ELSEVIER Theoretical Computer Science 173 (1997) 235–252

Theoretical Computer Science

Situated simplification

Andreas Podelski[a,*], Gert Smolka[b]

[a] *Max-Planck-Institut Informatik, Im Stadtwald, D-66123 Saarbrücken, Germany*
[b] *Programming Systems Lab, German Research Center for Artificial Intelligence (DFKI), Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany*

Abstract

Testing satisfaction of guards is the essential operation of concurrent constraint programming (CCP) systems. We present and prove correct, for the first time, an incremental algorithm for the simultaneous tests of entailment and disentailment of rational tree constraints to be used in CCP systems with deep guards (e.g., in AKL or in Oz). The algorithm is presented as the *simplification* of the constraints which form the (possibly deep) guards and which are *situated* at different nodes in a tree (of arbitrary depth). The nodes correspond to local computation spaces. In this algorithm, a variable may have multiple bindings (which each represent a constraint on that same variable in a different node). These may be realized in various ways. We give a simple fixed-point algorithm and use it for proving that the tests implemented by another, practical algorithm are correct and complete for entailment and disentailment. We formulate the results in this paper for rational tree constraints; they can be adapted to finite and feature trees.

1. Introduction

One idea behind concurrent constraint programming (CCP) is to base the satisfaction of guards (which is the condition driving the synchronization mechanism) on constraints. In this model, a constraint store is connected with several nodes α. Each of them is associated with a constraint φ_α (its guard) and with a guard check. The guard check consists of the tests of entailment and disentailment of φ_α by the constraint store. (If one of the two tests succeeds, an action may be triggered from that node.) The constraint store grows monotonically; upon each augmentation, the tests are repeated until one of the two tests succeeds. In the deep-guard model, each constraint φ_α may itself be a constraint store which also grows monotonically and which is itself connected with "lower" nodes, and so on. That is, at each instance one has a tree of nodes (or, *local computation spaces*) with constraints; a constraint in a node α is visible in all nodes lower than α. The problem is to determine which of these nodes are entailed [disentailed] by their parent node. At every next instance, this test will be

* Corresponding author. E-mail: podelski@mpi-sb.mpg.de.

repeated for a tree with augmented constraint stores in the nodes. Thus, an algorithm implementing the test has to be incremental. In this paper, we first define formally the general scheme of such an algorithm for an abstract constraint system as *situated simplification*. We then give a concrete algorithm for the case of constraints over rational trees in two different presentations, a high-level one based on fixed-point iteration and a more concrete one with more refined control and with implicit data representation. We prove the correctness of this algorithm in both presentations, relying on our concise formal account of the logical properties of rational-tree constraints.

CCP [14] comes out of concurrent and constraint logic programming, which originated with the Relational Language [4] and with Prolog-II [5], respectively. The computation model of concurrent logic programming languages [15] is based on committed-choice, a particular guard operator. In [10], the commit condition was analyzed as logical entailment. (The delay mechanism in Prolog-like languages as MuProlog [12, 11] and functional residuation in LIFE [1, 2] are based on entailment as well.) AKL [7] and Oz [8, 9] are two practical systems providing both for concurrent (as in multi-agent) and constraint logic programming in one uniform framework. Here, deep guards constitute the central mechanism to combine processes and ("encapsulated") search for problem-solving.

The first formal account (with proof of correctness) of an incremental algorithm for the simultaneous tests of entailment and disentailment is given in [3], for flat guards and a constraint system over feature trees. This algorithm is an instance of a general scheme called relative simplification. An abstract machine for the check of flat guards for constraint systems over trees is given (and proven correct) in [16]. It reflects the present implementations in AKL and Oz. The algorithm is guard-based in the sense that for every guard to be revisited (i.e., whose test is resumed), the entire local binding environment is re-installed (and removed afterwards, if the test still suspends). Its on-line complexity is quadratic, whereas it is quasi-linear for the Beauty & Beast algorithm given in [13]. That algorithm is variable-based in the sense that the bindings are installed for each variable independently and only when needed. The bindings are indexed by the guard; thus, they may remain being installed (i.e., this avoids re-installing/removing the local binding environment each time the test of the guard is resumed). The algorithm given in this paper picks up that idea, namely of indexing the bindings of the same variable by the different nodes (where each of the bindings represents a constraint on the variable in a different node). This provides for a high-level presentation. It now remains a matter of implementation how a variable's multiple bindings are realized. This may be done, for example, by re-installing/removing the guard environment as mentioned above, or by using lookup tables, which seems more efficient.

In this section, we have stated the problem informally and put it into a general context. In Section 3, we formalize it and give the general scheme called situated simplification for incremental algorithms solving the problem over an abstract constraint system. Before that, however, we discuss a motivating example informally, in Section 2. Section 4 lists those properties that are important for the tests of satisfiability and entailment over rational trees and that we need here. We give their proofs in a concise way. We then present a concrete situated simplification algorithm for constraints over rational trees in two different ways. The first one, in Section 5, is high-level in the description of its control strategy (which is by fixed-point iteration) and of its data representation. The second one, in Section 6, avoids redundant computations and data representations as much as possible using a task stack and using a store with implicit representations of constraint conjuncts. We close with a conclusion section.

2. Motivating example

The execution of the Oz program given in Fig. 1 will build up a tree of 5 computation spaces. The constraints are equations between terms, interpreted over the domain of rational trees. The effect of line 1 is to put the constraint $X=f(a,Y)$ into the ("global") constraint store at the root of the tree; we note this node β. Lines 2 to 11 code a disjunction consisting of two disjuncts. These have to be tested simultaneously for entailment and disentailment (the disjunction is suspended until one of the disjuncts is entailed or disentailed, i.e., "determined").[1] The tests of the two disjuncts are done in the two local computation spaces α_1 and α_2 which are directly below β.

The constraint $X = f(a,Y)$ does not determine the constraint $Y=c$. That is, the constraints in the computation spaces above α_2 (here, only β) do not determine the constraint in α_2 (which forms the second of the two disjuncts, given in line 10). This might change during the execution of other, not shown parts of the program, say, if $Y=c$ was added to β (then α_2 would be entailed), or if $Y=d$ was added to β (then α_2 would be disentailed).

The first of the two disjuncts (coded by lines 3 to 8) is a conjunction of the constraint $X=f(Z,Z)$ and another disjunction. The variable Z is quantified existentially over this conjunction, or: α_1 is the "home" of Z (whereas β is the home of X and Y). The computation space α_1 is above the two computation spaces γ_1 and γ_2 for the tests of the disjuncts $Y=Z$ and $Y=b$, respectively.

Now, γ_2 is disentailed since the conjunction $X = f(a,Y) \wedge X = f(Z,Z)$ of the constraints in computation spaces above γ_2, here α_1 and β, disentails the constraint $Y=b$. On the other hand, γ_1 is entailed since the conjunction $X = f(a,Y) \wedge X = f(Z,Z)$ of the constraints in computation spaces above γ_1, here also α_1 and β, entails the constraint $Y=Z$.

