted as $\mathrm{out}(v)$);
- $\mathsf{EMPTY\_PATH}\, v$: constructs an empty path $(v, [\,])$;
- $\mathsf{PATH\_SNOC}\, p\, e$: constructs a path $(v, [e_1, \ldots, e_n, e])$ by adding an edge e at the end of a path $p = (v, [e_1, \ldots, e_n])$.

3.4 Fixed-Point Problem

The abstract algorithm is formalized as a predicate that has the following form:

$$\mathsf{POTENTIAL}\, n\ g\ l\ s\ bs\ fe\ open\ closed\ outv\ xv.$$

Arguments of the predicate can be classified into the following three categories.

The first one is the execution counter. The argument n belongs to this category. This records number of steps in the execution. This also makes it possible to define the predicate by a recursive definition on natural numbers.

The second is that of parameters given as parts of specification of the problem. These values do not change during the execution. Arguments g, l, bs, and fe belong to this category, and the meaning of each argument is as follows:

- g : directed graph;

– l : semi-lattice;
– s : start point of search;
– bs : initial value at s;
– fe : assignment of a function to each edge ("$fe\,e\,x$" corresponds to $f_e(x)$).

The last category means internal states of the execution. These values are updated at each step of the execution. Arguments *open*, *closed*, *outv*, and *xv* belong to this category, and the meaning of each is as follows:

– *open* : not-processed vertices (corresponds to S in Fig. 1);
– *closed* : vertices processed at least once, but not in *open*;
– *outv* : edges incident from the expanded vertex and not processed yet;
– *xv* : assignment of an element of l to each vertex ("*xv v*" corresponds to x_v).

This predicate is defined by a recursive definition on natural numbers as in Fig. 3. This algorithm is considered to be terminated when both *open* and *outv* become empty sets. In this case, all of the internal states remain the same values as the previous states.

```
⊢def (∀g l s bs fe open closed outv xv.
    POTENTIAL 0 g l s bs fe open closed outv xv ⟺
    (open = {s}) ∧ (closed = {}) ∧ (outv = {}) ∧
    (xv = (λv. if (v = s) then bs else (Top(LEQ l))))) ∧
   (∀n g l s bs fe open closed outv xv.
    POTENTIAL(SUC n) g l s bs fe open closed outv xv ⟺
    (∃open' closed' outv' xv'.
     POTENTIAL n g l s bs fe open' closed' outv' xv' ∧
     (if (outv' = {})
      then
      (if (open' = {})
       then ((open = open') ∧ (closed = closed') ∧ (outv = outv') ∧ (xv = xv'))
       else (∃v. v ∈ open' ∧ (closed = {v} ∪ closed') ∧
             (open = open' \ {v}) ∧ (outv = INCIDENT_FROM g v) ∧ (xv = xv')))
      else (** (outv' ≠ {}) **)
      (∃e. e ∈ outv' ∧ (outv = outv' \ {e}) ∧
       (if (LEQ l (xv' (ed e)) (fe e (xv' (es e))))
        then ((open = open') ∧ (closed = closed') ∧ (xv = xv'))
        else ((open = {ed e} ∪ open') ∧ (closed = closed' \ {ed e}) ∧
              (xv = (λv. if (v = ed e)
                         then (Cap(LEQ l) (xv'(ed e)) (fe e (xv' (es e))))
                         else (xv' v)))))))))
```

Fig. 3. Definition of Abstract Algorithm

Note that the original algorithm introduced in the previous section forms a doubly-nested loop, while the predicate is defined by a single loop. The inner

loop of the original one is assimilated to the outer one, and an internal state $outv'$ of the previous step determines which loop we are executing; we are in the outer loop if and only if $outv' = \{\}$.

With this definition, we proved the following theorem that ensures the correctness of solutions.

