Decomposition Structures for Soft Constraint Evaluation Problems: An Algebraic Approach

Ugo Montanari[1], Matteo Sammartino[2], and Alain Tcheukam[3]

[1] University of Pisa
[2] University College London
[3] New York University, Abu Dhabi

Abstract. (Soft) Constraint Satisfaction Problems (SCSPs) are expressive and well-studied formalisms to represent and solve constraint-satisfaction and optimization problems. A variety of algorithms to tackle them have been studied in the last 45 years, many of them based on dynamic programming. A limit of SCSPs is its lack of compositionality and, consequently, it is not possible to represent problem decompositions in the formalism itself. In this paper we introduce *Soft Constraint Evaluation Problems* (SCEPs), an algebraic framework, generalizing SCSPs, which allows for the compositional specification and resolution of (soft) constraint-based problems. This enables the systematic derivation of efficient dynamic programming algorithms for any such problem.

1 Introduction

(Soft) Constraint Satisfaction Problems (SCSPs) are expressive and well-studied formalisms [20, 24] to represent and solve constraint-satisfaction and optimization problems [4]. A CSP consists of a network of hyperedges, interpreted as predicates on (variables associated to) the adjacent vertices. A *solution* is a variable assignment satisfying all the predicates (or providing a "best" level of satisfaction, in the soft version).

Finding a solution for an SCSP is in general an NP-complete problem. A variety of algorithms have been studied in the last 45 years, many of them based on *dynamic programming* [2]. Dynamic programming is a well-known method for solving optimization problems. It consists in: a) decomposing repeatedly the problem into smaller subproblems; b) solving subproblems in a bottom-up order, by combining solutions of smaller problems into those of bigger problems.

Key to the approach is the fact that repeated subproblems are only solved once. Different decompositions can have substantially different computational costs, and choosing a best one is known as *secondary optimization problem* of dynamic programming [3]. This is also an NP-complete problem. When the problem has a graphical representation, as in the case of CSPs, a class of tree-shaped structures, called *tree decompositions* [22, 19], have been used to represent dynamic programming hierarchies. The solution process corresponds to a bottom-up visit of the tree decomposition (see e.g. [12] for algorithms for CSPs based on tree decompositions).

A limit of SCSPs is the lack of compositionality and, consequently, of mechanisms to represent problem decompositions for dynamic programming in the formalism itself. In this paper we introduce a new, compositional framework for a wide class of constraint-based problems, which we call *Soft Constraint Evaluation Problems* (SCEPs), generalizing SCSPs. In this framework, both the *structure* and the *solution process* can be represented at the same time, with a formal connection between the two. This provides a correct-by-construction mechanism to decompose and solve SCEPs via dynamic programming.

SCEPs are specified via a simple syntax inspired by process algebras, with a natural interpretation in terms of constraints. As an example, the term:

$$p = (y)((x)A(x, y) \parallel (z)B(y, z))$$

represents a problem made of two constraints A and B, over x, y and y, z respectively, where $(\_)$ precedes $\_ \parallel \_$. Notice that y is shared.

The syntax is expressive enough to represent both the structure of the problem and a *decomposition* into subproblems. For instance, $A(x, y)$ being in the scope of (x) means that it must be solved w.r.t. x, which will produce a solution parametric in y. A fundamental role is played by the *axiom of scope extension*

$$(x)(p \parallel q) = (x)p \parallel q \qquad (x \text{ not free in } q)$$

which allows for the the manipulation of the subproblem structure of terms.

Given an SCEP, represented as the term p defined above, its *solution* is just the evaluation of p in a given SCEP algebra, i.e., an algebra providing an interpretation of basic constraints and operations. In other words, the solution can be computed via structural recursion on terms, using the interpreted operations. For instance, in a typical optimization problem, $\_ \parallel \_$ is interpreted as summing up each subproblem's contribution, e.g., cost, and $(x)\_$ as minimizing w.r.t. the variable x.

A key challenge here is achieving structural recursion in the presence of variable binding, such as the restriction operator (x) described above. In fact, if treated naively, variable binding leads to possibly ill-defined recursive definitions, where notions such as "free/bound variable" and "variable capture" need to be consistently taken into account. To tackle this, SCEP algebras are *permutation algebras* [15], including explicit variable permutations that enable a proper treatment of free and bound variables. This approach is equivalent to abstract syntax with binding via nominal sets (see, e.g., [21]).

The main contributions of this paper are as follows:

- In Section 3 we propose a *strong* axiomatization of SCEPs, and we present one of the main results of the paper: soundness and completeness of constraint networks w.r.t. our strong specification, namely networks form its initial algebra. Then we introduce a *weak* specification, where each term describes a specific decomposition. This enables decomposing and solving SCEPs, and in particular traditional constraint networks, in a unified framework.
- In Section 4 we show how SCSPs are an instance of SCEPs.

- In Section 5 we introduce the notion of *complexity* of term evaluation, and we characterize terms that are local optima w.r.t. complexity.
- In Section 6 we give a formal translation from tree decompositions to weak terms, which enables applying algebraic techniques to the former, and improving their complexity via the results of Section 5.
- In Section 7 we give a simple algorithm, inspired by *bucket elimination* [23, 5.2.4]. We show that our algorithm can achieve better decompositions than the latter one.
- Finally, in Section 8 we give a non-trivial example of a problem which can be represented and solved as an SCEP, but not as an SCSP.

2 Background

We recall some basic notions. A *ranked alphabet* $\mathcal{E}$ is a set equipped with an *arity* function $ar\colon \mathcal{E} \to \mathbb{N}$. A *labelled hypergraph* over a ranked alphabet $\mathcal{E}$ is a tuple $G = (V_G, E_G, a_G, lab_G)$, where: V_G is the set of vertices; E_G is the set of (hyper)edges; $a_G\colon E_G \to V_G^\star$ assigns to each hyperedge e the tuple of vertices attached to it ($V_G^\star$ is the set of tuples over V_G); $lab_G\colon E_G \to \mathcal{E}$ is a labeling function, assigning a label to each hyperedge e such that $|a_G(e)| = ar(lab_G(e))$.

Given two hypergraphs G_1 and G_2 over $\mathcal{E}$, a *homomorphism* between them is a pair of functions $h = (h_V\colon V_{G_1} \to V_{G_2}, h_E\colon E_{G_1} \to E_{G_2})$ preserving connectivity and labels, namely: $h_V \circ a_{G_1} = a_{G_2} \circ h_E$ and $lab_{G_2} \circ h_E = lab_{G_1}$. It is an *isomorphism* whenever h_V and h_E are bijections. We write $G_1 \uplus G_2$ for the component-wise disjoint union of G_1 and G_2.

2.1 Soft Constraint Satisfaction Problems

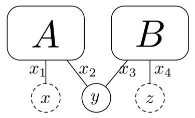
Let $\mathbb{V}$ be a denumerable set of variables and let $\mathcal{E}_C$ be a ranked alphabet of *soft constraints* (or just constraints). We assume that $\mathcal{E}_C$ also has a function $var\colon \mathcal{E}_C \to \mathbb{V}^\star$ (with $ar(A) = |var(A)|$, for all $A \in \mathcal{E}_C$), assigning a tuple of *distinct* canonical variables to each constraint. Canonical variables are such that $var(A) \cap var(B) = \varnothing$ if $A \neq B$. The structure of soft constraint problems can be described as a particular kind of hypergraphs labelled over $\mathcal{E}_C$.

Definition 1 (Concrete network). *A* concrete network *(of constraints) is a pair $I \blacktriangleright N$, where:*

- *$N = (V_N, E_N, a_N, lab_N)$ is a labelled hypergraph over $\mathcal{E}_C$ such that $V_N \subseteq \mathbb{V}$ and there are no isolated vertices, i.e., vertices v such that $v \notin a_N(e)$, for all $e \in E_N$;*
- *$I \subseteq V_N$ is a finite set of* interface variables.

In a concrete network, for every edge $e \in E_N$ we define a substitution of variables σ_e mapping component-wise the tuple of canonical variables $var(lab_N(e))$ to the actual variables $a_N(e)$ e is connected to. Hyperedges can be understood as *instances* of constraints, where canonical variables are replaced by concrete ones, describing how subproblems are connected. Interface variables are "external", in the sense that they allow networks to interact when composed.

Example 1. Let A and B be two constraints with $ar(A) = ar(B) = 2$ and $var(A) = \langle x_1, x_2\rangle$, $var(B) = \langle x_3, x_4\rangle$. Consider the labelled hypergraph N, with $V_N = \{x, y, z\}$, $E_N = \{e_1, e_2\}$, $a_N(e_1) = \langle x, y\rangle$, $a_N(e_2) = \langle y, z\rangle$, $lab_N(e_1) = A$, $lab_N(e_2) = B$. The concrete network $\{y\} \blacktriangleright N$ is depicted below:



Labels are placed inside the corresponding edge and connections to vertices are labelled with the corresponding canonical variable. Canonical variables will be often omitted in pictures of networks. Interface vertices, namely y, have solid outline, and non-interface ones, namely x and z, have dashed outline. As instantiations of the canonical to the concrete variables, we have $\sigma_{e_1} = \{x_1 \mapsto x, x_2 \mapsto y\}$, $\sigma_{e_2} = \{x_3 \mapsto y, x_4 \mapsto z\}$.

We now introduce *Soft Constraint Satisfaction Problems* (SCSPs in short) [4]. They are based on *c-semirings*, which are semirings $(S, +, \times, 0, 1)$ such that the additive operation $+$ is idempotent, 1 is its absorbing element and the multiplicative operation $\times$ is commutative.

Definition 2 (SCSP). *An SCSP is a tuple $(I \blacktriangleright N, \mathbb{D}, S, val)$ of a concrete network $I \blacktriangleright N$, a finite set $\mathbb{D}$, a c-semiring S and a set of functions $val_A \colon (var(A) \to \mathbb{D}) \to S$, one for each constraint A occurring in the network.*

In an SCSP, every constraint A is assigned a *value* val_A, that is a function giving a cost in S to every assignment in $\mathbb{D}$ of canonical variables of A. As a shorthand, for $e \in E_N$ and $A = lab_N(e)$, we write $val_e \colon (a_N(e) \to \mathbb{D}) \to S$ for the function $val_e = val_A(- \circ \sigma_e)$, giving a cost to every assignment to variables e is attached to, according to $var(A)$. Variables I are those of interest, i.e., those of which we want to know the possible assignments compatible with all the constraints. Values for each constraint are used to compute the solution for the SCSP, using the semiring operations, plus an operation of *projection* over variable assignments: given $\rho \colon X \to \mathbb{D}$ and $Y \subseteq X$, $\rho\downarrow_Y$ is the restriction of ρ to Y.

The solution is a function $sol \colon (I \to \mathbb{D}) \to S$: for each $\rho \colon I \to \mathbb{D}$

$$sol(\rho) = \sum_{\{\rho' \colon V_N \to \mathbb{D} \mid \rho'\downarrow_I = \rho\}} \big(val_{e_1}(\rho' \downarrow_{a_N(e_1)}) \times \cdots \times val_{e_n}(\rho' \downarrow_{a_N(e_n)})\big)$$

where $E_N = \{e_1, \ldots, e_n\}$. Notice that the function sol is computed via the pointwise application of semiring operations: each value function is applied to the (relevant part of the) variable assignment ρ, and then $\times$ is used on the results. In other words, $\times$ can be lifted to value functions, giving a natural interpretation of composition of two constraint networks N_1 and N_2:

$$val_{N_1} \otimes val_{N_2} = (\lambda\rho \colon (V_{N_1} \cup V_{N_2} \to \mathbb{D}) \to S).val_{N_1}(\rho\downarrow_{V_{N_1}}) \times val_{N_2}(\rho\downarrow_{V_{N_2}})$$

Example 2. SCSPs can be used to model and solve optimization problems where the goal is to minimize the total cost. Suppose we have two cost functions $A, B\colon \mathbb{D}^2 \to \mathbb{R}^+_\infty$, assigning a (possibly infinite) cost to pairs of values from a finite set $\mathbb{D}$. We want to find the minimum of $A(x,y) + B(y,z)$. This problem can be represented as a SCSP as follows. We introduce a constraint for each function, and we connect constraints to form the concrete network $\emptyset \blacktriangleright N$, where N is the hypergraph of Example 1. The interface is empty because we want to minimize w.r.t. all variables. In order to capture sums and minimization of constraints, we use the *weighted c-semiring* $S_W = (\mathbb{R}^+, \min, +, +\infty, 0)$. Then, the problem corresponds to the SCSP $(\emptyset \blacktriangleright N, \mathbb{D}, S_W, val)$, where $val(A)$ and $val(B)$ act as the functions A and B. The solution sol is a function $(\emptyset \to \mathbb{D}) \to \mathbb{R}^+$, i.e., a single value in $\mathbb{R}^+$, given by:

$$sol = \min_{d_1,d_2,d_3 \in \mathbb{D}} (A(d_1,d_2) + B(d_2,d_3))$$

which precisely computes the minimum of $A(x,y) + B(y,z)$.

