DFKI

Research
Report

93-22

Weak Looking-Ahead and its Application to Computer-Integrated Process Planning

Manfred A. Meyer and Jörg P. Müller

April 1993

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Erwin-Schrödinger-Straße
D-6750 Kaiserslautern, Germany
Tel.: + 49 (631) 302-3211
Fax: + 49 (631) 302-3210

Weak Looking-Ahead and its Application to Computer-Integrated Process Planning

Manfred A. Meyer and **Jörg P. Müller**
German Research Center for Artificial Intelligence (DFKI)
P. O. Box 20 80, D-W-6750 Kaiserslautern, Germany
email: meyer@dfki.uni-kl.de, jpm@dfki.uni-sb.de

Abstract

Constraint logic programming has been shown to be a very useful tool for knowledge representation and problem-solving in different areas. Finite Domain extensions of PROLOG together with efficient consistency techniques such as forward-checking and looking-ahead make it possible to solve many discrete combinatorial problems within a short development time. In this paper we present the weak looking-ahead strategy (WLA), a new consistency technique on finite domains combining the computational efficiency of forward-checking with the pruning power of looking-ahead. Moreover, incorporating weak looking-ahead into PROLOG's SLD resolution gives a sound and complete inference rule whereas standard looking-ahead itself has been shown to be incomplete. Finally, we will show how to use weak looking-ahead in a real-world application to obtain an early search-space pruning while avoiding the control overhead involved by standard looking-ahead.

This paper will also be published by Gordon and Breach Publishers in the Proceedings of the *Sixth International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE-93)*, Edinburgh, Scotland, June 1^{st}–4^{th}, 1993.

Contents

1 Introduction **3**

2 Finite Domain Consistency Techniques **3**

2.1 Forward-Checking - The Principle 4

2.2 Forward-Checking - Properties and Drawbacks 4

2.3 Looking-Ahead - The Principle 5

2.4 Looking-Ahead - Properties and Drawbacks 5

3 The Theoretical Background of WLA **6**

4 Using Weak Looking-Ahead for Tool-Selection **7**

4.1 The Problem Domain 8

4.2 The Lathe-Tool Selection Problem 8

4.3 The Variables 9

4.4 The Constraints 10

4.5 A FiDo Program for Tool Selection 11

4.6 Trace and Assessment of Program Execution 12

5 Conclusion **15**

1 Introduction

Many problems in different areas such as Operations Research, Hardware Design, and Artificial Intelligence applications can be regarded as constraint satisfaction problems (CSPs). Logic programming offers a convenient way of *representing* CSPs due to its relational, declarative and nondeterministic form. Unfortunately, standard logic programming languages such as PROLOG tend to be inefficient for *solving* CSPs, since what could be called constraints in PROLOG is used only in a passive *a posteriori* manner, leading to symptoms such as late recognition of failure, unnecessary and unintelligent backtracking and multiple computation of the same solutions.

There have been intensive research efforts in order to remedy this. One of them, which has caught increasing attention over the past few years, is the Constraint Logic Programming approach: By integrating a domain concept for logic variables and consistency techniques such as forward-checking or looking-ahead into PROLOG, the search space can be restricted in an *a priori* manner. Thus, a more efficient control strategy can be achieved, preserving the 'clean' dual PROLOG semantics.

When using these consistency techniques to implement real-world applications, it turned out that forward-checking and looking-ahead as provided in most finite-domain PROLOG extensions are not totally satisfactory: forward-checking itself often does not give any pruning at all until variables become singletons, whereas standard looking-ahead forces strong pruning of the search-space but induces serious control overhead.

In this paper we present a consistency technique on finite domains, which combines the efficiency of forward-checking with the pruning power of standard looking-ahead: The basic idea of this **weak looking-ahead (WLA)** strategy is to apply looking-ahead only once to a constraint and to use forward-checking for further restricting the domains of its arguments.

What makes this paper more than "just another paper about just another consistency technique" is the way weak looking-ahead came into being, which stood in a close relation to a real-life application: In the ARC-TEC project at DFKI we have been developing a knowledge-based system (μCAD2NC, [Boley *et al.*, 1991]) generating workplans for lathe CNC machines. It transforms CAD-like geometries of rotation-symmetric workpieces into abstract NC programs using declarative term representations for all processing steps. After we decided to solve the subproblem of selecting appropriate lathe tools for the various processing steps by using constraints, we experimented with several consistency algorithms. Soon we realized that on one hand, forward-checking is too weak for some applications where an earlier pruning of the search space is desired. On the other hand, using looking-ahead for this application is a bit like breaking the butterfly on the wheel. These observations entailed the wish for a new technique which causes only little more cost than forward-checking, but which can achieve much better pruning results in many cases.

