On Guarding Nested Fixpoints

Helmut Seidl Andreas Neumann

FB IV – Informatik, Universität Trier, D–54286 Trier, Germany
{seidl,neumann}@uni-trier.de

Abstract. For every hierarchical system of equations S over some complete and distributive lattice we construct an equivalent system with the same set of variables which additionally is *guarded*. The price to be paid is that the resulting right-hand sides may grow exponentially. We therefore present methods how the exponential blow-up can be avoided. Especially, the loop structure of the variable dependence graph is taken into account. Also we prove that size $\mathcal{O}(m \cdot |S|)$ suffices whenever S originates from a fixpoint expression where the nesting-depth of fixpoints is at most m. Finally, we sketch an application to regular tree pattern-matching.

Keywords: guardedness, μ-calculus, distributive lattices, loop-connectedness.

1 Introduction

Since Kozen's seminal paper [13] in 1983, the modal μ–calculus has been widely used for specification and verification of properties of concurrent processes. Fixpoint expressions or (slightly more convenient) *hierarchical* systems of equations, however, are considered to be difficult to understand – especially in presence of deep nesting of alternating fixpoints. Therefore, various kinds of normal forms have been suggested in order to ease both theoretical and practical manipulations. One useful additional property of fixpoint expressions (or hierarchical systems of equations) is *guardedness*.

A variable x is *guarded* in the expression e if x occurs only nested inside some application, i.e., as $f(\ldots x \ldots)$ for some operator f. A hierarchical system of equations is called guarded if it does not contain a cyclic variable dependence through unguarded variable occurrences only. Hierarchical systems of *Boolean* equations only use "$\sqcup$" (least upper bound) and "$\sqcap$" (greatest lower bound) in right-hand sides and thus no operators at all. Therefore, *all* occurrences of variables in right-hand sides are unguarded. Consequently, finding equivalent guarded systems means removing cyclic variable dependences completely. Such acyclic systems can be solved in polynomial (even linear) time. Therefore, finding equivalent guarded systems in general cannot be easier than computing solutions of hierarchical systems of Boolean equations.

For hierarchical systems of equations over more complicated complete lattices and with non-empty sets of operators, a guardedness transformation need not necessarily break *all* cyclic variable dependences. It does, however, eliminate

"useless" fixpoint iterations, namely those which can be removed without touching operator applications. This preprocessing has been used for simplifying proofs about the μ-calculus [20], in automata constructions [10, 16] or constructions of direct proofs of satisfiability [9, 11, 17].
Notions related to guardedness have been considered in various contexts. In recursive process definitions, guardedness is commonly assumed [1]. Guardedness plays an important role in universal algebra for polynomial equations to have *unique* solutions [5]. Uniqueness of solutions is also central when equations are to be solved over metric spaces [2] (see also section 7).
It is well-known that hierarchical systems of equations can always be transformed into equivalent guarded ones (see, e.g., [20, 11]) – if not even part of the "folklore". Here, however, we are interested in designing *efficient* transformation techniques which minimize the encountered overhead. We start our considerations by separating the transformation into two stages (section 3). The first stage performs a (appropriately generalized) control-flow analysis to determine which subexpressions may arrive at which unguarded variables. In the second phase then the actual transformation is performed. What is important here is that each phase operates on the original system – simultaneously on all levels of fixpoints. By this trick, we avoid a potential explosion in size through repeatedly feeding partially transformed systems (possibly of increased sizes) into the same algorithm (e.g., as in [20, 11]). While the number of variables of the system produced by the two-stage transformation has not increased, sizes of right-hand sides may have increased exponentially. In order to reduce this extra space, we take into account, *how* the new right-hand sides are constructed through fixpoint iteration. For arbitrary hierarchical systems, we obtain a new upper bound which is related to structural properties of the variable dependence graph (section 5). In the worst case, the blow-up in size of the system still can only be bounded to be exponential in the *alternation-depth* of the original system. Therefore we exhibit useful special classes where just a small polynomial increase suffices – independent of the alternation-depth (section 6). In case of equations over languages of finite trees, we finally show how guardedness transformations can be used to *replace* greatest fixpoints by least ones and thus to remove alternation of fixpoints altogether. This observation can be exploited for compiling powerful tree patterns to finite tree automata (section 7).

2 Hierarchical Systems of Equations

Instead of formally introducing fixpoint expressions, let us immediately consider the slightly more flexible concept of *hierarchical systems of equations*. Assume we are given a complete lattice $\mathbb{D}$ which, as such, is equipped with the binary operations "$\sqcup$" (least upper bound) and "$\sqcap$" (greatest lower bound). Let Σ denote a set of *further* operator symbols where each $f \in \Sigma$ denotes a monotonic function $[\![f]\!] : \mathbb{D}^k \to \mathbb{D}$ for some $k \geq 1$. Operator symbols from Σ denote "real" operations whose applications will not be touched by our transformations.
As right-hand sides of equations we allow expressions built up from formal variables from some set $\mathcal{Z}$ and constants by application of operators from Σ together

with "$\sqcup$" and "$\sqcap$". The set of all these expressions is denoted by $\mathcal{E}_{\Sigma,\mathbb{D}}(\mathcal{Z})$. Every expression $e \in \mathcal{E}_{\Sigma,\mathbb{D}}(\mathcal{Z})$ denotes a function $[\![e]\!] : (\mathcal{Z} \to \mathbb{D}) \to \mathbb{D}$. This function is monotonic, since "$\sqcup$", "$\sqcap$" and all operators from Σ are monotonic.
A *hierarchical* system of equations with free variables from $\mathcal{F}$ is a pair $(S, \mathcal{H})$. S is the finite basic set of equations $z = e_z, z \in \mathcal{Z}$, where for every z, e_z is an expression in $\mathcal{E}_{\Sigma,\mathbb{D}}(\mathcal{Z} \cup \mathcal{F})$, and $\mathcal{H}$ is a *hierarchy* on $\mathcal{Z}$. A hierarchy consists of a sequence $\mathcal{H} = (\langle \mathcal{Z}_r, \lambda_r \rangle, \ldots, \langle \mathcal{Z}_1, \lambda_1 \rangle)$ of mutually disjoint sets $\mathcal{Z}_k$ where $\mathcal{Z} = \mathcal{Z}_r \cup \ldots \cup \mathcal{Z}_1$ together with qualifications $\lambda_k \in \{\mu, \nu\}$. $\mathcal{Z}_k$ is also called the k-th *block* of variables, whereas length r of the hierarchy is called the *alternation-depth* of S. Intuitively, hierarchy $\mathcal{H}$ on S describes the nesting of scopes of variables within which fixpoint iteration is performed: iteration on variables from the same block is performed jointly whereas iteration on variables from block $\mathcal{Z}_i$ should be thought of as nested inside the iteration on variables from $\mathcal{Z}_j, i < j$.