In SCSPs the solution does not depend on the identity of non-interface variables, and this will also be true in our framework. We can then abstract away from those variables and take networks up to isomorphism. We say that two concrete networks $I_1 \blacktriangleright N_1$ and $I_2 \blacktriangleright N_2$ are isomorphic, written $I_1 \blacktriangleright N_1 \cong I_2 \blacktriangleright N_2$, whenever $I_1 = I_2$ and there is an isomorphism $\varphi\colon N_1 \to N_2$ such that $\varphi(x) = x$, for all $x \in I_1$.

Definition 3 (network). *A(n abstract) network $I \rhd C$ is an isomorphism class of concrete networks. We also write $I \rhd N$ to mean that $I \blacktriangleright N$ is a canonical representative of its class.*

In the following, we will depict abstract networks in the same way as concrete networks (see Example 1), implicitly assuming the choice of a canonical representative.

2.2 Tree decomposition

A decomposition of a graph can be represented as a *tree decomposition* [22, 19], i.e., a tree where each vertex is a piece of the graph. We introduce a notion of *rooted tree decomposition*. Recall that a *rooted tree* $T = (V_T, E_T)$ is a set of vertices V_T and a set of edges $E_T \subseteq V_T \times V_T$, such that there is a *root*, i.e. a vertex $r \in V_T$:

- with no ingoing edges: there are no edges (v, r) in E_T;
- such that, for every $v \in V_T$, $v \neq r$, there is a unique path from r to v, i.e., a unique sequence of edges $(r, u_1), (u_1, u_2), \ldots, (u_n, v)$, $n \geq 0$.

Definition 4 (Rooted tree decomposition of a hypergraph). *A* rooted tree decomposition *of a hypergraph G is a pair $\mathcal{T} = (T, X)$, where T is a rooted tree and $X = \{X_t\}_{t \in V_T}$ is a family of subsets of V_G, one for each vertex of T, such that:*

1. *for each vertex $v \in V_G$, there exists a vertex t of T such that $v \in X_t$;*
2. *for each hyperedge $e \in E_G$, there is a vertex t of T such that $a_G(e) \subseteq X_t$;*
3. *for each vertex $v \in V_G$, let $S_v = \{t \mid v \in X_t\}$, and $E_v = \{(x, y) \in E_T \mid x, y \in S_v\}$; then (S_v, E_v) is a rooted tree.*

We gave a slightly different definition of tree decomposition: the original one refers to a non-rooted, undirected tree. All tree decompositions in this paper are rooted, so we will just call them tree decompositions, omitting "rooted".

Tree decompositions are suited to decompose networks: we require that interface variables are located at the root.

Definition 5 (Decomposition of a network). *The decomposition of a network $I \rhd N$ is a decomposition of N rooted in r, such that $I \subseteq X_r$.*

2.3 Dynamic programming via tree decompositions

The general issue of assigning a tree-like structure to graphs and networks in order to efficiently solve optimization problems is an issue of paramount importance in optimization theory. It is known as the *dynamic programming secondary optimization problem* [3].

The dynamic programming strategy of reducing problems to subproblems needs to express optimal solutions in terms of parameters, which represent shared variables between subproblems. Such a decomposition can be formalized via a tree decomposition $\mathcal{T}$ of the graph, where each node t is a problem, its children are subproblems, and X_t are the problem's variables. The dynamic programming algorithm then is based on a bottom-up visit of the tree.

Usually, time and space requirements for computing parametric solutions are at least exponential in the number of variables. Thus the complexity of a problem is defined as the maximal number of parameters in its reductions, called *width*. Formally, we have $\texttt{width}(\mathcal{T}) = \max_{t \in T}\{|X_t|\}$. The *treewidth* of a graph is the minimal width among all of its tree decompositions[4]. If graphs in a certain class have bounded treewidth, then their complexity becomes linear in their size – possibly with a big coefficient which depends on the treewidth bound – usually a tremendous achievement. Finding the treewidth, which involves a minimization over all the decomposition of a graph, is NP-complete. Even if expensive, an efficient solution of the secondary optimization problem may be essential whenever the original problem must be solved many times with different data and thus several approaches have been proposed for solving the secondary problem approximately.

3 Soft Constraint Evaluation Problems (SCEPs)

In this section we introduce *Soft Constraint Evaluation Problems* (SCEPs). They are problems involving soft constraints, generalizing SCSPs. We work in an

[4] Width is conventionally defined as $\max_{t \in T}\{|X_t|\} - 1$. We drop "$-1$" so that it gives the actual number of parameters.

$(\mathbf{AX}_{\|})$

$p \parallel q \equiv_s q \parallel p \qquad (p \parallel q) \parallel r \equiv_s p \parallel (q \parallel r) \qquad p \parallel \mathtt{nil} \equiv_s p$

$(\mathbf{AX}_{(x)})$

$(x)(y)p \equiv_s (y)(x)p \quad (x)\mathtt{nil} \equiv_s \mathtt{nil}$

$(\mathbf{AX}_{\alpha})$

$(x)p \equiv_s (y)p[x \mapsto y] \qquad (y \notin fv(p))$

$(\mathbf{AX}_{SE})$

$(x)(p \parallel q) \equiv_s (x)p \parallel q \qquad (x \notin fv(q))$

$(\mathbf{AX}_{\pi})$

$p\, \mathsf{id} \equiv_s p \qquad (p\pi')\pi \equiv_s p(\pi \circ \pi')$

$(\mathbf{AX}^{p}_{\pi})$

$A(x_1, \ldots, x_n)\pi \equiv_s A(\pi(x_1), \ldots, \pi(x_n)) \qquad \mathtt{nil}\,\pi \equiv_s \mathtt{nil} \qquad (p \parallel q)\pi \equiv_s p\pi \parallel q\pi$

$((x)p)\pi \equiv_s (\pi(x))(p\pi)$

Fig. 1: Axioms of the strong SCEP specification.

algebraic setting: elements of the initial algebra describe the structure of SCEPs, and *evaluations* of such structure can be given in any other algebra satisfying the SCEP specification.

We write $Perm(\mathbb{V})$ for the set of permutations over $\mathbb{V}$, i.e., bijective functions $\pi\colon \mathbb{V} \to \mathbb{V}$. A *permutation algebra* is an algebra for the signature comprising all permutations and the formal equations $x\ \mathsf{id} = x$ and $(x\ \pi_1)\ \pi_2 = x\ (\pi_2 \circ \pi_1)$ (the application of a permutation is written in postfix notation). The SCEP signature equips permutation algebras with additional operators and equations.

Definition 6 (SCEP signature). *Recall that $\mathcal{E}_C$ is the ranked alphabet of constraints. The* SCEP signature *(s-signature in short) is given by the following grammar*

$$p, q := p \parallel q \ \mid\ (x)p \ \mid\ p\ \pi \ \mid\ A(\tilde{x}) \ \mid\ \mathtt{nil}$$

where $A \in \mathcal{E}_C$, $\pi \in Perm(\mathbb{V})$, $\{x\} \cup \tilde{x} \subseteq \mathbb{V}$ and $|\tilde{x}| = ar(A)$.

The *parallel composition* $p \parallel q$ represents the problem consisting of two subproblems p and q, possibly sharing some variables. The *restriction* $(x)p$ represents the fact that p has been solved w.r.t. x. The *permutation* $p\pi$ is p where variables have been renamed according to π. The *atomic SCEP* $A(\tilde{x})$ only involves an instance of the constraint A over variables $\tilde{x}$ (notice that the same variable may occur more than once in $\tilde{x}$). The constant $\mathtt{nil}$ represents the *empty problem.*

The *free* variables $fv(p)$ of p are

$$fv(p \parallel q) = fv(p) \cup fv(q) \qquad fv(x)p) = fv(p) \setminus \{x\} \qquad fv(p\pi) = \pi(fv(p))$$
$$fv(A(\tilde{x})) = \tilde{x} \qquad fv(\mathtt{nil}) = \emptyset$$

We write $v(p)$ for the set of all the variables occurring in p.

Definition 7 (Strong SCEP specification). *The* strong SCEP specification *(s-specification, in short) is formed by the signature in Definition 6 and the axioms in Fig. 1.*

The operator $\|$ forms a commutative monoid, meaning that problems in parallel can be solved in any order $(\mathbf{AX}_{\|})$. Restrictions can be α-converted $(\mathbf{AX}_{\alpha})$, i.e., the name of the variable w.r.t. which we solve the problem is irrelevant. Restrictions can also be swapped, i.e., we can solve w.r.t. variables in any order, and can be removed, whenever their scope is `nil` $(\mathbf{AX}_{(x)})$. The scope of restricted variables can be narrowed to terms where they occur free $(\mathbf{AX}_{SE})$. Notice that restriction is idempotent, namely $(x)(x)p \equiv_s (x)p$. Axioms regarding permutations say that identity and composition behave as expected $(\mathbf{AX}_{\pi})$ and that permutations distribute over syntactic operators $(\mathbf{AX}_{\pi}^{p})$. Permutations behave in a capture avoiding way, by replacing all names bijectively, including the bound one x. This can be understood as applying, at the same time, α-conversion and renaming of free variables on $(x)p$.

We assume a standard operation of definition $P(x_1, \ldots, x_n) \overset{\text{def}}{=} p$ where $x_1, \ldots, x_n$ is a sequence of distinct variables including $fv(p)$. We write $P(y_1, \ldots, y_n)$ for $p[x_1 \mapsto y_1, \ldots, x_n \mapsto y_n]$, where the substitution (not just a permutation) on p acts syntactically in a capture avoiding way. In this paper, we are interested in non-recursive (but well founded) definitions only. Definitions respect permutations, namely $P(x_1, \ldots, x_n)\pi \equiv_s P(\pi(x_1), \ldots, \pi(x_n))$.

We call *s-algebras* the algebras of the s-specification. Given an operation op in the s-specification, $op^{\mathcal{A}}$ denotes the interpretation of op in the s-algebra $\mathcal{A}$. We consider terms freely generated, modulo axioms of Fig. 1, in the style of [13], and we call them s-terms. They form an initial s-algebra $\mathcal{T}_s$. By initiality, for any s-algebra $\mathcal{A}$ and $p \in \mathcal{T}_s$, there is a unique interpretation $[\![p]\!]^{\mathcal{A}}$ of p as an element of $\mathcal{A}$, inductively defined as follows:

$$[\![p \parallel q]\!]^{\mathcal{A}} = [\![p]\!]^{\mathcal{A}} \parallel^{\mathcal{A}} [\![q]\!]^{\mathcal{A}} \qquad [\![(x)p]\!]^{\mathcal{A}} = (x)^{\mathcal{A}}\, [\![p]\!]^{\mathcal{A}} \qquad [\![p\pi]\!]^{\mathcal{A}} = [\![p]\!]^{\mathcal{A}}\, \pi^{\mathcal{A}}$$
$$[\![A(\tilde{x})]\!]^{\mathcal{A}} = A(\tilde{x})^{\mathcal{A}} \qquad [\![\mathtt{nil}]\!]^{\mathcal{A}} = \mathtt{nil}^{\mathcal{A}}$$

Here we use infix, prefix or postfix notation for functions $op^{\mathcal{A}}$ to reflect the syntax of s-terms. We use the expression *concrete* terms to indicate syntactic terms that are not considered up to axioms.