Finally, we observe that α_1 itself is not determined. The constraint $X=f(a,Y)$ does not determine $\exists Z\, X=f(Z,Z)$. This might change, for example, if the constraint $Y=a$ was added to β. This last case touches the issue of detecting "implicit equalities," here, between the first and second argument of the term $f(a,Y)$.

[1] In the following, we will use "determine" as a synonym of "entail or disentail".

```
X=f(a Y)
or
  Z in X=f(Z Z)
  or
     Y=Z
  []
     Y=b
  ro
[]
  Y=c
ro
```

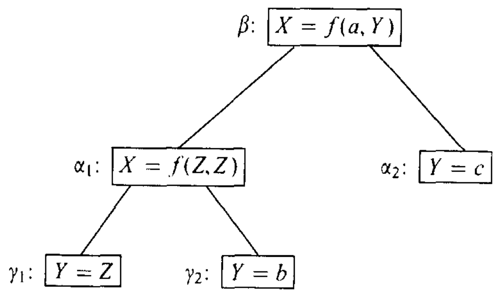

Fig. 1. Oz program and corresponding tree of 5 computation spaces.

How does our algorithm perform a test on the five computation spaces? It will simplify the constraint in each computation space to a new constraint, hereby taking into account the constraints in *all* the computation spaces above. This simplified constraint signals (by its syntactic form) whether the computation space is entailed or disentailed or neither.

For example, the simplified constraint in γ_1 is $\top$ (for *true*) and thus signals entailment ($\top$ is a special case of a constraint signaling entailment). The one in γ_2 is $\perp$ (for *false*) and thus signals disentailment. The one in α_1 is $Z = Y \wedge Y = a$ and, since it binds a variable (here Y) from a computation space above, it does not determine α_1. (It would be further simplified to $Z = Y$ if the constraint $Y = a$ was added to β, and this would signal entailment; note that $\exists Z\ Z = Y$ is equivalent to $\top$.) Finally, the simplified constraint in α_2 would be $Y = c$ and thus signals "not determined". (It would be further simplified to $\perp$ if $Y = a$ was added to β, and then signal disentailment.)

Intrinsic difficulties of tree-ordered constraint stores. The algorithm will compute with suitable representations of the constraints, namely by bindings on variables. These representations must specify to which node the binding belongs. Also, there might be several bindings on the same variable (in the example, two on X and three on Y). This will be handled by allowing multiple bindings and by indexing the bindings with the nodes. For a fixed node, the algorithm must accumulate the constraints in *all* the computation spaces above; i.e., it must represent their conjunction. What if there are several bindings on the same variable in this conjunction (in the example, if we take either of γ_1 or γ_2 as the fixed node, there are two bindings on X)? Could they be contradictory? Which one to choose? Also, if the test on a node depends on all nodes above it, then this means that each modification of the constraints of a node (by incremental adding of a conjunct) concerns potentially all the nodes below it. That is, their suspending tests may have to be resumed. Furthermore, acyclicity of bindings going through nodes above *and below* has to be rechecked. An additional difficulty, algorithmically and for proving correctness, comes from the fact that for the manipulation of constraints over rational trees, bindings previously considered must be memorized in order to avoid infinite loops and in order to prove the entailment of those bindings (as in the proof that $X=f(Y)$, $Y=f(X)$ and $Z=f(Z)$ together entail $X=Z$; here, the repeated encounter of $X=Z$ upon iterative application of the decomposition rule indicates the entailment of $X=Z$).

3. Situated simplification

In this section we will introduce some notions whose context is as follows. The algorithm to be presented in this paper is an instance of a new general scheme (called *situated simplification*) which is parameterized by the *constraint system*. An algorithm in this scheme will take a *constraint tree* as input. For each of the *nodes* of the constraint tree, the algorithm will determine whether the node has one of the two properties which we will introduce below, called *inconsistent* or *entailed*, or neither of them. It will do this by transforming the constraint tree into a *normal* one.

We assume a *constraint system*, i.e., a first-order theory Δ (the theory can be given as the set of all sentences valid in a given structure; e.g., of rational trees) and a set Con of first-order formulae called constraints. The set Con must be closed wrt. conjunction.[2] The set of variables is Var. For $\varphi \in \mathsf{Con}$, $\mathsf{free\_var}(\varphi)$ is the set of all free variables of φ.

We also assume a finite tree-ordered set[3] Nodes whose elements we refer to as *nodes* (or, local computation spaces). We note the tree-order "$\leqslant$". We read $\beta < \alpha$ as "β is above α" (which means, β is closer to the root in the tree order). In the notation in this paper, we will try to keep the order $\beta \leqslant \alpha \leqslant \gamma$ when we choose letters α, β, γ to name nodes.

Finally, we assume a function $\mathsf{home}:\mathsf{Var} \mapsto \mathsf{Nodes}$ which assigns each variable x one node α as its "home". We call the variables in $\mathsf{home}^{-1}(\alpha)$ the *local* variables of the node α. We often write them in a tuple $\bar{x}_\alpha$. Thus,

$$\bar{x}_\alpha = \{\text{local variables of } \alpha\} = \mathsf{home}^{-1}(\alpha).$$

The variables x with home in nodes strictly above α are the *global* variables of α. Thus,

$$\{\text{global variables of } \alpha\} = \bigcup\{\mathsf{home}^{-1}(\beta) \mid \beta < \alpha\}.$$

We may use "global" and "local" referring only implicitly to a fixed node α. The local and the global variables are the ones "visible in α". Thus,

$$\{\text{variables visible in } \alpha\} = \bigcup\{\mathsf{home}^{-1}(\beta) \mid \beta \leqslant \alpha\}.$$

[2] As usual, we identify a conjunction with the multiset of its conjuncts (and $\wedge$ with $\cup$) in our notation. Thus, $X=t \in \varphi$ means that $X=t$ is a conjunct in φ, and $\bigcup\{\varphi \mid \varphi \in M\}$ is the conjunction of all constraints φ in the set M.

[3] A tree-ordered set is isomorphic to a prefix-closed subset of some free monoid, where "$\leqslant$" corresponds to the relation "is prefix of".

The set Nodes and the assignment home of variables to their "home" nodes are fixed throughout the rest of this paper. We are interested in labelings of the nodes by constraints (hence the name "constraint tree") which respect the partition of the variables according to their home nodes in the following sense.

We define that a *constraint tree* T is a mapping

$$T : \mathsf{Nodes} \mapsto \mathsf{Con},\ \alpha \mapsto \varphi_\alpha$$

such that the free variables of each constraint φ_α are visible in its home node α, i.e.,

$$\mathsf{free\_var}(\varphi_\alpha) \subseteq \bigcup\{\mathsf{home}^{-1}(\beta) \mid \beta \leqslant \alpha\}.$$

We will write T also as the sequence $(\varphi_\alpha)_{\alpha \in \mathsf{Nodes}}$.