Example 1. Assume we are given the fixpoint expression

$$\mu x_1.a \sqcup (\mu x_2.f\,(x_1, x_2) \sqcup (x_1 \sqcap (\nu x_3.g\,x_3 \sqcup (x_2 \sqcap x_3))))$$

A representation of this expression by a hierarchical system is obtained by introducing an extra equation for each fixpoint subexpression. For our example this gives a set S consisting of:

$$x_1 = a \sqcup x_2 \qquad\qquad x_3 = g\,x_3 \sqcup (x_2 \sqcap x_3)$$
$$x_2 = f\,(x_1, x_2) \sqcup (x_1 \sqcap x_3)$$

Hierarchy $\mathcal{H}$ is obtained by dividing the set of fixpoint variables into blocks of fixpoints of the same kind for which fixpoint iteration can be performed jointly. For our example expression, we choose $\mathcal{H} = (\langle \mathcal{Z}_2, \mu \rangle, \langle \mathcal{Z}_1, \nu \rangle)$ where $\mathcal{Z}_1 = \{x_3\}$ and $\mathcal{Z}_2 = \{x_1, x_2\}$. □

Usually, if $\mathcal{H}$ is understood, we write S for the hierarchical system.
Fix some $1 \leq k \leq r$, and let $\mathcal{Z}^{(k)} = \mathcal{Z}_k \cup \ldots \cup \mathcal{Z}_1$. Then the k–th *subsystem* S_k of S is given by the set of equations $z = e_z, z \in \mathcal{Z}^{(k)}$, together with hierarchy $\mathcal{H}_k = (\langle \mathcal{Z}_k, \lambda_k \rangle, \ldots, \langle \mathcal{Z}_1, \lambda_1 \rangle)$. Note that the free variables of S_k are contained in $\mathcal{F} \cup \mathcal{Z}_r \cup \ldots \cup \mathcal{Z}_{k+1}$.
An *environment* ρ for S is a mapping $\rho : \mathcal{F} \to \mathbb{D}$. The *semantics* $[\![S]\!]\,\rho$ of S in $\mathbb{D}$ relative to environment ρ is a mapping $\mathcal{Z} \to \mathbb{D}$ defined by induction on the alternation-depth r. For $r \geq 1$, consider the monotonic function $G : (\mathcal{Z}_r \to \mathbb{D}) \to \mathcal{Z}_r \to \mathbb{D}$ given by $G\,\sigma\,z = [\![e_z]\!]\,(\rho + \sigma + [\![S_{r-1}]\!]\,(\rho + \sigma))$ where S_{r-1} is empty in case $r = 1$. Note that we use the "+"–operator to combine two functions with disjoint domains into one. In case $\mathcal{Z}_r$ is qualified as μ, let $\bar{\sigma}$ denote the least fixpoint of G. Otherwise, let $\bar{\sigma}$ denote the greatest fixpoint of G. Then $[\![S]\!]\,\rho$ is defined by $[\![S]\!]\,\rho\,z = \bar{\sigma}\,z$ if $z \in \mathcal{Z}_r$ and $[\![S]\!]\,\rho\,z = [\![S_{r-1}]\!]\,(\rho + \bar{\sigma})\,z$ otherwise.
The set $\mathcal{U}[\![e]\!]$ of *unguarded* variables occurring in e is inductively defined by:

$$\begin{array}{lll}
\mathcal{U}[\![d]\!] & = \emptyset & (d \in \mathbb{D}) \\
\mathcal{U}[\![z]\!] & = \{z\} & (z \text{ a variable}) \\
\mathcal{U}[\![f\,(e_1, \ldots, e_k)]\!] & = \emptyset & (f \in \Sigma) \\
\mathcal{U}[\![e_1 \sqcup e_2]\!] & = \mathcal{U}[\![e_1]\!] \cup \mathcal{U}[\![e_2]\!] & \\
\mathcal{U}[\![e_1 \sqcap e_2]\!] & = \mathcal{U}[\![e_1]\!] \cup \mathcal{U}[\![e_2]\!] &
\end{array}$$

Sequence $z_1, \ldots, z_m$ of variables of S is called *unguarded cycle* if z_m occurs

unguarded in e_{z_1} and likewise, for $j = 2, \ldots, m$, z_{j-1} occurs unguarded in e_{z_j}. System S is *guarded* iff S does not contain unguarded cycles.

Example 2. Consider the hierarchical system of equations from ex. 1. It contains, e.g., the unguarded cycle x_2, x_3. □

When analyzing guardedness of hierarchical systems, we find it useful to assume w.l.o.g. that right-hand sides e of variables are of one of the following forms:

1. e is a Boolean expression, i.e., in $\mathcal{E}_{\emptyset,\{\bot,\top\}}[\mathcal{Z} \cup \mathcal{F}]$; or
2. e is an operator application $f\,(e_1, \ldots, e_k)$, $k \geq 1$, or a constant from $\mathbb{D}$.

This special form can always be achieved, possibly by introduction of extra auxiliary variables for constants and operator applications.

Example 3. Consider our hierarchical system from ex. 1. We introduce extra variables y_1, y_2, y_3 for expressions a, $f\,(x_1, x_2)$ and $g\,x_3$, respectively, and obtain the equations:

$$x_1 = y_1 \sqcup x_2 \qquad y_1 = a \qquad x_3 = y_3 \sqcup (x_2 \sqcap x_3) \qquad y_3 = g\,x_3$$
$$x_2 = y_2 \sqcup (x_1 \sqcap x_3) \qquad y_2 = f\,(x_1, x_2)$$

The new hierarchy is obtained by adding the new variables to the corresponding blocks: $(\langle \{x_1, x_2, y_1, y_2\}, \mu\rangle, \langle \{x_3, y_3\}, \nu\rangle)$. □

3 The Basic Transformation

A lattice $\mathbb{D}$ is called *distributive* iff it has a least element $\bot$, a greatest element $\top$ and the equations $a \sqcap (b \sqcup c) = (a \sqcap b) \sqcup (a \sqcap c)$ and $a \sqcup (b \sqcap c) = (a \sqcup b) \sqcap (a \sqcup c)$ hold for all $a, b, c \in \mathbb{D}$. Let $\mathbb{B} = \{\bot \sqsubset \top\}$ denote the Boolean lattice, and for (finite) set $\mathcal{Y}$, $\mathbb{B}[\mathcal{Y}]$ denote the complete lattice consisting of all monotonic functions $(\mathcal{Y} \to \mathbb{B}) \to \mathbb{B}$. Facts 1 and 2 are well-known:

Fact 1 *Every element $\phi \in \mathbb{B}[\mathcal{Y}]$, can be uniquely represented by its minimal disjunctive normal form, i.e., $\phi = m_1 \sqcup \ldots \sqcup m_k$ where $m_i = \bigsqcap_{y \in Y_i} y$ for pairwise incomparable subsets $Y_i \subseteq \mathcal{Y}$.* □

