Constraint System Decomposition

Joan Thibault

Constraint System Decomposition

Joan Thibault

RESEARCH REPORT

N° 9478

July 2022

Project-Team HYCOMES

ISSN 0249-6399 ISRN INRIA/RR--9478--FR+ENG

Constraint System Decomposition

Joan Thibault*

Project-Team HYCOMES

Research Report n° 9478 — July 2022 — 68 pages

Abstract: Various classical problems in computer science can be formulated as Constraint Solving Problems (CSP), consisting in a query on a conjunction of constraints. Typical instances of such queries are satisfiability (with or without witness), optimization under constraints, model enumeration, model counting and canonicalization. Constraint systems can be Conjunctive Normal Form (CNF) formulas, as well as (Integer) Linear Programs ((I)LP) and, in its most generic form, Constraint Programs (CP).
In both industrial and academic contexts, instances are generally structured and, in most cases, sparse: each constraint involves only a small set of variables, and variables are only involved in a small set of constraints. Moreover, large practical instances tend to have a tree-like structure, which can efficiently be captured by the notion of treewidth, as commonly considered in the fixed-parameter tractability community.
Using dynamic programming to solve problems for which a "good" tree decomposition is available is rather old, and has been rediscovered many times in the history of computer science, under various names: *message passing* in *factor graphs*, *belief propagation* in *belief networks*, *arc consistency* in *constraint networks*, etc.
This work introduces the CoSTreD (Constraint System Tree Decomposition) method, based on symbolic representations and operators on them to improve the scalability of CSP solving.
CoSTreD is based upon two operators: a projection operator which allows to deal with satisfiability and canonicalization locally on the tree decomposition, and a coprojection operator, extending the method to optimization queries. We establish sufficient conditions under which these operators preserve the semantics of the CSP. Finally, CoSTreD is extended to deal with parameter (or mode) variables, mostly by (i) adapting the notion of tree decomposition to deal with parameter variables, (ii) using symbolic representations to avoid the combinatorial explosion of mode enumeration, and (iii) mitigating the contamination of constraints by parameter variables during message passing.

Key-words: tree decomposition, treewidth, message passing, belief propagation, factor graph, chordal networks, constraint networks, symbolic representation, symbolic computation, Snowflake

* Univ. Rennes, France

RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

Approche Décompositionnelle pour la compilation de Systèmes de Contraintes

Résumé : Plusieurs problèmes classiques en informatique peuvent être formulés comme des problèmes de résolution de contraintes (CSP), consistant en une requête sur une conjonction de contraintes. Les exemples typiques de telles requêtes sont la satisfiabilité (avec ou sans témoin), l'optimisation sous contraintes, l'énumération de modèles, le comptage de modèles et la canonisation. Les systèmes de contraintes peuvent être des formules CNF (Conjunctive Normal Form), ainsi que des programmes linéaires (entiers) ((I)LP) et, dans sa forme la plus générique, des systèmes de contraintes (CP).

Dans les contextes industriels et académiques, les instances sont généralement structurées et, dans la plupart des cas, creuses : chaque contrainte n'implique qu'un petit ensemble de variables, et les variables ne sont impliquées que dans un petit ensemble de contraintes. De plus, en pratique, les grandes instances ont tendance à avoir une structure arborescente, qui peut être facilement appréhendée par la notion de largeur arborescente, telle que communément admise dans la communauté de tractabilité à paramètre fixé (FPT).

L'utilisation de la programmation dynamique pour résoudre les problèmes pour lesquels une "bonne" décomposition arborescente est disponible est assez ancienne, et a été redécouverte de nombreuses fois dans l'histoire de l'informatique, sous différents noms : *message passing* dans les *factor graphs*, *belief propagation* dans les *belief networks*, *arc consistency* dans les *constraint networks*, etc.

Ce travail introduit la méthode CoSTreD (Constraint System Tree Decomposition), basée sur des représentations symboliques et des opérateurs sur celles-ci pour améliorer la scalabilité de la résolution de CSP.

CoSTreD est basée sur deux opérateurs : un opérateur de projection qui permet de traiter la satisfiabilité et la canonisation localement sur la décomposition de l'arbre, et un opérateur de co-projection, étendant cette méthode aux requêtes d'optimisation. Nous établissons des conditions suffisantes sous lesquelles ces opérateurs préservent la sémantique du CSP. Enfin, CoSTreD est étendu pour traiter les variables paramètres (ou mode), principalement en (i) adaptant la notion de décomposition d'arbre pour traiter les variables paramètres, (ii) utilisant des représentations symboliques pour éviter l'explosion combinatoire de l'énumération des modes, et (iii) atténuant la contamination des contraintes par les variables paramètres pendant le passage des messages.

Mots-clés : décomposition aborescente, largeur aborescente, réseaux de contraintes, représentation symbolique, calcul symbolique, calcul formel, Snowflake

Contents

Chapter 1

Introduction

Various classical problems in computer science can be formulated as Constraint Solving Problems (CSP), consisting in a query on a conjunction of constraints. Typical instances of such queries are satisfiability (with or without witness), optimization under constraints, model enumeration, model counting and canonicalization. Constraint systems can be Conjunctive Normal Form (CNF) formulas, as well as (Integer) Linear Programs ((I)LP) and, in its most generic form, Constraint Programs (CP).

In both industrial and academic contexts, instances are generally structured and, in most cases, sparse: each constraint involves only a small set of variables, and variables are only involved in a small set of constraints. Moreover, large practical instances tend to have a tree-like structure, which can efficiently be captured by the notion of treewidth, as commonly considered in the fixed-parameter tractability community [12].

Using dynamic programming to solve problems for which a "good" tree decomposition is available is rather old, and has been rediscovered many times in the history of computer science, under various names: *message passing* in *factor graphs*, *belief propagation* in *belief networks*, *arc consistency* in *constraint networks*, etc.

This work introduces the CoSTreD (Constraint System Tree Decomposition) method, based on symbolic representations and operators on them to improve the scalability of CSP solving.

CoSTreD is based upon two operators: a projection operator which allows to deal with satisfiability and canonization locally on the tree decomposition, and a co-projection operator, extending the method to optimization queries. We establish sufficient conditions under which these operators preserve the semantics of the CSP.

Furthermore, we show that one can implicitly enumerate all solutions of a given constraint system, by implicitly indexing each one of them by an integer and creating two circuits one which given such index returns the associated valuation, and conversely, given a valuation returns its associated index. This method also relies on two operators: the projection operator, and an isomorphism construction operator.

Finally, CoSTreD is extended to deal with mode variables, which coincides with the notion of parameter variables in mathematical programming. This is achieved by (i) adapting the notion of tree decomposition to deal with mode variables, (ii) using symbolic representations to avoid the combinatorial explosion of mode enumeration, and (iii) mitigating the contamination of constraints by mode variables during message passing.

It is crucial to note that, (1) mode variables shall not be confused with regular variables, nor assimilated with universally quantified variables (as they may appear in QBF formulas), (2) mode variables should not be "solved" in the same way as either existentially or universally

quantified variables as it would change the problem, and, (3), as far as we know, there is no efficient search-based techniques that avoids mode enumeration.