1. Algorithm POTENTIAL gives a fixed-point solution when it terminates.

$$\begin{aligned}\vdash\ & \forall g\ l\ s\ bs\ fe.\ s \in \mathsf{VS}\ g \wedge (\forall e.\ \mathsf{ed}\ e \neq s) \wedge \mathsf{HAS\_Top}\ l \wedge (bs \neq \mathsf{Top}(\mathsf{LEQ}\ l)) \wedge \\ & (\forall x\ e.\ e \in \mathsf{ES}\ g \wedge (fe\ e\ x = \mathsf{Top}(\mathsf{LEQ}\ l)) \Longrightarrow (x = \mathsf{Top}(\mathsf{LEQ}\ l))) \wedge \\ & (\forall e.\ e \in \mathsf{ES}\ g \Longrightarrow \mathsf{Mono}(\mathsf{LEQ}\ l)\ (\mathsf{LEQ}\ l)\ (fe\ e)) \Longrightarrow \\ & \quad (\forall n\ closed\ xv.\ \mathsf{POTENTIAL}\ n\ g\ l\ s\ bs\ fe\ \{\}\ closed\ \{\}\ xv \Longrightarrow \\ & \quad\quad \mathsf{IS\_FPSOLN}\ l\ g\ fe\ bs\ s\ xv)\end{aligned}$$

2. Algorithm POTENTIAL gives the path solution when it terminates.

$$\begin{aligned}\vdash\ & \forall g\ l\ s\ bs\ fe.\ s \in \mathsf{VS}\ g \wedge (\forall e.\ \mathsf{ed}\ e \neq s) \wedge \mathsf{HAS\_Top}\ l \wedge (bs \neq \mathsf{Top}(\mathsf{LEQ}\ l)) \wedge \\ & (\forall x\ e.\ e \in \mathsf{ES}\ g \wedge (fe\ e\ x = \mathsf{Top}(\mathsf{LEQ}\ l)) \Longrightarrow (x = \mathsf{Top}(\mathsf{LEQ}\ l))) \wedge \\ & (\forall e.\ e \in \mathsf{ES}\ g \Longrightarrow \mathsf{Distrib}(\mathsf{LEQ}\ l)\ (\mathsf{LEQ}\ l)\ (fe\ e)) \Longrightarrow \\ & \quad (\forall n\ closed\ xv.\ \mathsf{POTENTIAL}\ n\ g\ l\ s\ bs\ fe\ \{\}\ closed\ \{\}\ xv \Longrightarrow \\ & \quad\quad \forall v.\ v \in \mathsf{VS}\ g \Longrightarrow \mathsf{IS\_PATHSOLN}\ l\ g\ fe\ bs\ s\ v\ (xv\ v)))\end{aligned}$$

Predicates IS_FPSOLN and IS_PATHSOLN mean the last argument is a fixed-point solution and a path solution respectively. Note that distributivity is required in the case of path solution.

The first theorem required 13 intermediate lemmas (two of them are mentioned in Sect. 4). The second required additional 1 definition and 8 lemmas. Most of these lemmas are proved by induction over the number n of steps. As a whole, proofs of these theorems are accomplished by a HOL script approximately 1300 lines long.

4 Verification of A* Algorithm

In this section, we define the A* algorithm and show theorems we proved about that. Although the algorithm is defined on $\mathbb{R} \cup \{\top\}$ in general, we here formalize it on a general semi-lattice.

Since the A* algorithm can be expressed as an extension of the abstract algorithm as we mentioned in Sect. 2.2, its formalization becomes similar one to that of the abstract algorithm(Fig. 4). In the definition, $\mathsf{IS\_MINIMAL}\ r\ X\ x$ means $x \in X$ is a minimal element in a set X wrt. an order relation r.

Compared with the predicate POTENTIAL in the previous section, ASTAR has 4 additional arguments: *goal* and *hv* as parts of specification, *stat* and *po* as internal states. The meaning of each argument is as follows:

- *goal* : the goal vertex
- *hv* : assignment of an estimation function to each vertex
- *stat* : status of the execution