Fact 2 *Every mapping $\rho : \mathcal{Y} \to \mathbb{D}$, $\mathbb{D}$ a distributive lattice, can be uniquely extended to a mapping $\rho^* : \mathbb{B}[\mathcal{Y}] \to \mathbb{D}$ with the following properties:*

- $\rho^*\, y = \rho\, y$ *for every* $y \in \mathcal{Y}$;
- $\rho^*\, \bot = \bot$ *and* $\rho^*\, \top = \top$;
- $\rho^*\,(\phi_1 \sqcup \phi_2) = \rho^*\, \phi_1 \sqcup \rho^*\, \phi_2$;
- $\rho^*\,(\phi_1 \sqcap \phi_2) = \rho^*\, \phi_1 \sqcap \rho^*\, \phi_2$. □

A mapping with the properties listed in fact 2 is also called *morphism* (between distributive lattices). Fact 2 states that $\mathbb{B}[\mathcal{Y}]$ is the *free* distributive lattice (generated from $\mathcal{Y}$). Because of facts 1 and 2, we no longer distinguish between elements in $\mathbb{B}[\mathcal{Y}]$ and expressions built up from constants $\bot, \top$ and variables in $\mathcal{Y}$ by means of applications of $\sqcup$ and $\sqcap$. We obtain:

Proposition 3. *Assume S is a hierarchical system of equations $x = e_x$, $x \in \mathcal{X}$, over distributive lattice $\mathbb{D}$ with free variables from $\mathcal{F}$ where each right-hand side e_x is contained in $\mathbb{B}[\mathcal{X} \cup \mathcal{F}]$. Then for every environment $\rho : \mathcal{F} \to \mathbb{D}$,*

$$\rho^*\,([\![S]\!]\,\emptyset\,x) = [\![S]\!]\,\rho\,x \qquad \text{for all } x \in \mathcal{X}.$$ □

Here, the semantics of S on the left-hand side in the equation is computed over $\mathbb{B}[\mathcal{F}]$ w.r.t. the empty environment whereas on the right-hand side it is computed over $\mathbb{D}$ where the values for the free variables are taken from ρ.
Assume S is a hierarchical system of equations $z = e_z, z \in \mathcal{Z}$, of alternation depth r where the hierarchy of S is given by $\mathcal{H} = (\langle \mathcal{Z}_r, \lambda_r \rangle, \ldots, \langle \mathcal{Z}_1, \lambda_1 \rangle)$. Let $\mathcal{X}$ and $\mathcal{Y}$ denote the sets of variables with Boolean expressions as right-hand sides, and of variables with constants or operator applications as right-hand sides, respectively, and $\mathcal{Z}_k \cap \mathcal{X} = \mathcal{X}_k$. Moreover, let $\bar{S}$ denote the subsystem of S for variables in $\mathcal{X}$. Thus, all the variables in $\mathcal{Y}$ are free variables of $\bar{S}$.
For $k = 1, \ldots, r$, let $\mathcal{F}_k$ denote the set of free variables of subsystem S_k. Then construct $\mathbb{D}_k = \mathbb{B}[\mathcal{Y} \cup \mathcal{F}_k]$, and let $\bar{\sigma}_k : (\mathcal{X}_k \cup \ldots \cup \mathcal{X}_1) \to \mathbb{D}_k$ denote the semantics of $\bar{S}_k$ over $\mathbb{D}_k$ relative to the empty environment. We define a new hierarchical system S' with the same hierarchy as S but the following set of equations:

$$x = \bar{\sigma}_k\, x, \qquad x \in \mathcal{X}_k, 1 \leq k \leq r \quad y \qquad = e_y,\, y \in \mathcal{Y}$$

In S', variables from $\mathcal{X}_j$ may occur unguarded only in right-hand sides of variables from $\mathcal{X}_i$ where $i < j$. Thus, system S' is guarded.

Example 4. Consider the hierarchical system from ex. 3. The set of free variables is empty whereas the set $\mathcal{Y}$ of variables for operator applications and constants is given by $\mathcal{Y} = \{y_1, y_2, y_3\}$. We obtain:

$$\bar{\sigma}_2\, x_1 = y_1 \sqcup y_2 \qquad \bar{\sigma}_2\, x_2 = y_2 \sqcup (y_1 \sqcap y_3) \qquad \bar{\sigma}_1\, x_3 = y_3 \sqcup x_2$$

Consequently, the newly constructed hierarchical system is constituted by the same hierarchy $\mathcal{H} = (\langle \{x_1, x_2, y_1, y_2\}, \mu \rangle, \langle \{x_3, y_3\}, \nu \rangle)$ together with the equations:

$$\begin{array}{llll} x_1 = y_1 \sqcup y_2 & y_1 = a & x_3 = y_3 \sqcup x_2 & y_3 = g\, x_3 \\ x_2 = y_2 \sqcup (y_1 \sqcap y_3) & y_2 = f\,(x_1, x_2) & & \end{array}$$

□

We claim:

Theorem 1. *Assume S is a hierarchical system of equations over a complete and distributive lattice. Then S and the guarded system S' are equivalent.*

Proof. The following two observations can be deduced from prop. 3:

Fact 4 *Assume $1 \leq k \leq r$.*

1. *Then $\bar{\sigma}_{k-1}$ is the unique solution over $\mathbb{D}_{k-1}$ of the set of equations*
$$x = \bar{\sigma}_j\, x \quad x \in \mathcal{X}_j, j < k.$$
2. *Assume the k-th block of $\bar{S}$ is qualified μ (ν). Then $\bar{\sigma}_k$ is the least (greatest) solution over $\mathbb{D}_k$ of the set of equations*
$$x = e_x, \quad x \in \mathcal{X}_k \qquad x = \bar{\sigma}_{k-1}\, x, \quad x \in \mathcal{X}_j, j < k$$