Permutations in the specification allow computing the set of "free" variables, called *(minimal) support*, in any s-algebra.

Definition 8 (Support). *Let $\mathcal{A}$ be an s-algebra. We say that a finite $X \subset \mathbb{V}$* supports *$a \in \mathcal{A}$ whenever, for all permutations π acting as the identity on X, we have $a\pi^{\mathcal{A}} = a$. The minimal support $\mathtt{supp}(a)$ is the intersection of all sets supporting a.*

For instance, given an s-term $p \in \mathcal{T}_s$, $p\pi^{\mathcal{T}_s}$ applies π to all free names of p in a capture avoiding way. It is easy to verify that $\mathtt{supp}(p) = fv(p)$.

An important property of SCEP algebras, following from the theory of permutation algebras, is that $[\![p]\!]^{\mathcal{A}}$ depends on (at most) the free variables of p, formally:

Lemma 1. $\mathtt{supp}([\![p]\!]^{\mathcal{A}}) \subseteq \mathtt{supp}(p)$*, for all s-terms p and s-algebras $\mathcal{A}$.*

3.1 Weak specification

Our syntax is expressive enough to describe both the problem's structure and its decomposition into subproblems. For instance, the structurally congruent concrete terms

$$(y)(x)(z)(A(x,y) \parallel B(y,z)) \qquad (y)((x)A(x,y) \parallel (z)B(y,z))$$

are equivalent s-terms, and so they describe the same problem, but the information about which subproblems to solve w.r.t. x and z, represented as the subterms in the scope of (x) and (z), is different. To distinguish different decompositions, we introduce a *weak* SCEP specification where $(\mathbf{AX}_{SE})$ is dropped to avoid the rearrangement of restrictions.

Definition 9 (Weak SCEP specification). *The* weak SCEP specification *(w-specification, in short), is the s-specification without* $(\mathbf{AX}_{SE})$*, and where the axiom* $(x)\mathtt{nil} \equiv_s \mathtt{nil}$ *is replaced with*

$$(\mathbf{AX}^{w}_{(x)}) \qquad (x)p \equiv_w p \qquad (x \notin fv(p)) \ .$$

The axiom $(\mathbf{AX}^{w}_{(x)})$ is needed to discard "useless" variables. In the s-specification, it can be derived using other axioms, including $(\mathbf{AX}_{SE})$. This is not possible in the w-specification, so we need to state it explicitly.

Algebras of the w-specification are called w-algebras and the terms modulo its axioms are called w-terms, forming the initial w-algebra; w-terms can be understood as networks having a *hierarchical structure*, made of scopes determined by restrictions. We are interested in two forms of w-terms.

Definition 10 (Normal and canonical forms). *A w-term is said to be in* normal form *whenever it is of the form* $(\tilde{x})(A_1(\tilde{x}_1) \parallel A_2(\tilde{x}_2) \parallel \cdots \parallel A_n(\tilde{x}_n))$*, where* $\tilde{x} \subseteq \tilde{x}_1 \cup \cdots \cup \tilde{x}_n$*. It is in* canonical form *whenever it is obtained by the repeated application of the directed version of* $(\mathbf{AX}_{SE})$*:* $(x)(p \parallel q) \to (x)p \parallel q$ $(x \notin fv(q))$ *until termination. For both forms, we assume that subterms of the form* $(\tilde{x})\mathtt{nil}$ *(where* $\tilde{x}$ *may be empty) are removed using* $(\mathbf{AX}_{(x)})$ *and* $(\mathbf{AX}_{\parallel})$*.*

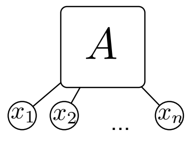
Normal and canonical forms exist in both concrete (no axioms) and abstract (up to weak axioms) versions. Normal and canonical forms are somewhat dual: normal forms have all restrictions at the top level, whereas in canonical forms every restriction (x) is as close as possible to the atomic terms where x occurs. Notice that an s-term may have more than one canonical form, whereas normal forms are unique (both up to w-specification axioms).

3.2 Soundness and completeness of networks

We now show that networks form an s-algebra, and that this algebra is isomorphic to $\mathcal{T}_s$. In other words, we show that the s-specification is sound and complete w.r.t. networks.

Theorem 1. *Let $\mathcal{N}$ be the smallest algebraic structure defined as follows. Constants are:*

$$A^{\mathcal{N}}(x_1, x_2, \ldots, x_n) =$$



$$\mathtt{nil}^{\mathcal{N}} = \emptyset \rhd 1_N$$

and operations are:

$$(I \rhd N)\pi^{\mathcal{N}} = \pi(I) \rhd N_\pi \qquad (x)^{\mathcal{N}}(I \rhd N) = I \setminus \{x\} \rhd N$$
$$I_1 \rhd N_1 \parallel^{\mathcal{N}} I_2 \rhd N_2 = I_1 \cup I_2 \rhd N_1 \uplus_{I_1,I_2} N_2$$

where: N_π is N where each vertex v is replaced with $\pi(v)$; $N_1 \uplus_{I_1,I_2} N_2$ is the disjoint union of N_1 and N_2 where vertices in $I_1 \cup I_2$ with the same name are identified; and 1_N is the network with no vertices and edges. Then $\mathcal{N}$ is an s-algebra.

Even if not depicted, when the same variable x occurs twice in $A(x_1, x_2, \ldots, x_n)$, the corresponding hyperedge has two tentacles connected to the same vertex x. Theorem 1 implies that there is a unique evaluation of s-terms: given p, the corresponding network $[\![p]\!]^{\mathcal{N}}$ can be computed by structural recursion. We show that any network is the evaluation of an s-term. In order to do this, we first give translations between concrete networks and s-terms in normal forms over the same set of variables, which will also be useful later.

Definition 11 (Translation functions). *Let $I \blacktriangleright N$ be a concrete network. Let $e_1, \ldots, e_n$ be its edges, and let $A_i = lab_N(e_i)$, $\tilde{x}_i = a_N(e_i)$. Then we define*

$$\mathtt{term}(I \blacktriangleright N) = (V_N \setminus I)(A_1(\tilde{x}_1) \parallel \cdots \parallel A_n(\tilde{x}_n))$$

Vice versa, given a concrete term in normal form $p = (\tilde{x})(A_1(\tilde{x}_1) \parallel \cdots \parallel A_n(\tilde{x}_n))$ we define $\mathtt{net}(p) = fv(p) \blacktriangleright N_p$, where:

- *$V_{N_p} = v(p)$;*
- *$E_{N_p} = \{e^{(i)}_{A_i(\tilde{x}_i)} \mid A_i(\tilde{x_i})$ is an atomic subterm of $p\}$;*
- *a_{N_p} and lab_{N_p} map $e^{(i)}_{A_i(\tilde{x}_i)}$ to $\tilde{x}_i$ and A_i, respectively.*

Notice that we assume an indexing on atomic subterms of p. This allows $\mathtt{net}$ to map two identical subterms to different edges.

Example 3. Consider the term in normal form $p = (x)(z)(A(x, y) \parallel B(y, z))$, then $\mathtt{net}(p)$ is the concrete network depicted in Example 1.

Completeness is a consequence of the following theorem.

Theorem 2. *Given two s-terms in normal form n_1 and n_2, if $\mathtt{net}(n_1) \cong \mathtt{net}(n_2)$ then $n_1 \equiv_s n_2$. As a consequence, $[\![p_1]\!]^{\mathcal{N}} = [\![p_2]\!]^{\mathcal{N}}$ implies $p_1 \equiv_s p_2$, for any two s-terms.*

4 SCSPs as SCEPs

We now show how SCSPs are represented and solved as SCEPs. Consider the SCSPs definable over a fixed c-semiring S, a fixed domain of variable assignments $\mathbb{D}$ and a fixed family of value functions val_A, one for each atomic constraint. SCEPs for such SCSPs can be defined as follows: networks are the underlying ones of SCSPs, and the SCEP algebra for evaluations is formed by value functions. Here by value function we mean functions of the form $(\mathbb{V} \to \mathbb{D}) \to S$. This is different from Section 2.1, where the domain of value functions are variable assignments $I \to \mathbb{D}$, with I a finite set. We will see that the new formulation is equivalent, and allows for simpler algebraic operations, because they do not depend on the "types" of assignments.

Theorem 3. *Let $\mathcal{V}$ be the smallest algebraic structure defined as follows. For any $\rho \colon \mathbb{V} \to \mathbb{D}$, constants are:*

$$A^{\mathcal{V}}(x_1, x_2, \ldots, x_n)\rho = val_A(\rho \downarrow_{\{x_1,x_2,\ldots,x_n\}} \circ \hat{\sigma}) \qquad \mathtt{nil}^{\mathcal{V}}\rho = 1$$

and operations are:

$$((x)^{\mathcal{V}}\phi)\rho = \sum_{d \in \mathbb{D}} \phi(\rho[x \mapsto d]) \qquad (\phi\pi^{\mathcal{V}})\rho = \phi(\rho \circ \pi) \qquad (\phi_1 \parallel^{\mathcal{V}} \phi_2)\rho = \phi_1\rho \times \phi_2\rho$$

where $\hat{\sigma}$ maps $var(A)$ to $\langle x_1, x_2, \ldots, x_n \rangle$, component-wise. Then $\mathcal{V}$ is an s-algebra.

Notice that $\parallel^{\mathcal{V}}$ is the extension of the $\otimes$ operator of Section 2.1 to arbitrary value functions, but it is simpler: projections are not needed here, because variable assignments all have the same type, namely $\mathbb{V} \to \mathbb{D}$.

Now we show that the evaluation function $[\![-]\!]^{\mathcal{V}}$, applied to a network $I \rhd N$, gives the solution of the SCSP defined over that network. Notice that $[\![I \rhd N]\!]^{\mathcal{V}}$ has type $(\mathbb{V} \to \mathbb{D}) \to S$, but its domain should be of the form $I \to \mathbb{D}$. However, $[\![I \rhd N]\!]^{\mathcal{V}}$ has the following property.

Property 1 (Compactness). We say that $\phi \colon (\mathbb{V} \to \mathbb{D}) \to S$ is *compact* if $\rho \downarrow_{\mathtt{supp}(\phi)} = \rho' \downarrow_{\mathtt{supp}(\phi)}$ implies $\phi\rho = \phi\rho'$, for all $\rho, \rho' \colon \mathbb{V} \to \mathbb{D}$.

Now, by Lemma 1, we have $\mathtt{supp}([\![I \rhd N]\!]^{\mathcal{V}}) \subseteq \mathtt{supp}(I \rhd N) = I$. Therefore compactness means that $[\![I \rhd N]\!]^{\mathcal{V}}$ only depends on assignments to interface variables. The interpretation of constants is clearly compact and, by structural induction, we can show that compound terms are. We have our main result.

Theorem 4. *Given an SCSP with underlying network $I \rhd N$ and value functions val_A, we have that $I \rhd N$ evaluated in $\mathcal{V}$, namely $[\![I \rhd N]\!]^{\mathcal{V}}$, is its solution.*

We stress that SCEPs are more general than SCSPs: an example will be shown in Section 8.

5 Evaluation complexity

Although all the s-terms corresponding to the same network have the same evaluation in any algebra $\mathcal{A}$, different ways of computing such an evaluation, represented as different w-terms, may have different computational costs. As already mentioned, finding the best one amounts to giving a solution for the secondary optimization problem.

We introduce a notion of complexity of w-terms to measure the computational costs of such evaluations.

Definition 12. *Given a w-term p, its complexity $\langle\!\langle p \rangle\!\rangle$ is defined as follows:*

$$\langle\!\langle p \parallel q \rangle\!\rangle = \max\{\langle\!\langle p \rangle\!\rangle, \langle\!\langle q \rangle\!\rangle, |fv(p \parallel q)|\} \qquad \langle\!\langle (x)p \rangle\!\rangle = \langle\!\langle p \rangle\!\rangle \qquad \langle\!\langle p\pi \rangle\!\rangle = \langle\!\langle p \rangle\!\rangle$$
$$\langle\!\langle A(\tilde{x}) \rangle\!\rangle = |set(\tilde{x})| \qquad \langle\!\langle \mathtt{nil} \rangle\!\rangle = 0$$

The complexity of p is the maximum "size" of elements of $\mathcal{A}$ computed while inductively constructing $[\![p]\!]^{\mathcal{A}}$, the size being given by the number of variables in the support. Notice that all the concrete terms corresponding to the same abstract w-term have the same complexity.