We refer to the constraint φ_α as the "constraint situated in α". We define the "context of α" as the constraint $\varphi_{<\alpha}$ which is the conjunction of the constraints situated in nodes strictly above α; i.e.,

$$\varphi_{<\alpha} = \bigcup\{\varphi_\beta \mid \beta < \alpha\}.$$

The constraint "visible in α" is noted $\varphi_{\leqslant\alpha}$ and defined accordingly.

The node α is *inconsistent* (or, disentailed) if $\varphi_{\leqslant\alpha}$ is unsatisfiable, which is the same as

$$\Delta \models \varphi_{<\alpha} \rightarrow \neg\exists\bar{x}_\alpha\varphi_\alpha.$$

Note that this may also be expressed by $\Delta \models \varphi_{<\alpha} \rightarrow (\exists\bar{x}_\alpha\,\varphi_\alpha \leftrightarrow \bot)$.

The node α is *entailed* if (1) it is consistent and (2) the constraint situated in α is entailed (modulo local variables) by the context of α. Condition (2) is formally

$$\Delta \models \varphi_{<\alpha} \rightarrow \exists\bar{x}_\alpha\varphi_\alpha.$$

Note that condition (2) may also be expressed by $\Delta \models \varphi_{<\alpha} \rightarrow (\exists\bar{x}_\alpha\,\varphi_\alpha \leftrightarrow \top)$. The constraint tree $(\varphi_\alpha)_{\alpha\in\mathsf{Nodes}}$ is *normal* if it satisfies conditions 1 and 2 below:

1. If α is inconsistent, then the constraint φ_α is the constraint *false*.
2. If α is entailed, then the constraint $\exists\bar{x}_\alpha\,\varphi_\alpha$ is equivalent to *true*.

Formally, this means the same as the following:

1. If $\Delta \models \varphi_{<\alpha} \rightarrow \neg\exists\bar{x}_\alpha\,\varphi_\alpha$, then $\varphi_\alpha = \bot$.
2. If $\Delta \models (\exists)\varphi_{<\alpha}$ and $\Delta \models \varphi_{<\alpha} \rightarrow \exists\bar{x}_\alpha\,\varphi_\alpha$, then $\Delta \models \exists\bar{x}_\alpha\,\varphi_\alpha$.

Two constraint trees are *equivalent* if for every node α, the two constraints visible in α in each constraint tree are equivalent. Formally, for two constraint trees $(\varphi_\alpha)_{\alpha\in\mathsf{Nodes}}$ and $(\varphi'_\alpha)_{\alpha\in\mathsf{Nodes}}$, if

$$\Delta \models \varphi_{\leqslant\alpha} \leftrightarrow \varphi'_{\leqslant\alpha}.$$

Note that the two constraint trees have the same set of nodes, namely Nodes, which is fixed (as is the home mapping on variables).

We name *situated simplification* a procedure which transforms a constraint tree into an equivalent normal constraint tree.

Situated simplification implements the simultaneous tests of entailment and disentailment of φ_α by the context of α for every $\alpha \in \mathsf{Nodes}$.

4. Rational trees

We assume a signature containing the function symbols (or constructor symbols) which we note f, g, h, a, b, c, etc. (we assume the existence of at least two different symbols but nothing else on the signature; in particular, it may be finite or infinite). We call *constructions* the terms of the form $f(\bar{x})$ and note Struct the set that they form. Here, $\bar{x}$ denotes an ordered tuple $(x_1,\ldots,x_n)$ of length n according to the arity of the function symbol f, with *pairwise different* variables $x_1,\ldots,x_n$.

The set of constraints Con is the set of possibly existentially quantified conjunctions φ of equations between variables $x \in \mathsf{Var}$ and terms $t \in \mathsf{Var} \cup \mathsf{Struct}$ (the restriction to terms of depth at most one is for presentation only; it is, of course, not a proper restriction). Formally, we have the following abstract syntax for constraints:[4]

$$\varphi ::= x = y \mid y = f(\bar{x}) \mid \exists x\, \varphi \mid \varphi_1 \wedge \varphi_2 \mid \top \mid \bot$$

A tree τ may be represented as a set of pairs (w, f) where the function symbol f is the labeling of the node with the path $w \in \{1,2,\ldots\}^\star$. The empty path ε refers to the root of the tree. We write the free-monoid concatenation of paths v and w simply vw; we have $\varepsilon w = w\varepsilon = w$.

The set t must satisfy several conditions in order to be a tree: The labeling is unique, the root is always a node of t, and the direct descendants of each node conform to the arity of its function symbol (i.e., $(w,f),(w,g) \in t$ implies $f = g$), $(\varepsilon, f) \in t$ for some function symbol f), and if $(w,f) \in t$ then $(wi, f_i) \in t$ function symbol f_i iff $1 \leqslant i \leqslant$ arity of f).

The tree τ is rational iff it has only finitely many subtrees, which are the trees $w^{-1}\tau = \{(v,f) | (wv,f) \in \tau\}$ for some path w.

The application of a function f to trees $t_1,\ldots,t_n$ yields the tree

$$f(t_1,\ldots,t_n) = \{(\varepsilon, f)\} \cup \{(iw, g) \mid (w,g) \in t_i,\ i = 1,\ldots,n\}.$$

Given a constraint φ, we say that the variable x is *determined* if φ contains an equation between x and a construction. A constraint $\varphi = \{x_i = f_i(\bar{u}_i) \mid i = 1,\ldots,n\}$ where the determined variables $x_1,\ldots,x_n$ are pairwise different is called a *linear system*.

From now on, Δ is the theory of rational trees over the given signature of function symbols. We will use the following three facts about trees, the first one for the consistency test and the other two for the entailment test. The first one is *the* characteristic property of rational trees (cf., for example, [6]).

[4] We use $=$ for both the logical equality symbol and the meta-level identity; no ambiguity will arise.

Fact 1. *A linear system φ is satisfiable; i.e., $\Delta \models (\exists)\,\varphi$.*

The first fact is a logical consequence of the next one. (On the other hand, given a proof of the first fact, the second fact could have been proven from the first. Namely, the value of a non-determined variable never contains an occurrence of the value of a determined variable and, thus, may be chosen arbitrarily in any solution for φ.)