□

With fact 4 we prove for $k = 1, \ldots, r$ all ρ and $z \in \mathcal{Z}_k$ that $[\![S_k]\!]\, \rho\, z = [\![S'_k]\!]\, \rho\, z$. Assume this assertion holds for S_{k-1} (if it exists). We successively will transform S_k into the system S'_k. Each of the applied steps will preserve the semantics for the variables in $\mathcal{Z}_k$.
Step 0: We replace the subsystem S_{k-1} of S_k (if existing) with S'_{k-1}. Subsystems S_{k-1} and S'_{k-1} are equivalent by induction hypothesis. In the following, we only

transform the k-th block and within this block only the equations with left-hand sides not from $\mathcal{Y}$. Let us call this the *Boolean part* of the k-th block.
Step 1: We *add* a fresh variable x' to (the Boolean part of) the k-th block for every $x \in \mathcal{X}_j$, $j < k$, together with the equation $x' = \bar{\sigma}_j\, x$. Thus, the new right-hand side of variable x' is a copy of the right-hand side of the corresponding variable x. Therefore they evaluate to the same values, and we can replace every occurrence of $x \in \mathcal{X}_j, j < k$, in the Boolean part of the k-th block with the corresponding variable x'.
Step 2: By Bekic principle [3, 16], the resulting k-th block is equivalent to a block where the right-hand sides of the x' equal the (unique) solution $\{x' \mid x \in \mathcal{X}_j, j < k\} \to \mathbb{D}_{k-1}$ of the corresponding subset of equations. By fact 4.1, this solution is given by $x' \mapsto \bar{\sigma}_{k-1}\, x$. Therefore in step 2, we *replace* the right-hand side of $x', x \in \mathcal{X}_j, j < k$, with $\bar{\sigma}_{k-1}\, x$.
Step 3: In the Boolean part of the k-th block, we now rename every variable $x \in \mathcal{X}_k$ with corresponding x', and add new equations $x = x', x \in \mathcal{X}_k$.
Thus, step 3 consists in splitting of the fixpoint computation for the k-th block into an inner iteration on the primed variables x' within the Boolean part, nested inside an iteration on the unprimed variables x. This again preserves the semantics (see, e.g., [3, 16]).
Step 4: Assume w.l.o.g. that block k in S_k is qualified μ. By fact 4.2, the least solution of the set of equations over $\mathbb{D}_k$ with left-hand sides $x', x \in \mathcal{X}_j, j \leq k$, is given by $x' \mapsto \bar{\sigma}_k x$. Therefore again by Bekic principle, we now can replace the right-hand sides of all x' with $\bar{\sigma}_k\, x$.

Example 5. Consider the hierarchical system of equations from ex. 3 and let $k = 2$. Then S_1' is given by the equations

$$x_3 = y_3 \sqcup x_2 \qquad y_3 = g\, x_3$$

together with the hierarchy $(\langle \{x_3, y_3\}, \nu \rangle)$. This part of the system will remain unchanged throughout the construction. The only block of equations which we are going to modify is the k-th (i.e., second) block. Initially, it is given by:

$$\begin{array}{ll} x_1 = y_1 \sqcup x_2 & y_1 = a \\ x_2 = y_2 \sqcup (x_1 \sqcap x_3) & y_2 = f\,(x_1, x_2) \end{array}$$

Step 1 adds variable x_3' with right-hand side $\bar{\sigma}_1\, x_3 = y_3 \sqcup x_2$ and results in the set of equations:

$$\begin{array}{lll} x_1 = y_1 \sqcup x_2 & x_3' = y_3 \sqcup x_2 & y_1 = a \\ x_2 = y_2 \sqcup (x_1 \sqcap x_3') & & y_2 = f\,(x_1, x_2) \end{array}$$

Notice that the reference to x_3 in the equation for x_2 has been replaced by a reference to the new variable x_3'. Step 2 is vacuous in this example. Step 3 then renames x_i with x_i' $(i = 1, 2)$ and then adds the equations $x_i = x_i'$. It results in the set of equations:

$$\begin{array}{llll} x_1 = x_1' & x_1' = y_1 \sqcup x_2' & x_3' = y_3 \sqcup x_2' & y_1 = a \\ x_2 = x_2' & x_2' = y_2 \sqcup (x_1' \sqcap x_3') & & y_2 = f\,(x_1, x_2) \end{array}$$

The least solution of the equations for x_1', x_2', x_3' over $\mathbb{D}_2 = \mathbb{B}[\{y_1, y_2, y_3\}]$ is

$$\bar{\sigma}\, x_1' = y_1 \sqcup y_2 \qquad \bar{\sigma}\, x_2' = y_2 \sqcup (y_1 \sqcap y_3) \qquad \bar{\sigma}\, x_3' = y_3 \sqcup y_2$$

which precisely equals $x_i' \mapsto \bar{\sigma}_2\, x_i$. Thus, $\bar{\sigma}$ gives the new right-hand sides for the x_i' in step 4. □

After step 4, the primed variables do no longer occur in right-hand sides – besides in the equations $x = x'$. Therefore, we can replace these equations by $x = \bar{\sigma}_k\, x$ and subsequently remove all variables x' together with their defining equations. The resulting system precisely equals S'_k, and we are done. □

Our construction of an equivalent guarded system is an improvement of the "classical" folklore method for fixpoint expressions sketched in [20, 17, 11] which introduces a huge (even doubly exponential) increase in size. Note, however, that for the case of fixpoint expressions, our methods will allow us even to construct an equivalent guarded hierarchical system of *polynomial* size (see section 6). The present transformation prepares the ground for such further improvements in the construction since it allows to clearly separate the transformation into two stages. The first stage computes the mappings $\bar{\sigma}_k$, $k = 1, \ldots, r$, whereas only the second stage ultimately transforms S.
Let $n \leq m \leq |S|$ denote the numbers of elements in $\mathcal{X}$ and in $\mathcal{X} \cup \mathcal{Y} \cup \mathcal{F}$, respectively. Then each value in $\mathbb{D}_k$ can be represented by an expression of size $\mathcal{O}(m \cdot 2^m)$. The overall size of the transformed system therefore is bounded by $\mathcal{O}(|S| + n \cdot m \cdot 2^m)$. In order to improve on the (potentially) exponential space to store the expressions $\bar{\sigma}_k\, x$ we take into account *how* the $\bar{\sigma}_k$ can be constructed.

4 Blind Algorithms

Assume we are given a (finite) set S of equations over some lattice $\mathbb{D}$ without free variables and cyclic variable dependences. Then S has a unique solution σ which, given a suitable topological ordering $x_1 < \ldots < x_n$ of the variables, can be computed by successively evaluating the right-hand sides for $x_i, i = 1, \ldots, n$. In this sense, we can view S as a *straight-line program* computing variable assignment σ. Therefore, our goal can be rephrased to design efficient straight-line programs that compute variable assignments $\bar{\sigma}_k$. In contrast to straight-line programs, we will allow redefinitions of variables and use programming-language constructs as **for**-loops or **switch**-statements whose conditions, however, may not depend on $\mathbb{D}$-valued variables. Formally, this can be assured by viewing the lattice elements as abstract values for which there are assignments and operations $\sqcup$ and $\sqcap$, but which are lacking any kind of comparison. Let us call such algorithms *blind*.
Every terminating blind algorithm can be unrolled into a finite sequence of variable assignments. By possibly introducing auxiliary variables, we can always bring this sequence into single-assignment form. Therefore, we obtain:

Fact 5 *For every terminating blind algorithm computing $\sigma : \mathcal{X} \to \mathbb{D}$, there is a straight-line program computing a variable assignment σ' which uses the same number of operations in $\mathbb{D}$ such that $\sigma\, x = \sigma'\, x$ for all $x \in \mathcal{X}$.* □

We conclude that time complexity of blind algorithms for computing $\bar{\sigma}_k$, $k = 1, \ldots, r$, directly can be translated into the *output space* of corresponding guardedness transformations. In the following, we therefore will design efficient blind algorithms for computing the semantics of hierarchical equation systems over distributive complete lattices with $\Sigma = \emptyset$, i.e., operators only from $\{\sqcup, \sqcap\}$.

```
forall (x ∈ 𝒳)  x = ⊥;
for (j = 1, j ≤ k, j++) {
    forall (x ∈ 𝒳)  x' = e_x;
    forall (x ∈ 𝒳)  x = x';
}
```

Fig. 1. Lock-Step Iteration.