2 Finite Domain Consistency Techniques

Over the past years, increasing attention has been paid to using constraints in logic programming [Jaffar *et al.*, 1986; Jaffar and Lassez, 1987; Jaffar and Michaylov, 1987; van Hentenryck, 1989] for it presents a very powerful incorporation of the advantages both of logic

programming (declarativity, relational form, nondeterminism) and consistency techniques for constraint solving problems. By using consistency techniques it is possible to overcome the basic shortcomings of logic programming languages, which are mainly caused by their poor, mostly backtracking-like control strategies. Techniques such as forward-checking and looking-ahead are used to restrict the domains of variables in an active manner and to achieve an *a priori* pruning of the search space.

In this section, we will give a short informal description of both forward-checking and looking-ahead according to [van Hentenryck, 1989]). In the next section, we present the weak looking-ahead method which is essentially based on these techniques.

2.1 Forward-Checking - The Principle

The idea of forward-checking is formally expressed by the forward-checking inference rule (FCIR). Informally, a constraint $\mathcal{C}$ can be used in a forward-checking manner as soon as all except one of its domain-variable arguments, say X, are instantiated to a ground value. Then, $\mathcal{C}$ is called *forward-checkable*. $\mathcal{C}$ can be considered a unary predicate $\mathcal{C}'(X)$, and the set of possible values that can be given to X can be restricted to those elements a satisfying $\mathcal{C}'(a)$.

For example, let $\mathcal{C}(X, Y, Z)$ be X `\=` $Y + Z$, with $X = 4$, $Z = 1$ and Y ranging over $\{1, 2, 3\}$. Then $\mathcal{C}'(V) \equiv V$ `\=` 3. Thus, the domain of Y can be restricted to the set $\{1, 2\}$. Furthermore, if the domain of a variable becomes singleton, the variable is instantiated to the singleton value. Thus, other constraints can become forward-checkable, keeping constraint propagation going on.

2.2 Forward-Checking - Properties and Drawbacks

Forward-checking has turned out to be one of the most popular consistency techniques for several reasons:

- Forward-checking is a technique which can be easily implemented. For example, [de Schreye *et al.*, 1990; Müller, 1991] use PROLOG systems with coroutining facilities to implement it, whereas [Hein, 1992; Stein, 1992] show how forward-checking can be integrated into PROLOG by adding a set of new WAM instructions.
- It yields reasonable pruning results for many applications, keeping the computational costs fairly low.
- There exist sound and complete proof procedures based on normal SLD resolution combined with forward-checking (see [van Hentenryck, 1989]).

The main drawback of forward-checking is its strong applicability precondition: A predicate can be executed by the FCIR only if all except one of its variables are instantiated to a ground value. Thus:

- For predicates with many arguments and/or many variables, at a given point of computation, there is only a relatively small probability that forward-checking can be applied to them.

- Especially when computation starts, it is very often the case that no constraint is forward-checkable. That means that choices have to be made, i.e. variables are instantiated in a more or less random manner. Thus, the devil of backtracking which we would like to exorcize by the use of consistency techniques, returns through the back door.

- Some constraints, such as $=, >, <$ should not be executed by forward-checking at all, because they embody a great deal of structural information about the relation between their arguments[1].

2.3 Looking-Ahead - The Principle

Looking-Ahead [van Hentenryck, 1987a; van Hentenryck, 1987b; Mackworth, 1977; Lauriere, 1978] offers a powerful possibility to reduce the number of values that can be assigned to variables of a constraint, even if this constraint is not yet forward-checkable.

For every domain variable X appearing as an argument of an N-ary constraint $\mathcal{C}$, and for every value within the domain of X, it must be checked whether there exists at least one admissible value from the domain of each domain variable Y appearing in $\mathcal{C}$ so that the constraint $\mathcal{C}$ is satisfied. The arguments of $\mathcal{C}$ which are no domain variables must be ground.

For example, let $\mathcal{C}(X, Y, Z)$ be $X > Y + Z$, where X, Y, and Z range over $\{1, 2, 3, 4\}$. Using looking-ahead, the domains can be immediately restricted to $X = \{3, 4\}, Y = Z = \{1, 2\}$. Note that if we used forward-checking instead, no pruning at all would be achieved since no forward-condition would be fulfilled. From now on, each time a value is removed from one of the domains, looking-ahead has to be repeated.