By some aspects CoSTreD extend various previous work, such as the work of Capelli and Mengel [1] on the efficient use of KR for fixed-treewidth Quantified Boolean Formula (QBF) using the d-DNNF [4] representation. However, the advantages of CoSTreD with respect to previous work are four-fold:

- In terms of use cases the sole requirement is the implementation of the projection and co-projection operators. These operators can be efficiently implemented in various logics, including Boolean logic, linear programming, and systems of polynomial (in)equations.
- the correctness and complexity analysis do not rely on any particular underlying representations, e.g. for Boolean constraints any representation with an asymptotic $\mathcal{O}(2^n)$ memory consumption and computation time for basic operators suffices.
- In terms of genericity, the high-level point of view of CoSTreD allows a modular approach, hence maximizing re-usability, and shortening testing and deployment time when implementing a new instance. We efficiently implemented this approach within the Snowflake library [13] using OCaml's functors to effectively deal with this level of genericity.
- The clear separation of the problem into components allows to break-down various analysis at component level, among which the search for a good ordering of the variables. Properly ordering variables when dealing with either search-based or symbolic methods is a constantly reoccurring problem, being able to mostly bypass it for only a small cost is an accomplishment by itself.

The remainder of this report is organized as follows

- Chapter 3 [Constraint System Algebra] lays down the theoretical ground for the remainder of this report. We introduce the notion of Constraint System Algebra (CSA) which are, intuitively, formulas F where the top-layer operator (denoted $+$) is commutative and distributed over the operator of the second layer (denoted $\times$) which is also commutative. We show that for any such CSA, one may easily evaluate the result of such expression by performing "message passing"-like operations on a tree-decomposition. The main limitation of this generic framework is that it does not allow to link the evaluation of the expression with the set of valuations of the variables. The remaining chapters present use cases where we are able to make explicit such a relation.
- Chapter 4 [Propagation and Consistency] studies the case of Boolean CSA, within the generic setting of graph-decompositions. We show that propagating information within a graph-decomposition may not converge, furthermore, convergence of the message passing process may not be enough to compute the result of the expression.
- Chapter 5 [Tractability of Tree-Decomposition] proves that message-passing with tree-decompositions suffices to ensure termination and correctness of the method.
- Chapter 6 [Tree-Based Reduction] introduces a co-projection operator which allows to maintain, in the particular case of monotonic operators (e.g. $\max/+$ CSA), the relation between the result of the expression and the valuations of the variables, during the message-prassing process.
- Chapter 7 [Cartesian Operators] shows that, for Cartesian operators ($\uplus/\times$ CSA), one can compute, using a message-passing technique, an enumeration function of a constraint system.
- Chapter 8 [Parametric Reduction] shows that all the above use cases can be extended to a multimode setting, where both the expression and its result are parameterized by mode variables.

Chapter 2

Related Work

In the following, we shall discuss related work from the Weighted Constraint Satisfiability Problem(wCSP) [11] community. As far as we know, the multimode extension (mwCSP) of wCSP is new, hence we leave it out of this survey section and postpone its formal definition to Chapter 8

We insist that here the notion of "constraint" should be understood rather widely, from clauses (hence the satisfiability problem on CNF) to system of linear (in)equations or even polynomial and ideals. Also including the wide range of constraints that one may find in constraint programming (including set-based, integer-based, graph-based, word-based, etc. constraints).

Independently of the exact nature of the constraints constituting a particular constraint system (also called constraint network by Rinar Dechter [5]) it is quite common in the literature to consider system which are sparse, that is, every variable occur in a small number number of constraints and conversely, every constraints involves a small number of variables.

Exploiting sparsity as is, is not that simple, as it does not usually lead to any particular kind of high-level semantic properties.

However, one may usually assume a property stronger than sparsity which is that the *primal graph*[1] has a low treewidth [10], that is, the system can be organized in a tree-like manner.

In our particular setting this hypothesis is quite reasonable as many typical networks are actually tree-shaped such as city-wide electrical or heat networks, corridors in a building, etc.

N.B.: exhibiting this tree-like structure is already hard by itself as finding a "good" tree-decomposition is NP-hard for any known relevant notion of "good", hence we generally rely on scalable heuristics such as min-degree or min-fill-in. For more details about the relevance of using heuristic, we refer the reader to the work the study of Jégou et al. [7] which compares the effect of using several tree-decomposition heuristics on the effective computation time on several benchmarks.

The most common counter example to such strong hypothesis are two-dimensions grids (or higher), as far as we know no generic method has been found to efficiently and generically deal with this topology, hence we leave it out of this short survey.

When solving sparse constraint system a common technique to alleviate this property is to use *message-passing* based techniques, in particular its refinement to tree-decompositions : *belief propagation*.

Although message-passing-based techniques are rather efficient in order to deal with very large systems, we need methods that scale near-linearly with the size of the system (but not necessarily linear with the size of the components of the tree-decomposition).

[1] the primal graph of a constraint system is the graph where a vertex is associated to every variable in the constraint system, and every vertex/variable is related iff they appear in a common constraint, see Definition 3.1

We start by reviewing linear-time instances of belief-propagation in the case of CSP (that is, without the optimization component):

- system of linear equation, using partial Gauss-Jordan elimination[9]
  - component elimination is cubic in the number of variables involved
- system of linear inequation, using Fourier-Motzkin lemma, beware of the combinatorial explosion of constraint, but it is quite simple to keep it in check in the case of zones (difference-bounded-matrices, DBM) and octagons
  - component elimination is quadratic in the number of equations involved
- system of polynomial equations [3] [2]
  - component elimination is double exponential in the number of variables involved

Essentially, the core idea is always the same, using dynamic programming to solve the problem one component at a time, but instead of performing actual case enumeration (like it would usually be the case), use a symbolic projection operator to improve efficiency.

The list of problem one can solve in linear time further decreases when dealing with VCSP, our main reference being the work of Lange and Swoboda [8] on applying message passing technique on 0-1 ILP (Integer Linear Programming) by mean of Binary Decision Diagrams (BDD).

With regard to this work, our research report offers more genericity and clearer axiomatization on how to further generalize our work, furthermore, although this detail is mostly syntactic, our implementation deals with more arbitrary relations between variables by opposition to their work where relations must be linear.

Furthermore, we must insist on the fact that our theory and implementation are able to deal with multimode VCSP, hence strictly extending the scope of previous work which either focused solely on VCSP or even more specifically on CSP.

Finally, many search-based techniques exploiting the sparsity of the primal graph in term of a good tree-decomposition exists, see Rina Dechter's book [5] for common references, however, as far as we know there is no extension of these techniques to deal with the case of multimode queries which is our case here.

Chapter 3

Constraint System Algebra

3.1 Constraint System Algebra

This section introduces the very broad context in which the present report evolves, allowing to establish a simple yet powerful framework to express most kind of constraint-based reasoning. This section contains a curated list of problem falling in the following framework.

In order to clearly state what this work is about, we first need to kill the elephant in the room, and speak of what it is not about.

We are given two (possibly infinite) sets $\mathcal{A}$ (standing for domain) and $\mathcal{B}$ (standing for co-domain).