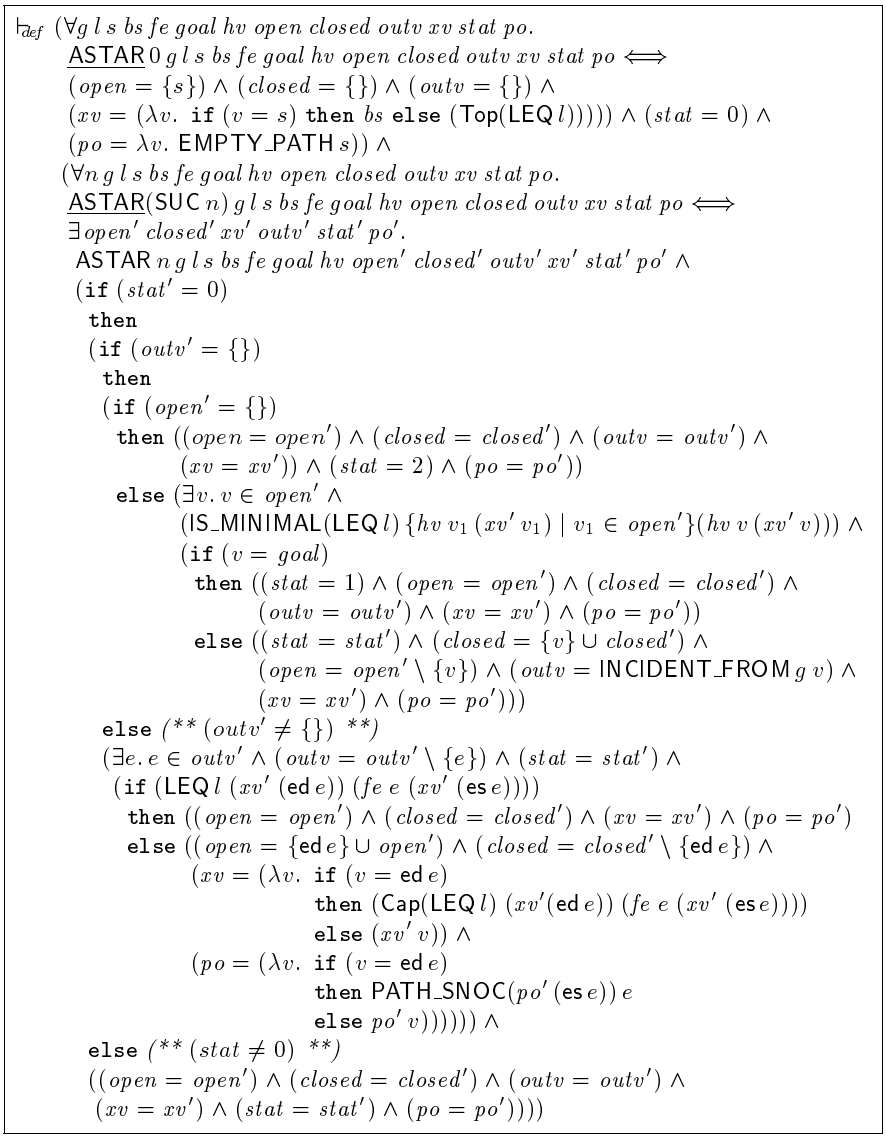

Fig. 4. Definition of A* Algorithm

– po : assignment of an optimal path from s to each vertex

Arguments hv and $stat$ should be explained in further detail. In Sect. 2.2, estimation $\mathrm{h}(v)$ for a vertex v was a real number, and $x_v + \mathrm{h}(v)$ was used as a measure when selecting a vertex from S. In the formalization, however, we do not restrict the lattice L to $\mathbb{R} \cup \{\top\}$, but make $hv\,v$ a function from l to l and $hv\,v\,(xv\,v)$ is used as a measure for the selection. Obviously this formalization generalizes the usual formulation; set $hv\,v = \lambda x.\, x + \mathrm{h}(v)$. Execution status $stat$ has one of 0, 1 or 2 as its value. Value 0 means the execution is now proceeding, 1 means that we have already reached the goal vertex and the execution has been finished, and 2 means we could not reach the goal vertex (i.e., $goal$ is not reachable from s in the given problem.), and the execution has been finished.

Internal values are updated as in POTENTIAL's case while $stat = 0$, and these values stop changing when $stat$ becomes 1 or 2. The most significant difference between ASTAR and POTENTIAL is how to select vertex from $open$ as we mentioned in Sect. 2.2. Here we select a vertex $v(\in open)$ such that $hv\,v\,(xv\,v)$ is *minimal* in the sense of the order "$\mathsf{LEQ}\,l$" in the semi-lattice l, since there may be no *minimum* value(here we do not restrict "$\mathsf{LEQ}\,l$" to a total order). We later discuss this decision in Sect. 5.2.

Most of the properties of POTENTIAL can be imported to that of ASTAR, using the following theorem:

$$\begin{array}{l}\vdash \forall g\,l\,n\,s\,bs\,fe\,goal\,hv\,open\,closed\,outv\,xv\,stat\,po.\\ \quad (\mathsf{ASTAR}\,n\,g\,l\,s\,bs\,fe\,goal\,hv\,open\,closed\,outv\,xv\,stat\,po \Longrightarrow\\ \quad\ (\exists m.\,(\mathsf{POTENTIAL}\,m\,g\,l\,s\,bs\,fe\,open\,closed\,outv\,xv)))\end{array}$$

The above theorem itself can be proved by expanding ASTAR and POTENTIAL with their definitions, and by induction over n. With the above theorem, a theorem whose form is "$\forall g\,l \ldots.\ \mathsf{ASTAR} \ldots \Longrightarrow P$"can be derived from a theorem "$\forall g\,l \ldots.\ \mathsf{POTENTIAL} \ldots \Longrightarrow P$".