2.4 Looking-Ahead - Properties and Drawbacks

By using looking-ahead, the search space can be pruned at an early stage of computation. However, the trouble with standard looking-ahead is that it is a very expensive method of ensuring arc-consistency. Therefore, for most applications it is considered inappropriate [de Schreye *et al.*, 1990; Dechter, 1989]. Nevertheless, it would be a shame to forgo all the benefits brought about by the strong pruning capabilities of the Looking-Ahead Inference Rule (LAIR). In the next section, we present *weak looking-ahead*, which can be regarded a compromise between forward-checking and looking-ahead. Let us assume that, in our above looking-ahead example, we would perform the first looking-ahead step as shown, but after that, we would *not* do any more looking-ahead, but instead solve the (now simplified) problem by normal resolution or by forward-checking. This procedure expresses the main idea of the weak looking-ahead strategy which we will point out in more detail in the following.

[1] For example, the information that two variables X and Y are equal should not only be used if X or Y are ground. Rather, the equality constraint should be maintained from the moment it has been stated (see [Müller, 1991]).

3 The Theoretical Background of WLA

The basic theoretic work in the area of using consistency techniques in logic programming has been done by van Hentenryck [van Hentenryck, 1989]. This research has contributed a great deal to both forming a solid framework and preserving the logic part of the programming languages developed while achieving a much better control behaviour than that achieved by standard logic programming languages such as PROLOG. This is an aspect of crucial significance, because, to quote Jaffar and Michaylov [Jaffar *et al.*, 1990]: *"Forsaking the logic in PROLOG in order to remove some limitations of the language is like throwing out the baby with the bath water."*

In this section, we will give a formal definition of the weak looking-ahead inference rule, and we will present the basic formal properties such as soundness and completeness of the proof procedure defined on top of WLA. The terminology we use and the sense we use it are basically the same as in [van Hentenryck, 1989].

The weak looking-ahead strategy combines the use of LAIR and FCIR. A similar technique has been informally proposed in [de Schreye *et al.*, 1990] as "first-order looking-ahead". We present a generalized technique we call weak looking-ahead. This name seems more appropriate for expressing what the underlying algorithm really does. The basic idea of WLA is that each constraint can be selected by the looking-ahead part *not more than once*, and that this should happen at an appropriate time. After this, only the FCIR (or normal inference) can be applied to it. This idea is covered by the following definitions.

Definition 1 *An atom $p(t_1, ..., t_n)$ is called WLA-checkable if p is a constraint and*

- *$p(t_1, ..., t_n)$ is lookahead-checkable and has not yet been selected by the WLA, or*
- *$p(t_1, ..., t_n)$ is forward-checkable and has already been selected by the WLA.*

Definition 2 (WLA) *Let P be a program, $G_i = \ ?{-}A_1, ..., A_k, ..., A_m$ a goal and σ_{i+1} a substitution. G_{i+1} is derived by the WLA along with σ_{i+1} from G_i and P if A_k is WLA-checkable with $x_1, ..., x_n$ being the WLA variables in A_k and the following holds:*

- *If A_k is lookahead-checkable and the WLA has not been applied to A_k in the actual proof, then:*

  For each x_j, the new domain e_j is $e_j = \{v_j{\in}d_j \mid \exists v_1{\in}d_1, ..., v_{j-1}{\in}d_{j-1}, v_{j+1}{\in}d_{j+1}, ..., v_n{\in}d_n$ such that $\sigma(A_k)$ with $\sigma = \{x_1{\leftarrow}v_1, ..., x_n{\leftarrow}v_n\}$ is a logical consequence of $P\}$.

  For each x_j, if e_j has become a singleton, i.e. $e_j = \{c\}$, then the new value y_j is the constant c, otherwise a new variable ranging over e_j. σ_{i+1} then is defined as $\sigma_{i+1} = \{x_1 \leftarrow y_1, ..., x_n \leftarrow y_n\}$.

  G_{i+1} is either ?- $\sigma_{i+1}(A_1, ..., A_{k-1}, A_{k+1}, ..., A_m)$, if at most one y_j is a domain variable, or G_{i+1} is ?-$\sigma_{i+1}(A_1, ..., A_m)$, otherwise.

- *If A_k is forward-checkable, then:*

  Let x_d be the forward variable inside A_k. Then, the new domain e is defined as $e = \{a \in d \mid P \models A_k\{x_d \leftarrow a\}\} \neq \emptyset$.

σ_{i+1} *is defined as* $\sigma_{i+1} = \{x_d \leftarrow c\}$, *if* $e = \{c\}$, *i.e.* e *has become a singleton. Otherwise,* $\sigma_{i+1} = \{x_d \leftarrow y_e\}$, *where* y_e *is a new domain variable ranging over* e.