We assume that $\mathcal{B}$ comes with a structure of commutative semi-ring (CSR), that is,

- $\mathcal{B}$ has two internal operators $+$ and $\times$
- $(\mathcal{B}, +)$ is a commutative monoid with an identity element denoted $\bot$, that is,
  - $+$ is associative : $(x + y) + z = x + (y + z)$,
  - $+$ is commutative : $x + y = y + x$
  - $\bot$ is the identity element : $x + \bot = x$
- $(\mathcal{B}, \times)$ is a commutative monoid with an identity element denoted $\top$
- $\times$ distributes over $+$: $x \times (y + z) = (x \times y) + (x \times z)$ (we will mostly use this property in the reverse direction in order to perform curated factorization)
- $\bot$ is an absorbing element with respect to $\times$: $x \times \bot = \bot$.

The particular choice of $\bot$ and $\top$ in the usual place of 0 and 1 is to make explicit that any CSR can be understood as an extension of the Boolean semi-ring.

This choice allows to naturally inject Boolean-level reasoning in various transformation (see Chapter 6 [Tree Based Reduction]).

Remark 3.1. *Also in many cases $\bot$ and $\top$ are actually exogenous with the elements that one would have naturally dealt with. For example in the context of the $(\mathbb{N}, \max, +)$ "algebra", one may see that* max *is lacking an identity element, we usually extend $\mathbb{N}$ with $+\infty$, so that it becomes a CSR. However, then, if one want to state that every occurrences of $\mathbb{N}$ are bounded by k, one should instead say that "finite occurrences" are bounded by k.*

Instead, using $\bot$ to extend $\mathbb{N}$ allows to make explicit its foreign nature and stream-line the transfer of notion from $\mathbb{N}$ to $\mathbb{N} \cup \{\bot\}$ by simply starting the sentence by "non-$\bot$ elements are ...".

Let $X = \{x_i\}_i$ be set of variables with value in $\mathcal{A}$.

If the domain $\mathcal{A}$ is infinite, we further assume that $(\mathcal{B}, +)$ is a complete monoid, hence $\sum$ can be indexed by infinite sets.

We call $\sigma \in \mathcal{A}^X$ a valuation of the variables.

We may define a generic notion of constraint as a subset $\mathcal{C} \subseteq \mathcal{A}^X \to C$ of functions from $\mathcal{A}$ to $\mathcal{B}$.

We assume that $\mathcal{C}$ contains all constant functions, and is stable by $+$ and $\times$ operators (lifted from $\mathcal{B}$ to $\mathcal{A} \to \mathcal{B}$).

Furthermore, we may define a constraint system F (standing for formula) as a bag[1] of constraints, that is $F = \{\!\{C_1, \ldots, C_m\}\!\}$.

Finally, we may define the semantic $[\![F]\!]$ of F as:

$$\forall \sigma \in \mathcal{A}^X,\ [\![F]\!](\sigma) := \prod_{C \in F} C(\sigma)\ .$$

Furthermore, we define the result $\mathrm{R}(F)$ of F as :

$$\mathrm{R}(F) := \sum_{\sigma \in \mathcal{A}^X} \underbrace{\prod_{C \in F} C(\sigma)}_{=[\![F]\!](\sigma)}\ .$$

We encapsulate these notions as a sextuplet $\texttt{CSA} = (\mathcal{A}, \mathcal{B}, +, \times, \bot, \top)$ (the n-ary extensions $\sum$ and $\prod$ are made implicit in this definition) that we call Constraint System Algebra (CSA).

Such basic definition actually encompasses a large class of problem, for example:

- assuming $\texttt{CSA} = (\mathcal{A}, \mathbb{B}, \vee, \wedge, \bot, \top)$ then, any constraint system F denotes a constraint system (with variables having values in $\mathcal{A}$), with $[\![F]\!]$ its classical semantic of "all constraint must be satisfied" and $R(F)$ its classical semantic of "there exists a solution". Various domains and constraints may be used
  - if $\mathcal{A} = \mathbb{B}$ and constraint are disjunction, then F is a CNF (Conjunctive Normal Form) constraint system
  - if $\mathcal{A} = \mathbb{B}$ and constraint are arbitrary Boolean constraint system or a single-output circuit, then F is a circuit-based constraint system
  - if $\mathcal{A} = \mathbb{R}$ (or $\mathbb{Q}$) and constraints are linear equalities, we retrieve linear systems (which can be solved using Gauss-Jordan elimination)
  - if $\mathcal{A} = \mathbb{N}$ and constraints are linear (in)equalities, we retrieve integer linear programs (which is NP-Complete)
  - if $\mathcal{A} = \mathbb{N} \mid \mathbb{R}$ and constraints are linear (in)equalities, we retrieve mixed integer linear programs (which is also NP-Complete).
- assuming $\texttt{CSA} = (\mathcal{A}, \mathbb{N} \cup \{\bot\}, \max, +, \bot, 0)$ then, any constraint system F denotes a weighted constraint system, with $[\![F]\!]$ its classical semantic of "all hard constraint are satisfied and overall weight is the sum of individual weight" and $R(F)$ its classical semantic of "the maximal overal weight reachable by the constraint system is". Which allows to encore any optimization extension of the problem mentioned above.
- assuming $\texttt{CSA} = (\mathcal{A}, [0, 1], \max, \min, 0, 1)$ then, any constraint system F can be interpreted as a fuzzy-logic constraint system.
- assuming $\texttt{CSA} = (\mathcal{A}, \mathbb{N} \cup \{\bot, \top\}, \max, \min, \bot, \top)$, then we may encode max flow problems : we maximize across all path the minimal value (bottleneck) of each weighted arc.

[1] a bag (or multiset) behaves similarly to a set, except that elements have a multiplicity, effectively making $\{\!\{C, C\}\!\} \neq \{\!\{C\}\!\}$ by opposition to sets

- assuming $\texttt{CSA} = (\mathcal{A}, [0,1], +, \times, 0, 1)$ with each constraint representing the (parametered) probability of the corresponding event, $[\![F]\!]$ represent the (parametered) combined probability of all event $R(F)$ represents integral of the distribution. Although denoted $+$, we assume that it actually encodes for a linear combination operator which guarantees that the sum of all weights is 1, hence preserving the $[0,1]$ bound.
- assuming $\texttt{CSA} = (\mathcal{A}, \mathbb{N}, +, \times, 0, 1)$ with each weighted constraint representing the number of internal configuration of a sub-system (0 representing the unsatisfiability or nonexistence of a sub-system). Then, for any constraint system F, $[\![F]\!]$ denotes the number of indistinguishable configuration of the sub-systems, and, $R(F)$ denotes the total number of configuration.

3.1.1 Constraint System Decomposition

In practical instances, either industrial or academic, the naive computation of the result of such constraint system leads to untractable computation, in order to extend the class of constraint system that are tractable, one must efficiently identify and exploit some (or several) kind(s) of structure(s).

Present report focuses on exploiting the sparsity of practical large scale formulas, that is, every constraint only involves a few variables, conversely every variable is only involved in a few constraints.