Fact 2. *For all values of the non-determined variables in a linear system φ there exist values for its determined variables $x_1,\dots,x_n$ such that φ holds, i.e., $\Delta \models (\forall)\ \exists(x_1,\dots,x_n)\varphi$.*

Proof. Given φ, we define the relation $x \rightsquigarrow_w y$ ("x leads to y") by: $x \rightsquigarrow_\varepsilon x$, and if $x \rightsquigarrow_w y$ and $y = f(y_1,\dots,y_k,\dots,y_n) \in \varphi$ then $x \rightsquigarrow_{wk} y_k$. We extend any valuation v defined on the non-determined variables of φ by setting

$$v(x) = \begin{array}{l} \{(w,f) \mid x \rightsquigarrow_w y, y = f(\bar{u}) \in \varphi\} \\ \cup\{(wv,f) \mid x \rightsquigarrow_w y,\ y \text{ is non-determined, } (v,f) \in v(y)\}. \end{array}$$

For all determined variables x, every subtree of $v(x)$ is either of the form $w^{-1}v(x) = v(z)$ where z is the variable occurring in φ such that $x \rightsquigarrow_w z$, or it is a subtree of such a $v(z)$ for a non-determined variable z. Thus, $v(x)$ is a rational tree, and v satisfies all equations $x = f(\bar{u}) \in \varphi$. □

The next fact says when equations between determined variables are entailed.[5]

Fact 3. *The constraint φ entails the conjunction ψ of variable–variable equations if for every conjunct $x = y$ of ψ there exist determining equations $x = f(u_1,\dots,u_m)$ and $y = f(v_1,\dots,v_m)$ in φ such that the variables u_j and v_j are equated in φ or in ψ or they are the same variable (i.e., $u_j = v_j \in \varphi \cup \psi$ or $u_j = v_j$ for $j = 1,\dots,m$).*

Proof. We first note that two rational (or infinite, or finite) trees τ_1 and τ_2 are equal iff for all n they are equal up to depth n.[6] Given a valuation v which satisfies φ (i.e., $\Delta, v \models \varphi$), we prove, by induction over n,

$$\text{for all } x = y \in \psi,\ v(x) \text{ and } v(y) \text{ are equal up to depth } n. \qquad (1)$$

We assume (1) for n' with $n' < n$ and $x = y \in \psi$. Then there exist $x = f(u_1,\dots,u_m)$, $y = f(v_1,\dots,v_m) \in \varphi$ as in the formulation of Fact 3. Thus, if $n = 0$ then (1) holds. Otherwise, for $j = 1,\dots,m$, $u_j = v_j$ or $u_j = v_j \in \varphi$ or $u_j = v_j \in \psi$. In any of the three cases (in the last one by induction), $v(u_j)$ and $v(v_j)$ are equal up to depth $n - 1$, and hence, (1) holds for n. □

[5] This is a simple fact about rational trees, and finite trees as well. It is orthogonal to the algorithmic problem of the entailment test for rational trees which is caused by cycles in the determining equations.
[6] Formally, one may define the restriction of a tree τ to depth n inductively by $f(\tau_1,\dots,\tau_m)|_0 = f(a,\dots,a)$ and $f(\tau_1,\dots,\tau_m)|_{n+1} = f(\tau_1|_n,\dots,\tau_m|_n)$, for some constant symbol a.

5. Fixed-point algorithm

In this section, we will represent the two kinds of rational-tree constraints by bindings (either to a variable or to a construction) which are marked by the node to which the constraint belongs. We call the corresponding representation of a whole constraint tree a *decoration*. Then we will describe an algorithm which works by generating many new bindings (a lot of which will be redundant). Since it will never remove a binding (and not use new constructions), however, the termination follows from the finiteness of all possible bindings. The successive generations of bindings are justified by either the logical properties of equality, or by the fact that a constraint is visible in all nodes below the one to which it belongs, or by one logical property of the rational-tree constraint system (namely, the injectivity of function symbols). If there are no more justified generations of bindings possible, then the bindings (reduced to a non-redundant subset) represent a constraint tree which is *normal* (and, thus, exhibits which nodes are inconsistent or entailed). This follows from Theorem 1, which states how the bindings exhibit directly which nodes are inconsistent or entailed.

We will next define a representation for rational-tree constraint trees.

A *decoration D* is a labeling of nodes α by finite relations $\overset{\alpha}{=} \subseteq \mathsf{Var} \times (\mathsf{Var} \cup \mathsf{Struct})$ such that all variables occurring in $\overset{\alpha}{=}$ are visible in the node α. We write the relationship as $x \overset{\alpha}{=} y$ or $x \overset{\alpha}{=} f(\bar{u})$, respectively. For each node α, we define its "context relation"

$$\overset{<\alpha}{=} = \bigcup \{ \overset{\beta}{=} \mid \beta < \alpha \}.$$

A decoration D defines a constraint tree $(\varphi_\alpha)_{\alpha \in \mathsf{Nodes}}$ by $\varphi_\alpha = \{ x = t \mid x \overset{\alpha}{=} t \}$.

For decorations, the notions of equivalence and of inconsistent (or of entailed) nodes and of determined variables are obtained by referring to the defined constraint trees.

A decoration D is *complete* if for all variables x, y, constructions $f(\bar{u}), f(\bar{v})$ and nodes α,

1. $\overset{\alpha}{=} \cap (\mathsf{Var} \times \mathsf{Var})$ is an equivalence relation,
2. $\overset{<\alpha}{=} \subseteq \overset{\alpha}{=}$,
3. $x \overset{\alpha}{=} y$, $x \overset{\alpha}{=} f(\bar{u})$ implies $y \overset{\alpha}{=} f(\bar{u})$,
4. $x \overset{\alpha}{=} f(\bar{u})$, $x \overset{\alpha}{=} f(\bar{v})$ implies $\bar{u} \overset{\alpha}{=} \bar{v}$.

Given any decoration D, each of the conditions above can be made to be satisfied by adding pairs to the relations $\overset{\alpha}{=}$ (which is an equivalence transformation on the defined constraint tree). Going iteratively through the four conditions yields a monotonically growing family of relations. Since for each α, $\overset{\alpha}{=}$ is a subset of $\mathsf{Var} \times (\mathsf{Var} \cup \mathsf{Struct})$ ranging only over the variables and constructions occurring in D, the iteration reaches a fixed point in finitely many steps (note our assumptions that each relation $\overset{\alpha}{=}$ is finite and that the fixed set Nodes is finite). We have given an algorithm which proves the following statement.

Proposition 1. *For every decoration D there exists a least complete decoration D′ containing D (i.e., D ⊆ D′). Moreover, such a decoration D′ is equivalent to D.*

A complete decoration is interesting because it exhibits which nodes of the defined constraint tree are inconsistent and which are entailed.

Theorem 1. *If D is a complete decoration, then:*

1. *The node α is inconsistent iff $x \overset{\alpha}{\doteq} f(\bar{u})$ and $x \overset{\alpha}{\doteq} g(\bar{v})$ and $f \neq g$ for some variable x and constructions $f(\bar{u})$, $g(\bar{v})$;*
2. *The node α is entailed iff α is consistent and the following two conditions hold.*
   (a) *If x is global and $x \overset{\alpha}{\doteq} f(\bar{u})$ then there exists a variable y and a construction $f(\bar{v})$ with $x \overset{\alpha}{\doteq} y$ and $y \overset{<\alpha}{\doteq} f(\bar{v})$ (i.e., y is determined in the context of α).*
   (b) *If x and y are global and $x \overset{\alpha}{\doteq} y$ then either $x \overset{<\alpha}{\doteq} y$ or $x \overset{<\alpha}{\doteq} f(\bar{u})$ and $y \overset{<\alpha}{\doteq} f(\bar{v})$ for some constructions $f(\bar{u})$, $f(\bar{v})$ (i.e., both x and y are determined in the context of α).*