G_{i+1} = ?- $\sigma_{i+1}(A_1, ..., A_{k-1}, A_{k+1}, ..., A_m)$.

Properties of Weak Looking-Ahead The main point of the above definition is point 6, which uses SLDFC resolution (SLD resolution with forward-checking, cf. [van Hentenryck, 1989]), whose soundness and completeness have been proved, in order to finish the proof after some prepruning has been done by using the LAIR in a definite way. Thus, if we want to prove soundness and completeness of the WLA, we basically have to check the LAIR part. Since this part gets involved not more than once for each goal, and since this happens as early as possible (due to point 2 of the definition), the disadvantages of the LAIR such as its incompleteness and the high computational overhead can be avoided.

Proposition 1 (Soundness of WLA) *Let* P *be a program, and let* G_i *be the goal* ?- $A_1, ..., A_k, ..., A_m$, *and* A_k *be WLA-checkable. Let the goal* G_{i+1} *be derived by WLA along with* σ_{i+1} *from* G_i *and* P *as* G_{i+1} = ?-$\sigma_{i+1}(A_1, ..., A_{k-1}, A_{k+1}, ..., A_m)$. G_i *is a logical consequence of* P *iff* G_{i+1} *is a logical consequence of* P.

The next result concerns the completeness of WLA. This means that we can define a complete proof procedure using weak looking-ahead.

Definition 3 (SLDW-resolution) *A first-order resolution proof procedure is called SLDW-resolution, if it uses weak looking-ahead for WLA-checkable goals and normal SLDD-derivation (SLD-derivation with domain variables) for other goals.*

We can prove the completeness of such a proof procedure by making use of the completeness of the FCIR, showing that by applying the LAIR not more than once to each goal, no solutions are lost. The completeness result is expressed by the following proposition:

Proposition 2 (Completeness of WLA) P *be a logic program,* G *be a goal. If there exists an SLDD-refutation of* $P \cup \{G\}$, *then there also exists an SLDW-refutation of* $P \cup \{G\}$. *Moreover, if* σ *is the answer substitution from the SLDD-refutation of* $P \cup \{G\}$, *and* ρ *is the answer substitution from the SLDW-refutation of* $P \cup \{G\}$, *then* $\rho \leq \sigma$.

For the proofs of the propositions 1 and 2 we refer to [Müller, 1991]. In the next section we will demonstrate weak looking-ahead by an example from our application domain.

4 Using Weak Looking-Ahead for Tool-Selection

In this section we will show the usability of the weak looking-ahead inference rule by an example from the concrete domain of the ARC-TEC project at DFKI that constitutes an AI approach towards implementing the idea of computer-integrated manufacturing (CIM). Along with conceptual solutions, it provides a continuous sequence of software tools for the **A**cquisition, **R**epresentation, and **C**ompilation of **TEC**hnical knowledge (cf. [Bernardi *et al.*, 1991]). It combines the KADS knowledge-acquisition methodology [Wielinga *et al.*,

1992], the KL-ONE representation theory [Brachman and Schmolze, 1985], and WAM compilation [Hein and Meyer, 1992] and constraint-handling technologies [Meyer *et al.*, 1992]. For its evaluation, an expert system for production planning has been developed.

4.1 The Problem Domain

The input to the production planning system is a very low-level description of a rotational-symmetric workpiece as it comes from a CAD system. Geometrical description of the workpiece's surfaces and topological neighborhood relations are the central parts of this representation. If possible at all, production planning with these data starting from (nearly) first principles would require very complex algorithms. Thus, planning strategies on such a detailed level are neither available nor do they make sense. Instead human planners [Schmalhofer *et al.*, 1991] have a library of skeletal plans in their minds. Each of these plans is associated with a more or less abstract description of a (part of a) workpiece, which are called workpiece features [Klauck *et al.*, 1991]. Such a feature is defined by its association to a corresponding manufacturing method.

The generation of an abstract feature description of the workpiece is the first step of the production planning process. The obtained features characterize the workpiece with respect to its production. In a second step the skeletal plans (associated to the features) are retrieved and merged resulting in an abstract NC program, which is then transformed into code for the concrete CNC machine.

4.2 The Lathe-Tool Selection Problem

The application problem we are dealing with for the rest of this paper will be to find appropriate lathe tools to manufacture the workpiece. According to the shape, the material and other attributes of the lathe part to be manufactured, the workplan consists of a number of different steps. A typical workplan may provide one step for roughing, another step for finishing and a third (facultative) step for doing the fine finishing of the lathe part. However, a workplan can be much more complicated. For each processing step, appropriate tools have to be chosen.