We introduce the notion of support of a constraint, which is a superset of the significant (i.e. non-useless) variables. That is, for every constraint C, we denote $\texttt{supp}\,(C) \subseteq X$ such that $C \in \mathcal{A}^{\texttt{supp}(C)} \to \mathcal{B}$ (for any subset of variables $Y \subseteq X$, we implicit cast $\mathcal{A}^Y \to \mathcal{B}$ to $\mathcal{A}^X \to \mathcal{B}$ by ignoring all variables in $Y^{\complement} = X - Y$ making them useless).

For self-contentness we remind the reader the definition of *primal graph* and *tree decomposition*

Definition 3.1 (Primal Graph)**.** *Let F be a constraint system, we define its primal graph* $\mathrm{PG}(F)$ *as :*

$$\mathrm{PG}(F) := \left(X, \bigcup_i \texttt{supp}\,(f_i)^2 \right) .$$

That is,
- $\mathrm{PG}(F)$ *is an undirected graph, such that*
- *vertices of* $\mathrm{PG}(F)$ *are variables in X, and,*
- *there exists an edge $(x, y) \in X^2$ iff there exists a common constraint f_i, such that $x, y \in \texttt{supp}\,(f_i)$.*

Definition 3.2 (Tree Decomposition)**.** *Given an undirected graph $G = (V, E)$, we define a decomposition $D = (B, I)$ of G as a graph which vertices B are subsets of vertices in V and edges I are called interconnection. A decomposition, must follow the following properties*
- *D is a forest (that is, a collection of trees).*
- *(vertex coverage) $\forall v \in V, \exists b \in B, v \in b$, and,*
- *(edge coverage) $\forall (u, v) \in E, \exists b \in B, u \in b \wedge v \in b$, and,*
- *(locality) $\forall v \in V, \{b \in B \mid v \in b\}$ defines a connected sub-graph of D.*

Additionally, we say that a decomposition is minimal iff
- *(non-emptiness) $\forall b \in B, b \neq \emptyset$, and,*

- *(minimality rule)* $\forall b, b' \in B, b \not\subseteq b'$

Remark 3.2 (Tree Decomposition vs Graph Decomposition)**.** *Chapter 4 introduces a generalization of tree decomposition we call graph decomposition.*

In particular,

- *a graph decomposition in the shape of a forest (that is, a collection of trees) is actually a tree decomposition, and,*
- *a graph decomposition in the shape of a collection of paths is actually a path decomposition [6].*

Hence, any result on graph decomposition induces its counter part on tree and path decompositions.

Definition 3.3 (Constraint System Tree Decomposition)**.** *We define a constraint system tree decomposition (CSTD)* $\mathbf{F} = (D, F)$*, where*

- D *is graph decomposition on* X
- F *is a mapping from components of the graph decomposition* D *to constraints, that is for each component* $b \in B$*, we denote* f_b *its associated constraint.*
- *(support inclusion)* $\forall b \in B, \texttt{supp}\,(f_b) \subseteq b$

Lemma 3.1 (Separation Property of Tree Decompositions)**.** *Let* $G = (V, E)$ *be a graph, and* $D = (B, I)$ *be a tree-decomposition of* G*, then for any component* $b \in B$*.*

- *We denote* $b_1, \ldots, b_{b^\circ}$ *the neighbors of* b *in* D*, and,*
- *we denote* D_i *the sub-tree-decomposition with* b_i *as root, and,*
- *we denote* G_i *the sub-graph induced by* D_i*.*
- *For any two indices* $i \neq j$*,*
  1. $b_i \cap b_j \subseteq b$*, and,*
  2. $\texttt{supp}\,(D_i) \cap \texttt{supp}\,(D_j) = b_i \cap b_j \subseteq b$*, and,*
  3. $G_i.V \cap G_j.V = b_i \cap b_j \subseteq b$

Proof. *Let* $b \in B$ *a component of* D*.*

- *We denote* $b_1, \ldots, b_{b^\circ}$ *the neighbors of* b *in* D*, and,*
- *we denote* D_i *the sub-tree-decomposition with* b_i *as root, and,*
- *we denote* G_i *the sub-graph induced by* D_i*.*
- *Lets prove (2), hence deducing (3) and (1),*
- *Lets assume that there exists* $v \in V$ *such that* $v \in \texttt{supp}\,(D_i) \cap \texttt{supp}\,(D_j)$*,*
- *However,* $B_v = \{b \in B \mid v \in b\}$ *must be connected in* D*,*
- *Furthermore,* b*,* b_i *and* b_j *are on the only path from* D_i *to* D_j*,*
- *Hence,* $b, b_i, b_j \in B_v$*, i.e.,* $v \in b, b_i, b_j$
- *Hence,* $\texttt{supp}\,(D_i) \cap \texttt{supp}\,(D_j) \subseteq b_i \cap b_j \subseteq b$*,*
- *However,* $b_i \in D_i$ *and* $b_j \in D_j$*, hence* $\texttt{supp}\,(D_i) \cap \texttt{supp}\,(D_j) = b_i \cap b_j$*,*
- *Thus (2) holds, hence, (3) and (1) hold as well.*

□

Hence, given F a constraint system and $D = (B, I)$ a graph-decomposition of its primal graph, we may turn F into the constraint system decomposition $\mathbf{F} = (D, F')$ by first partitioning the constraint of F according to the components of D (that is, we put each constraint in exactly one bag), which is licit by the commutativity of $\times$ then we merge constraints appearing in the same bag, which is licit as the set of constraint is stable by $\times$, without loss of generality we add the constant function $\top$ to all empty bags.

Reasoning in the generic case of graph-decomposition is known to be quite hard, in the case that both operators are idempotent (e.g. $\wedge/\vee$ or $\max$ / $\min$) we may still derive some result, see Chapter 4 [Propagation and Consistency] for more details.

Hence, we strengthen the sparsity assumption into the low treewidth assumption, that is the primal graph has an overall tree-like structure (classic examples of sparse but high treewidth setting are grid-like configuration, with two or more dimensions).

Remark 3.3. *Ideally, one wants as small components as possible, in practice, one may prefer some compromise between the size and the number of components, anyway for all known relevant metric of "good" tree-decomposition, the problem of exhibiting one is NP-Complete. Hence, one usually rely on various heuristics, min-degree and min-fill-in offering a reasonable first choice as they provide "good" results while being simple to implement efficiently.*

We refer the reader to the comparative study of Jégou et al. [7] on the relative impact of a choice of heuristic and exact methods.

We introduce a reduction operator which given a constraint C and a subset of variables $Y \subseteq X$ returns a constraint $\sum_Y C$ defined as : (denoting $Z = X - Y$),

$$\sum_Y C := \sigma_Z \mapsto \sum_{\sigma_Y} C(\sigma_Z, \sigma_Y) \ .$$

Using distributivity of $\times$ over $+$, one may show that, $R(f(y) \times g(y, z)) = R(f \times \sum_z g)$

From there, one may define the leaf reduction operator which takes a CSD $\mathbf{F}$ a leaf s and its parent d in the tree, and we denote $\mathbf{F}[s \twoheadrightarrow d] = \mathbf{F}'$ the CSD where s (and F_s) has been removed from $\mathbf{F}$ and $F'_d := F_d \times \sum_{(s-d)} F_s$.