The following two theorems, which are used in the proof of the correctness of POTENTIAL, also play important roles in the correctness of ASTAR. They correspond to Assertion 1 and 3 in [15].

– For each edge $e = (v, w)$, $x_w \leq f_e(x_v)$, if e have already been processed.

$$\begin{array}{l}\vdash \forall g\,l\,n\,s\,bs\,fe\,open\,closed\,outv\,xv.\\ \quad \mathsf{POTENTIAL}\,n\,g\,l\,s\,bs\,fe\,open\,closed\,outv\,xv\\ \quad\ \Longrightarrow \forall e.\,e \in \mathsf{ES}\,g \wedge \mathsf{es}\,e \in closed \wedge e \notin outv\\ \qquad\qquad \Longrightarrow (\mathsf{LEQ}\,l)\,(xv\,(\mathsf{ed}\,e))\,(fe\,e\,(xv\,(\mathsf{es}\,e)))\end{array}$$

– For each edge $e = (v, w)$ such that $v \in closed$, either $w \in open \cup closed$, or e is being processed.

$$\begin{array}{l}\vdash \forall g\,l\,s\,bs\,fe.\ \mathsf{HAS\_Top}\,l \wedge (bs \neq \mathsf{Top}(\mathsf{LEQ}\,l)) \wedge\\ \quad (\forall x\,e.\,e \in \mathsf{ES}\,g \wedge (fe\,e\,x = \mathsf{Top}(\mathsf{LEQ}\,l)) \Longrightarrow (x = \mathsf{Top}(\mathsf{LEQ}\,l))) \Longrightarrow\\ \quad \forall n\,open\,closed\,outv\,xv.\\ \quad\ \mathsf{POTENTIAL}\,n\,g\,l\,s\,bs\,fe\,open\,closed\,outv\,xv\\ \quad\ \Longrightarrow \forall e.\,e \in \mathsf{ES}\,g \wedge \mathsf{es}\,e \in closed\\ \qquad\qquad \Longrightarrow \mathsf{ed}\,e \in open \vee \mathsf{ed}\,e \in closed \vee e \in outv\end{array}$$

With importing the above theorems from POTENTIAL, we proved the following lemma.

$$
\begin{array}{l}
\vdash \forall g\, l\, n\, s\, bs\, fe\, goal\, hv\, open\, closed\, outv\, xv\, po. \\
\quad \mathsf{ASTAR}\, n\, g\, l\, s\, bs\, fe\, goal\, hv\, open\, closed\, outv\, xv\, 1\, po \wedge \\
\quad \mathsf{HAS\_Top}\, l \wedge (bs \neq \mathsf{Top}(\mathsf{LEQ}\, l)) \wedge \\
\quad (\forall x\, e.\, e \in \mathsf{ES}\, g \wedge (fe\, e\, x = \mathsf{Top}(\mathsf{LEQ}\, l)) \Longrightarrow (x = \mathsf{Top}(\mathsf{LEQ}\, l))) \wedge \\
\quad (\forall e.\, e \in \mathsf{ES}\, g \Longrightarrow \mathsf{Mono}(\mathsf{LEQ}\, l)\, (\mathsf{LEQ}\, l)\, (fe\, e)) \wedge \\
\quad (\forall bv\, v.\, \mathsf{IS\_LEQ\_PATHSOLN}\, l\, g\, fe\, (hv\, v\, bv)\, bv\, v\, goal) \wedge \\
\quad (\forall v.\, (v \in open) \Longrightarrow (\mathsf{LEQ}\, l)\, (xv\, goal)\, (hv\, v\, (xv\, v))) \\
\quad \Longrightarrow \mathsf{IS\_LEQ\_PATHSOLN}\, l\, g\, fe\, (xv\, goal)\, bs\, s\, goal
\end{array}
$$

where $\mathsf{IS\_LEQ\_PATHSOLN}\, l\, g\, fe\, x\, b\, v\, w$ means x is a lower bound of

$$\{((fe\, e_n) \circ \cdots \circ (fe\, e_1))(b) \mid (v, [e_1, \ldots, e_n]) \text{ is a path from } v \text{ to } w\}.$$

A condition that uses this predicate in the above lemma corresponds to the condition that "$\mathrm{h}(v)$ gives a lower bound of distance from v to the goal."