This tool selection heavily depends on a lot of geometrical (e.g. the edge-angle) as well as technological parameters (e.g. material, process etc.). Moreover, the tool system itself consists of subparts that have to be combined, e.g. the tool holder, the material of the plate and its geometry. In practice, there are a lot of restrictions, 'which holder to use for which plate', 'which kind of plate geometry to use for which workpiece contour' and so on. Figure 1 shows a typical lathe workpiece together with the selected tools for the different manufacturing features and lathe-turning steps.

The lathe-tool selection problem can naturally be formulated as a constraint satisfaction problem (CSP). To keep things simple, we may assume that a lathe tool consists of two basic parts: the *cutting plate*, which actually cuts the material, and the *tool holder*, which serves to hold the cutting plates. We can exchange either the cutting plate only or both plate and holder. In our application, we are now concerned with finding a well-suited tool—or rather: a number of well-suited tools—starting from a set of constraints which describe the actual problem, i.e. information about the process to be performed, about the lathe part

Figure 1: An example workpiece with its selected lathe tools

to be processed, and internal information about the compatibility of holders and cutting-plates as well as about holder and plate geometries. Tool selection will then result in a set of possible holder/tool combinations for each skeletal plan or manufacturing feature. Using this information, the planning layer can finally perform the optimizations necessary to obtain a (sub)optimal workplan.

4.3 The Variables

When formalizing the tool selection problem as a CSP, the first thing we have to do is to restrict the number of input parameters, which crucially determines the complexity of the problem, since each parameter corresponds to a variable in the constraint net. For our small example we will use the following variables:

- `Holder`: This variable denotes the tool holder. In the beginning, it ranges over the domain of all holders. During constraint propagation, it will be restricted to the set of holders which can currently be chosen.
- `Plate`: This variable denotes the cutting plate to be chosen. Analogously to the holder variable, it ranges over the set of all cutting plates and will be restricted subsequently.
- `Process`: This variable corresponds to the actual kind of processing.
- `WP-material`: This variable contains the material of the lathe workpiece.
- `Beta-Max`: This variable denotes the maximal angle β appearing within the range of one feature of the workpiece.[2]

[2] In general, each of these features corresponds to a single working process.

- `Edge-Angle`: This variable embodies the most important geometrical attribute of a cutting-plate, its edge-angle ε.

- `TC-Edge-Angle`: The tool cutting edge-angle χ is a geometrical characteristic of the tool holder. It denotes the angle between the horizontal cutting direction and the marginal cutting axis of the holder.

Figure 2 gives a better understanding of the geometrical items introduced above.

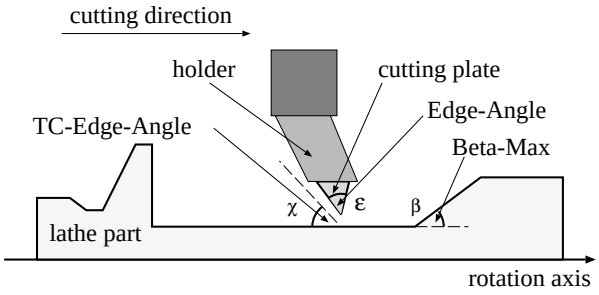

Figure 2: The Angle Constraint

4.4 The Constraints

Having identified the problem variables, the constraints can be put on the variables. In the following, we will consider only the most important constraints:

- `holder_tcea(Holder, TC-Edge-Angle)`: This constraint describes the functional relation between a holder and its tool-cutting edge-angle. It is represented as a primitive or database constraint by enumerating all the possible combinations.

- `plate_ea(Plate, Edge-Angle)`: This constraint is a database-constraint, too. It denotes the fact that each plate has its own edge-angle[3].

- `compatible(Holder, Plate)`: This constraint expresses the compatibility condition between tool holders and cutting plates.

- `hard_enough(Plate, WP-Material)`: For materials with different degrees of hardness, different cutting-plates have to be used. Processing hardened steel, e.g., may require ceramic or even diamond cutting plates, whereas aluminum can be cut with other, cheaper plates. Note, however, that hardness is just one of many attributes of a material which are important in order to choose the right cutting plate.

- `process_holder(Process, Holder)`: For the different steps of processing, different types of holders are appropriate.

[3]Of course, we could have implemented the plate as a more complex data structure containing its edge-angle as an attribute. For the sake of uniformity, we implemented it as a constraint, just as we did with the `holder_tcea` constraint.