It is important to notice that variables in s are not variables of $\mathbf{F}[s \twoheadrightarrow d]$ which is critical in the case of non-idempotent CSA (such as $(\mathcal{A}, \mathbb{N}, +, \times, 0, 1)$).

By iterating the leaf reduction operator until no variables remain (we consider that the tree-decomposition is rooted with the empty component), one may compute the result of the constraint system while having a WCET within $\mathcal{O}(|D| f(\mathrm{tw}(D)))$ where f is the parametered cost of the reduction operator, and $\mathrm{tw}(D)$ is the cardinal of the largest component of the decomposition D.

3.1.2 Extensions and Limitations

We may naturally extend the scope of CSA to multimode (aka parametric) computations, by extending X with an extra set of so called mode variables M, while the notion of semantics extends naturally, one must change the notion of result so that it is no longer a value but a function, such that

$$\forall \sigma \in \mathcal{A}^M,\ R(f)(\sigma_M) = R(f(\sigma_M, \cdot)) \ .$$

Chapter 8 [Parametric Reduction] deals with the extension of the notion of propagation and reduction to the multimode case.

An important remark is that the current definition of CSA mostly focuses on the result and how to successively turn a valuation into an intermediary, then combine all intermediary values into a result. However, it does not relate back this result to the valuations that influenced it.

The remainder of this report is dedicated to the computation of such relevant relation between the result and the valuation:

- Chapter 4 [Propagation and Consistency] shows that in the case of doubly idempotent CSA (that is $+$ and $\times$ are idempotent, e.g., $\wedge/\vee$, $\max$ / $\min$) one may define a confluent propagation operator (termination is only ensured if the domain $\mathcal{A}$ is finite). Going forward, unless mentioned otherwise, we further assume that the decomposition is a tree-decomposition.
- Chapter 5 [Tractability of Tree-Decomposition] shows that in the case of doubly idempotent CSA, one may compute the result along with the set of valuation which realize it.
- Chapter 6 [Tree-Based Reduction] shows that in the case of an idempotent CSA (that is, $+$ is idempotent, e.g., $\max/+$ or $\min/+$), one may compute the result along with the set of valuation which realize it. Actually, this chapter starts by introducing an axiomatization which relax the notion of CSA, allowing to focus solely on the notion of projection (and reduction operator).
- Chapter 7 [Cartesian Reduction] shows that in the case of Cartesian-like operators, one may explicit a bijection between the result and the set-annotated valuations.

3.2 Notations

For the sake of formalization, we introduce the following notations

- a domain $\mathcal{A}$ to which variables belong
- a finite set X of n variables,
  - Variables are denoted $X = \{x_1, \dots, x_n\}$.
  - We call valuation a mapping from X to $\mathcal{A}$, generally denoted $\sigma \in \mathcal{A}^X$
- a finite set F of m constraint, called *constraint system*, that is, Boolean functions f_i on $\mathcal{A}^X$.
  - Constraints are denoted $F = \{f_1, \dots, f_m\}$
  - If the function returns true ($\top$), the constraint is satisfied, otherwise, it returns false ($\bot$) and the constraint is not satisfied.
  - For convenience of notation, we may use $\sigma \models f$ whenever $f(\sigma) = \top$
- the conjunction operator $\wedge$,
- we define the semantics of F as $[\![F]\!] = \bigwedge_i f_i$.

Remark 3.4. *We do not require the domain $\mathcal{A}$ to be finite, although necessary to ensure the finiteness of the lattice structure in Chapter 4, hence the termination of the propagation algorithm. However, using tree decomposition in Chapter 5, we show that there always exists a terminating strategy*

At this stage, constraint should only be interpreted as their semantic of Boolean function, hence, two constraints are equivalent iff they are equal, implementation level consideration are addressed at the end of this chapter.

Definition 3.4 (Existential Projection Operator)**.** *We denote the existential projection operator as $\exists(Y)f$ or $\exists_Y f$ and defined as,*

for any subset $Y \subseteq X$ of variables and constraint f, we denote $\exists_Y f = f' \in \mathcal{A}^{Y^c} \to \mathbb{B}$, such that, for any valuation $\sigma_Y \in \mathcal{A}^{Y^c}$, then $\sigma_Y \models f'$ iff there exists $\sigma \in \mathcal{A}^X$ an extension of σ_Y such that $\sigma \models f$.

To avoid unnecessary domain castings, we always assume that the result of existential projection is plunged back into $\mathcal{A}^X \to \mathbb{B}$ by re-introducing variables in Y as useless variables of the result.

Definition 3.5 (Existential Restriction Operator)**.** *We denote $\exists_{|Y} f = \exists_{Y^c} f$ the existential restriction of f to Y, that is, the existential projection of all variables which are not in Y.*

Definition 3.6 (Useless Variables, Significant Variables, Support Variables)**.** *Let $f \in \mathcal{A}^X \to \mathbb{B}$ a constraint and $x \in X$ a variable, we say that x is a useless variables, iff*

$$\forall \sigma \in \mathcal{A}^X, v \in D, f(\sigma) = f(\sigma[x \leftarrow v]) \ .$$

That is, x is said useless if the result is not impacted by its value.

We say that a variables is significant if its not useless,

We denote $\mathtt{supp}(f) \subseteq X$ the set of all significant variables of f, also called, its supporting variables.

Given a set of constraint F, we denote $\mathtt{supp}(F) = \bigcup_i \mathtt{supp}(f_i)$

Remark 3.5 (Syntax Level Distinction)**.** *At the semantic level there is no distinction between significant variables and supporting variables, however, both sets may be distinct upon implementation.*

Usually, we call supporting variables, variables which explicitly appear in the definition of f, however, some of these variables may actually be useless, for example in the expression $(x \vee y) \wedge x$, y is a supporting variables (in a syntactic sense) although it is not significant.

For now we will not further detail possible distinctions between both sets as they can be considered one and the same at the semantic level.

- *"support variables" is a syntactic notion*
- *"useless/significant variables" are semantic notions*

Lemma 3.2 (Support and Conjunction, Existential Projection, Existential Restriction)**.** *Let two Boolean functions f and g, then $\mathtt{supp}(f \wedge g) \subseteq \mathtt{supp}(f) \bigcup \mathtt{supp}(g)$.*

Let f a Boolean function, and $Y \subseteq X$ a subset of variables, then $\mathtt{supp}(\exists_Y f) \subseteq \mathtt{supp}(f) - Y$.

Let f a Boolean function, and $Y \subseteq X$ a subset of variables, then $\mathtt{supp}\left(\exists_{|Y} f\right) \subseteq \mathtt{supp}(f) \bigcap Y$.

Lemma 3.3 (Support and Semantic)**.** *Let F be constraint system, then,*

$$\mathtt{supp}\left(\llbracket F \rrbracket\right) \subseteq \mathtt{supp}\left(F\right)$$

That is, a supporting variable of the semantic of F, must be in the support of F.

