On Correctness and Completeness
of an n Queens Program

WŁODZIMIERZ DRABENT

Institute of Computer Science, Polish Academy of Sciences,
ul. Jana Kazimierza 5, 01-248 Warszawa, Poland
and
Department of Computer and Information Science,
Linköping University S – 581 83 Linköping, Sweden
(e-mail: drabent@ipipan.waw.pl*)*

Abstract

Thom Frühwirth presented a short, elegant, and efficient Prolog program for the n queens problem. However, the program may be seen as rather tricky and one may not be convinced about its correctness. This paper explains the program in a declarative way and provides proofs of its correctness and completeness. The specification and the proofs are declarative, that is they abstract from any operational semantics. The specification is approximate, it is unnecessary to describe the program's semantics exactly. Despite the program works on non-ground terms, this work employs the standard semantics, based on logical consequence and Herbrand interpretations. Another purpose of the paper is to present an example of precise declarative reasoning about the semantics of a logic program.

KEYWORDS: logic programming, declarative programming, program completeness, program correctness, specification, nonground answers

1 Introduction

Thom Frühwirth (1991) presented a short, elegant, and efficient Prolog program for the n queens problem. However, the program may be seen as rather tricky and one may not be convinced about its correctness. The author's description is rather operational. So it should be useful to explain the program declaratively and to provide formal proof that it is correct.

In imperative and functional programming, program correctness implies that the program produces the "right" results. In logic programming, which is nondeterministic, the situation is different. One also needs the program to be complete, that is to produce all the results required by the specification. (In particular, the empty program producing no answers is correct whatever the specification is.)

This paper provides proofs of correctness and completeness of the n queens program; the proofs are declarative, that is they abstract from any operational semantics.

The paper is organized as follows. After technical preliminaries, Section 3 presents the n queens program together with an informal description of its declarative semantics.

It also discusses how to overcome the difficulties with constructing its specification. The next section presents a formal specification. Proofs of correctness and completeness of the program are subjects of, respectively, Sections 5 and 6. Section 7 discusses the approach. The last section concludes the paper.

2 Preliminaries

Basics. This paper considers definite clause logic programs. We employ the standard terminology and notation (Apt 1997) and do not repeat here standard definitions and results. We assume a fixed alphabet of function and predicate symbols. The Herbrand universe will be denoted by $\mathcal{HU}$, the Herbrand base by $\mathcal{HB}$, and the set of all terms (atoms) by $\mathcal{TU}$ (respectively $\mathcal{TB}$); $\mathcal{HB}_p$ is the set of ground atoms with the predicate symbol p. By $\mathbb{N}$ we denote the set of natural numbers. We sometimes do not distinguish a number $i \in \mathbb{N}$ from its representation as a term, $s^i(0)$.

We use the list notation of Prolog. We assume that $[e_1, \ldots, e_n|e]$ stands for e when $n = 0$. A list (respectively open list) of length $n \geq 0$ is a term $[e_1, \ldots, e_n] \in \mathcal{TU}$ ($[e_1, \ldots, e_n|v] \in \mathcal{TU}$, where v is a variable); $e_1, \ldots, e_n$ are the members of this (open) list. We generalize the notion of (open) list membership, and say that $e \in \mathcal{TU}$ is **member** of a term $t \in \mathcal{TU}$ if $t = [e_1, \ldots, e_{k-1}, e|e']$, for some terms $e_1, \ldots, e_{k-1}, e'$, where $k > 0$. In such case, we also say that e is the k-th member of t. Note that this kind of membership is defined by the Prolog built-in predicate member/2. As in Prolog, each occurrence of _ stands for a distinct variable.

We follow the approach of Apt (1997) to SLD-resolution. So we consider queries instead of goals. Queries are conjunctions of atoms. By an **answer** of a program P, we mean any query Q such that $P \models Q$ (Q is a logical consequence of P). So an answer is a query to which a computed or correct answer substitution has been applied; Apt (1997) calls it computed/correct instance of a query. (It does not matter whether correct or computed answer substitutions are considered here, due to soundness and completeness of SLD-resolution.) $\mathcal{M}_P$ stands for the least Herbrand model of a program P. By the *relation* defined by a predicate p in P, we mean $\{\vec{t} \in \mathcal{TU}^n \mid P \models p(\vec{t})\}$, where n is the arity of p.

Specifications. In this paper, the treatment of specifications and reasoning about correctness and completeness follows that of Drabent (2016); missing proofs and further explanations can be found there. For further discussion, examples and references, see also Drabent (2018) and Drabent and Miłkowska (2005).

By a **specification** we mean an Herbrand interpretation $S \subseteq \mathcal{HB}$. A program P is **correct** w.r.t. a specification S when $\mathcal{M}_P \subseteq S$. This implies that $S \models Q$ (S is a model of Q) for any answer Q of P. Note that $S \models Q$ means that each ground instance of each atom of Q is a member of S. A program P is **complete** w.r.t. S when $S \subseteq \mathcal{M}_P$. This implies that, for any ground query Q, if $S \models Q$, then Q is an answer of P. So Q is an instance of an answer in each SLD-tree for P and any query Q_0 more general than Q.[1]

---

[1] As Prolog implements SLD-resolution except for the occur-check, the following holds in practice. If a program P, complete w.r.t. S, is executed as a Prolog program with a query Q_0, and the computation terminates, then all the answers for Q_0 described by S are computed. More precisely, if $S \models Q$ for a ground instance Q of Q_0, then Q is an instance of some answer produced by the computation. Similarly, assume that P is correct w.r.t. S and is executed with some query. Assume also that the occur-check is not needed in this computation (P with the query is occur-check free (Apt 1997)). Then $S \models Q$ for each obtained answer Q. These two facts hold for any selection rule.