5 The General Case

Let S denote a set of equations $x = e_x, x \in \mathcal{X}$, without free variables. over a distributive lattice $\mathbb{D}$ where $\Sigma = \emptyset$. The first algorithm one may think of is *lock-step* iteration as in fig. 1. This algorithm successively computes the n-th approximation of the least fixpoint. Since all values of the next round are computed w.r.t. the old values of the variables, we use a set $\{x' \mid x \in \mathcal{X}\}$ of fresh variables to receive the new values. These are then copied into the $x \in \mathcal{X}$.
The algorithm from fig. 1 finds the least fixpoint after $k = \#\mathcal{X}$ rounds. A straight forward application to hierarchical systems would successively remove fixpoints outside-in by an appropriate unrolling. The structure of variable dependences, however, is not taken into account. Therefore, we prefer to replace lock-step iteration with a *Round-Robin* strategy. For (intra-procedural) data-flow analysis, such an approach has been considered, e.g., by Kam and Ullman [12].
The (variable) *dependence graph* of the set of equations S is the directed graph $G = (\mathcal{X}, E)$ where E consists of all edges (x_1, x_2) where variable x_1 occurs in the right-hand side of variable x_2. A set B of edges of G is called *set of back-edges* if G without edges from B is a dag. The maximal number of edges from B on any cycle-free path in G is called *loop-connectedness* of G (relative to B).
Notice that the loop-connectedness of G relative to B is at most $\#B$ or even $\#\{v \mid (u, v) \in B\}$ which sometimes is less. Deciding in general whether the loop-connectedness relative to some B is $\geq k$ for arbitrary k is NP-complete [7]. Determining a set of back-edges which minimizes the loop-connectedness seems to be an even harder problem. In case, however, graph G is "well-structured" (*reducible*), polynomial algorithms are known both for computing such minimal B as well as the corresponding loop-connectedness [7] (see [8] for precise definitions of reducibility). The polynomial algorithm for reducible graphs also provides us with a heuristics (running in linear time) to compute small sets B of back-edges in arbitrary graphs: just determine a DFS forest T of G, and then choose B as the set of all edges (u, v) of G where v is an ancestor of u w.r.t. T. The resulting set B is at least *locally minimal* in so far as no proper subset is a set of back-edges for G as well.
A good choice for a set B of back-edges as well as a safe approximation of the loop-connectedness relative to B will do for all our subsequent constructions. Worse B as well as less accurate approximations for the loop-connectedness may result in larger outputs but will not affect the correctness of the construction. *Any* choice of B, however, will provide us with an algorithm which is *not worse* than the lock-step algorithm. For the following let us fix a suitable set B of

```
for (i = 1, i ≤ n, i++)  x_i = ⊥;
for (j = 1, j ≤ k, j++)
    for (i = 1, i ≤ n, i++)  x_i = e_i;
```

Fig. 2. Round-Robin Iteration.

back-edges. Let $<$ denote a topological ordering of the variables in $\mathcal{X}$ w.r.t. to the dag obtained from G by removing all edges from B. Assume this ordering is given as $x_1 < x_2 < \ldots < x_n$ where the right-hand side for x_i is given by e_i. Then the new strategy loops through all variables where for variable x_{i+1} we use those values for variables $x_1, \ldots, x_i$ which already have been computed within the same round. The version for least solutions is shown in fig. 2. The dual version for greatest fixpoints differs in that values of variables x_i are initialized with $\top$ (instead of $\bot$). For the case of least solutions, we prove:

Proposition 6. *The Round-Robin algorithm of fig. 2 computes the least solution of S in $c+1$ rounds where c equals the loop-connectedness of S (relative to B).*

Example 6. Consider the set of equations (with $\Sigma = \emptyset$) given by:

$$x_1 = y_1 \sqcup x_2 \qquad x_2 = y_2 \sqcup (x_1 \sqcap x_3) \qquad x_3 = y_3 \sqcup x_2$$

The variable dependence graph is shown in fig. 3. One set of back-edges is given

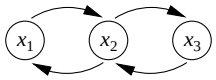

Fig. 3. The Variable Dependences for Ex. 6.

by $B = \{(x_1, x_2), (x_2, x_3)\}$. The loop-connectedness relative to B is 2. Another set of back-edges, however, is given by $B' = \{(x_3, x_2), (x_1, x_2)\}$. For set B', the loop-connectedness equals 1 implying that Round-Robin iteration according to ordering $x_2 < x_1 < x_3$ terminates already after 2 rounds. Starting with initial values $\bot$, we obtain:

	x_2	x_1	x_3
1	y_2	$y_1 \sqcup y_2$	$y_3 \sqcup y_2$
2	$y_2 \sqcup (y_1 \sqcap y_3)$	$y_1 \sqcup y_2$	$y_3 \sqcup y_2$

□

Proof of prop. 6. For simplicity, let us assume that right-hand sides e_i are of one of the forms $a \in \mathbb{D}$, x_j, $x_j \sqcup x_k$ or $x_j \sqcap x_k$ for $x_j, x_k \in V$. The sets $\mathcal{I}_i$ of *intersection trees* for x_i, $i = 1, \ldots, n$, inductively are defined as follows:

- If $e_i \in \mathbb{D}$, then $x_i \in \mathcal{I}_i$;
- If $e_i \equiv x_j$, then $x_i(I) \in \mathcal{I}_i$ for every $I \in \mathcal{I}_j$;
- If $e_i \equiv x_j \sqcup x_k$, then $x_i(I) \in \mathcal{I}_i$ for every $I \in \mathcal{I}_j \cup \mathcal{I}_k$;
- If $e_i \equiv x_j \sqcap x_k$, then $x_i(I_j, I_k) \in \mathcal{I}_i$ for every $I_i \in \mathcal{I}_j$ and $I_k \in \mathcal{I}_k$.

Fig. 4. The Graphs $G^{(1)}$ and $G^{(2)}$ for Ex. 3.