Proof. *If a variable is useless in every constraint then, it is useless in the semantic interpretation.* □

Chapter 4

Propagation and Consistency

This section shows how to exploit structural analysis in constraint problem with sparse and structured primal graph, assuming that one as an "efficient" existential projection operator.

We define a first version of adjacent propagation directly based upon the definition of a set of constraints.

Definition 4.1 (Adjacent Propagation (First Draft))**.** *Let $F = \{f_1, \ldots, f_m\}$ ba constraint system and $s, d \in \{1, \ldots, m\}$ be two indices.*

Then, we denote $F[s \rightsquigarrow d] := F' = \{f'_1, \ldots, f'_m\}$, defined as

- $\forall i \neq d, f'_i := f_i$, *and,*
- $f'_d := f_d \wedge \exists_{\lvert \mathtt{supp}(f_s) \cap \mathtt{supp}(f_d)} f_s$.

Lemma 4.1 (Compatibility of Adjacent Propagation (First Draft) with Primal Graph)**.** *Let F be a constraint system, then* $\mathrm{PG}(F[i \rightsquigarrow j]) \subseteq \mathrm{PG}(F)$.

The remainder of this chapter is upon the definition of "adjacent propagation", which essentially states a way of propagating information from one constraint to another inside a formula, while preserving both the semantic and the structure, that is, the set of satisfying valuations and the primal graph.

The propagation process is based upon the following lemma,

Lemma 4.2 (Saturation Equivalence)**.** *For any Boolean functions f and g, $f \subseteq g$ iff $f = f \wedge g$.*

Proof. *Trivial using using functional extensionality and Boolean reasoning* □

This lemma states, that one can always add a term g to a set of functions f which is redundant with terms which are already in this set. These extra terms are used to make the initial terms of formula "communicate" and reach some level of "consensus" or consistency. Our objective is to define a term which is minimal in size, while being maximal in term of the information it contains.

Lemma 4.3 (Existential Restriction is Redundant)**.** *For any Boolean function f, and subset Y of its variables, we have, $f \subseteq \exists_{|Y} f$.*

Lemma 4.4 (Saturation with Existential Restriction)**.** *For any two functions f and g and subset Y of their variables, $f \wedge g = f \wedge \left(\exists_{|Y} f \wedge g\right)$.*

Although, it does not look like much, this property is the semantic foundation of the present work. On the remainder of this chapter, we show that this formula can be used to locally propagate information, hence, preserving the primal graph (see Definition 3.1) of the formula.

Lemma 4.5 (Primal Graph and Saturation with Existential Restriction)**.** *Let F be a set of constraint, $f \in F$ a constraint of F, and $Y \subseteq X$ a subset of variables, we denote $F' = F \cup \{\exists_{|Y} f\}$.*
Then, $[\![F']\!] = [\![F]\!]$, $\mathtt{supp}(F') = \mathtt{supp}(F)$, $\mathrm{PG}(F) = \mathrm{PG}(F')$.

This last lemma shows that one can add the extra term $\exists_{|Y} f_i$ while preserving both the semantic and the macro-structure.

4.1 Constraint System Decomposition

We can now formalize how this term can be used to make constraints "communicate", while preserving a more local notion of structure, and eventually reaching a confluent fixpoint.

We want to base the "communication" between constraints on top of the notion of support, however, whenever "communication" happens it update a component possibly changing its set of supporting variables, which may lead to some unwanted complications.

Remark 4.1 (Conservative Constraint Merging)**.** *It is possible to reduce the number of components by merging together any two constraint f_1, f_2 such that $\mathtt{supp}(f_1) \subseteq \mathtt{supp}(f_2)$ into a new constraint $f_2' := f_1 \wedge f_2$. Such transformation has no impact on the primal graph.*

A minor issue of this definition is that a variable in some constraint may become useless due to adjacent propagation, to avoid the change in dimension of the formula, we introduce the notion graph decomposition allowing to (1) permanently "set" the support of each constraint, and (2) restrict which constraints are allowed to communicate.

Definition 4.2 (Graph Decomposition)**.** *Let X a set of items (e.g. vertices of a graph or variables of a formula), we call $D = (B, I)$ a graph decomposition on X where*

- *$B \subseteq \mathcal{P}(V)$ are called bags or component of the decomposition*
- *$I \subseteq B^2$ are called interconnection*
- *(support coverage) $\forall v \in V, \exists b \in B, v \in b$*
- *(locality) for any vertex $v \in V$, the sub-graph of D induced[a] by $\{b \in B \mid v \in b\}$ is connected.*
  More precisely:
  - *denoting $B_v := \{b \in B \mid v \in b\} \subseteq B$,*
  - *we have that $D[B_v]$ (the sub-graph induced of D induced by B_v) is connected*

[a]Given a graph $G = (V, E)$ and $U \subseteq V$ a subset of vertices V, $G[U]$ denotes the sub-graph induced by U, that is, $G[U] := (U, E \cap U^2)$

Definition 4.3 (Graph Decomposition of a Graph)**.** *Let $G = (V, E)$ an undirected graph, and $D = (B, I)$ be a graph decomposition on V.*

Then, we say that D is a graph decomposition of G iff

- *(edge coverage) $\forall (s, d) \in E, \exists b \in B, s \in b \wedge d \in b$*

Definition 4.4 (Graph Decomposition of a Constraint System)**.** *Let X be a set of variables, let F be a constraint system on X, let D be a graph decomposition on X.*

Then, we say that D a graph decomposition of F iff

- *(constraint coverage) $\forall f \in F, \exists b \in B, \mathtt{supp}(f) \subseteq b$.*

Definition 4.5 (Constraint System Decomposition)**.** *We define a constraint system decomposition (CSD) $\mathbf{F} = (D, F)$, where*

- *D is graph decomposition on X*
- *F is a mapping from components of the graph decomposition D to constraints, that is for each component $b \in B$, we denote f_b its associated constraint.*
- *(support inclusion) $\forall b \in B, \mathtt{supp}(f_b) \subseteq b$*

Remark 4.2. *In constraint system decomposition $\mathbf{F} = (D, F)$, D is a graph decomposition of F, but that is already enforced by the other two axioms.*

Remark 4.3 (Converting Constraint System into Constraint System Decomposition)**.** *One may convert an arbitrary constraint system F into a constraint system decomposition $\mathbf{F} = (D, F')$, assuming that the graph decomposition D is a graph decomposition of F.*

Then, for each constraint $f \in F$, we arbitrary select a component $b \in B$, such that $\mathtt{supp}(f) \subseteq b$, denoted $b(f)$.

Then, for every component b we define $f'_b := \bigwedge_{f \in F, b(f)=b} f$.

Hence, defining $F' := \{f'_b\}_b$ a more convenient constraint system.

One may show that $[\![\mathbf{F}]\!] = [\![F']\!] = [\![F]\!]$.

Definition 4.6 (Semantic Interpretation)**.** *We define the semantic interpretation $[\![\mathbf{F}]\!]$ of $\mathbf{F} = (D, F)$ as $[\![\mathbf{F}]\!] := \bigwedge_{b \in B} f_b$.*

Given two CSD $\mathbf{F}$ and $\mathbf{G}$, we say that they are equivalent (denoted $\mathbf{F} \equiv \mathbf{G}$) iff they are based upon the same decomposition and $[\![\mathbf{F}]\!] = [\![\mathbf{G}]\!]$.