Dealing with the n queens program we face a usual phenomenon: Often it is inconvenient (and unnecessary) to specify $\mathcal{M}_P$ exactly, that is to provide a specification S for which the program is both correct and complete, $S = \mathcal{M}_P$. It is useful to use instead an **approximate specification**, which is a pair (S_{compl}, S_{corr}) of specifications for, respectively, completeness and correctness. We say that a program P is *fully correct* w.r.t. (S_{compl}, S_{corr}) when $S_{compl} \subseteq \mathcal{M}_P \subseteq S_{corr}$. The approximation is *exact* if $S_{compl} = S_{corr}$.

The choice of an approximate specification depends on the properties of interest. See for instance Drabent (2019) or Drabent and Miłkowska (2005) for various specifications for APPEND describing various properties of the program.

Proving program correctness. An obvious sufficient condition for correctness is provided by Theorem 1 below. According to Deransart (1993), the condition is due to Clark (1979).

Theorem 1

For a program P and a specification S, if $S \models P$, then P is correct w.r.t. S.

Proof

As S is an Herbrand model of P, the least Herbrand model of P is a subset of S. $\qquad\square$

As S is an Herbrand interpretation, $S \models P$ means that for each ground instance $H \leftarrow B_1, \ldots, B_n$ $(n \geq 0)$ of a clause of P, if $B_1, \ldots, B_n \in S$ then $H \in S$.

Proving program completeness. First, we introduce some auxiliary notions.

Definition 2

A ground atom H is **covered** by a clause C w.r.t. a specification S if H is the head of a ground instance $H \leftarrow B_1, \ldots, B_n$ $(n \geq 0)$ of C, such that $B_1, \ldots, B_n \in S$ (Shapiro 1983).

A ground atom H is *covered by a program P* w.r.t. S if H is covered w.r.t. S by some clause $C \in P$.

Definition 3

A **level mapping** is a function $|\ |: \mathcal{HB} \to \mathbb{N}$. A program P is **recurrent** w.r.t. a level mapping $|\ |$ (Bezem 1993) when, for each ground instance $H \leftarrow B_1, \ldots, B_n$ $(n \geq 0)$ of a clause of P and each $i \in \{1, \ldots, n\}$, we have $|H| > |B_i|$.

The completeness proof presented in this paper will be based on the following lemma, which is an immediate corollary of (Drabent 2016, Theorem 5.6 and Proposition 5.4) or of (Deransart and Małuszyński 1993, Theorem 6.1).

Lemma 4

Let P be a program, and S a specification. If each atom $A \in S$ is covered by P w.r.t. S, and P is recurrent w.r.t. some level mapping, then P is complete w.r.t. S.

A note on built-ins. The presented approach can be generalized in a rather obvious way to Prolog with some built-ins. We focus here on Prolog arithmetic. A program P using arithmetic predicates (like is/2, or >/2) can be understood (Apt 1997) as augmented with an infinite set of ground unit clauses defining the ground instances of arithmetic relations. Apt denotes the set by $P(Ar)$. Such clauses are for example (in the infix form) 4 is 2+2, and 2+2 < 7. To deal with correctness or completeness of such program, we assume that

the specification is augmented with $P(Ar)$ (more precisely, that the set of atoms with arithmetic predicates in the specification is $P(Ar)$). We also assume that $|B| = 0$ for each $B \in P(Ar)$. Now the sufficient conditions for correctness and completeness apply. (As they are obviously satisfied by $P(Ar)$, the condition for correctness needs to be checked only for the clauses from P, and that for completeness only for atoms with non built-in predicate symbols.)

This approach abstracts from run-time errors. So completeness w.r.t. S means that if $S \models Q$ and Q is a ground instance of a query Q_0, then Q is an instance of an answer of a Prolog computation starting with Q_0, unless a run-time error or infinite loop is encountered.

3 The n queens program

This section presents the n queens program of Frühwirth (1991), provides its informal declarative description, and discusses how to construct its specification. Possible inaccuracies due to informal approach will be corrected in the next sections, dealing with a formal specification and proofs.

The problem is to place n queens on an $n \times n$ chessboard so that there is exactly one queen on each row and each column, and at most one queen on each diagonal. The main idea of the program is to describe the position of the queens by a data structure in which it is impossible that two queens are placed on the same row, column, or a diagonal. In this way, the constraints of the problem are treated implicitly and efficiently.

This paper considers the version of the program which represents natural numbers as terms in a standard way. Another version employs Prolog arithmetic. The specifications and proofs of Sections 4, 5 and 6 can be, in a rather obvious way, transformed to ones dealing with the latter version, following *A note on built-ins* from the previous section.