Example 4. Consider the w-terms from Section 3.1

$$p = (y)(x)(z)(A(x,y) \parallel B(y,z)) \qquad q = (y)((x)A(x,y) \parallel (z)B(y,z)).$$

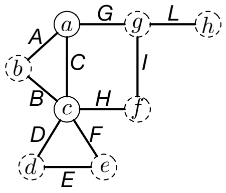
Even though they are s-congruent, and thus represent the same problem, we have $\langle\!\langle p \rangle\!\rangle = 3$ and $\langle\!\langle q \rangle\!\rangle = 2$. In fact, in order to evaluate p in any algebra, one has to evaluate $A(x,y) \parallel B(y,z)$, and then solve it w.r.t. all its variables. Intuitively, $A(x,y) \parallel B(y,z)$ is the most complex subproblem one considers in p, with 3 variables, hence $\langle\!\langle p \rangle\!\rangle = 3$. Instead, the evaluation of q requires solving $A(x,y)$ and $B(y,z)$ w.r.t. x and z, which are problems with 2 variables, and then putting the resulting partial solutions in parallel. The solution process for q never considers subproblems with more than 2 variables, hence $\langle\!\langle q \rangle\!\rangle = 2$.

The soundness of this definition follows from Lemma 1: if $[\![p']\!]^{\mathcal{A}}$ is computed while constructing $[\![p]\!]^{\mathcal{A}}$, we have $\mathtt{supp}([\![p']\!]^{\mathcal{A}}) \subseteq \mathtt{supp}(p')$, and this relation among supports does not depend on the choice of $\mathcal{A}$. The interesting cases are $(x)p$ and $p \parallel q$: the computation of $[\![(x)p]\!]^{\mathcal{A}}$ relies on that of $[\![p]\!]^{\mathcal{A}}$, whose support may be bigger, so we set the complexity of $(x)p$ to that of p; computing $[\![p \parallel q]\!]^{\mathcal{A}}$ requires computing $[\![p]\!]^{\mathcal{A}}$ and $[\![q]\!]^{\mathcal{A}}$, but the support of the resulting element of $\mathcal{A}$ is (at most) the union of those of p and q, so we have to find the maximum value among $\langle\!\langle p \rangle\!\rangle$, $\langle\!\langle q \rangle\!\rangle$ and the overall number of free variables.

Complexity is well-defined only for w-terms, because applying $(\mathbf{AX}_{SE})$ may change the complexity. Indeed, we have the following results for w-terms.

Lemma 2. *Given $(x)(p \parallel q)$, with $x \notin fv(q)$, we have $\langle\!\langle (x)p \parallel q \rangle\!\rangle \leq \langle\!\langle (x)(p \parallel q) \rangle\!\rangle$.*

As an immediate consequence, all the canonical forms of a term always have lower or equal complexity than the normal form.



(a) Network

$$\begin{tikzcd}
 & t_1, X_{t_1} = \{a,c,f\} \arrow[dl] \arrow[d] \arrow[dr] & \\
t_2, X_{t_2} = \{a,b,c\} & t_3, X_{t_3} = \{c,d,e\} & t_4, X_{t_4} = \{a,f,g\} \arrow[d] \\
 & & t_5, X_{t_5} = \{g,h\}
\end{tikzcd}$$

(b) One of its tree decompositions

Fig. 2: Example network and tree decomposition.

Theorem 5. *Given a term p, let n be its normal form. Then, for all canonical forms c of p we have $\langle\!\langle c \rangle\!\rangle \leq \langle\!\langle n \rangle\!\rangle$.*

Of course, different canonical forms may have different complexities. However, due to Lemma 2, canonical forms may be considered as *local minima* of complexity w.r.t. the application of axioms of the strong specification.

6 Tree decompositions as w-terms

In this section we provide a translation from tree decompositions to w-terms. This enables applying algebraic techniques to tree decompositions, and improving their complexity by bringing the corresponding w-terms in canonical form.

Given a network $I \rhd N$, let $\mathcal{T} = (T, X)$ be one of its tree decompositions. Its *completed* version $\mathcal{CT} = (\mathcal{T}, \{t_x\}_{x \in E_N \cup V_N})$ explicitly associates components of N to vertices of T: for each $v \in V_N$ (resp. $e \in E_N$), t_v (resp. t_e) is the vertex closest to the root of T such that $v \in X_{t_v}$ (resp. $a_N(e) \subseteq X_{t_e}$). By the definition of rooted tree decomposition (Definition 4), such vertices t_x exist (properties 1 and 2), and can be characterized as the roots of the subtrees of T induced by x (by $a_N(x)$, if x is an edge), according to property 3.

We now translate $\mathcal{CT}$ into a w-term. Given a vertex t of T, let

$$V(t) = \{v \in V_N \mid t_v = t\} \qquad E(t) = \{e \in E_N \mid t_e = t\} \ .$$

Suppose t has children $t_1, \dots, t_n$ and $E(t) = \{e_1, \dots, e_k\}$, with $n, k \geq 0$. Let $\tilde{x} = V(t) \setminus I$. The w-term $\chi(t)$ is inductively defined as follows:

$$\chi(t) = (\tilde{x})(A_1(\tilde{x}_1) \parallel \cdots \parallel A_k(\tilde{x}_k) \parallel \chi(t_1) \parallel \cdots \parallel \chi(t_n))$$

where $A_i = lab_N(e_i)$ and $\tilde{x}_i = a_N(e_i)$. When $k = 0$ and/or $n = 0$, the corresponding part of the parallel composition degenerates to `nil`. We assume that subterms of the form $(\tilde{x})$`nil` are removed via $(\mathbf{AX}_{(x)})$ and $(\mathbf{AX}_{\parallel})$.

Example 5. Consider the network in Fig. 2a, whose underlying graph is taken from [6]. A tree decomposition for it is shown in Fig. 2b. Recall that interface variables have solid outline, namely they are a and c. Its completed version has: $t_a = t_c = t_f = t_1$, $t_b = t_2$, $t_e = t_d = t_3$, $t_h = t_5$ and $t_g = t_4$, $t_{(a,b)} = t_2$, $t_{(a,c)} = t_1$,

$t_{(a,g)} = t_4$, $t_{(b,c)} = t_2$, $t_{(c,d)} = t_3$, $t_{(c,e)} = t_3$, $t_{(c,f)} = t_1$, $t_{(d,e)} = t_3$, $t_{(f,g)} = t_4$, $t_{(g,h)} = t_5$. Therefore we have

$$\begin{aligned}
\chi(t_1) &= (f)(C(a,c) \parallel H(c,f) \parallel \chi(t_2) \parallel \chi(t_3) \parallel \chi(t_4)) \\
\chi(t_2) &= (b)(A(a,b) \parallel B(b,c)) \\
\chi(t_3) &= (e)(d)(D(c,d) \parallel E(d,e) \parallel F(c,e)) \\
\chi(t_4) &= (g)(I(f,g) \parallel G(a,g) \parallel \chi(t_5)) \\
\chi(t_5) &= (h)L(g,h)
\end{aligned}$$

Again, notice that interface variables a and c are not restricted in $\chi(t_1)$.

Definition 13 (`wterm`). *Given a tree decomposition $\mathcal{T}$ rooted in r, the corresponding w-term* $\texttt{wterm}(\mathcal{T})$ *is $\chi(r)$ computed on the completed version of $\mathcal{T}$.*

We have that $\texttt{wterm}(\mathcal{T})$ correctly represents the network $\mathcal{T}$ decomposes.

Proposition 1. *Let $\mathcal{T}$ be a rooted tree decomposition for $I \rhd N$. Then $[\![\texttt{wterm}(\mathcal{T})]\!]^{\mathcal{N}} = I \rhd N$.*

We now have one of our main results, relating the width of $\mathcal{T}$ and the complexity of the corresponding w-term.

Proposition 2. *Given a tree decomposition $\mathcal{T}$, $\langle\!\langle \texttt{wterm}(\mathcal{T}) \rangle\!\rangle \leq \texttt{width}(\mathcal{T})$.*

7 Computing canonical decompositions

We now give a simple algorithm to compute canonical term decompositions. The algorithm is shown in Fig. 3. It is based on *bucket elimination* [23, 5.2.4], also known as adaptive consistency. However, we will show that bucket elimination may also produce non-canonical decompositions, whereas our algorithm produces all and only canonical terms.

Bucket elimination works as follows. Given a CSP network of constraints, its variables are ordered, and constraints are partitioned into buckets: each constraint is placed in the bucket of its last variable in the order. At any step the bucket of the last variable, say x, is eliminated by synthesising a new constraint involving all and only the variables in the bucket different than x. This constraint is put again in the bucket of its last variable. The solution is produced when the last bucket is eliminated. Notice that one can also eliminate a subset of the variables, and obtain a solution parametric in the remaining variables.

In our algorithm, putting a constraint in the bucket of its last variable corresponds to applying the scope extension axiom. The algorithm takes an s-term in normal form as input, represented as $(R)A$, where A is a multiset of atomic terms and R is the set of variables to be eliminated. This notation amounts to taking the term up to weak axioms. A total order on R is given as input as well. The algorithm operates as follows. It picks the max variable (line 3) and partitions the input w-term into subterms according to whether the chosen

Inputs: s-term $(R)A$ in normal form; a total order O_R over R.
Output: w-term P in canonical form.

```
1   $P \leftarrow (R)A$
2   **while** $O_R \neq \emptyset$
3       $x \leftarrow$ extract $\max O_R$
4       $O_R \leftarrow O_R \setminus \{x\}$
5       find all terms $A' \subseteq A$ such that $x \in fv(A')$
6       **if** $A' = \{(R')P'\}$ where $P'$ has no top-level restriction
7           $Q \leftarrow$ call the algorithm on $(x)P'$ with order $\{(x, x)\}$
8           $P'' \leftarrow (R')Q$
9       **else** $P'' \leftarrow (x)A'$
10      $P \leftarrow (R \setminus \{x\})A \setminus A' \cup \{P''\}$
11  **return** $P$
```

Fig. 3: Algorithm to compute canonical w-terms: P, P', P'' and Q denote w-terms, R and R' are sets of restricted variables, and A, A' are multisets of atomic or restriction-rooted w-terms.

variable occurs free or not (line 5). When line 5 returns a singleton $\{(R')P'\}$, the algorithm attempts at pushing the variable x further inside P', achieving the same effect as $(\mathbf{AX}_{SE})$. This is done by first calling the algorithm on $(x)P'$ and then restricting R' in the resulting term. This operation can be understood as a sequence of restriction swaps that bring x closer to P'. We have that the algorithm returns all and only the canonical forms of $(R)A$.

Theorem 6. *C is a canonical form of $(R)A$ if and only there is O_R^C such that the algorithm in Fig. 3 with inputs $(R)A$ and O_R^C outputs C.*

It is easy to see that the worst case complexity for the algorithm is given by the product of the number of variables by the number of atomic terms. In fact, this is the maximal number of times the test $x \in fv(A')$ is executed in line 5. The same worst case complexity holds for the ordinary bucket algorithm. However, for every total ordering assigned to variables, the complexity of the canonical form produced by our algorithm is lower or equal than that of the bucket elimination algorithm.