Proof. Given a complete decoration and a node α fixed, we may construct a function $\mathsf{r} : \mathsf{Var} \mapsto \mathsf{Var}$ such that (1) $\mathsf{r}(x) = \mathsf{r}(y)$ iff $x \overset{\alpha}{\doteq} y$, and (2) $\mathsf{r}(x)$ is local only if x is local. That is, r assigns each variable a – with preference global – representative of its equivalence class, the equivalence being $\overset{\alpha}{\doteq} \bigcap (\mathsf{Var} \times \mathsf{Var})$. (We will make use of the preference of a global over a local variable as the representative when we introduce Eq. (2).) The constraint

$$\varphi_\alpha^{\mathsf{r}} = \{\mathsf{r}(x) = f(\mathsf{r}(\bar{u})) \mid x \overset{\alpha}{\doteq} f(\bar{u})\} \cup \{x = \mathsf{r}(x) \mid x \overset{\alpha}{\doteq} \mathsf{r}(x),\ x \neq \mathsf{r}(x)\}$$

is equivalent to φ_α (in the empty theory, by the laws for equality),

$$\models \varphi_\alpha \leftrightarrow \varphi_\alpha^{\mathsf{r}}.$$

If the condition in Statement 1 of the theorem holds, then clearly φ_α is unsatisfiable. Otherwise, we can write $\varphi_\alpha^{\mathsf{r}}$ in the form

$$\varphi_\alpha^{\mathsf{r}} = \{x_1 = f_1(\bar{u}_1), \dots, x_n = f_n(\bar{u}_n)\} \cup \{x_{n+1} = y_{n+1}, \dots, x_m = y_m\}, \tag{2}$$

where the variables $x_1, \dots, x_m$ are pairwise different. The first part is a linear system and, by Fact 1, has a solution over rational trees. Again by the laws for equality, this solution may be completed to be one for the second part too. This proves Statement 1.

If condition (a) in Statement 2 is violated, then $\varphi_{<\alpha} \wedge x = g(\bar{v})$ is satisfiable and disentails $x = f(\bar{u})$ and, hence, φ_α. If condition (b) is violated, then $\varphi_{<\alpha} \wedge x = f(\bar{u}) \wedge y = g(\bar{v})$ is consistent and disentails $x = y$ and, hence, φ_α. Thus, $\varphi_{<\alpha}$ does not entail φ_α.

If conditions (a) and (b) hold, then Fact 3 says that all equalities between global variables in φ_α are redundant with respect to $\varphi_{<\alpha}$. The equalities with at least one local, existentially quantified variable are redundant too. Thus, $\varphi_{<\alpha} \wedge \varphi_\alpha$ is equivalent to $\varphi_{<\alpha} \wedge \varphi_{\alpha,\mathsf{local}}^{\mathsf{r}}$, where

$$\varphi_{\alpha,\mathsf{local}}^{\mathsf{r}} = \{x = t \in \varphi_\alpha^{\mathsf{r}} \mid x \text{ local}\}.$$

But Fact 2 says that $\exists \bar{x}_\alpha \varphi_{\alpha,\mathsf{local}}^{\mathsf{r}}$ is valid. This proves Statement 2. □

Remark. In the theorem above, Statement 2 holds with respect to finite trees too. Thus, the algorithm can be adapted to finite trees simply by adding the occurs-check to the test of a node's consistency.

In fact, the description above yields that if each of two constraints is satisfiable over finite trees (and hence, also over rational trees), then the entailment relation between them is the same for finite and for rational trees.

The theorem above expresses that the fixed-point algorithm implements situated simplification. Namely, a complete decoration D can be assigned a constraint tree $(\psi_\alpha)_{\alpha\in\mathsf{Nodes}}$ as follows. If there exist $x \overset{\alpha}{=} f(\bar{u})$ and $x \overset{\alpha}{=} g(\bar{v})$ with $f \neq g$, then $\psi_\alpha = \bot$. Otherwise,

$$\psi_\alpha = \varphi^{\mathsf{r}}_\alpha - \varphi_{<\alpha} - \{x = y \mid x \text{ and } y \text{ are determined in } \varphi_{<\alpha}\}.$$

The constraint tree thus defined is equivalent to the one defined by D and it is normal. We omit the tedious but straightforward proof.

6. Practical algorithm

We will first define an efficient (i.e., non-redundant) representation for the equations visible in the nodes of a constraint tree and then investigate a "solved form" for such a representation, i.e., a form for consistent nodes exhibiting which of them are entailed. We next define a representation of a constraint tree specifying the constraint tree as a triple of (yet) unsolved equations, solved ones and inconsistent nodes. Naturally, the operational service of the practical algorithm is to "solve" such a representation of a constraint tree, namely, to transform it into an equivalent one where all consistent nodes are represented by equations in solved form. We will finally give such an algorithm and prove it correct.

6.1. *Sets of situated bindings*

A *set of situated bindings* B is a set of elements $(x,\alpha,t) \in \mathsf{Var} \times \mathsf{Nodes} \times (\mathsf{Var} \cup \mathsf{Struct})$, where x and the variables of t are visible in α, which satisfies the following conditions.

1. ("No cycles on the same path")
If $\alpha_1,\ldots,\alpha_n \leqslant \alpha$ and $\{(x_1,\alpha_1,x_2),\ldots,(x_n,\alpha_n,x_{n+1})\} \subseteq B$, then $x_1 \neq x_{n+1}$.

2. ("No binding of global to local variable")
If $(x,\alpha,y) \in B$ and x is global, then y is global too.

3. ("No two bindings of the same variable in the same node")
If $(x,\alpha,s),(x,\alpha,t) \in B$, then $\beta = \alpha$ and $s = t$.

4. ("If two bindings of the same variable are on the same path, then the above one to a construction and the lower one to a variable")
If $\beta < \alpha$ and $(x,\beta,s),(x,\alpha,t) \in B$, then $s \in \mathsf{Var}$ and $t \in \mathsf{Struct}$.

A set of situated bindings B defines a constraint tree by $\varphi_\alpha = \{x = t \mid (x,\alpha,t) \in B\}$ and thereby the notions of inconsistent and of entailed nodes.

Thanks to the first condition above, the following definition is well-founded.[7]

$$\mathsf{vderef}(x,\alpha,B) = \begin{cases} \mathsf{vderef}(y,\alpha,B) & \text{if there exists } (x,\beta,y) \in B \\ & \quad\text{with } \beta \leqslant \alpha, \\ x & \text{otherwise.} \end{cases}$$

Note that thanks to condition 4 each $(x,\beta,y) \in B$ with $\beta \leqslant \alpha$ is necessarily unique.

A set of situated bindings B defines a decoration $(\overset{\alpha}{\underset{B}{=}})_{\alpha \in \mathsf{Nodes}}$ by

$$\begin{array}{ll} x \overset{\alpha}{\underset{B}{=}} y & \text{iff } \mathsf{vderef}(x,\alpha,B) = \mathsf{vderef}(y,\alpha,B), \text{ and} \\ x \overset{\alpha}{\underset{B}{=}} f(\bar{u}) & \text{iff } (\mathsf{vderef}(x,\alpha,B),\beta,f(\bar{u})) \in B \text{ for some } \beta \leqslant \alpha. \end{array}$$

Note that this decoration satisfies the first two, but generally not the last two conditions for a complete decoration.