I is called *cycle-free* iff no path in I has more than one occurrence of the same variable. Each intersection tree $I \in \mathcal{I}_i$ represents a value $[\![I]\!]$, namely, the meet over all values corresponding to the leafs of I. Formally, $[\![I]\!]$ is defined as follows:

$$\begin{array}{ll} [\![x_i]\!] & = d \quad \text{if } e_i \equiv d \in \mathbb{D} \\ [\![x_i(I)]\!] & = [\![I]\!] \\ [\![x_i(I_1, I_2)]\!] & = [\![I_1]\!] \sqcap [\![I_2]\!] \end{array}$$

Let $\sigma : \mathcal{X} \to \mathbb{D}$ denote the least solution of S. Then $\sigma\, x_i = \bigsqcup\{[\![I]\!] \mid I \in \mathcal{I}_i\}$, $x_i \in \mathcal{X}$. Now consider an intersection tree $I \in \mathcal{I}_i$ which is not cycle-free. Then we can construct a cycle-free $I' \in \mathcal{I}_i$ such that $[\![I]\!] = [\![I']\!] \sqcap d$ for some $d \in \mathbb{D}$ implying that $[\![I]\!] \sqsubseteq [\![I']\!]$. Hence, it suffices to take least upper bounds just over *cycle-free* intersection trees. Thus the following claim implies our assertion:
Claim: Assume $j \geq 1$. After round j, the value of x_i is an upper bound for $[\![I]\!]$ whenever $I \in \mathcal{I}_i$ is cycle-free and has at most $j-1$ back-edges on every path from a leaf to the root. □
In presence of distributivity, our prop. 6 can be seen as a generalization of Kam and Ullman's result [12] to more general forms of systems of equations.
Let us apply prop. 6 to hierarchical systems with $\Sigma = \emptyset$. We propose an iteration strategy which for alternation-depth r consists in r nested **for**-loops. Each iteration of the outermost loop first evaluates the variables from block r; then it descends into an iteration on the variables of the lower blocks.
Assume $G = (\mathcal{X}, E)$ is the variable dependence graph of hierarchical system S. We construct directed graphs $G^{(k)}$, $k = 1, \ldots, r$, as follows.

- The set of vertices of $G^{(k)}$ is given by $\mathcal{X}$;
- The set of edges of $G^{(k)}$ consists of all pairs (z, x) where $x \in \mathcal{X}_k$ and z occurs in e_x (*primary* edges) together with all pairs (x, z) where $x \in \mathcal{X}_k$, $z \in \mathcal{X}_j, j < k$, and there is a path in $G^{(k-1)}$ from x to z (*derived* edges).

Let c_k denote the minimal loop-connectedness of $G^{(k)}$ relative to sets of back-edges consisting of primary edges only. Then the variables in $\mathcal{X}_k$ can be arranged in such a way that $(c_k + 1)$ iterations of the k-th **for**-loop are sufficient. We call c_k the k-th *derived* loop-connectedness.

Example 7. Consider the hierarchical equation system of ex. 3. The graphs $G^{(1)}$ and $G^{(2)}$ are shown in fig. 4. Since $G^{(1)}$ has only a self-loop, derived loop-connectedness c_1 equals 0. The other graph, $G^{(2)}$, has already been considered in ex. 6. There we found as set of back-edges $B' = \{(x_3, x_2), (x_1, x_2)\}$. Since all edges in B' are primary, we conclude that $c_2 = 1$. □

Let n_k denote the number of variables of the k-th block. By construction, $c_k \leq n_k$ for all k. Recall that $(n_1 + 1) \cdot \ldots \cdot (n_r + 1) \leq (\frac{n}{r} + 1)^r$ where $n = n_1 + \ldots + n_r$. Combining theorem 1 with the sketched blind algorithm, we obtain:

Theorem 2. *Assume S is a hierarchical system of equations with n variables and alternation-depth r over a complete and distributive lattice, and let $c_1, \ldots, c_r$ denote the sequence of derived loop-connectednesses. Then an equivalent*[1] *guarded hierarchical system can be constructed of size*

$$\mathcal{O}(r \cdot (c_1 + 1) \ldots (c_r + 1) \cdot |S|) \leq \mathcal{O}(r \cdot (\tfrac{n}{r} + 1)^r \cdot |S|). \qquad \square$$

The size of the resulting system is linear in the size of S but still may be exponential in the alternation-depth. Observe, however, that the left estimation of theorem 2 usually is sharper than the right bound which just counts variables and ignores variable dependences.

6 Polynomial Special Cases

Often, the variable dependences of (hierarchical) systems are not "arbitrary". In particular, this is the case when the system is derived from a fixpoint expression as in ex. 1. The key idea for this case is to recursively descend into strongly connected components. We obtain a forest-like decomposition similar to [4, 6].
Assume G is a directed graph. A *decomposition forest* (df for short) w for a set V of nodes of G is defined as follows. If $V = \emptyset$, then $w = \epsilon$ (the empty list of trees). Otherwise, let G' denote the subgraph of G with nodes in V.
Case 1: G' is strongly connected. Then $w = (w', x)$ where $x \in V$ and w' is a df for $V \backslash \{x\}$. We call x *exit* of strong component G'.
Case 2: G' is not strongly connected. Then $w = w_1 \ldots w_k$, $k > 1$, where w_j is a df for V_j, and the sequence $V_1, \ldots, V_k$ is a topological ordering of the strong components of G', i.e., whenever an edge of G' goes from V_i to V_j, then $i \leq j$.
Thus, a df is obtained from G by recursively applying two steps: first, decomposition into strongly connected components; second, extracting exits from these.
$\mathsf{depth}(w, x)$ of variable x relative to df w equals the number of parentheses within which x is nested. Formally, if $w = (w_1, x_1) \ldots (w_k, x_k)$ then $\mathsf{depth}(w, x_j) = 1$ for $j = 1, \ldots, k$, and for x occurring in w_j, $\mathsf{depth}(w, x) = 1 + \mathsf{depth}(w_j, x)$. The depth of w then is the maximal depth of a variable occurring in w.
Every directed graph has decomposition forests, but only "well-structured" graphs have decomposition forests which are *exit-post-dominated*. Here, w is an *exit-post-dominated* df (edf for short) iff w is a df where for every subtree (v, h) of w and every edge (x, y), x in (v, h) and y not in (v, h) implies $x = h$.