Example 6. Let us apply the algorithm to the following term in normal form:

$$P = (\{x_1, x_2, x_3, x_4\})\{A(x_1, x_2), B(x_1, x_4), C(x_1, x_3), D(x_3, x_4)\}$$

with $O_R = x_4 < x_3 < x_2 < x_1$. Line 3 picks x_1 and line 5 gives $A' = \{A(x_1, x_2), B(x_1, x_4), C(x_1, x_3)\}$. As A' is not a singleton, P becomes

$$(\{x_2, x_3, x_4\})\{D(x_3, x_4), (x_1)\{A(x_1, x_2), B(x_1, x_4), C(x_1, x_3)\}\} \ .$$

In the next iteration x_2 is picked from O_R, and we have $A' = (x_1)\{A(x_1, x_2), B(x_1, x_4), C(x_1, x_3)\}$. Now A' is a singleton, so the algorithm is called on

$$(x_2)\{A(x_1,x_2),B(x_1,x_4),C(x_1,x_3)\}$$

with $\{(x_2,x_2)\}$ order. The restriction (x_2) is pushed further inside, and the term

$$\{(x_2)A(x_1,x_2),\{B(x_1,x_4),C(x_1,x_3)\}\}$$

is returned. Line 8 will prepend (x_1) to the term above, and line 9 will construct the following term

$$(\{x_3,x_4\})\{D(x_3,x_4),(x_1)\{(x_2)A(x_1,x_2),\{B(x_1,x_4),C(x_1,x_3)\}\}\}.$$

which is then returned. The next two iterations will pick x_3 and x_4, and the **then** and **else** cases of line 6 are executed respectively. In the end we get the term (in usual notation):

$$C=(x_4)(x_3)(D(x_3,x_4)\parallel(x_1)((x_2)A(x_1,x_2)\parallel B(x_1,x_4)\parallel C(x_1,x_3)))$$

Bucket elimination corresponds to always executing line 9, even when A' is a singleton. In this case the result would be:

$$P'=(x_4)(x_3)(D(x_3,x_4)\parallel(x_2)(x_1)(A(x_1,x_2)\parallel B(x_1,x_4)\parallel C(x_1,x_3)))$$

which is not in canonical form and has worse complexity. In fact, we have $\langle\!\langle C\rangle\!\rangle=3<\langle\!\langle P'\rangle\!\rangle=4$.

8 Example

In this section we present an example of an optimization problem which is an SCEP and cannot be represented as an SCSP.

Consider a social network, based on an overlay network, where certain meeting activities for a group of sites require the existence of routing paths between every pair of collaborating sites. Under the assumption that the network is composed of end-to-end two-way connections with independent probabilities of failure, we want to find the probability of a given group of sites staying connected.

We formalize the problem as an SCEP as follows. We consider networks that are undirected, binary graphs with no loops (but possibly with circuits), modelling the overlay network. Each edge has an associated probability of failure. The solution of the problem is the probability of some interface vertices staying connected. To achieve this, the idea is evaluating networks $I\rhd N$ into an algebra of probability distributions P on the partitions $Part(I)$ of I. Thus every partition of I, characterizing a certain level of connectivity, is assigned a probability. Consequently, if J is the group of sites we are interested in and N is the hypergraph for the whole network, then the solution is obtained by computing the probability distribution P for $J\rhd N$ and by selecting $P(\{J\})$. Notice that the size of the values of our algebra grows very rapidly with the cardinality n of I. In fact, the number of possible partitions for a set of n elements is the

Bell number, inductively given by $B_0 = 1$, $B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k$. Thus if a vector representation is chosen, the amount of memory needed to represent a value of the algebra grows very rapidly with the number of interface vertices.

We now define the evaluation from networks and we show that it induces an s-algebra. For the case of constants, we assume for simplicity that we have two kinds of edges: A-labelled ones (more reliable) and B-labelled ones (less reliable), both with two vertices x, y. Given $\Pi_1 = \{\{x\}, \{y\}\}$ and $\Pi_2 = \{\{x, y\}\}$, we have

$$\begin{aligned} \llbracket I \rhd N_A \rrbracket^{\mathcal{D}} \Pi_1 &= q_A \quad & \llbracket I \rhd N_A \rrbracket^{\mathcal{D}} \Pi_2 &= 1 - q_A \\ \llbracket I \rhd N_B \rrbracket^{\mathcal{D}} \Pi_1 &= q_B \quad & \llbracket I \rhd N_B \rrbracket^{\mathcal{D}} \Pi_2 &= 1 - q_B \end{aligned}$$

where N_A (resp. N_B) is a network with a single A-labelled (resp. B-labelled) hyperedge, and q_A (resp. q_B) is the probability of the former (resp. latter) hyperedge failing, i.e., of x and y being in different sets of the partition. We have $\texttt{nil}^{\mathcal{D}}\emptyset = 1$. Permutations are defined straightforwardly:

$$\llbracket I \rhd N\pi \rrbracket^{\mathcal{D}} \Pi = \llbracket I \rhd N \rrbracket^{\mathcal{D}} \Pi\pi^{-1} \ ,$$

where $\Pi \in Part(I\pi)$. Permutations are applied to sets and partitions in the obvious way. Parallel composition is more complicated:

$$\llbracket I_1 \rhd N_1 \parallel I_2 \rhd N_2 \rrbracket^{\mathcal{D}} \Pi = \sum_{\{(\Pi_1, \Pi_2) \mid \Pi_1 \cup \Pi_2 = \Pi\}} \llbracket I_1 \rhd N_1 \rrbracket^{\mathcal{D}} \Pi_1 \times \llbracket I_2 \rhd N_2 \rrbracket^{\mathcal{D}} \Pi_2.$$

where $\Pi \in Part(I_1 \cup I_2)$, and each Π_1,Π_2 must belong to $Part(I_1)$ and $Part(I_2)$, respectively. Here the union operation $\cup$ produces the finest partition coarser than the two components and $\times$ is the multiplication on reals. The last operation is restriction:

$$\llbracket (x)I \rhd N \rrbracket^{\mathcal{D}} \Pi = \sum_{\{\Pi' \in Part(I \cup \{x\}) \mid \Pi' - x = \Pi\}} \llbracket I \rhd N \rrbracket^{\mathcal{D}} \Pi'$$

where $\Pi' - x$ removes x from its set in Π'. Here probability values are accumulated for all the cases where a certain partition of interface vertices is guaranteed, independently of the set where variable x is located.

Theorem 7. *The image of* $\llbracket - \rrbracket^{\mathcal{D}}$ *is an s-algebra.*

As a family of overlay networks we choose *wheels* of N vertices where each vertex is also connected to a central *control* vertex. Accordingly, connections in the ring have low failure probability (label A), while the connections to the center have high failure probability (label B). We want to find out how much the connection probability between two adjacent vertices in the ring deteriorates when the direct link between them breaks down.

The formal definition of our networks is given in Fig. 4. They consist of radius elements R_i, recursively composed in parallel; rings are closed ($W_k(v, x)$) by connecting the last (v) and the first (x) radius; the failed network is $FW_k(v, x)$ where the ring is interrupted because $A(v, x)$ is missing. Fig. 4 shows $W_2(v, x)$.

$$
\begin{aligned}
R_0(x,y,z) &= A(x,y) \parallel B(x,z)\\
R_{i+1}(x,y,z) &= (v)(R_i(x,v,z) \parallel R_i(v,y,z))\\
W_k(v,x) &= (z)(R_k(x,v,z) \parallel A(v,x) \parallel B(v,z))\\
FW_k(v,x) &= (z)(R_k(x,v,z) \parallel B(v,z))
\end{aligned}
$$

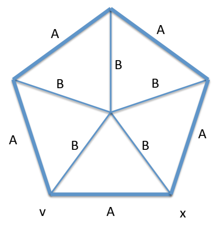


Fig. 4: Formal specification of a *wheel* network and depiction of W_2.

It is easy to see that $W_k(v,x)$ is a wheel with $N = 2^k + 1$ radii, which is specified by a number of simple well-founded non-recursive defining equations linear in k. The top-down recursive evaluation of $W_k(v,x)$ is clearly exponential in k. The complexity of bucket elimination is the same. However, the bottom-up dynamic programming evaluation is much more efficient: its complexity is linear in k and logarithmic in the size N of the problem, thanks to the presence of repetitive subterms.

8.1 Non-existence of a SCSP formulation

As mentioned, the problem does not fit the SCSP format. To show why, given the SCEP defined above, let us try to construct an equivalent SCSP.

We can safely assume that the network $I \rhd N$ is the same in both cases. The carrier of our algebra consists of the probability distributions on the partitions $Part(I)$ of the interface variables I of the network. To fit the SCSP definition, a partition in $Part(I)$ can be represented as (the kernel of) an assignment of variables I. Thus the solution function $sol : D(Part(I))$ computes the probability $sol(\Pi)$ associated to a given partition Π of interface variables. Without discussing how to impose a semiring structure on probabilities, notice that the solution in the SCSP case, for any two networks N_1 and N_2 whose union is N, is given by $val_{N_1}(\Pi_1) \otimes val_{N_2}(\Pi_2)$, where Π_1 and Π_2 are the restrictions (projections) of Π to the vertices of N_1 and N_2, respectively. The solution only examines the probabilities caused by *the same* Π on the two sub-networks. This limitation is incompatible with the definition of parallel composition in our example, where to compute the outcome of a resulting partition in the composed network, the probabilities must be considered computed by all pairs of partitions in the component networks whose union (as described earlier when defining $\parallel^{\mathcal{D}}$) is the given partition.

8.2 Implementation

The main issue in the implementation is how to represent the values of the domain and how to implement the operations. Probability distributions can be represented as vectors indexed by the partitions of the set of the interface

k	N	A	B	F	msec	W	msec	A	B	F	msec	W	msec
1	3	0.01	0.1	0.00217	17	0.00002	22	0.1	0.3	0.07043	17	0.00704	19
2	5	0.01	0.1	0.03154	78	0.00031	105	0.1	0.3	0.30817	74	0.03081	80
3	9	0.01	0.1	0.0697	183	0.00069	190	0.1	0.3	0.54609	172	0.00546	184
4	17	0.01	0.1	0.14157	409	0.00141	426	0.1	0.3	0.8046	435	0.00804	452
5	33	0.01	0.1	0.26908	620	0.00269	623	0.1	0.3	0.96379	625	0.09637	661

Table 1: Example values.

vertices, which grow very rapidly with the number of vertices. To allow for fast insertion and retrieval, it is convenient to represent partitions as strings and to order them. A simple representation starts ordering the vertices within the sets of vertices, and eventually the sets of vertices in the partitions according to their first element. It is interesting to observe that it is convenient not to represent sets of vertices which are singletons. Omitting them makes partitions untyped, and thus simplifies the computation of parallel composition.

A natural way to compute parallel composition takes all pairs (Π_1, Π_2) of partitions, determines $\Pi_1 \cup \Pi_2 = \Pi$ and increments by $p_1\Pi_1 \times p_2\Pi_2$ the entry of Π in the result. The union can be computed efficiently with merge-find-like algorithms, thus the cost of multiplication is essentially quadratic with the number of partitions. Similarly, the cost of $(x)p$ is essentially linear with the number of partitions of p: every value $p\Pi$ increments the entry $\Pi - x$ of the result.

We ran experiments on a 2.2 GHz Intel Core i7 with 4 GB RAM. In Table 1 we see the connection probability between v and x with and without failure (i.e. for F_k and W_k) for various values of k, together with the corresponding computing time. Each case is computed for failure probabilities $q_A = 0.01, q_B = 0.1$, and $q_A = 0.1, q_B = 0.3$. Notice that it is always the case that failure probability for W_k equals the product of the failure probability for F_k and of q_A. This is obvious, since the edge A and the network F_k are composed in parallel to obtain W_k, and thus their failure probabilities should be multiplied.

9 Conclusion

We have presented a class of constraint algebras, which generalize SCSPs. Vertices of constraint networks are implicitly represented as support elements of a permutation algebra. This allows for the evaluation of terms of the algebras in rather abstract domains. Applying directionally the scope extension axiom until termination yields terms for efficient dynamic programming strategies. An example has also been shown about computing the connection probability of communication networks. This problem can be represented using our algebras, but not as an SCSP.

Our framework is a significant step towards the use of existing techniques and tools for algebraic specifications in the context of constraint-based satisfaction and optimization. While some evidence of the approach we foresee are given in the paper (improved bucket elimination and doubly exponential speed up in a recursive, well-founded definition), further results are left for future work. A direction to explore is using more sophisticated term substitutions (e.g., second order substitutions, in the line of [14]) for defining complex networks inductively. In this paper definitions are restricted to deterministic, non-recursive instances: dropping these restrictions would lead us to the realm of DATALOG constraint programming, with tabling, possibly suggestive in the presence of programmable evaluation strategies.