Here is the main part of the program (with abbreviated predicate names and with the original comment); it will be named NQUEENS.

```
pqs(0,_,_,_).                                              (1)
pqs(s(I),Cs,Us,[_|Ds]):-
        pqs(I,Cs,[_|Us],Ds), pq(s(I),Cs,Us,Ds).           (2)
% pq(Queen,Column,Updiagonal,Downdiagonal)  places a single queen
pq(I,[I|_],[I|_],[I|_]).                                   (3)
pq(I,[_|Cs],[_|Us],[_|Ds]):- pq(I,Cs,Us,Ds).              (4)
```

Solutions to the n queen problem are provided by those answers of NQUEENS that are of the form $pqs(n, q, t_1, t_2)$, where n is a number and q a list of length n. A number $j \in \{1, \ldots, n\}$ being the k-th member of q means that the queen of row j is placed on column k. (The role of t_1, t_2 will be explained later.) So to obtain the solutions, one can use a query $Q_{\text{in},n} = pqs(n, q_0, \_, \_)$, where q_0 is a list of n distinct variables.

We quote the original description of the program, as it is an example of nondeclarative viewing of logic programs:

> Observing that no two queens can be positioned on the same row, column or diagonals, we place only one queen on each row. Hence we can identify the queen by its
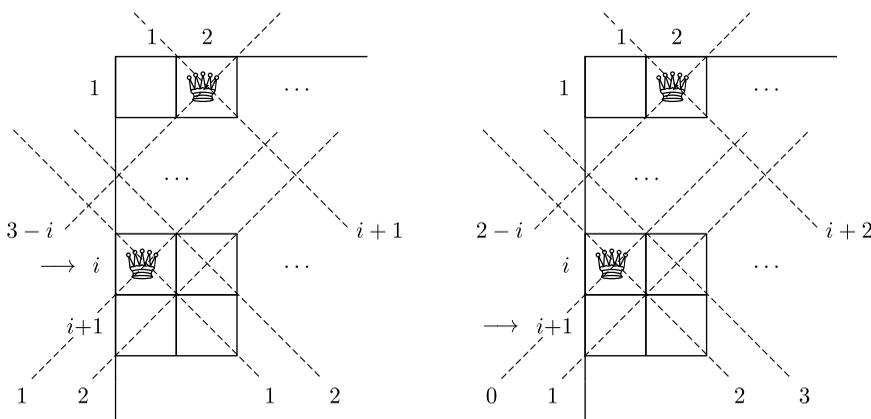
Fig. 1. Numbering of rows and columns. Numbering of up-diagonals ($\nearrow$-diagonals) and down-diagonals ($\searrow$-diagonals) in the context of row i (left), and $i+1$ (right), where $i > 2$.

row-number. Now imagine that the chess-board is divided into three layers, one that deals with attacks on columns and two for the diagonals going up and down respectively. We indicate that a field is attacked by a queen by putting the number of the queen there.

Now we solve the problem by looking at one row at a time, placing one queen on the column and the two diagonal-layers. For the next row/queen we use the same column layer, to get the new up-diagonals we have to move the layer one field up, for the down-diagonals we move the layer one field down.

This does not have much to do with the logic of the program; in particular, the relations defined by the program are not described. Instead, actions of the program are described. Also, the description does not seem to justify why the program is correct. Let us try to treat the program declaratively, abstracting from the operational semantics.

Chessboard representation. Assume that columns and rows of the $n \times n$ chessboard are numbered from 1 to n, from left to right and from top to bottom, respectively. So in an up-diagonal ($\nearrow$-diagonal), all the squares have the same sum of the row number and the column number. In a down-diagonal ($\searrow$-diagonal), the difference of the two numbers is the same. Each queen is identified by its row number.

In contrast to the numbering of rows and columns, the numbering of diagonals is not fixed, it is specific to the context of the currently considered row; the diagonal number m includes the m-th square of the row, for $m \in \{1, \dots, n\}$ (Figure 1). So, in the context of row i, a queen j (i.e. that of row j) placed on a column k is on the up-diagonal of number $k + j - i$, and on the down-diagonal of number $k + i - j$. Consider, for instance, queen 1 placed on column 2 (Figure 1). Then, in the context of row i, it is on the up-diagonal $3 - i$, and on the down-diagonal $1 + i$.

Given a set $A \subseteq \{1, \dots, n\}$ of queens, by a **correct placement** of queens A we mean placing them on the chessboard so that each row, column, up-diagonal and down-diagonal contains at most one queen from A.

When the initial query is $Q_{\text{in},n}$ as described above, the program represents the position of queens (in the context of a given row i) by a list and two open lists. In a general case, this representation consists of three terms of the form $[t_1, \dots, t_l | t]$. They represent,

respectively, the columns, up-diagonals, and down-diagonals, so that

> if column (up-diagonal, down-diagonal) k, where $k > 0$, contains the queen j
> then j is the k-th member of the term representing the columns (respectively (5)
> up-diagonals, down-diagonals).

If a column (or a diagonal) k contains no queen then the k-th member of the respective term is arbitrary[2] or does not exist. Such representation guarantees that at most one queen can be placed on each column and each diagonal with a positive number.[3]

For example, a chessboard with two queens $1, i$ placed as in Figure 1 is represented as follows: The columns are represented by $[i, 1|\ldots]$. In the context of row i, the up-diagonals are represented by $[i|\ldots]$ (queen 1 is not represented here, as its up-diagonal has number $3 - i \leq 0$); the down-diagonals are represented by $[i, \ldots, 1|\ldots]$, where 1 is the $(i+1)$-th member of the term (as the queen 1 is on the down-diagonal number $i+1$). In the context of row $i + 1$, both queens are not represented in a term representing the up-diagonals, and the down-diagonals are represented with $[t, i, \ldots, 1|\ldots]$, where 1 is the $(i+2)$-th member of the term (and t is arbitrary).