We call a set B of situated bindings *complete* if the following two conditions hold:

1. If (x,α,y), $(x,\beta,f(\bar{u})) \in B$ and $\beta \leqslant \alpha$ then $y \overset{\alpha}{\underset{B}{=}} f(\bar{u})$.
2. If $(x,\beta_1,f(\bar{u}))$, $(y,\beta_2,f(\bar{v})) \in B$ and $x \overset{\alpha}{\underset{B}{=}} y$ and β_1, $\beta_2 \leqslant \alpha$ then $\bar{u} \overset{\alpha}{\underset{B}{=}} \bar{v}$.

We now have the following characterization.

Proposition 2. *B is a complete set of situated bindings iff $(\overset{\alpha}{\underset{B}{=}})_{\alpha \in \mathsf{Nodes}}$ is a complete decoration.*

Given a set of situated bindings B, we obtain B^- by removing "secondary bindings" in B, i.e.,

$$B^- = B - \{(x,\alpha,y) \mid \text{exists } (x,\beta,f(\bar{u})) \in B,\ \beta \leqslant \alpha\}.$$

Theorem 1 and Proposition 2 immediately yield the following characterization of entailed nodes.

Proposition 3. *Given a complete set of situated bindings B, a node α is entailed iff all bindings $(x,\alpha,t) \in B^-$ are on local variables x only. All nodes are consistent.*

With respect to situated simplification, the statement above means the following.

Corollary 1. *If B is a complete set of situated bindings, then B^- defines an equivalent normal constraint tree (with consistent nodes only).*

[7] Note that the function vderef always yields a variable (and never a construction). This allows us to express, in the definition of $\overset{\alpha}{\underset{B}{=}}$, an "explicit equality" between two variables. (An "implicit equality" is one between two variables bound to equal constructions; e.g., between x and y when $(x,\alpha,f(\bar{u}))$, $(y,\alpha,f(\bar{u})) \in B$, or when $(x,\alpha,f(x))$, $(y,\alpha,f(y)) \in B$, and so on.)

6.2. Configurations

A *configuration* is a triple (E,B,I) consisting of a multiset E of elements $(x,\alpha,t) \in \mathsf{Var} \times \mathsf{Nodes} \times (\mathsf{Var} \cup \mathsf{Struct})$ (which we call situated equations), a set of situated bindings B and a set $I \subseteq \mathsf{Nodes}$ of *inconsistent nodes* which never contains a node occurring in either B or E and is downward closed (i.e., if $\beta \leqslant \alpha$ and $\beta \in I$ then $\alpha \in I$).

A configuration defines a constraint tree over rational trees by

$$\varphi_\alpha = \begin{cases} \bot & \text{if } \alpha \in I, \\ \{x = t \mid (x,\alpha,t) \in B \cup E\} & \text{otherwise.} \end{cases}$$

Given a configuration, the notions of equivalence and of inconsistent and entailed nodes refer to the defined constraint tree.

We call a configuration (E,B,I) *normal* if $E = \emptyset$ and B is a complete set of situated bindings.

The operational service to be provided by our algorithm is indicated by the following characterization (namely, to transform a configuration into a normal one).

Proposition 4. *Given a normal configuration (B,E,I), a node α is*
1. *inconsistent iff $\alpha \in I$, and*
2. *entailed iff all variables x with a binding $(x,\alpha,t) \in B^-$ are local variables of α.*

The next remark says that the algorithm implements situated simplification; it is a reformulation of Corollary 1.

Corollary 2. *Given a normal configuration (B,E,I), the equivalent configuration (E,B^-,I) obtained by removing secondary bindings in B defines a normal constraint tree.*

6.3. Normalization of configurations

We consider the procedure given in Fig. 2.

Starting with an initial configuration (E_0,B_0,I_0), each execution of the body of the while loop yields a new triple (E_i,B_i,I_i), for $i = 1,\ldots,N$ where $N \leqslant \omega$. It might be a useful exercise for the reader to reformulate the algorithm using configuration-rewrite rules.

It is important to note that the algorithm can start with any configuration (and not just with one where B and I are empty). The algorithm is to be used on-line, i.e., where the computation tree and the set of situated equations E are augmented incrementally. The algorithm is incremental since B and I grow then incrementally too.

We will next explain some lines of the algorithm. In line 3, the result of vderef applied to x is again a variable, by the definition of vderef. This variable might itself be bound to a construction. If the binding lies in α, line 10 will take care of that case, and line 14 if the binding lies in a node β above α.

```
1   while $E \neq \emptyset$
2       choose $(x, \alpha, t) \in E$
3       $x :=$ vderef $(x, \alpha, B)$
4       if $t = y$ (i.e., $t \in$ Var) then
5         $y :=$ vderef $(y, \alpha, B)$
6         if $x = y$ then
7           skip
8         [] $x \neq y$ then
9           if $x$ global and $y$ local for $\alpha$ then swap$(x, y)$ fi
10          for all $(x, \gamma, s) \in B$ with $\alpha \leq \gamma$
11              remove $(x, \gamma, s)$ from $B$, add $(y, \gamma, s)$ to $E$
12          for all $(y, \gamma, z) \in B$ with $\alpha < \gamma$
13              if vderef$(z, \gamma, B) = x$ then remove $(y, \gamma, z)$ from $B$ fi
14          if exists $(x, \beta, s) \in B$ with $\beta < \alpha$ then add $(y, \alpha, s)$ to $E$ fi
15          add $(x, \alpha, y)$ to $B$
16        fi
17      [] $t = f(\bar{u})$ (i.e., $t \in$ Struct) then
18        if exists $(x, \beta, g(\bar{v})) \in B$ with $\beta \leq \alpha$, $f \neq g$ then
19          for all $(z, \gamma, s) \in B \cup E$ with $\alpha \leq \gamma$
20              remove $(z, \gamma, s)$ from $B$ and $E$, add $\gamma$ to $I$
21        [] exists $(x, \beta, f(\bar{v})) \in B$ with $\beta \leq \alpha$, then
22          add $(\bar{u}, \alpha, \bar{v})$ to $E$
23        [] not exists $(x, \beta, g(\bar{v})) \in B$ with $\beta \leq \alpha$ ($f = g$ or $f \neq g$), then
24          for all $(x, \gamma, s) \in B$ with $\alpha \leq \gamma$
25          remove $(x, \gamma, s)$ from $B$ and add to $E$
26          add $(x, \alpha, f(\bar{u}))$ to $B$
27        fi
28      fi
29      remove $(x, \alpha, t)$ from $E$
30  end
```

Fig. 2. The algorithm transforming a configuration into a normal one.

If the term t is a variable, then its deref value is one too (again, by the definition of vderef). Line 9 ensures that we do not bind a global to a local variable. This corresponds to condition 2 in the definition of a set of situated bindings in Section 6.1 (which plays a role for the entailment condition).