Related work. Other compositional constraint definitions have been proposed in the literature: in [7] constraints are modeled in a *named* semiring, and in [4] the semiring operations are extended point-wise to functions mapping variable assignments to semiring values. However, in the former case no explicit evaluation is performed, while in the latter no restriction operation is considered. Other approaches are: [5], where compositionality is achieved via complex categorical structures, and [25], where compositionality is not tackled. In a previous workshop paper [18], some early results were given by two of the authors. However, while the algebraic specification is essentially the same, the interpretation domain was restricted to SCSPs for optimization, without reference to SCEPs. Moreover, no proof was given that SCSPs actually satisfy the specification. Furthermore, the connection with the classical tree decomposition was just hinted.

The problem of how to represent parsing trees for (hyper)graphs has been studied in depth in the literature. In particular, we mention the notion of Courcelle graph algebras [9] and of graph grammars for hyperedge replacement [8], which assign a complexity value to the parsing steps. Typical results are about classes of graphs with parsings of bound complexity, having properties that can be proved or computed in linear time. While these results are analogous to ours for some aspects, they do not apply specifically to SCSPs or SCEPs. Instead, tree decomposition and secondary optimization problems have been studied for CSP in [16]. However our approach has a simpler and more effective compositional structure and an up-to-date foundation for name handling.

The role of bounded treewidth CSP has been studied also in connection with the general area of computing homomorphisms between relational structures [11, 10, 17] and k-consistency [1].

Acknowledgements. We thank Nicklas Hoch and Giacoma Valentina Monreale for their collaboration in an earlier version of this work. We also thank an anonymous reviewer for suggesting the example where bucket elimination does not produce a canonical term.

References

1. Albert Atserias, Andrei A. Bulatov, and Víctor Dalmau. On the power of k-consistency. In *ICALP*, pages 279–290, 2007.
2. Richard Bellman. The theory of dynamic programming. *Bulletin of the American Mathematical Society*, 60(6):503–516, 1954.

3. Umberto Bertelè and Francesco Brioschi. On non-serial dynamic programming. *J. Comb. Theory, Ser. A*, 14(2):137–148, 1973.
4. Stefano Bistarelli, Ugo Montanari, and Francesca Rossi. Semiring-based constraint satisfaction and optimization. *J. ACM*, 44(2):201–236, 1997.
5. Christoph Blume, H. J. Sander Bruggink, Martin Friedrich, and Barbara König. Treewidth, pathwidth and cospan decompositions with applications to graph-accepting tree automata. *J. Vis. Lang. Comput.*, 24(3):192–206, 2013.
6. Hans L. Bodlaender and Arie M. C. A. Koster. Combinatorial optimization on graphs of bounded treewidth. *Comput. J.*, 51(3):255–269, 2008.
7. Maria Grazia Buscemi and Ugo Montanari. Cc-pi: A constraint-based language for specifying service level agreements. In *ESOP*, pages 18–32, 2007.
8. David Chiang, Jacob Andreas, Daniel Bauer, Karl Moritz Hermann, Bevan Jones, and Kevin Knight. Parsing graphs with hyperedge replacement grammars. In *ACL*, pages 924–932, 2013.
9. Bruno Courcelle and Mohamed Mosbah. Monadic second-order evaluations on tree-decomposable graphs. *Theor. Comput. Sci.*, 109(1&2):49–82, 1993.
10. Víctor Dalmau and Peter Jonsson. The complexity of counting homomorphisms seen from the other side. *Theor. Comput. Sci.*, 329(1-3):315–323, 2004.
11. Víctor Dalmau, Phokion G. Kolaitis, and Moshe Y. Vardi. Constraint satisfaction, bounded treewidth, and finite-variable logics. In *CP*, pages 310–326, 2002.
12. Rina Dechter. *Constraint processing.* Elsevier Morgan Kaufmann, 2003.
13. Hartmut Ehrig and Bernd Mahr. *Fundamentals of Algebraic Specification 1: Equations und Initial Semantics*, volume 6 of *EATCS Monographs on Theoretical Computer Science.* 1985.
14. Marcelo P. Fiore and Ola Mahmoud. Second-order algebraic theories - (extended abstract). In *MFCS*, pages 368–380, 2010.
15. Fabio Gadducci, Marino Miculan, and Ugo Montanari. About permutation algebras, (pre)sheaves and named sets. *Higher-Order and Symbolic Computation*, 19(2-3):283–304, 2006.
16. Vibhav Gogate and Rina Dechter. A complete anytime algorithm for treewidth. In *UAI*, pages 201–208, 2004.
17. Martin Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *J. ACM*, 54(1), 2007.
18. Nicklas Hoch, Ugo Montanari, and Matteo Sammartino. Dynamic programming on nominal graphs. In *GaM 2015*, pages 80–96, 2015.
19. Ton Kloks. *Treewidth, Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science.* Springer, 1994.
20. Ugo Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Inf. Sci.*, 7:95–132, 1974.
21. A. M. Pitts. *Nominal Sets: Names and Symmetry in Computer Science*, volume 57 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, 2013.
22. Neil Robertson and Paul D. Seymour. Graph minors. III. planar tree-width. *J. Comb. Theory, Ser. B*, 36(1):49–64, 1984.
23. Francesca Rossi, Peter van Beek, and Toby Walsh, editors. *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence.* Elsevier, 2006.
24. Francesca Rossi, Peter van Beek, and Toby Walsh. Constraint programming. In *Handbook of Knowledge Representation*, pages 181–211. 2008.
25. Alexander Schiendorfer, Alexander Knapp, Jan-Philipp Steghöfer, Gerrit Anders, Florian Siefert, and Wolfgang Reif. Partial valuation structures for qualitative soft constraints. In *Software, Services, and Systems*, pages 115–133, 2015.

A Omitted results and proofs

Lemma A.1. $\mathtt{supp}(I \rhd N) = I$.

Proof. Let us spell out the definition of the support of a network. A finite $X \subseteq \mathbb{V}$ supports $I \rhd N$ whenever, for all permutations π that act as the identity on X,

$$(I \rhd N)\pi^{\mathcal{N}} = \pi(I) \rhd N_\pi = I \rhd N$$

Clearly I supports $I \rhd N$: if π does not touch I, it maps $I \rhd N$ to $I \rhd N_\pi \cong I \rhd N$, because N and N_π only differ for a bijective renaming of non-interface vertices. Now, to prove that I is indeed the minimal support, suppose $\mathtt{supp}(I \rhd N) = I \setminus \{x\}$, with $x \in V_N$, and take a permutation π that is the identity on $I \setminus \{x\}$ and swaps x and y, with $y \notin V_N$. Then $\pi(I) \rhd N_\pi$ is $I \rhd N$ where the interface vertex x has been replaced with y, but this is a different abstract network from $I \rhd N$. In fact, there is no isomorphism between the two networks, as isomorphisms must fix interface vertices. Therefore x must be part of the minimal support. $\square$

Proof (of Theorem 1). We have to check that all the axioms hold. We assume the following interpretation of substitutions of variables: $(I \rhd N)[x \mapsto y] = I[x \mapsto y] \rhd N[x \mapsto y]$, where $N[x \mapsto y]$ is N with the vertex x replaced by y.

($\mathbf{AX}_{\|}$) commutativity and associativity follow from the same properties of set union, and the fact that abstract networks are up to isomorphism, so disjoint union of networks is commutative and associative as well. For the unity, we have

$$I \rhd N \|^{\mathcal{N}} \mathtt{nil}^{\mathcal{N}} = I \cup \emptyset \rhd N \uplus_{I,\emptyset} 0_N = I \rhd N$$

($\mathbf{AX}_{(x)}$) for $(x)^{\mathcal{N}}(y)^{\mathcal{N}} I \rhd N$, both sides of the axiom are $I \setminus \{x, y\} \rhd N$.

($\mathbf{AX}_{\alpha}$) given $(x)^{\mathcal{N}} I \rhd N$, we can assume y is not a vertex of N. If it is, we can apply an isomorphism to the network, mapping y to some $z \notin V_N$. In fact, $y \notin \mathtt{supp}(I \rhd N) = I$, so isomorphisms need not fix it.

$$\begin{aligned}
(x)^{\mathcal{N}} I \rhd N &= I \setminus \{x\} \rhd N \\
&\cong (I \setminus \{x\})[x \mapsto y] \rhd N[x \mapsto y] \\
&= I[x \mapsto y] \setminus \{y\} \rhd N[x \mapsto y] \\
&= (y)^{\mathcal{N}} I[x \mapsto y] \rhd N[x \mapsto y] \\
&= (y)^{\mathcal{N}} (I \rhd N[x \mapsto y])
\end{aligned}$$

($\mathbf{AX}_{SE}$) take $I_1 \rhd N_1$ and $I_2 \rhd N_2$, with $x \notin \mathtt{supp}(I_1 \rhd N_1) = I_1$.

$$\begin{aligned}
(x)^{\mathcal{N}}(I_1 \rhd N_1 \|^{\mathcal{N}} I_2 \rhd N_2) &= (x)^{\mathcal{N}}(I_1 \cup I_2 \rhd N_1 \uplus_{I_1,I_2} N_2) \\
&= I_1 \cup I_2 \setminus \{x\} \rhd N_1 \uplus_{I_1,I_2} N_2 \\
&= I_1 \rhd N_1 \|^{\mathcal{N}} I_2 \setminus \{x\} \rhd N_2 \qquad (x \notin I_1) \\
&= I_1 \rhd N_1 \|^{\mathcal{N}} (x)^{\mathcal{N}} I_2 \rhd N_2
\end{aligned}$$

(AX$_\pi$) the identity axiom is obvious. For composition we have

$$\begin{aligned}(I \rhd N\pi'^{\mathcal{N}})\pi^{\mathcal{N}} &= (\pi'(I) \rhd N_I^{\pi'})\pi^{\mathcal{N}}\\ &= \pi(\pi'(I)) \rhd (N_I^{\pi'})^{\pi}_{\pi'(I)}\\ &= (\pi \circ \pi')I \rhd N_I^{\pi\circ\pi'}\\ &= I \rhd N(\pi \circ \pi')^{\mathcal{N}}\end{aligned}$$

(AX$^p_\pi$) It is obvious for constants. For the other axioms we have:

$$\begin{aligned}(I_1 \rhd N_1 \parallel^{\mathcal{N}} I_2 \rhd N_2)\pi^{\mathcal{N}} &= (I_1 \cup I_2 \rhd N_1 \uplus_{I_1,I_2} N_2)\pi^{\mathcal{N}}\\ &= \pi(I_1) \cup \pi(I_2) \rhd (N_1)_\pi \uplus_{\pi(I_1),\pi(I_2)} (N_2)_\pi\\ &= \pi(I_1) \rhd (N_1)_\pi \parallel^{\mathcal{N}} \pi(I_2) \rhd (N_2)_\pi\\ &= (I_1 \rhd N_1)\pi^{\mathcal{N}} \parallel^{\mathcal{N}} (I_2 \rhd N_2)\pi^{\mathcal{N}}\end{aligned}$$

$$\begin{aligned}((x)^{\mathcal{N}} I \rhd N)\pi^{\mathcal{N}} &= (I \setminus \{x\} \rhd N)\pi^{\mathcal{N}}\\ &= \pi(I) \setminus \{\pi(x)\} \rhd N_\pi\\ &= (\pi(x))^{\mathcal{N}}(\pi(I) \rhd N_\pi)\\ &= (\pi(x))^{\mathcal{N}}(I \rhd N\pi)\end{aligned}$$

Lemma A.2. *Given an s-term p, $[\![p]\!]^{\mathcal{N}} = fv(p) \rhd N$, for some N.*

Proof. It it is easy to check that $[\![\mathtt{term}(I \blacktriangleright N)]\!]^{\mathcal{N}}$ and $I \blacktriangleright N$ only differ from the identity of non-interface variables, which correspond in a structure-preserving way.

Proof (of Theorem 2). Let $I_i \blacktriangleright N_i = \mathtt{net}(n_i)$, for $i = 1, 2$. Then, by hypothesis, $I_1 \blacktriangleright N_1 \cong I_2 \blacktriangleright N_2$, therefore:

- $fv(p_1) = I_1 = I_2 = fv(p_2)$;
- they have the same number of edges;
- if the isomorphism maps $e_1 \in E_{N_1}$ to $e_2 \in E_{N_2}$, then these edges are attached to the same interface variables, in the same order. Non-interface variables can be arbitrary, but still in bijective correspondence.