Constraint System Decomposition (CSD) allows to define adjacent propagation in a way that leaves "supporting variables" unchanged as they are actually set by the graph decomposition.

Definition 4.7 (Adjacent Propagation)**.** *Given* $\mathbf{F} = (D, F)$*, and* $(s, d) \in I$ *an interconnection of* D*, we define* $\mathbf{F}[s \rightsquigarrow d]$ *as* $\mathbf{F}' = (D, F')$*, where:*

- $\forall i, i \neq d \Rightarrow f'_i := f_i$
- $f'_d := f_d \wedge \exists_{(s-d)} f_s$

Proof (Proof that $\mathbf{F}'$ is a CSD with respect to D)**.** *First, we show support inclusion of each constraint :*

- $\forall i, i \neq d \Rightarrow \mathtt{supp}\,(f'_i) = \mathtt{supp}\,(f_i) \subseteq b_i$*, and,*
- $\mathtt{supp}\,(f'_d) \subseteq \mathtt{supp}\,(f_d) \bigcup \underbrace{\mathtt{supp}\left(\exists_{(s-d)} f_s\right)}_{\subseteq s-(s-d) \subseteq d} \subseteq d.$

□

Now that adjacent propagation is defined, it opens various questions on its behavior

- as it operates in a finite structure, adjacent propagation is bound to reach a fixpoint at some point.
- is this fixpoint unique ?
- can we characterize fixpoint(s) ? and, if several, can we characterize the "lowest" for some well-chosen order ?
- does this operator make the formula in some well-known structure ?

Remark 4.4 (Adjacent Propagation versus Message Passing)**.** *Adjacent propagation is very similar to message passing, the main difference lies in intention*

- *on the one hand, message passing, allows to transmit some information from one component to the next,*
- *on the over hand, adjacent propagation is semantically defined as the maximal amount of information one can transmit (using* $\exists$*)*

To avoid unintentional confusion between both notions, we keep distinct terms for both, but as adjacent propagation is message passing with a maximal message (by definition of $\exists$*)*

- *on the one hand, any property or technique available for message passing is also available with adjacent propagation, and,*
- *on the other hand, any "message" respecting the expressiveness of our framework is necessarily an approximation of adjacent propagation.*

Finally, adjacent propagation is defined for constraint system decomposition, how one can turn an arbitrary formula into constraint system decomposition ?

All these questions are addressed in the following sections.

4.2 Partially Ordering Constraint System Decomposition

For convenience, if not mentioned otherwise, we assume a global graph decomposition $D = (B, I)$ for the remainder of this section. In particular, one may denote $\mathbf{F} = F = \{f_b\}_{b \in B}$ (furthermore, $\mathbf{G} = G = \{g_b\}_{b \in B}$)

For the sake of self-contentness, we remind the reader of the generally admitted definition of partial order.

Definition 4.8 (Partial Order)**.** *Let S be a set of element and $R \in S \times S \to \mathbb{B}$ a binary relation (used with infix notation).*

Then, we say that R is a partial order iff

- *R is transitive : $x\ R\ y \wedge y\ R\ z \implies x\ R\ z$, and,*
- *R is reflexivity : $x\ R\ x$, and,*
- *R is anti-symmetric : $(x\ R\ y \wedge y\ R\ x) \iff x = y$*

Definition 4.9 (Partial Order on Constraint System Decomposition : More Saturated Than)**.** *We define the following partial order $\preccurlyeq$ on constraint system decomposition (with the same graph decomposition) Given two **equivalent** constraint system decomposition $\mathbf{F}$ and $\mathbf{G}$, we say that $\mathbf{F}$ is more saturated than $\mathbf{G}$ (denoted $\mathbf{F} \preccurlyeq \mathbf{G}$) iff $\forall b$, $f_b \subseteq g_b$.*

We notice that $\preccurlyeq$ defines a partial order on constraint system decomposition.

Furthermore, we define strictly more saturated than as $\mathbf{F} \prec \mathbf{G} := \mathbf{F} \preccurlyeq \mathbf{G} \wedge \mathbf{F} \neq \mathbf{G}$, that is, $\mathbf{F}$ is more saturated than $\mathbf{G}$ and for at least one component $b \in B$, $f_b \subsetneq g_b$.

Furthermore, if the domain $\mathcal{A}$ is finite, then $\prec$ is a well-founded[a] partial order.

[a] well-founded : all decreasing chains are finite, additionally, all increasing chains are finite.

Proof (Proof of $\preccurlyeq$ is a partial order)**.** *We may prove that $\preccurlyeq$ is a partial order as a Cartesian-exponent (finitely iterated Cartesian product) of a partial order ($\subseteq$) intersected with an equivalence relation $\equiv$.* □

Proof (Proof of $\prec$ is well-founded)**.** *The lattice is finite, as the domain $\mathcal{A}$ is finite, the number n of variable is finite and the number m of constraint is finite as well.* □

Definition 4.10 (Acceptable Decomposition)**.** *Let X be a set of variables, let f be a function on X and D a graph decomposition on X.*

We denote $\mathrm{CSD}(D, f)$ *the set of constraint system decomposition F with decomposition D and semantic f:*

$$\mathrm{CSD}(D, f) = \{\mathbf{F} = (D, F) \mid [\![F]\!] = f\} \ .$$

We say that a decomposition is acceptable D with respect to a function f iff

$$\mathrm{CSD}(D, f) \neq \emptyset \ .$$

Lemma 4.6 (Characterization Of Equivalent Constraint System Decomposition)**.** *Let $\mathbf{F} = (D, F)$ be a CSD, then* $\mathrm{CSD}(D, [\![F]\!])$ *is the set of all CSD equivalent to $\mathbf{F}$ (according to Definition 4.6).*

We may denote $\bar{\mathbf{F}} = \mathrm{CSD}(\mathbf{F}) = \mathrm{CSD}(D, [\![F]\!])$.

Proof. *Unfold*
- *Definition 4.6 [Semantic Interpretation], and,*
- *Definition 4.10 [Acceptable Decomposition].*

□

Lemma 4.7 (Equivalent Constraint Systems are Semi-Lattices)**.** *Let f be a function a D an acceptable decomposition of f, then, $(\mathrm{CSD}(D, f), \prec, \wedge$-wise$)$ is a finite semi-lattice.*

Proof (Proof of $(\prec, \wedge$-wise$)$ is a semi-lattice on equivalent CSD)**.** *First, we prove that the above triplet define semi-lattice.*
- *Starting from, $(\mathbb{B}, \wedge, \vee)$ is a lattice*
- *Hence, $(\mathcal{A}^X \to \mathbb{B}, <, \wedge, \vee)$ is a lattice as a Cartesian exponent of a lattice*
- *Hence, $($constraint system decomposition$, <, \wedge, \vee)$ is a lattice, as a finite number of Cartesian product of lattices (or Cartesian exponent depending on the exact representation).*
- *Hence, $(\mathrm{CSD}(D, f), \prec, \wedge)$ defines a set of semi-lattices of the above, exactly one per equivalence class (according to "$\equiv$" on constraint system decomposition). ($\wedge$ being an internal law of theses sub-lattices).*