Lines 10–11 ensure conditions 3 and the part of 4 which concerns the lower parts of paths through α.

Lines 12–13 ensure that condition 1 holds even after line 15 has been executed. Namely, one has to avoid cyclic references which go through nodes above *and below* α. It is important to note that we can restrict ourselves to removing bindings (y, γ, z) where $\alpha < \gamma$, and not $\alpha \leqslant \gamma$. This reason is that y is the result of applying the function vderef$(_, \alpha, B)$. Thus, there cannot be a binding (y, α, z).

In line 14, the term s is necessarily a construction (if it were a variable, the value of vderef could not be x). Thus, condition 4 holds even after line 15 has been executed.

We need, however, ensure that condition 1 of the definition of a *complete* set of situated bindings will hold (which plays a role for the disentailment test).

If the term t is a construction, then there are three cases. All of them are easy to deal with (1) (Lines 18–20). The variable x is bound to a construction with a different function symbol. Then the node α and all nodes below it are inconsistent. We need remove their bindings according to the definition of a configuration in Section 6.2 (2) (Lines 21–22). The variable x is bound to a construction with the same function symbol. Then we need ensure that condition 2 of the definition of a complete set of situated bindings will hold (3) (Lines 23–26). The variable x is not bound to any construction ("x is free"). Then we only need to ensure conditions 3 and 4 in the definition of a set of situated bindings in Section 6.1.

Example from Section 2. The initial configuration (E_0, B_0, I_0) which corresponds to the execution of the Oz program given in Section 2 is given by $B_0 = \emptyset$, $I_0 = \emptyset$, and

$$E_0 = \{(x,\beta,f(y_1,y)),(y_1,\beta,a),(x,\alpha_1,f(z_1,z)),(z_1,\alpha_1,z),(y,\gamma_1,z),$$
$$(y,\gamma_2,b),(y,\alpha_2,c)\}.$$

Note that we need introduce auxiliary (existentially quantified) variables y_1 and z_1 because constructions are of the form $f(\bar{x})$ where the variables in the tuple $\bar{x} = (x_1,\dots,x_n)$ are pairwise different.

We will choose (and remove) and add elements of E in a stack-like manner. That is, the algorithm will first move $(x,\beta,f(y_1,y))$ and (y_1,β,a) from E to B (using lines 23–26). Then it will remove $(x,\alpha_1,f(z_1,z))$ from E and add (z_1,α_1,y_1) and (z,α_1,y) to B (after adding the two bindings temporarily to E, using lines 21–22).

After applying **vderef** twice, the binding (z_1,α_1,z) gets installed in B as (y_1,α_1,y). Here, line 14 is applied; i.e., (y,α_1,a) is put into E and eventually installed in B.

After applying **vderef** on z, the binding (y,γ_1,z) is simply removed from E (using lines 6–7). So the node γ_1 does not contain any bindings. Thus, in the constraint tree, the constraint in the node γ_1 is the empty conjunction, which is $\top$ (for *true*).

Using lines 18–20, we remove the binding (y,γ_2,b) from E and add γ_2 to I.

Finally, the binding (y,α_2,c) is moved from E to B, by use of lines 23–26.

Then, the outcome of the algorithm is the configuration (B,E,I) where $E = \emptyset$, $I = \{\gamma_2\}$, and

$$B = \{(x,\beta,f(y_1,y)),(y_1,\beta,a),(z_1,\alpha_1,y_1),(z,\alpha_1,y),(y_1,\alpha_1,y),(y,\alpha_1,a),(y,\alpha_2,c)\}.$$

If we eliminate the existentially quantified variables y_1 and z_1, then $x = f(a,y)$ is the constraint of the node β, $z = y \wedge y = a$ the one of α_1, $\top$ the one of γ_1, $\bot$ the one of γ_2, and $y = c$ the one of α_2. v.

Other examples. We will now give some examples in order to motivate particular lines of the algorithm. Always, we assume $\beta \leqslant \alpha \leqslant \gamma$.

The configuration with $B = \{(x,\beta,f(x)),\ (y,\beta,f(y))\}$ and $E = \{(x,\alpha,y)\}$ will lead to applications of lines 14–15 (add (x,α,y) to B and $(y,\alpha,f(y))$ to E) and lines 21–22 (add (x,α,y) to E) and lines 6–7 and then terminate. The node α is entailed since both x and y are bound to constructions.

The configuration with $B = \{(x,\beta,f(u)),\ (x,\alpha,y)\}$ and $E = \{(y,\beta,g(v))\}$ will lead to applications of lines 14–15 (add (x,α,y) to B and $(y,\alpha,f(u))$ to E and afterwards to B) and line 25 (move $(y,\alpha,f(u))$ from B to E) and then to line 18 (add α to I).

The configuration with $B = \{(y,\beta,f(v)),\ (x,\alpha,f(u))\}$ and $E = \{(y,\alpha,x)\}$ where $\mathsf{home(x)} = \alpha$ will lead to applications of line 9 (swap x and y), and then line 10–11 (remove $(x,\alpha,f(u))$ from B, add $(y,\alpha,f(u))$ to E). After adding (u,α,v) to E (by application of line 21) and then moving the binding from E to B, the algorithm terminates. The node α is not determined.

The configuration with $B = \{(w,\beta,u),\ (u,\alpha,v)\}$ and $E = \{(v,\beta,w)\}$ will lead to an application of lines 12–13 (that is, (u,α,v) is removed from B) before the installation of (v,β,w) in B.

Thus, if we put the two preceding examples together, the configuration with $B = \{(y,\beta,f(v)),\ (x,\alpha,f(u)),\ (w,\beta,u)\}$ and $E = \{(y,\alpha,x),\ (v,\beta,w)\}$ where $\mathsf{home(x)} = \alpha$ will lead to a configuration without a binding on α. That is, α is entailed.

6.4. Correctness and termination

We consider any sequence $((E_i,B_i,I_i))_{i=0,\dots,N}$ starting in a configuration (E_0,B_0,I_0) and obtained by successive execution of the body of the while loop. The proofs of the next propositions are obvious.

Proposition 5. *Each triple (E_i,B_i,I_i) is a configuration.*

Proposition 6. *If the procedure terminates in (E_N,B_N,I_N) then (E_N,B_N,I_N) is a normal configuration.*

Proposition 7. *The step from (E_i,B_i,I_i) to $(E_{i+1},B_{i+1},I_{i+1})$ is an equivalence transformation on the defined constraint trees.*

Theorem 2. *The sequence $((E_i,B_i,I_i))_{i=0,\dots,N}$ must be finite; i.e., the procedure given in Fig. 2 always terminates.*

Proof. Every configuration $(E_{i+1},B_{i+1},I_{i+1})$ is obtained from (E_i,B_i,I_i) by one of five cases inside the body of the while loop. Hence, we have one of the following possibilities.