Example 8. Consider the hierarchical system from ex. 3. Then a df for the variable dependence graph of $\bar{S}$ is given by $w = (((\epsilon, x_3), x_2), x_1)$. Df w is indeed exit-post-dominated. Observe that the exits in this decomposition are nothing but the fixpoint variables of the expression. Another edf, however, which has smaller depth is given by $w' = ((\epsilon, x_1)(\epsilon, x_3), x_2)$. $\square$

The post-dominator relation can be computed in polynomial time [19]. Therefore, it takes only polynomial time to decide whether or not a graph has an edf and, in case it has, to construct such an edf. Our new algorithm for equation systems

[1] up to extra auxiliary variables, of course

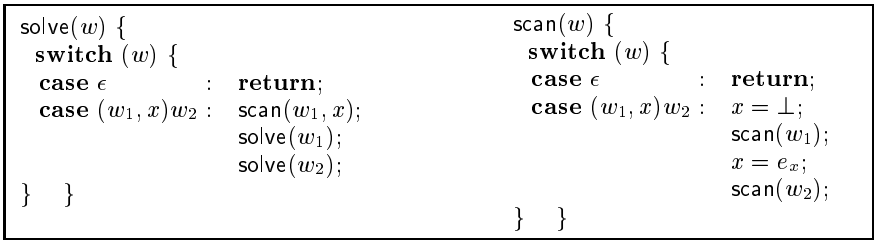

Fig. 5. The EDF-Algorithm for w.

over distributive lattices with $\Sigma = \emptyset$ corresponding to edf's is shown in fig. 5. It processes one strong component after the other; within a strong component, it first performs a scan over the whole component. This scan evaluates each variable within the strong component from left to right. After this scan, the final value for the exit has been reached. Then the algorithm descends one level down the edf w. Note that this recursive call of solve reinitializes each variable in the subforest. In case, we are just interested in the least solution, this reinitialization can be abandoned. This is no longer possible, however, for alternating fixpoints.

Proposition 7. *Assume S is a set of equations $x = e_x, x \in \mathcal{X}$, without free variables over a distributive lattice where $\Sigma = \emptyset$, and w is an edf of the variable dependence graph of S. Then:*

1. *The edf-algorithm from fig. 5 computes the least solution of S.*
2. *It evaluates each variable x exactly* depth(w, x) *times.* □

The correctness of the edf-algorithm crucially depends on w being exit-post-dominated. The advantage of this algorithm (whenever applicable), however, is that some variables may be evaluated significantly fewer times than others. Also, if we are only interested in the variables of an upper fragment of w, reevaluation of the remaining variables can be discarded. The other advantage is that it can be extended to hierarchical systems of equations – without further increase in complexity. Assume S is a hierarchical system of equations where $\Sigma = \emptyset$. Let G denote the variable dependence graph (i.e., the one obtained from S by ignoring the hierarchy). An edf w for G is *leveled* iff for each subtree $t = (w', h)$ of w, variable h has a block number which is at least as big as the block number of every variable occurring in t. We say that S is *expression-like* iff G has a leveled edf.

The main motivation for this definition is that the hierarchical equation systems derived from fixpoint expressions naturally have leveled edf's as defined above. Expression-*like* hierarchical systems, however, are more "liberal" than fixpoint expressions, e.g., by allowing sharing of identical subsystems.

Let us now modify the algorithm from fig. 5 by changing procedure scan to initialize $x = \top$ whenever x is a greatest-fixpoint variable and $x = \bot$ otherwise.

Proposition 8. *Assume S is a hierarchical system of equations without free variables over a distributive lattice with $\Sigma = \emptyset$. If w is a leveled edf for the dependence graph of S, then the modified edf-algorithm for w computes $[\![S]\!]\,\emptyset$.* □

Example 9. Consider the hierarchical system from ex. 1 together with edf $w' = ((\epsilon, x_1)(\epsilon, x_3), x_2)$. First, let us determine the values of $\bar{\sigma}_2$ for x_1 and x_2 in $\mathbb{B}[\{y_1, y_2, y_3\}]$. The computation of the modified edf-algorithm produces the following sequence of variable evaluations:

$$x_1 \;:\; y_1 \qquad x_3 \;:\; y_3 \qquad x_2 \;:\; \boxed{y_2 \sqcup (y_1 \sqcap y_3)} \qquad x_1 \;:\; \boxed{y_1 \sqcup y_2}$$

Final results for $\bar{\sigma}_2$ are enclosed into frame boxes. Notice that we avoided reevaluation of x_3, since the values of $\bar{\sigma}_2$ are just needed for variables from the second block. In order to compute $\bar{\sigma}_1$ for x_3 from the first block, we switch the lattice to $\mathbb{D}_1 = \mathbb{B}[\{y_3, x_1, x_2\}]$. One single evaluation step yields $\bar{\sigma}_1\, x_3 = y_3 \sqcup x_2$. □

Combining theorem 1 with prop. 8, we obtain:

Theorem 3. *Assume S is a hierarchical system of equations over a complete and distributive lattice where $\bar{S}$ is expression-like. Then an equivalent guarded hierarchical system can be constructed of size $\mathcal{O}(m \cdot |S|)$ where m is the depth of a leveled edf for the dependence graph of $\bar{S}$.* □

Applied to some fixpoint expression e, theorem 3 states that an equivalent guarded hierarchical system of equations can be constructed which is just a factor m larger (m the depth of nesting of fixpoints in e).
Another important subclass of (hierarchical) systems of equations is obtained by restricting the usage of "$\sqcap$". Assume S is a hierarchical system of equations with $\Sigma = \emptyset$. S is called *disjunctive* iff each right-hand side e is of the form

$$e ::= x \mid d \mid e_1 \sqcup e_2 \mid e \sqcap d$$

where x denotes a variable and d elements in $\mathbb{D}$. This special form is the generalization of disjunctive Boolean equation systems as considered by Mader [14] to arbitrary distributive lattices. They closely correspond to distributive fixpoint expressions [18]. A simplification of the ideas from the latter paper can be applied to reduce the alternation-depth beforehand to (at most) 2. Using this transformation together with the technique from section 5, we obtain:

Theorem 4. *Assume S is a hierarchical system of equations over a complete and distributive lattice where $\bar{S}$ is disjunctive. Then an equivalent guarded hierarchical system can be constructed of size $\mathcal{O}((n+1) \cdot (c+1) \cdot |S|)$ where n is the number of greatest fixpoint variables and c is the loop-connectedness of the variable dependence graph of $\bar{S}$.* □

7 Applications

Guardedness transformations are especially useful whenever operator applications are "contracting" in some sense. Let us make this idea precise. Let $\mathbb{D}$ denote a complete lattice. Let us consider a metric δ on $\mathbb{D}$ which satisfies the following properties:

$$
\begin{array}{lrcl}
(1) & \delta(d_1, d_2) & \leq & \mathsf{max}(\delta(d_1, d), \delta(d, d_2)) \\
(2) & \delta(d \sqcup d_1, d \sqcup d_2) & \leq & \delta(d_1, d_2) \\
(3) & \delta(d \sqcap d_1, d \sqcap d_2) & \leq & \delta(d_1, d_2)
\end{array}
$$

for all $d, d_1, d_2 \in \mathbb{D}$. A metric satisfying (1) is also called *ultra-metric*. In case, inequalities (2) and (3) hold, we call δ *invariant*. We illustrate these definitions by the following example.