Rationale. Now we informally describe the purpose of the predicates of the program. This presentation is preliminary; the aim is to facilitate introducing the actual specification.

The role of pq is to define (a relation consisting of) tuples $(i, cs, us, ds) \in \mathcal{TU}^4$, where

$$\text{for some } j > 0, i \text{ is the } j\text{-th member of each } cs, us, ds. \tag{6}$$

This will be used to assure that a queen i is placed on a column, up-diagonal and down-diagonal of the same number.

The role of pqs is to define tuples $(i, cs, us, [t|ds]) \in \mathbb{N} \times \mathcal{TU}^3$ such that

> $i > 0$ and cs, us, ds represent as in (5)
> a correct placement of queens $1, \ldots, i$ on, respectively,
> the columns, up-diagonals and down-diagonals, (7)
> where the diagonals are numbered in the context of row i,

and additionally all the tuples from $\{0\} \times \mathcal{TU}^3$. For example, NQUEENS has an answer A describing placing two queens on a 4×4 chessboard, $A = pqs(2, cs, us, [\_|ds])$, where $cs = [1, \_, 2, \_]$, $us = [\_, \_, 2|\_]$, and $ds = [\_, 1, 2|\_]$. The argument tuple of A satisfies (7).

Now we understand, for instance, why in clause (2) the third argument in the head $pqs(s(I), Cs, Us, [\_|Ds])$ differs from that in the body atom $pqs(I, Cs, [\_|Us], Ds)$. This is because if, according to (7), term $[\_|Us]$ represents in the context of row I the up-diagonals with positive numbers, then its tail Us does this in the context of row $s(I)$. Similar reasoning applies to the fourth argument and down-diagonals.

Note that property (7) is not closed under substitution. (E.g. answer A above has an instance $A' = pqs(2, [1, 1, 2, \_], us, ds)$ which places the same queen on two columns, and two queens on a down-diagonal.) So what we described differs from the relation actually defined by pqs, and our description needs to be corrected.

---

[2] It is a variable when the initial query is $Q_{\text{in},n}$.
[3] Diagonals with non-positive numbers are dealt with in contexts in which their numbers are positive.

Informal specification. Note first that property (6) is closed under substitution (due to employing the generalized notion of member). Thanks to this our specification for pq is obvious:

$$S_{pq} = \{ \, pq(\, i, \, [c_1, \ldots, c_k, i|c], \, [u_1, \ldots, u_k, i|u], \, [d_1, \ldots, d_k, i|d] \,) \in \mathcal{HB} \mid k \geq 0 \, \}. \qquad (8)$$

Note that this specification is exact, in the sense that $\{(3), (4)\}$ (the fragment of NQUEENS defining pq) is both correct and complete w.r.t. S_{pq}.

The difficulty in constructing a specification for pqs is that the program has also answers which represent incorrect placement of queens. This cannot be avoided, as any answer of NQUEENS like A above (i.e. representing a correct placement of queens with some columns empty) has instances which violate the conditions of the puzzle. It may seem that we face a contradictory task: the role of our specification is to describe correct placements, but it has to include some incorrect ones.

The idea to overcome the difficulty is to care only about those atoms $pqs(i, cs, us, ds) \in \mathcal{HB}$, where cs is a list of distinct members. It leads to the following informal specification for correctness for pqs:

$$\begin{aligned}
&\text{the set of those } pqs(i, cs, us, ds) \in \mathcal{HB} \text{ where}\\
&\quad i \in \mathbb{N}, \; 1, \ldots, i \text{ are members of term } cs, \text{ and}\\
&\quad \text{if } i > 0 \text{ and } cs \text{ is a list of distinct members then} \qquad\qquad (9)\\
&\qquad ds = [t|ds'] \text{ (for some } t, ds'), \text{ and}\\
&\qquad \text{condition (7) holds for } i, cs, us, ds'.
\end{aligned}$$

It follows that if cs is a list of length i then it is a list of distinct members $1, \ldots, i$, and hence it is a solution of the i queens problem. Now our specification for NQUEENS is the union of the sets (8) and (9). Note that it serves its purpose, as correctness w.r.t. it implies that the program solves the problem (each answer for the initial query $Q_{\text{in},n}$ represents a solution). Note also that the specification contains atoms which are not answers of NQUEENS, for example $pqs(1, [1, 1], [], [])$.

Such informal specification facilitates understanding of the program and makes possible informal but precise reasoning about the program. For an example, consider a ground instance of clause (2)

$$pqs(s(i), cs, us, [t|ds]) \; \leftarrow \; pqs(i, cs, [t'|us], ds), \; pq(s(i), cs, us, ds).$$

Let us denote the body atoms by B_1 and B_2, respectively. Assume that they are as described by the specification, that is $B_1 \in (9)$, $B_2 \in (8)$. We show that also the head is as described by the specification, that is, it is in (9). By $B_2 \in (8)$, $s(i)$ is a member of cs. Assume that cs is a list of distinct members. So by $B_1 \in (9)$ and by (7), cs, $[t'|us]$ and the tail of ds represent a correct placement of queens $1, \ldots, i$ in the context of row i. Hence, this placement in the context of row $i+1$ is represented by cs, us, ds. By $B_2 \in (8)$ we have that, in the same context, cs, us, ds represent placing the queen $i + 1$. So its column is distinct from those occupied by queens $1, \ldots, i$, the same holds for its up- and down-diagonals. Thus, cs, us, ds represent a correct placement of queens $1, \ldots, i+1$. Hence the head of the clause instance is in the set (9).