The main theorem on the A* algorithm is that it gives a path solution when it normally terminates (i.e., when $stat = 1$). Since a path solution is expressed as the greatest lower bound of a certain set, we have to assert 1) the solution is a lower bound of the set, and 2) it is the greatest one among any lower bound of the set. The former condition is achieved by the above lemma. Then the latter one is asserted by adding an extra condition that "$\mathsf{LEQ}\, l$" is total, which enables us to prove that "$po\, goal$" actually records a path from s to $goal$ and its distance coincide with "$xv\, goal$". As a result, we obtain the following theorem:

$$
\begin{array}{l}
\vdash \forall g\, l\, n\, s\, bs\, fe\, goal\, hv\, open\, closed\, outv\, xv\, po. \\
\quad \mathsf{ASTAR}\, n\, g\, l\, s\, bs\, fe\, goal\, hv\, open\, closed\, outv\, xv\, 1\, po \wedge \\
\quad \mathsf{HAS\_Top}\, l \wedge (bs \neq \mathsf{Top}(\mathsf{LEQ}\, l)) \wedge \\
\quad (\forall x\, e.\, e \in \mathsf{ES}\, g \wedge (fe\, e\, x = \mathsf{Top}(\mathsf{LEQ}\, l)) \Longrightarrow (x = \mathsf{Top}(\mathsf{LEQ}\, l))) \wedge \\
\quad (\forall e.\, e \in \mathsf{ES}\, g \Longrightarrow \mathsf{Mono}(\mathsf{LEQ}\, l)\, (\mathsf{LEQ}\, l)\, (fe\, e)) \wedge \\
\quad (\forall bv\, v.\, \mathsf{IS\_LEQ\_PATHSOLN}\, l\, g\, fe\, (hv\, v\, bv)\, bv\, v\, goal) \wedge \\
\quad (\forall b.\, hv\, goal\, b = b) \wedge (\mathsf{IS\_TOTAL}\, l) \\
\quad \Longrightarrow \mathsf{IS\_PATHSOLN}\, l\, g\, fe\, bs\, s\, goal\, (xv\, goal)
\end{array}
$$

5 Insights from Our Formalization

This section describes some insights resulting from our formalization. We mention two topics: Modification of the original proof and an observation on the formalization of the A* algorithm.

5.1 Modifications to the Original Proof

As mentioned before, we made several modifications to [15]. This section describes these modifications in detail.

First, we modified the definition of fixed-point solutions in Sect. 2.1, so that reachability from s is taken into account. The original definition of a fixed-point solution is characterized by the following equation [15]:

$$\begin{cases} x_s = b_s \\ x_v = \bigwedge_{t(e)=v} f_e(x_{h(e)}) \quad (v \neq s) \end{cases}$$

However, the algorithm P (Fig.1) may not compute a fixed-point solution, if there is no restriction on reachability, since an isolated vertex can have non-$\top$ value of L.

In the original work, directed graphs were restricted to be finite. This was because the existence of the GLB (Greatest Lower Bound) of an arbitrary subset of L was used when proving propositions having the following form:

For each step n, if a is the GLB of a set $X(n)(\subseteq L)$ then $P(a)$

If we try to prove the above proposition by induction over n, we should prove "$P(a)$ holds" from the following assumptions:

- (Assumption1) For each step n, if a' is the GLB of a set $X(n)$ then $P(a')$
- (Assumption2) a is the GLB of a set $X(n+1)$.

Then nothing can be derived from Assumption1, since we cannot assert the existence of the GLB of $X(n)$ without using some kinds of completeness on L or finiteness of a directed graph.

Therefore, we replace the above proposition with a more general one:

For each step n, if a is a LB (lower bound) of a set $X(n)(\subseteq L)$ then $P(a)$

Note that the new version results in no loss of generality whenever the GLB of $X(n)$ exists. Then all we have to do is to prove "$P(a)$ holds" from the following assumptions:

- (Assumption1') For each step n, if a' is a LB of a set $X(n)$ then $P(a')$
- (Assumption2') a is a LB of a set $X(n+1)$.