By definition of the net function, these statements imply that n_1 has an atomic subterm $A(\tilde{x})$ if and only n_2 has an atomic subterm $A(\tilde{x}')$, where components of $\tilde{x}$ and $\tilde{x}'$ that belong to $fv(n_1)$ (or, equivalently, to $fv(n_2)$) are equal. All other components are bound variables, corresponding up to α-conversion. In other words, $n_1 \equiv_s n_2$, as required. □

Proof (of Theorem 3). Given a substitution of variables $[x \mapsto y]$, its interpretation on cost functions is $(\phi[x \mapsto y])\rho = \phi(\rho \circ [x \mapsto y])$, where $[x \mapsto y]$ is extended to be a function $\mathbb{V} \to \mathbb{V}$ in the obvious way.

We have to check all the axioms.

($\mathbf{AX}_{\|}$) follows from monoidality of $\times$.
($\mathbf{AX}_{(x)}$) we have

$$(\,(x)^{\mathcal{V}}((y)^{\mathcal{V}}\phi)\,)\rho = \sum_{d_1\in\mathbb{D}}\sum_{d_2\in\mathbb{D}}\phi\rho[x\mapsto d_1][y\mapsto d_2]$$

which is not affected by swapping x and y, and

$$((x)^{\mathcal{V}}\mathtt{nil}^{\mathcal{V}})\rho = \sum_{d\in\mathbb{D}}\mathtt{nil}^{\mathcal{V}}\rho = \sum_{d\in\mathbb{D}}0 = 0 = \mathtt{nil}^{\mathcal{V}}\rho$$

($\mathbf{AX}_{\alpha}$) suppose $y\notin\mathtt{supp}(\phi)$, then we have

$$\begin{aligned}((x)^{\mathcal{V}}\phi)\rho &= \sum_{d\in\mathbb{D}}\phi(\rho[x\mapsto d])\\ &= \sum_{d\in\mathbb{D}}\phi(\rho[y\mapsto d]\circ[x\mapsto y]) \qquad (1)\\ &= \sum_{d\in\mathbb{D}}\phi[x\mapsto y](\rho[y\mapsto d])\\ &= ((y)^{\mathcal{V}}\phi[x\mapsto y])\rho\end{aligned}$$

where (1) follows from compactness of ϕ and the fact that $\rho[x\mapsto d]$ and $\rho[y\mapsto d]\circ[x\mapsto y]$ have the same action on $\mathtt{supp}(\phi)$.
($\mathbf{AX}_{SE}$) suppose $x\notin\mathtt{supp}(\phi_1)$,then we have

$$\begin{aligned}((x)^{\mathcal{V}}(\phi_1\parallel^{\mathcal{V}}\phi_2))\rho &= \sum_{d\in\mathbb{D}}\phi_1(\rho[x\mapsto d])\times\phi_2(\rho[x\mapsto d])\\ &= \sum_{d\in\mathbb{D}}\phi_1\rho\times\phi_2(\rho[x\mapsto d]) \qquad (2)\\ &= \phi_1\rho\times\sum_{d\in\mathbb{D}}\phi_2(\rho[x\mapsto d]) \qquad (3)\\ &= (\rho_1\parallel^{\mathcal{V}}(x)^{\mathcal{V}}\phi_2)\rho\end{aligned}$$

where (2) follows from compactness of ϕ_1 and (3) from distributivity.
($\mathbf{AX}_{\pi}$) we have

$$((\phi\pi'^{\mathcal{V}})\pi^{\mathcal{V}})\rho = \phi((\rho\circ\pi)\circ\pi') = \phi(\rho\circ(\pi\circ\pi')) = (\phi(\pi\circ\pi')^{\mathcal{V}})\rho$$

by associativity of function composition. The other axiom is obvious.
($\mathbf{AX}_{\pi}^{p}$) we omit the obvious axioms:

$$\begin{aligned}(A^{\mathcal{V}}(x_1,\dots,x_n)\pi)\rho &= A^{\mathcal{V}}(x_1,\dots,x_n)(\rho\circ\pi)\\ &= val_A((\rho\circ\pi)\downarrow_{\{x_1,\dots,x_n\}}\circ\hat{\sigma})\\ &= val_A(\rho\downarrow_{\{\pi(x_1),\dots,\pi(x_n)\}}\circ([x_1\mapsto\pi(x_1),\dots,x_n\mapsto\pi(x_n)]\circ\hat{\sigma})\\ &= A^{\mathcal{V}}(\pi(x_1),\dots,\pi(x_n))\rho\end{aligned}$$

$$
\begin{aligned}
((\phi_1 \parallel^{\mathcal{V}} \phi_2)\pi^{\mathcal{V}})\rho &= (\phi_1 \parallel^{\mathcal{V}} \phi_2)(\rho \circ \pi) \\
&= \phi_1(\rho \circ \pi) \times \phi_2(\rho \circ \pi) \\
&= (\phi_1\pi^{\mathcal{V}} \parallel^{\mathcal{V}} \phi_2\pi^{\mathcal{V}})\rho
\end{aligned}
$$

$$
\begin{aligned}
(((x)^{\mathcal{V}}\phi)\pi^{\mathcal{V}})\rho &= \sum_{d \in \mathbb{D}} \phi(\rho \circ \pi)[x \mapsto d] \\
&= \sum_{d \in \mathbb{D}} \phi(\rho[\pi(x) \mapsto d] \circ \pi) \\
&= \sum_{d \in \mathbb{D}} (\phi\pi^{\mathcal{V}})(\rho[\pi(x) \mapsto d]) \\
&= ((\pi(x))^{\mathcal{V}}(\phi\pi^{\mathcal{V}}))\rho
\end{aligned}
$$

Proof (Proof of Theorem 4). By compactness, $[\![I \rhd N]\!]^{\mathcal{V}}$ can be regarded as a function of type $(I \to \mathbb{D}) \to S$. Now, consider the normal form n for (the concrete version of) $I \rhd N$ (so $[\![I \rhd N]\!]^{\mathcal{V}} = [\![n]\!]^{\mathcal{V}}$). It is straightforward to check that $[\![n]\!]^{\mathcal{V}} = \mathit{sol}$, as defined in Section 2.1.

Lemma A.3. *Let r be the root of T. Then $\mathit{fv}(\chi(r)) = I$.*

Proof. Straightforward, observing that I are the only variables that are not restricted in the inductive computation of $\chi(r)$.

Proof (of Proposition 1). Let r be the root of $\mathcal{T}$. By definition of completed rooted tree decomposition, for ever edge $e \in E_N$ there is a unique vertex t of T such that $e \in E(t)$. It follows by a simple induction that edges $e \in E_N$ and atomic subterms in $\texttt{wterm}(\mathcal{T})$ are in one-to-one correspondence: each atomic subterm $A(\tilde{x})$ corresponds to an edge e in N such that $\mathit{lab}_N(e) = A$ and $a_N(e) = \tilde{x}$. Notice that there may be many occurrences of the same $A(\tilde{x})$, corresponding to different edges with the same label and vertices. Therefore the hypergraph component of $[\![\texttt{wterm}(\mathcal{T})]\!]^{\mathcal{N}}$ is exactly N. It remains to prove that its inteface is I. This follows from Lemma A.2 and Lemma A.3.

Proof (of Proposition 2). By definition, $\texttt{wterm}(\mathcal{T}) = \chi(r)$ is of the form

$$
(\tilde{x})(A_1(\tilde{x}_1) \parallel \cdots \parallel A_k(\tilde{x}_k) \parallel \chi(t_1) \parallel \cdots \parallel \chi(t_n)) \ . \tag{4}
$$

Let $p' = A_1(\tilde{x}_1) \parallel \cdots \parallel A_k(\tilde{x}_k) \parallel \chi(t_1) \parallel \cdots \parallel \chi(t_n)$. Then we have:

$$
\begin{aligned}
\langle\!\langle \texttt{wterm}(\mathcal{T}) \rangle\!\rangle &= \langle\!\langle p' \rangle\!\rangle \qquad \text{(definition of } \langle\!\langle\rangle\!\rangle) \\
&= \max\{\langle\!\langle A_1(\tilde{x}_1) \rangle\!\rangle, \dots, \langle\!\langle A_k(\tilde{x}_k) \rangle\!\rangle, \langle\!\langle \chi(t_1) \rangle\!\rangle, \dots, \langle\!\langle \chi(t_n) \rangle\!\rangle, \mathit{fv}(p')\} \\
&= \max\{|\tilde{x}_1|, \dots, |\tilde{x}_k|, \langle\!\langle \chi(t_1) \rangle\!\rangle, \dots, \langle\!\langle \chi(t_n) \rangle\!\rangle, \mathit{fv}(p')\} \\
&= \max\{\langle\!\langle \chi(t_1) \rangle\!\rangle, \dots, \langle\!\langle \chi(t_n) \rangle\!\rangle, \mathit{fv}(p')\}
\end{aligned}
$$

where the last equations follow from $\tilde{x}_i \subseteq \mathit{fv}(p')$, for $i = 1, \dots, k$.

We will prove the claim for a weaker form of tree decompositions. We say that $\mathcal{T}$ is a *pre-decomposition* of a network $I \rhd N$ whenever it agrees with Definition 4, except that X_t is allowed to contain vertices than are not in V_N, i.e., $V_N \subseteq \bigcup X_t$. Clearly a tree decomposition is a pre-decomposition. Moreover, it makes sense to compute the w-term $\texttt{wterm}(\mathcal{T})$ for a pre-decomposition $\mathcal{T}$, because additional vertices not in V_N become restrictions of variables that do not occur anywhere in the term; these restrictions can be dropped using $\mathbf{(AX^{w}_{(x)})}$.

We proceed by induction on the structure of a pre-decomposition $\mathcal{T}$. Given a vertex t' of $\mathcal{T}$, we denote by $\mathcal{T}_{t'}$ the sub-pre-decomposition rooted in t': it is easy to check that $\mathcal{T}_{t'}$ is a pre-decomposition for the network $[\![\chi(t')]\!]^{\mathcal{N}} = [\![\texttt{wterm}(\mathcal{T}_{t'})]\!]^{\mathcal{N}}$. Notice that, even if $\mathcal{T}$ is a proper tree decomposition, $\mathcal{T}_{t'}$ may still be a pre-decomposition, because some variables in $X_{t''}$, with t'' an ancestor of t' in $\mathcal{T}$, may be in $X_{t'}$, by (3) of Definition 4, but not in $\chi(t')$, thus they are not vertices of $[\![\texttt{wterm}(\mathcal{T}_{t'})]\!]^{\mathcal{N}}$.

Suppose $\mathcal{T}$ has only one vertex. Then $\texttt{wterm}(\mathcal{T})$ is of the form

$$(\tilde{x})(A_1(\tilde{x}_1) \parallel \cdots \parallel A_k(\tilde{x}_k))$$

and we have

$$\langle\!\langle \texttt{wterm}(\mathcal{T}) \rangle\!\rangle = \max\{|\tilde{x}_1|, \dots, |\tilde{x}_k|\} \leq |\tilde{x}| + |\tilde{x}_1| + \dots |\tilde{x}_k| = \texttt{width}(\mathcal{T}).$$

For the induction step, let the root r of $\mathcal{T}$ have children $t_1, \dots, t_n$. The term $\texttt{wterm}(\mathcal{T})$ is of the form (4) and, for $i = 1, \dots, n$ and $j = 1, \dots, k$, we have:

- $\langle\!\langle \chi(t_i) \rangle\!\rangle = \langle\!\langle \texttt{wterm}(\mathcal{T}_{t_i}) \rangle\!\rangle \leq \texttt{width}(\mathcal{T}_{t_i}) \leq \texttt{width}(\mathcal{T})$, by induction hypothesis.
- $|fv(p')| \leq \texttt{width}(\mathcal{T})$, because $fv(p')$ is the union of $fv(\texttt{wterm}(\mathcal{T}))$ and $\tilde{x}$, which are both contained in X_r: the former because $\mathcal{T}$ pre-decomposes a network whose interface vertices are $fv(\texttt{wterm}(\mathcal{T}))$, by Proposition 1 and Lemma A.2; the latter by definition of $\texttt{wterm}(\mathcal{T})$. By definition of width, $|X_r| \leq \texttt{width}(\mathcal{T})$.