The reasoning of the last paragraph explains the clause and convinces us about its correctness. Actually it is an informal outline of a central part of a correctness proof of the program, based on Theorem 1. In the next section, the specification outlined here is made formal and is augmented by a specification for completeness.

4 Approximate specification

This section presents a pair of specifications for correctness and for completeness of NQUEENS, formalizing the ideas from the previous section. We often do not distinguish between number i and the queen i.

The specification for predicate pq is obvious. Both for correctness and for completeness it is S_{pq} from (8) in the previous section.

In order to formulate the specification for pqs, we introduce some additional notions. Assume a queen j, that is that of row j, is in column k of the chessboard described by a list of columns cs. This means that the k-th member of cs is j. Then, in the context of row i, the numbers of the two diagonals containing this queen are defined as follows.

Definition 5
Let a number j be the k-th member of a list cs.

    The **up-diagonal number** of j, w.r.t. i in cs is $k + j - i$.

    The **down-diagonal number** of j, w.r.t. i in cs is $k + i - j$.

Obviously, queens j, j' are on the same up (respectively down) diagonal iff for some i they have the same up (down) diagonal number w.r.t. i.[4] Note that "for some i" can be replaced by "for all i," so we can skip "w.r.t. i" when stating that some queens have distinct up- (respectively down-) diagonal numbers.

Now we are ready to introduce the core of our specification.

Definition 6
A triple of terms $(cs, us, ds) \in \mathcal{TU}^3$ *represents a correct placement* up to row m in the context of row i (in short: is **correct** up to m w.r.t. i) when $0 \le m \le i$ and

    cs is a list of distinct members, and each $j \in \{1, \ldots, m\}$ is its member,

    the up (respectively down) diagonal numbers of $1, \ldots, m$ in cs are distinct,   (10)

    for each $j \in \{1, \ldots, m\}$,

        if the up (down) diagonal number of j w.r.t. i in cs is $l > 0$

        then the l-th member of us (respectively ds) is j.   (11)

For example, $([1, a, 2, b], [c, d, 2], [e, 1, 2])$ is correct up to 2 w.r.t. 2, and the same holds w.r.t. 3 for $([1, a, 2, b], [d, 2], [f, e, 1, 2])$; both triples represent a correct placement of queens $1, 2$ on a 4×4 chessboard. Given that queens (of rows) $1, \ldots, m$ are placed on the columns as described by cs, condition (11) assures that us, ds describe the placement of these queens on the diagonals with positive numbers. (An up-diagonal number in (10), (11) may be negative, as $j \le i$.) Obviously, (10) implies that cs describes a correct placement of queens $1, \ldots, m$.

Now the specification for pqs is

$$S_{pqs} = \big\{\, pqs(0, cs, us, ds) \mid cs, us, ds \in \mathcal{HU} \,\big\} \cup$$

$$\left\{\, pqs(i, cs, us, [t|ds]) \in \mathcal{HB} \;\middle|\; \begin{array}{l} i > 0, \quad 1, \ldots, i \text{ are members of } cs, \\ \text{if } cs \text{ is a list of distinct members then} \\ (cs, us, ds) \text{ is correct up to } i \text{ w.r.t. } i. \end{array} \,\right\},$$

---

[4] As an up-diagonal consists of those squares for which the sum $k + j$ of the column number k and the row number j is the same. Similarly, for a down-diagonal $k - j$ is constant

and our specification of NQUEENS for correctness is

$$S = S_{pq} \cup S_{pqs}.$$

Note that correctness w.r.t. S implies the required property of the program. Take an instance A of query $Q_{\text{in},n}$, so $A = pqs(n, cs', us', ds') \in \mathcal{TU}$, where $n > 0$ and cs' is a list of length n. If $S \models A$ then cs' is a solution of the n queens problem (as, for each ground instance $pqs(n, cs, us, u)$ of A, u is of the form $[t|ds]$ and $1, \ldots, n$ are members of cs; thus cs is a list of distinct members $1, \ldots, n$, so (cs, us, ds) is correct up to n w.r.t. n, hence cs represents a solution of the n queens problem).

While specifying completeness, we are interested in ability of the program to produce all solutions to the problem. This leads to the following specification for completeness:

$$S_{pqs}^{0} = \left\{ pqs(i, cs, us, [t|ds]) \in \mathcal{HB} \; \middle| \; \begin{array}{l} i > 0, \\ (cs, us, ds) \text{ is correct up to } i \text{ w.r.t. } i. \end{array} \right\}.$$

We conclude this section with a property which will be used later on.

Lemma 7
Assume $0 < m \le i$. Consider two conditions

$$(cs, [t|us], ds) \text{ is correct up to } m \text{ w.r.t. } i, \tag{12}$$

$$(cs, us, [t'|ds]) \text{ is correct up to } m \text{ w.r.t. } i+1. \tag{13}$$

For any $t, t' \in \mathcal{HU}$, (12) implies (13). For any $t' \in \mathcal{HU}$, (13) implies $\exists t \in \mathcal{HU}$ (12).