Since it is easy to prove that "each LB of $X(n+1)$ is also a LB of $X(n)$" in our case, one can use Assumption1' and prove the new version without assuming finiteness of a directed graph.

5.2 A* Algorithm

In Sect. 4, we gave a lemma on the A* algorithm with respect to the predicate ASTAR. However, if we examine its proof carefully, we notice that properties of ASTAR are not used in the proof. In fact, we can prove this as a property of

POTENTIAL:

$$\begin{array}{l}\vdash \forall g\, l\, n\, s\, bs\, fe\, goal\, hv\, open\, closed\, outv\, xv.\\ \quad \mathsf{POTENTIAL}\, n\, g\, l\, s\, bs\, fe\, open\, closed\, outv\, xv \wedge\\ \quad \mathsf{HAS\_Top}\, l \wedge (bs \neq \mathsf{Top}(\mathsf{LEQ}\, l)) \wedge\\ \quad (\forall x\, e.\, e \in \mathsf{ES}\, g \wedge (fe\, e\, x = \mathsf{Top}(\mathsf{LEQ}\, l)) \Longrightarrow (x = \mathsf{Top}(\mathsf{LEQ}\, l))) \wedge\\ \quad (\forall e.\, e \in \mathsf{ES}\, g \Longrightarrow \mathsf{Mono}(\mathsf{LEQ}\, l)\, (\mathsf{LEQ}\, l)\, (fe\, e)) \wedge\\ \quad (\forall bv\, v.\, \mathsf{IS\_LEQ\_PATHSOLN}\, l\, g\, fe\, (hv\, v\, bv)\, bv\, v\, goal) \wedge\\ \quad (outv = \{\}) \wedge (goal \notin closed) \wedge\\ \quad (\forall v.\, (v \in open) \Longrightarrow (\mathsf{LEQ}\, l)\, (xv\, goal)\, (hv\, v\, (xv\, v)))\\ \quad \Longrightarrow \mathsf{IS\_LEQ\_PATHSOLN}\, l\, g\, fe\, (xv\, goal)\, bs\, s\, goal\end{array}$$

In the definition of the A* algorithm(Fig. 4), a vertex that is minimal in some sense is selected from a set *open*. However, the above theorem means that vertex selection is not essential to the *correctness* of the A* algorithm (it only affects its efficiency, i.e., the size of the search space), while a condition

$$\forall v.\, (v \in open) \Longrightarrow (\mathsf{LEQ}\, l)\, (xv\, goal)\, (hv\, v\, (xv\, v))$$

at the final stage (i.e., when the goal vertex is selected) is. The above condition means the value assigned to the goal vertex is less than or equal to $hv\, v\, (xv\, v)$ for all $v \in open$. Note that this condition is satisfied at the final stage in the usual A* algorithm with an assumption that $\mathsf{LEQ}\, l$ is a total order.

This fact implies the possibility of variants of the A* algorithm that may not select a vertex v such that $x_v + \mathrm{h}(v)$ attains the minimum value in *open*. For example, one can consider an algorithm that selects a vertex from *open* so that the search may not fall into an infinite loop, even for a problem when the usual A* algorithm fails to terminate. All we should state is that the above condition is satisfied at the final stage, i.e., selection criteria at the middle of execution have nothing to do with the correctness of the solution.

In Sect. 4, we added an assumption that "$\mathsf{LEQ}\, l$" is a total order to assert the computed solution is the greatest one among any lower bound of a certain set. However it can be achieved without a total order condition, but with a distributivity on "$\mathsf{LEQ}\, l$". The theorem about this fact is also expressed as an property of POTENTIAL, and can be used both in POTENTIAL's case and in ASTAR's case:

$$\begin{array}{l}\vdash \forall g\, l\, s\, bs\, fe.\, s \in \mathsf{VS}\, g \wedge \mathsf{HAS\_Top}\, l \wedge\\ \quad (\forall e.\, e \in \mathsf{ES}\, g \Longrightarrow \mathsf{Distrib}(\mathsf{LEQ}\, l)\, (\mathsf{LEQ}\, l)\, (fe\, e))\\ \quad \Longrightarrow \forall n\, open\, closed\, outv\, xv.\\ \quad \mathsf{POTENTIAL}\, n\, g\, l\, s\, bs\, fe\, open\, closed\, outv\, xv\\ \quad \Longrightarrow \forall v\, a.\, \mathsf{IS\_LEQ\_PATHSOLN}\, l\, g\, fe\, a\, bs\, s\, v\\ \quad \Longrightarrow (\mathsf{LEQ}\, l)\, a\, (xv\, v)\end{array}$$

This observation was made by Bruno Martin [12]. We are now planning to rewrite the proof of the correctness of the A* algorithm according to this observation.