Example 10. Let $\mathbb{T}_\Sigma$ denote the set of languages of *finite* trees over signature Σ. Then $\mathbb{T}_\Sigma$ is a complete and distributive lattice (w.r.t. set inclusion as natural ordering). On $\mathbb{T}_\Sigma$ we define $\delta(L_1, L_2) = 2^{-h}$ where h is the *minimal* depth of a tree in the symmetric difference of L_1 and L_2. Then δ is a metric which satisfies inequalities (1), (2) and (3). □

Operator $f : \mathbb{D}^k \to \mathbb{D}$ is called *contracting*, iff there exists some $0 < \lambda < 1$ such that for all $d_i, d_i' \in \mathbb{D}$ and every j, $\delta(f(d_1, \ldots, d_k), f(d_1', \ldots, d_k')) \leq \lambda \cdot \delta(d_j, d_j')$.

Example 11. Consider the distributive and complete lattice $\mathbb{T}_\Sigma$ from ex. 10. For $a \in \Sigma$, let $[\![a]\!]$ denote the operation of formal application of a. Then $[\![a]\!]$ is contracting with factor $\lambda = \frac{1}{2}$. □

The following theorem is analogous to Banach's fixpoint theorem.

Theorem 5. *Assume $\mathbb{D}$ is a complete lattice and all operators are contracting w.r.t. some invariant ultra-metric on $\mathbb{D}$. Then every finite system of equations over $\mathbb{D}$ without unguarded cycles has a* unique *solution.* □

Proof. W.l.o.g. let us assume that no right-hand side contains unguarded variable occurrences. By structural induction, we find that for variable assignments σ_1, σ_2 and expression e without unguarded variable occurrences, $\delta([\![e]\!]\,\sigma_1, [\![e]\!]\,\sigma_2) \leq \lambda \cdot \mathsf{max}\{\delta(\sigma_1\, x, \sigma_2\, x) \mid x \in \mathcal{X}\}$ for some $0 \leq \lambda < 1$. Now let σ_1 and σ_2 denote the least and greatest solutions of S, respectively, and assume that $\sigma_1 \neq \sigma_2$. Thus, $r = \mathsf{max}\{\delta(\sigma_1\, x, \sigma_2\, x) \mid x \in \mathcal{X}\} > 0$, and there exists some $x \in \mathcal{X}$ such that $r = \delta(\sigma_1\, x, \sigma_2\, x) = \delta([\![e_x]\!]\,\sigma_1, [\![e_x]\!]\,\sigma_2) \leq \lambda \cdot r$ – a contradiction. □

Note that we did not assume that $\mathbb{D}$ is a *complete* metric space. Existence of solutions follows since $\mathbb{D}$ is a complete lattice. The metric is only used to guarantee unicity of solutions.

In [15], we have proposed techniques for pattern matching in finite trees. Here, patterns denote recognizable tree languages for which the element problem must be solved. As a convenient and expressive specification language for recognizable sets we suggested fixpoint expressions. Expressions containing just least fixpoints naturally correspond to (alternating) finite tree automata. In order to allow easy complementation, greatest fixpoints are useful as well. According to ex. 10 and 11, Theorem 5 exhibits an interesting method how greatest fixpoints can be removed. Given a fixpoint expression over $\mathbb{T}_\Sigma$, we proceed as follows:

(0) We construct an equivalent hierarchical system of equations;

(1) We construct an equivalent guarded hierarchical system of equations;

(2) We replace all greatest fixpoints by least ones.

Acknowledgments: We thank André Arnold, Damian Niwinski and Igor Walukiewicz for many inspiring discussions and helpful remarks.

References

1. H.R. Andersen, C. Stirling, and G. Winskel. A Compositional Proof System for the Modal Mu-Calculus. In *IEEE Conf. on Logic in Computer Science (LICS)*, 144–153, 1994.
2. A. Arnold and M. Nivat. Metric Interpretations of Infinite Trees and Semantics of Non-Deterministic Recursive Programs. *Theoretical Computer Science (TCS)*, 11:181–205, 1980.
3. H. Bekic. Definable Operations in General Algebras, and the Theory of Automata and Flowcharts. Technical Report, IBM Labor, Wien, 1967.
4. F. Bourdoncle. Efficient Chaotic Iteration Strategies with Widenings. In *Int. Conf. on Formal Methods in Programming and their Applications*, 128–141. LNCS 735, 1993.
5. B. Courcelle. Basic Notions of Universal Algebra for Language Theory and Graph Grammars. *Theoretical Computer Science (TCS)*, 163(1&2):1–54, 1996.
6. C. Fecht and H. Seidl. A Faster Solver for General Systems of Equations. *Science of Computer Programming (SCP)*, 1999. To appear.
7. A.C. Fong and J.D. Ullman. Finding the Depth of a Flow Graph. *J. of Computer and System Sciences (JCSS)*, 15(3):300–309, 1977.
8. M.S. Hecht. *Flow Analysis of Computer Programs.* North Holland, 1977.
9. R. Kaivola. A Simple Decision Method for the Linear Time Mu-Calculus. In *Int. Workshop on Structures in Concurrency Theory (STRICT) (ed. J. Desel)*, 190–204. Springer-Verlag, Berlin, 1995.
10. R. Kaivola. Fixpoints for Rabin Tree Automata Make Complementation Easy. In *23rd Int. Coll. on Automata, Languages and Programming (ICALP)*, 312–323. LNCS 1099, 1996.
11. R. Kaivola. *Using Automata to Characterise Fixpoint Temporal Logics.* PhD thesis, Dept. of Computer Science, Univ. of Edinburgh, 1996.
12. J.B. Kam and J.D. Ullman. Global Data Flow Analysis and Iterative Algorithms. *J. of the Association for Computing Machinary (JACM)*, 23(1):158–171, 1976.
13. D. Kozen. Results on the Propositional μ-Calculus. *Theoretical Computer Science (TCS)*, 27:333–354, 1983.
14. A. Mader. *Verification of Modal Properties Using Boolean Equation Systems.* PhD thesis, TU München, 1997.
15. A. Neumann and H. Seidl. Locating Matches of Tree Patterns in Forests. Technical Report 98-08, University of Trier, 1998. Short version in *18th Int. Conf. on Foundations of Software Technology and Theoretical Computer Science (FST&TCS)*, LNCS 1530, 134-145, 1998.
16. D. Niwinski. Fixed Point Characterization of Infinite Behavior of Finite State Systems. *Theoretical Computer Science (TCS)*, 189:1–69, 1997.
17. D. Niwinski and I. Walukiewicz. Games for the μ-Calculus. *Theoretical Computer Science (TCS)*, 163(1&2):99–116, 1996.
18. H. Seidl and D. Niwinski. On Distributive Fixpoint Expressions. Technical Report TR 98-04 (253), University of Warsaw, 1998. To appear in RAIRO.
19. R.E. Tarjan. Finding Dominators in Directed Graphs. *SIAM J. of Computing*, 2(3):211–216, 1974.
20. I. Walukiewicz. Notes on the Propositional μ-Calculus: Completeness and Related Results. Technical Report NS-95-1, BRICS Notes Series, 1995.