Proof
Assume that cs is a list of distinct members and each $j \in \{1, \ldots, m\}$ is a member of cs. We will consider here the diagonal numbers in cs. Obviously, the up- (down-) diagonal numbers w.r.t. i (of $1, \ldots, m$) are distinct iff the diagonal numbers w.r.t. $i+1$ are.

Let $j \in \{1, \ldots, m\}$. Then l is the down-diagonal number of j w.r.t. i iff $l_1 = l+1$ is the down-diagonal number of j w.r.t. $i+1$. Note that $l > 0$ (as $j \le i$). So for down-diagonals, conditions (11) for i, j, l and ds, and (11) for $i+1$, j, l_1 and $[t'|ds]$ are equivalent.

Number l is the up-diagonal number of j w.r.t. i iff $l_2 = l-1$ is the up-diagonal number of j w.r.t. $i+1$. For $l_2 > 0$ we, similarly as above, obtain that for up-diagonals conditions

$$\text{(11) for } i, j, l \text{ and } [t|us], \tag{14}$$

$$\text{(11) for } i+1, j, l_2 \text{ and } us \tag{15}$$

are equivalent. For $l_2 = 0$, (14) vacuously implies (15), and (15) implies that (14) holds for some t, namely $t = j$. For $l_2 < 0$ the equivalence of (14) and (15) is trivial.

This completes the proof of both implications of the lemma. $\qquad\square$

5 Correctness proof

Following Theorem 1, to prove correctness of program NQUEENS w.r.t. specification S, one has to show that S is a model of each clause of the program. In other words to show, for each ground instance of a clause of the program, that the head is in S provided the body atoms are in S. For the unit clauses of NQUEENS

$$pq(I, [I|\_], [I|\_], [I|\_]).$$
$$pqs(0, \_, \_, \_).$$

it is obvious that each ground instance of the clause is in S. Consider clause (4). For any its ground instance

$$pq(i, [t_1|cs], [t_2|us], [t_3|ds]) \leftarrow pq(i, cs, us, ds),$$

it immediately follows from the definition of S_{pq} that if the body atom is in S (thus in S_{pq}) then its head is in $S_{pq} \subseteq S$.

The nontrivial part of the proof is to show that S is a model of clause (2). Consider its ground instance

$$pqs(s(i), cs, us, [t|ds]) \leftarrow pqs(i, cs, [t_1|us], ds), \ pq(s(i), cs, us, ds).$$

Let H be its head, and B_1, B_2 the body atoms. Assume $B_1, B_2 \in S$. Now (by $B_2 \in S$) $s(i)$ is the l-th member of cs, us, ds (for some $l > 0$). Note that l is the up (down) diagonal number of $s(i)$ w.r.t. $s(i)$ in cs. So condition (11) holds for $s(i)$ w.r.t. $s(i)$.

Consider first the case of $i = 0$. Then (cs, us, ds) is correct up to $s(0)$ w.r.t. $s(0)$, provided that cs is a list of distinct members. Hence $H \in S$.

Consider $i > 0$. Note first that $1, \ldots, s(i)$ are members of cs ($s(i)$ as explained above, and $1, \ldots, i$ by $B_1 \in S$). Assume that cs is a list of distinct members. Then (by $B_1 \in S$) $(cs, [t_1|us], ds')$ is correct up to i w.r.t. i, where ds' is the tail of ds. Hence, by Lemma 7, $\beta = (cs, us, ds)$ is correct up to i w.r.t. $s(i)$. As shown above, (11) holds for $s(i)$ w.r.t. $s(i)$ (where l is both the up- and the down-diagonal number of $s(i)$). Thus, (11) holds for $1, \ldots, s(i)$ w.r.t. $s(i)$. Hence, no up (or down) diagonal number of a $j \in \{1, \ldots, i\}$ is l. As the latter diagonal numbers are distinct (due to β being correct up to i), (10) holds for $1, \ldots, s(i)$.

Hence β is correct up to $s(i)$ w.r.t. $s(i)$. Thus $H \in S$. This completes the proof.

6 Completeness proof

As explained in Section 4, we are interested in completeness of NQUEENS w.r.t. specification S_{pqs}^0. However, the sufficient condition of Lemma 4 does not hold for this specification. Instead let us use

$$S^0 = S_{pq} \cup S_{pqs}^0 \cup \{\, pqs(0, cs, us, ds) \mid cs, us, ds \in \mathcal{HU} \,\}$$

as the specification for completeness.[5]

We first show that each atom from specification S^0 is covered by program NQUEENS. Each atom

$$A = pq(\, i, [c_1, \ldots, c_k, i|c], \ [u_1, \ldots, u_k, i|u], \ [d_1, \ldots, d_k, i|d]\,)$$

from S_{pq} is covered by NQUEENS w.r.t. S^0; for $k = 0$ by clause (3) as A is its instance; for $k > 0$ by clause (4) due to its instance $A \leftarrow pq(\, i, [c_2, \ldots, c_k, i|c], [u_2, \ldots, u_k, i|u],$

---

[5] This is a common phenomenon in mathematics; an inductive proof of a property may be impossible, unless the property is strengthened. Actually, the same happened in the case of correctness. We are interested in correctness of NQUEENS w.r.t. $S_{pqs} \cup \mathcal{HB}_{pq}$. However, $S_{pqs} \cup \mathcal{HB}_{pq}$ is not a model of the program and Theorem 1 is not applicable. Instead, we used a stronger specification $S = S_{pqs} \cup S_{pq}$. Obviously, correctness (completeness) w.r.t. a specification implies correctness (completeness) w.r.t. any its superset (subset).