6 Related Work

Formalizations on graphs and graph algorithms have been achieved for several theorem proving systems as an end in themselves or as a basis for other work. Formalizations of several kinds of graphs (directed [17], undirected [4], and planar [18]) can be seen even in a single system. Hasselink [10] constructed a mechanical proof of a distributed algorithm for minimum-weight spanning trees in NQTHM. Basin and Kaufmann [1] compared two systems, NQTHM and Nuprl, using Ramsey's theorem on graphs as an example. Graph algorithms can also be used as a target of program verification. An algorithm that tests the acyclicity of a graph with an efficient depth first search algorithm is verified by the KIV system [9] and an executable program is then extracted. Chou and Peled [5] verified the partial order reduction algorithm as an example of model-checking algorithms in HOL. As in our work, they also verified a meta-theory (abstract level of the algorithm) that led to a deeper understanding of the algorithm.

7 Concluding Remarks

We have succeeded in formally verifying the correctness of the A* algorithm. The verification of A* is based on that of the abstract algorithm, POTENTIAL. In particular, most properties needed for the correctness of A* could be verified as properties of POTENTIAL. This process led us to realize that some assumptions on the A* algorithm were redundant for its correctness, because the related properties of POTENTIAL could be demonstrated without them.

The definition of A* was written by hand, and the basic relation between A* and POTENTIAL, i.e., the fact that any property that holds for POTENTIAL also holds for A* was also proved, though not automatically. However, the definition of A* was derived from that of POTENTIAL by the addition of some parameters and conditions. If we could define A* by specifying only such differences, it would be possible to derive automatically the relation between the two as a theorem. Since the definitions are inductive, we need a mechanism to give a new inductive definition from an existing inductive definition by stating such differences. In other words, "inheritance" of inductive definitions should be supported.

We have not proved the termination of the A* algorithm. In order to demonstrate this, we need another formulation of A* that contains additional internal states such as a trace of the execution. Such a formulation should also be derived from the original definition of A*.

The efficiency of A* has also not been formalized. We showed that some assumptions on A* were not necessary for its correctness, but they are required for the arguments on its efficiency. Formalizing efficiency arguments in general will be our future work. For the efficiency arguments on A*, we will need yet another formulation of A*.

This work is also motivated by the goal of verifying the correctness of the verifiers such as model checkers based on computation tree logic. We plan to formalize the semantics of computation tree logic [7], and verify the correctness of

the algorithm for model checking as a concrete example of the abstract algorithm in our framework.

Assume that a state transition graph is given. Generate a new graph by reversing the direction of each edge, and adding a new vertex s and edges from s to the original vertices. The added edge from s to v is denoted by e_v. Let p be a predicate on a state, i.e., a function from each state to a truth value. We use the boolean lattice $\{\top, \bot\}$ as the lattice for assigning values to vertices. If a transition from a vertex v to a vertex v' exists in the original graph, the generated graph has an edge e from v' to v. For such an edge, define $f_e(x) = \mathsf{p}(v) \wedge x$. As for a newly added edge e_v from s, define $f_{e_v}(x) = \top$. We show that the formula $\mathbf{A}\Box\,\mathsf{p}$ (p always holds on any infinite path) corresponds to the path solution under this setting. Let x_v be the path solution of the above problem. Then

$$x_v = \bigwedge_{p' \in \mathrm{path}(s,v)} f_{p'}(b_s) \leq f_p(b_s) = \mathsf{p}(v_1) \wedge \ldots \wedge \mathsf{p}(v_n)$$

holds for any path $p = v_0, v_1, \ldots, v_n$ from $s(= v_0)$ to $v(= v_n)$. Hence if $x_v = \top$,

$$\mathsf{p}(v_1) = \ldots = \mathsf{p}(v_n) = \top$$

holds. This means that p holds at vertices $v_n(= v), \ldots, v_1$.

Conversely, if p holds at all the vertices on a path from v to v' for any v' in the original state transition graph, then

$$f_p(b_s) = \mathsf{p}(v_1) \wedge \ldots \wedge \mathsf{p}(v_n) = \top$$

holds in the generated graph. Therefore, $x_v = \bigwedge f_p(b_s) = \top$ holds for the path solution x_v. As a result, $x_v = \top$ if and only if the formula $\mathbf{A}\Box\,\mathsf{p}$ holds at the state v.