By definition, $\texttt{width}(\mathcal{T})$ is the maximum of the values listed above. Since these values are all bound by $\texttt{width}(\mathcal{T})$, we get the claim. □

Proof (of Theorem 6).

$\Longrightarrow$: We can compute an ordering on R using the inductive structure of C. If

$$C = (R)\{A_1(\tilde{x}_1), \dots, A_n(\tilde{x}_n)\}$$

then O^C_R can be any ordering, as A' in line 5 will always be $\{A_1(\tilde{x}_1), \dots, A_n(\tilde{x}_n)\}$. Otherwise, if

$$C = (R')\{A_1(\tilde{x}_1), \dots, A_n(\tilde{x}_n), C_1, \dots, C_m\}$$

Then $C_i = (R_i)P_i$ and, by induction, there is a normal form $(R'_i)A_i$ for them and an ordering O_i for R'_i. Clearly we have $R = R' \cup \bigcup_{i=1,\dots,m} R'_i$, so we can form an ordering O_R for R as follows: $x <_{O_R} y$ if and only if

- there is R'_i such that $x \in R'_i$ and $y \in R$, or;

- $x <_O y$, $O \in \{O_R, O_1, \ldots, O_n\}$, or;
- there are R'_i and R'_j, with $i < j$, such that $x \in R'_i$ and $y \in R'_j$.

This is well-defined, as R,R'_1,…,R'_n are pairwise disjoint. To see that the algorithm in Fig. 3 produces C, observe that $(R)A$ can be written as

$$(R'_1)\ldots(R'_m)(R')(\{A_1(\tilde{x}_1), \ldots, A_n(\tilde{x}_n)\} \cup A_1 \cup \cdots \cup A_m)$$

Therefore line 5 will subsequently pick all variables in R'_1 until line 10 produces

$$(R'_2)\ldots(R'_m)(R')(\{A_1(\tilde{x}_1), \ldots, A_n(\tilde{x}_n), C_1\} \cup A_2 \cup \cdots \cup A_m)$$

then R'_2, and so on, until all A_i all turned into C_i. Finally, C is returned.

$\Longleftarrow$: suppose, by contradiction, that C is not canonical. Then there are a subterm $(R)\{P_1, \ldots, P_n\}$ in C, $x \in R$ and P_i such that $x \notin fv(P_i)$. Consider the step of the algorithm where x is selected by line 3. After the selection, line 5 returns $A' = \{(R')\{P_1, \ldots, P_n\}\}$, with $R' \subseteq R$. We must have:
- $R' \neq \emptyset$, because otherwise line 5 would have returned a set of terms without P_i, as $x \notin fv(P_i)$.
- $P_x = \{P_i \mid x \in fv(P_i)\} \neq \emptyset$, because $x \in fv(\{(R')\{P_1, \ldots, P_n\}\}) \subset fv(\{P_1, \ldots, P_n\}) = \bigcup_i fv(P_i)$.

Then the guard of line 6 holds, so the algorithm is called on $\{P_1, \ldots, P_n\}$ and $\{x\}$ (line 7).
During the recursive call, x is selected again by line 3, but now line 5 returns P_x. Finally, possibly after other recursive calls, line 10 is reached and the algorithm returns some term $S \cup \{S'\}$, where $S = \{P_1, \ldots, P_n\} \setminus P_x$.
Now we are back in the original call of the algorithm. The term $S \cup \{S'\}$ is assigned to Q in line 7. Then $(R')(S \cup \{S'\})$ is assigned to P'', which becomes a subterm of C in line 10. This is a contradiction: we assumed that $(R)\{P_1, \ldots, P_n\}$ is a subterm of C, but the algorithm cannot construct it in a subsequent step of the algorithm, because its terms in parallel have already been partitioned within $(R')(S \cup \{S'\})$. Notice that S is neither empty nor the whole $\{P_1, \ldots, P_n\}$, because $S \cap P_x = \emptyset$ and $P_i \in S$.

Proof (of Theorem 7). We have to check all the axioms. Instead of networks, we will use terms to simplify notation. Recall that, by Lemma A.2, p corresponds to a network with interface variables $\texttt{supp}(p)$.

($\mathbf{AX}_\parallel$) Commutativity and identity are immediate: union on partitions and product on reals are commutative; composing with `nil` does not change neither partitions (the empty set has only one possible partition) nor their probabilities (multiplied by 1). Finally, parallel composition is associative. Consider terms p_1, p_2, p_3, and let $I_i = \texttt{supp}(p_i)$, $i = 1, 2, 3$. Both $[\![(p_1 \parallel p_2) \parallel p_3]\!]^{\mathcal{D}}$ and $[\![p_1 \parallel (p_2 \parallel p_3)]\!]^{\mathcal{D}}$ return the same value: after coercion to $I_1 \cup I_2 \cup I_3$, both of them consider all the triples $\Pi_1 \in Part(I_1)$, $\Pi_2 \in Part(I_2)$ and $\Pi_3 \in Part(I_3)$ with $\Pi_1 \cup \Pi_2 \cup \Pi_3 = \Pi$, multiply the values of $[\![p_1]\!]^{\mathcal{D}}\, \Pi_1$, $[\![p_2]\!]^{\mathcal{D}}\, \Pi_2$ and $[\![p_3]\!]^{\mathcal{D}}\, \Pi_3$ and sum up all of them.

($\mathbf{AX}_{(x)}$) Both $[\![(x)(y)p]\!]^{\mathcal{D}}$ and $[\![(y)(x)p]\!]^{\mathcal{D}}$ return the same value: probabilities values are accumulated for a certain partition of interface vertices independently of the set where variables x and y are located. If the interface set is empty, restriction has no effect.

($\mathbf{AX}_{\pi}$) The identity axiom is obvious. For the composition axiom, we have:

$$\begin{aligned}
[\![p\pi'\pi]\!]^{\mathcal{D}}\, \varPi &= [\![p\pi']\!]^{\mathcal{D}}\, (\varPi\pi^{-1}) \\
&= [\![p]\!]^{\mathcal{D}}\, (\varPi\pi^{-1}\pi'^{-1}) \\
&= [\![p]\!]^{\mathcal{D}}\, (\varPi(\pi\circ\pi')^{-1}) \\
&= [\![p(\pi\circ\pi')]\!]^{\mathcal{D}}\, \varPi.
\end{aligned}$$

($\mathbf{AX}_{\pi}^{p}$) They hold obviously for constants. For parallel composition we have:

$$\begin{aligned}
[\![((p_1 \parallel p_2)\pi)]\!]^{\mathcal{D}}\, \varPi &= [\![(p_1 \parallel p_2)]\!]^{\mathcal{D}}\, \varPi\pi^{-1} \\
&= \sum_{\{(\varPi_1,\varPi_2)\mid \varPi_1\cup\varPi_2 \;=\varPi\pi^{-1}\}} [\![p_1]\!]^{\mathcal{D}}\, \varPi_1 \times [\![p_2]\!]^{\mathcal{D}}\, \varPi_2 \\
&= \sum_{\{(\varPi_1',\varPi_2')\mid \varPi_1\cup\varPi_2=\varPi\}} [\![p_1]\!]^{\mathcal{D}}\, \varPi_1'\pi^{-1} \times [\![p_2]\!]^{\mathcal{D}}\, \varPi_2'\pi^{-1} \\
&= \sum_{\{(\varPi_1',\varPi_2')\mid \varPi_1\cup\varPi_2=\varPi\}} [\![p_1]\!]^{\mathcal{D}}\, \pi\varPi_1' \times [\![p_2]\!]^{\mathcal{D}}\, \pi\varPi_2' \\
&= [\![((p_1\pi \parallel p_2\pi)]\!]^{\mathcal{D}}\, \varPi.
\end{aligned}$$

For restriction we have:

$$\begin{aligned}
[\![((x)p)\pi]\!]^{\mathcal{D}}\, \varPi &= [\![((x)p)]\!]^{\mathcal{D}}\, \varPi\pi^{-1} \\
&= \sum_{\{\varPi'\mid \varPi'-\pi^{-1}(x)=\varPi\pi^{-1}\}} [\![p]\!]^{\mathcal{D}}\, \varPi' \\
&= \sum_{\{\varPi''\mid \varPi''-\pi^{-1}(x)=\varPi\}} [\![p]\!]^{\mathcal{D}}\, \varPi''\pi^{-1} \\
&= \sum_{\{\varPi''\mid \varPi''-\pi^{-1}(x)=\varPi\}} [\![p\pi]\!]^{\mathcal{D}}\, \varPi'' \\
&= [\![(\pi(x))(p\pi)]\!]^{\mathcal{D}}\, \varPi.
\end{aligned}$$

($\mathbf{AX}_{\alpha}$) We have:

$$\begin{aligned}
[\![(y)p[x\mapsto y]]\!]^{\mathcal{D}}\, \varPi &= [\![(y)p[x\mapsto y]]\!]^{\mathcal{D}}\, \varPi \\
&= \sum_{\{\varPi'\mid \varPi'-y=\varPi\}} [\![p[x\mapsto y]]\!]^{\mathcal{D}}\, \varPi' \\
&= \sum_{\{\varPi'\mid \varPi'-y=\varPi\}} [\![p]\!]^{\mathcal{D}}\, \varPi'[y\mapsto x]
\end{aligned}$$

$$= \sum_{\{\Pi'' \mid \Pi'' - x = \Pi\}} [\![p]\!]^{\mathcal{D}}\, \Pi'' \tag{5}$$
$$= [\![(x)p]\!]^{\mathcal{D}}\, \Pi$$

where, for (5), we have $\Pi \in Part(\mathtt{supp}(p[x \mapsto y] \setminus \{y\})$, so $\Pi' - y = \Pi$ if and only if $\Pi'[y \mapsto x] - x = \Pi[y \mapsto x] = \Pi$. Then we can set $\Pi'' = \Pi'[y \mapsto x]$.

($\mathbf{AX}_{SE}$) We have:

$$\begin{aligned}
&[\![(x)(p_1 \parallel p_2)]\!]^{\mathcal{D}}\, \Pi \\
&\quad = \sum_{\{\Pi' \mid \Pi' - x = \Pi\}} \sum_{\{(\Pi'_1, \Pi'_2) \mid \Pi'_1 \cup \Pi'_2 = \Pi'\}} [\![p_1]\!]^{\mathcal{D}}\, \Pi'_1 \times [\![p_2]\!]^{\mathcal{D}}\, \Pi'_2 \\
&\quad = \sum_{\{(\Pi'_1, \Pi'_2) \mid (\Pi'_1 \cup \Pi'_2) - x = \Pi\}} [\![p_1]\!]^{\mathcal{D}}\, \Pi'_1 \times [\![p_2]\!]^{\mathcal{D}}\, \Pi'_2 \\
&\quad = \sum_{\{(\Pi'_1, \Pi'_2) \mid \Pi'_1 - x \cup \Pi'_2 - x = \Pi\}} [\![p_1]\!]^{\mathcal{D}}\, \Pi'_1 \times [\![p_2]\!]^{\mathcal{D}}\, \Pi'_2 \qquad (x \notin \mathtt{supp}(p_2)) \\
&\quad = \sum_{\{(\Pi''_1, \Pi'_2) \mid \Pi''_1 \cup \Pi''_2 = \Pi\}} [\![p_1]\!]^{\mathcal{D}}\, \Pi''_1 - x \times [\![p_2]\!]^{\mathcal{D}}\, \Pi''_2 - x \\
&\quad = \sum_{\{(\Pi''_1, \Pi'_2) \mid \Pi''_1 \cup \Pi''_2 = \Pi\}} \Big(\sum_{\{\Pi''' \mid \Pi''' - x = \Pi''_1\}} [\![p_1]\!]^{\mathcal{D}}\, \Pi'''_1 \Big) \times [\![p_2]\!]^{\mathcal{D}}\, \Pi''_2 \\
&\quad = [\![((x)p_1) \parallel p_2]\!]^{\mathcal{D}}\, \Pi.
\end{aligned}$$