$[d_2, \ldots, d_k, i|d]$) (as its body atom is in S^0). Also, each atom $pqs(0, cs, us, ds)$ is covered, as it is an instance of clause (1).

The nontrivial part of the proof is to show that each $A \in S^0_{pqs}$ is covered. Consider such atom, it is of the form

$$A = pqs(s(i), cs, us, [t|ds]),$$

where $i \geq 0$ and (cs, us, ds) is correct up to $s(i)$ w.r.t. $s(i)$. So cs is a list of distinct members, and each $j \in \{1, \ldots, s(i)\}$ is a member of cs. Let $s(i)$ be the l-th member of cs. Thus l is the up- and down-diagonal number of $s(i)$ w.r.t. $s(i)$ in cs, and (by Definition 6) $s(i)$ is the l-th member of us and of ds.

We show that A is covered by clause (2) w.r.t. S^0, due to its instance

$$A \leftarrow B_1, B_2. \qquad \text{where } B_1 = pqs(i, cs, [t'|us], ds), \quad B_2 = pq(s(i), cs, us, ds)$$

(and $t' \in \mathcal{HU}$ will be determined later). We have $B_2 \in S^0$ (as $s(i)$ is the l-th member of cs, us and ds). If $i = 0$ then $B_1 \in S^0$, thus A is covered by (2).[6]

Assume $i > 0$. As (cs, us, ds) is correct up to $s(i)$ w.r.t. $s(i)$, it is correct up to i w.r.t. $s(i)$, and by Lemma 7, $(cs, [t'|us], ds')$ is correct up to i w.r.t. i, for some $t' \in \mathcal{HU}$, where ds' is the tail of ds. Hence for such t' we have $B_1 \in S^0_{pqs} \subseteq S^0$, thus A is covered by (2).

This completes the proof that each $A \in S^0$ is covered by NQUEENS w.r.t. S^0. It remains to find a level mapping under which NQUEENS is recurrent. Consider the level mapping defined by

$$\begin{aligned} |\,pqs(i, cs, us, ds)\,| &= |i| + |cs|, \\ |\,pq(i, cs, us, ds)\,| &= |cs|, \end{aligned} \qquad \text{where} \qquad \begin{aligned} |\,[h|t]\,| &= 1 + |t|, \\ |\,s(t)\,| &= 1 + |t|, \\ |f(t_1, \ldots, t_n)| &= 0, \end{aligned}$$

for any ground terms $i, cs, us, ds, h, t, t_1, \ldots, t_n$, and any n-ary function symbol f distinct from s and from $[\,|\,]$ ($n \geq 0$). An easy inspection shows that under this level mapping NQUEENS is recurrent. Hence by Lemma 4, the program is complete w.r.t. S^0.

Additionally, it follows by (Apt 1997, Corollary 6.9) that the program terminates for initial queries $Q_{in,n}$, as each $Q_{in,n}$ is bounded w.r.t. $|\,|$ (Apt 1997, Definition 6.7).

7 Comments

A practical consequence of the correctness, completeness, and termination proven above is as follows. Assume that NQUEENS with query $Q_{in,n}$ is executed by Prolog, however with the occur-check. Then the computation will terminate, and the answers will represent all the solutions to the n queens problem. This holds for arbitrary selection rule. The occur-check is actually not needed (Drabent 2021), so this also holds for Prolog without the occur-check.

Let us now discuss general applicability of the presented approach. Note first that finding an appropriate specification was a crucial step to treat NQUEENS within the standard declarative semantics.

Our construction of the approximate specification exemplifies a general pattern (Drabent 2016; 2018). Some ground atoms may be irrelevant for the program properties

---

[6] Note that in this case A is covered w.r.t. S^0 but not w.r.t. $S^0_{pq} \cup S^0_{pqs}$. This is why we use $S^0 \supset S^0_{pq} \cup S^0_{pqs}$ as a specification.

we are interested in. (In our case, they are atoms $pqs(i, cs, us, ds) \in \mathcal{HB}_{pqs}$, where $i = 0$, or $i \in \mathbb{N}$, $1, \ldots, i$ are members of cs, and cs is not a list of distinct members.) So $\mathcal{HB}$ is divided into a set $S_{irr} \subseteq \mathcal{HB}_{pqs}$ of irrelevant atoms and $S_{rel} = \mathcal{HB} \setminus S_{irr}$, the set of relevant ones. From the latter, a set $S_c \subseteq S_{rel}$ of "correct" ones is chosen, so that correctness w.r.t. specification $S_{irr} \cup S_c$ implies the program properties of interest.

In many cases, S_c describes the ground atoms which the program should compute, so S_c is used as a specification for completeness, and $S_{irr} \cup S_c$ as one for correctness.

A contribution of our example is dealing with the fact that some answers of the program represent expected solutions, but have instances which apparently should be considered incorrect. The problem was overcome by including such instances into S_{irr}. In this way, we do not need to consider nonground answers, instead it is sufficient to consider their ground instances from $\mathcal{HB} \setminus S_{irr}$. This idea should be applicable to other programs posing similar problems.