The formula $\mathbf{E}\Box\,\mathsf{p}$ (p always holds on a certain infinite path), however, corresponds to a solution of the following equation.

$$x_v = \bigvee_{t(e)=v} f_e(x_{h(e)})$$

The correspondence between the formula and a fixed-point solution also exists in this case. If x_v is the maximal solution of the above equation (maximal fixed-point), then $x_v = \top$ holds if and only if there exists an infinite path on which p always holds. Defining an algorithm that computes the maximal solution will also be our future work.

Acknowledgements

We thank Bruno Martin from ENS Lyon, who was visiting the University of Tokyo. The first formal verification of the A* algorithm is done by him using another strategy. We also thank anonymous referees for constructive comments.

References

1. David Basin and Matt Kaufmann. The Boyer-Moore prover and Nuprl: An experimental comparison. In Gérard Huet and Gordon Plotkin, editors, *Logical Frameworks*, pages 89–119. Cambridge University Press, 1991.
2. Robert S. Boyer and J Strother Moore. *A Computational Logic*. Academic Press, New York, 1979.
3. Albert J Camilleri and Wishnu Prasetya. Cpo theory, 1994. Available from `http://www.cl.cam.ac.uk/ftp/hvg/hol88/contrib/cpo/`.
4. C.-T. Chou. A formal theory of undirected graphs in higher-order logic. In *Proceedings of the 7th International Workshop on Higher Order Logic Theorem Proving and Its Applications*, volume 859 of *LNCS*, pages 144–157, Valletta, Malta, September 1994. Springer-Verlag.
5. Ching-Tsun Chou and Doron Peled. Verifying a model-checking algorithm. In *Tools and Algorithms for the Construction and Analysis of Systems*, number 1055 in LNCS, pages 241–257, Passau, Germany, 1996. Springer-Verlag.
6. A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.
7. E. Allen Emerson. *Handbook of Theoretical Computer Science*, chapter 16. Elsevier Science Publishers B.V., 1990.
8. M. J. C. Gordon and T. F. Melham, editors. *Introduction to HOL: A theorem proving environment for higher order logic*. Cambridge University Press, 1993.
9. R. Hähnle, M. Heisel, W. Reif, and W. Stephan. An interactive verification system based on dynamic logic. In J. Seikmann, editor, *8th International Conference on Automated Deduction*, number 230 in LNCS, pages 306–315, Oxford, 1986. Springer-Verlag. Also note `http://www.informatik.uni-ulm.de/pm/kiv/kiv.html`.
10. Wim H. Hesselink. The verified incremental design of a distributed spanning tree algorithm — extended abstract. Computing Science Reports Groningen CS–R9602, November 1997. Available from `http://www.cs.rug.nl/~wim/ghs/whh168.ps`.
11. G. Kildall. A unified approach to global program optimization. In *POPL*, pages 194–206, 1973.
12. Bruno Martin, July 1997. Private Communication.
13. Nils J. Nilsson. *Principles of Artificial Intelligence*. Tioga Publishing, 1980.
14. N. Shankar, S. Owre, and J. Rushby. The PVS proof checker: A reference manual. Technical report, Computer Science Lab, SRI Intl., 1993.
15. Tetsuo Tamai. A class of fixed-point problems on graphs and iterative solution algorithms. In *Logic and Software Engineering*, pages 102–121. World Scientific, 1996.
16. P. Windley. Abstract theories in HOL. In *Higher Order Logic Theorem Proving and its Applications: Proceedings of the IFIP TC10/WG10.2 Workshop*, volume A-20 of *IFIP Transactions*, pages 197–210, Leuven, Belgium, September 1992. North-Holland/Elsevier.
17. W. Wong. A simple graph theory and its application in railway signalling. In *Proceedings of the 1991 International Workshop on the HOL Theorem Proving System and its Applications*, pages 395–409, Davis, California, USA, August 1991. IEEE Computer Society Press, 1992.
18. Mitsuharu Yamamoto, Shin-ya Nishizaki, Masami Hagiya, and Yozo Toda. Formalization of planar graphs. In *8th International Workshop on Higher-Order Logic Theorem Proving and Its Applications*, volume 971 of *LNCS*, pages 369–384. Springer-Verlag, 1995.