1. $(E_{i+1},B_{i+1},I_{i+1}) = (E_i - \{(x,\alpha,x)\},B_i,I_i)$ for some variable x,
2. $I_{i+1} = I_i$, and

$B_{i+1} = B_i \cup \{(x,\alpha,y)\} - B$ where $B \subseteq \{(x,\alpha,f(\bar{u}))\} \cup \{(z,\gamma,t) \mid \alpha < \gamma\}$,

3. $I_{i+1} = I_i \uplus I$ where $I \neq \emptyset$,

4. $(E_{i+1}, B_{i+1}, I_{i+1}) = (E_i - \{(x, \alpha, f(\bar{u}))\} \cup E, B_i, I_i)$ where $E \subseteq \{(u, \gamma, v) \mid \alpha \leqslant \gamma\}$,
5. $I_{i+1} = I_i$, and $B_{i+1} = B_i \cup \{(x, \alpha, f(\bar{u}))\} - B$ where $B \subseteq \{(x, \gamma, t) \mid \alpha < \gamma\}$.

In each of these cases, $(E_{i+1}, B_{i+1}, I_{i+1}) \prec (E_i, B_i, I_i)$ where $\preccurlyeq$ is the lexicographic ordering on reversed configuration-triples, with, componentwise,

1. $I \leqslant I'$ if $I \supseteq I'$,
2. $B \leqslant B'$ if $B \sqsupseteq B'$, where $\sqsupseteq$ is the multiset ordering induced by

$$(x, \alpha, y) > (x, \alpha, f(\bar{u})) > (x, \gamma, t) \quad \text{if } \alpha < \gamma,$$

3. $E \leqslant E'$ if $E \sqsubseteq E'$, where $\sqsubseteq$ is the multiset ordering induced by

$$(z, \gamma, t) < (u, \alpha, v) < (x, \alpha, f(\bar{u})) \quad \text{if } \alpha < \gamma.$$

Since there are no infinitely decreasing $\prec$-chains, the sequence $((E_i, B_i, I_i))_{i=0,\ldots,N}$ is finite. $\square$

7. Conclusion and future work

We have given the first formal account of an algorithm for checking entailment and disentailment of deep guards. We have formulated the results in this paper for rational tree constraints; they can be adapted to finite and to feature trees.

A first conclusion one may draw is that the machinery needed for deep guards is principally not more complicated than for flat guards. The sole difference lies in the administration of the tree order for (1) the implementation of the function vderef(x, α, B), which goes over at most two levels of the computation tree in the case of flat guards, but arbitrarily many in the case of deep guards, and (2) the removal of situated bindings (x, γ, s) from nodes γ below a given node α, thus, essentially, for the test of $\leqslant$-comparison between nodes.

This work is the preliminary for (on-line) complexity analysis and for comparing different realizations of our algorithm. The difficulty seems here to determine the complexity of finding all situated bindings (x, γ, s) with $\gamma \leqslant \alpha$. It will be interesting to measure the performance of the implementations already existing in AKL and Oz, which are guard-based, against a variable-based implementation using hash-tables for the lists of situated bindings of each variable. The theoretical on-line complexity seems better for the latter which avoids re-installing multiple bindings. In the case of flat guards over rational trees, this has been shown in [13]: It has quasi-linear as opposed to quadratic cost. Interesting, though mainly theoretically, is also the problem of the optimal amortized-time complexity of the vderef(x, α, B) function, which is about path-compression for bindings which go through several nodes.

Acknowledgements

We thank Peter Van Roy for discussions. We thank the anonymous referees for insightful remarks. This work is partially supported by the ESPRIT project ACCLAIM (EP 7195). Gert Smolka has also been supported by the BMBF Project Hydra (contract ITW 9105) and the Esprit Project CCL (contract EP 6028).

References

[1] H. Aït-Kaci and A. Podelski, Towards a meaning of LIFE, in: J. Maluszyński and M. Wirsing, eds., *Proc. 3rd Internat. Symp. on Programming Language Implementation and Logic Programming*, Lecture Notes in Computer Science, Vol. 528 (Springer, Berlin, 1991) 255–274.

[2] H. Aït-Kaci and A. Podelski, Functions as passive constraints in life, *ACM Trans. Programming Languages Systems (TOPLAS)* **16** (1994) 1279–1318.

[3] H. Aït-Kaci, A. Podelski and G. Smolka, A feature-based constraint system for logic programming with entailment, *Theoret. Comput. Sci.* **122** (1994) 263–283.

[4] K.L. Clark and S. Gregory, A relational language for parallel programming, in: *Proc. ACM Conf. on Functional Programming Languages and Computer Architecture* (1981) 171–178.

[5] A. Colmerauer, Prolog II reference manual and theoretical model, Tech. Report, Groupe Intelligence Artificielle, Université Aix - Marseille II, October 1982.

[6] B. Courcelle, Fundamental properties of infinite trees, *Theoret. Comput. Sci.* **25** (1983) 95–169.

[7] S. Haridi and S. Janson, Kernel Andorra Prolog and its computation model, in: D.H.D. Warren and P. Szeredi, eds., *Proc. 7th Internat. Conf. on Logic Programming* (MIT Press, Cambridge, MA, 1990) 31–48.

[8] M. Henz, M. Mehl, M. Müller, T. Müller, J. Niehren, R. Scheidhauer, C. Schulte, G. Smolka, R. Treinen and J. Würtz, The Oz Handbook, Research Report RR-94-09, Deutsches Forschungszentrum für Künstliche Intelligenz, Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany, 1994. Available through anonymous ftp from duck.dfki.uni-sb.de.

[9] M. Henz, G. Smolka and J. Würtz, Oz – a programming language for multi-agent systems, in: Ruzena Bajcsy, ed., *13th Internat. Joint Conf. on Artificial Intelligence*, Vol. 1, Chambéry, France (Morgan Kaufmann, Los Altos, CA, 1993) 404–409.

[10] M.J. Maher, Logic semantics for a class of committed-choice programs, in: J.-L. Lassez, ed., *Proc. 4th Internat. Conf. on Logic Programming* (MIT Press, Cambridge, MA, 1987) 858–876.

[11] L. Naish, Automating control for logic programs, *J. Logic Programming* **2** (1985) 167–184.

[12] L. Naish, The Mu-Prolog 3.2db reference manual, Tech. Report, Department of Computer Science, University of Melbourne, Victoria, Australia, 1985.

[13] A. Podelski and P. Van Roy, The Beauty and the Beast algorithm: quasi-linear incremental tests of entailment and disentailment, in: *Proc. Internat. Symp. on Logic Programming (ILPS)* (MIT Press, Cambridge, MA, 1994) 359–374.

[14] V. Saraswat and M. Rinard, Concurrent constraint programming, in: *Proc. 17th ACM Conf. on Principles of Programming Languages*, San Francisco, CA (1990) 232–245.

[15] E. Shapiro, The family of concurrent logic programming languages, *ACM Comput. Surveys* **21** (1989) 413–511.

[16] G. Smolka and R. Treinen, Records for logic programming, *J. Logic Programming* **18** (1994) 229–258.