8 Conclusions

The paper provides an example of precise reasoning about the semantics of a logic program. It presents detailed proofs of correctness and completeness of the n queens program of Frühwirth (1991). The program is short, but may be seen as tricky or nonobvious. The approach is declarative; the specifications and proofs abstract from any operational semantics, the program is treated solely as a set of logical formulae. Note that, in many cases, approaches based on the operational semantics are proposed for reasoning about declarative properties of logic programs (Apt 1997; Bossi and Cocco 1989; Pedreschi and Ruggieri 1999). This seems to introduce unnecessary complications (cf. (Drabent and Miłkowska 2005, Section 3.2)).

The program uses nonground data, like open lists with some elements being variables. Moreover, some of its answers (which represent solutions to the n queens problem) have instances that represent incorrect positioning of the queens. So one may expect that approaches based on the standard semantics and Herbrand interpretations are inapplicable here. Actually, this is not the case. We discuss difficulties with constructing a specification, show how to overcome them, and provide a formal specification based on Herbrand interpretations. Then we prove that the program is correct and complete with respect to the specification. Building the specification is a crucial part of this work.

It may seem that s-semantics (Falaschi *et al.* 1989) is suitable here, as it explicitly deals with non-ground answers. However, the approach employed in this paper seems preferable, as analogical specification and proofs employing the s-semantics (Drabent 2020) turn out to be more complicated.

Our specification is approximate (see Section 2). Constructing an exact specification of the program would be too troublesome and would result in more complicated correctness and completeness proofs. This is quite common in logic programming—one often does not need to know the exact semantics of one's program. Some features of the program are of no interest, for instance they may be irrelevant to its intended usage. So we do not need to describe them.

The paper deals with a single program. In Section 7, we discuss the presented ideas as an instance of a more general approach.

The detailed proofs presented here may be seen as too impractical due to numerous details. This is however usually the case when proving program properties. Experience from imperative and logic programming shows that program correctness really does depend on many details (see e.g. the example proofs in Apt *et al.* (2009) and in the papers mentioned above). Maintaining the details by means of some proof assistant is outside of the scope of this paper. On the other hand, in the author's opinion proofs like those presented here can be informally performed by programmers, possibly in a less detailed way, during actual programming. Two fragments of such informal reasoning, with various levels of precision, are shown in Section 3 (in subsections *Rationale* and *Informal specification*). We expect that formal proof methods, like those discussed here, can teach programmers a systematic way of reasoning about their programs in practice.

Acknowledgement

Comments of anonymous referees and of Michael Maher were instrumental in improving the presentation.

Conflicts of interest

The author declares none.

References

Apt, K. R. 1997. *From Logic Programming to Prolog.* International Series in Computer Science. Prentice-Hall.

Apt, K. R., de Boer, F. S. and Olderog, E.-R. 2009. Modular verification of recursive programs. In *Languages: From Formal to Natural, Essays Dedicated to Nissim Francez.* Lecture Notes in Computer Science, vol. 5533. Springer, 1–21.

Bezem, M. 1993. Strong termination of logic programs. *Journal of Logic Programming 15,* 1&2, 79–97.

Bossi, A. and Cocco, N. 1989. Verifying correctness of logic programs. In *TAPSOFT, Vol. 2*, J. Díaz and F. Orejas, Eds. Lecture Notes in Computer Science, vol. 352. Springer, 96–110.

Clark, K. L. 1979. Predicate logic as computational formalism. Tech. Rep. 79/59, Imperial College, London. December.

Deransart, P. 1993. Proof methods of declarative properties of definite programs. *Theoretical Computer Science 118,* 2, 99–166.

Deransart, P. and Małuszyński, J. 1993. *A Grammatical View of Logic Programming.* The MIT Press.

Drabent, W. 2016. Correctness and completeness of logic programs. *ACM Transactions on Computational Logic 17,* 3, 18:1–18:32.

Drabent, W. 2018. Logic + control: On program construction and verification. *Theory and Practice of Logic Programming 18,* 1, 1–29.

Drabent, W. 2019. On correctness of an n queens program. *CoRR abs/1909.07479.* http://arxiv.org/abs/1909.07479. A preliminary version of the current paper, with extended introductory sections.

Drabent, W. 2020. S-semantics – an example. *CoRR abs/2006.06077.* http://arxiv.org/abs/2006.06077. Presented at LOPSTR 2021.

DRABENT, W. 2021. A note on occur-check. In *Proceedings 37th International Conference on Logic Programming (Technical Communications)*, A. Formisano et al Eds. Electronic Proceedings in Theoretical Computer Science, vol 345. Open Publishing Association, 54–67.

DRABENT, W. AND MIŁKOWSKA, M. 2005. Proving correctness and completeness of normal programs – a declarative approach. *Theory and Practice of Logic Programming 5,* 6, 669–711.

FALASCHI, M., LEVI, G., PALAMIDESSI, C. AND MARTELLI, M. 1989. Declarative modeling of the operational behavior of logic languages. *Theoretical Computer Science 69,* 3, 289–318.

FRÜHWIRTH, T. 1991. `nqueens`. A post in `comp.lang.prolog`. 1991-03-08. Also in (Sterling and Shapiro 1994, Section 4.1, Exercise (v)).

PEDRESCHI, D. AND RUGGIERI, S. 1999. Verification of logic programs. *Journal of Logic Programming 39,* 1–3, 125–176.

SHAPIRO, E. 1983. *Algorithmic Program Debugging.* The MIT Press.

STERLING, L. AND SHAPIRO, E. 1994. *The Art of Prolog*, 2 ed. The MIT Press.