□

Lemma 4.8 (Adjacent Propagation Is Contracting)**.** *Let $D = (B, I)$ be a graph decomposition and let $\mathbf{F} = (D, F)$ be a constraint system decomposition and two adjacent components $(b_1, b_2) \in I$ in $\mathbf{F}$, then, $\mathbf{F}[b_1 \rightsquigarrow b_2] \preccurlyeq \mathbf{F}$.*

Proof. *Unfold definition and using, on the one hand, the property that $F \wedge G \subseteq F$, on the other hand, using Lemma 4.2 [Saturation Equivalence] to prove semantic preservation.* □

Lemma 4.9 (Adjacent Propagation Sequence Is Contracting)**.** *Let $\mathbf{F}$ a constraint system decomposition and p a sequence of adjacent propagation, then $\mathbf{F}[p] \preccurlyeq \mathbf{F}$*

Proof. *By induction over the sequence and using above lemma.* □

4.2.1 Local Consistency and Confluence

This section introduces a local notion of consistency with respect to a function and a graph decomposition.

Definition 4.11 (Local Consistency)**.** *Let* $\mathbf{F} = F$ *a constraint system decomposition, and* $i = (s, d) \in I$ *an interconnection, we say that* i *is locally consistent (or saturated) in* $\mathbf{F}$ *iff* $\mathbf{F}[s \rightsquigarrow d] = \mathbf{F}$ *(that is,* $f_s \subseteq \exists_{|s} f_d$*).*

Furthermore, we say that a CSD is locally consistent iff for all interconnection $i \in I$*,* i *is locally consistent in* $\mathbf{F}$*.*

Lemma 4.10 (Monotony of adjacent propagation with respect to $\preccurlyeq$)**.** *Let* $D = (B, I)$ *be a graph decomposition,* $\mathbf{G}$ *and* $\mathbf{F}$ *two equivalent CSD, and, two adjacent components* $(s, d) \in I$ *in* $\mathbf{F}$*.*

$$\mathbf{G} \preccurlyeq \mathbf{F} \Rightarrow \mathbf{G}[s \rightsquigarrow d] \preccurlyeq \mathbf{F}[s \rightsquigarrow d] \ .$$

Proof. *We denote* $\mathbf{G}' := \mathbf{G}[s \rightsquigarrow d]$ *and* $\mathbf{F}' := \mathbf{F}[s \rightsquigarrow d]$*.*
By unfolding definition, one may prove that, for any component $b \in B \backslash \{d\}$*,* $g'_b = g_b \subseteq f_b = f'_b$*. Furthermore,*

$$g'_d \ =^{*1} \ g_d \wedge \exists_{|d} g_s \ \subseteq^{*2} \ f_d \wedge \exists_{|d} f_d \ =^{*3} \ f'_d$$

Justified by
1. *unfolding definition of adjacent propagation*
2. *by monotony of* $\wedge$ *and* $\exists$ *with respect to* $\subseteq$ *and using* $G_d \subseteq F_d$ *and* $G_s \subseteq F_s$
3. *unfolding definition of adjacent propagation*

□

Lemma 4.11 (Monotony of adjacent propagation sequences with respect to $\preccurlyeq$)**.** *Let* $\mathbf{F}$ *and* $\mathbf{G}$ *two constraint system decomposition, and* p *a sequence of adjacent propagation, then*

$$\mathbf{G} \preccurlyeq \mathbf{F} \Rightarrow \mathbf{G}[p] \preccurlyeq \mathbf{F}[p] \ .$$

Proof. *By induction over the sequence and using above lemma.* □

Lemma 4.12 (Weak Quasi-Commutativity of Adjacent Propagation)**.** *Let* $\mathbf{F}$ *a constraint system decomposition, let* p_1 *and* p_2 *two sequences of adjacent propagation, then :*
1. $\mathbf{F}[p_1][p_2] \preccurlyeq \mathbf{F}$*, and,*
2. $\mathbf{F}[p_1][p_2] \preccurlyeq \mathbf{F}[p_1]$*, and,*
3. $\mathbf{F}[p_1][p_2] \preccurlyeq \mathbf{F}[p_2]$*, and,*
4. $\mathbf{F}[p_1][p_2] \preccurlyeq \mathbf{F}[p_1] \wedge \mathbf{F}[p_2]$*.*

Proof. *The proof mostly relies on (strong) consistency of adjacent propagation with "*$\preccurlyeq$*" along with the semi-lattice structure of* $\preccurlyeq$
1 and 2 $\mathbf{F}[p_1][p_2] \preccurlyeq \mathbf{F}[p_1] \preccurlyeq \mathbf{F}$ *(using consistency Lemma 4.9*
3 $\mathbf{F}[p_1][p_2] \preccurlyeq \mathbf{F}[p_2]$ *(using monotony Lemma 4.11 with* $\mathbf{F}[p_1] \preccurlyeq \mathbf{F}$ *and* $[p_2]$*)*

4 Using semi-lattice structure of $\preccurlyeq$ with respect to $\wedge$.

□

Benoit: This looks like a variation of the Knaster-Tarski theorem

Joan: [RQ] this theorem does not seem used apart from citation

Theorem 4.1 (Confluence of Adjacent Propagation)**.** *Let $\mathbf{F}$ a constraint system decomposition and p_1 and p_2 two sequences of adjacent propagation, such that $\mathbf{F}[p_1]$ and $\mathbf{F}[p_2]$ are locally consistent, then $\mathbf{F}[p_1] = \mathbf{F}[p_2]$.*

Proof.
- *First, using stability of locally consistent constraint system decomposition by adjacent propagation $\mathbf{F}[p_1][p_2] = \mathbf{F}[p_1]$ and $\mathbf{F}[p_2][p_1] = \mathbf{F}[p_2]$.*
- *Using above Lemma 4.12 [Weak Quasi-Commutativity of Adjacent Propagation], we have $\mathbf{F}[p_1] = \mathbf{F}[p_1][p_2] \preccurlyeq \mathbf{F}[p_2]$ and $\mathbf{F}[p_2] = \mathbf{F}[p_2][p_1] \preccurlyeq \mathbf{F}[p_1]$.*
- *That is, $\mathbf{F}[p_1] \preccurlyeq \mathbf{F}[p_2]$ and $\mathbf{F}[p_2] \preccurlyeq \mathbf{F}[p_1]$.*
- *Hence, using antisymmetry of $\preccurlyeq$, $\mathbf{F}[p_1] = \mathbf{F}[p_2]$.*

□

In essence, the above theorem combined with well-foundedness states that every constraint system decomposition eventually (in a finite number of steps) reduces (via iterated adjacent propagation) to a unique locally consistent constraint system decomposition.

4.2.2 Local Projection and Bottom Element

In the previous section we prove that equivalent constraint systems are semi-lattices (see Lemma 4.7).

In this section, we prove that these lattices have a bottom element (that is a unique minimal element with respect to the "more saturated than" partial order).

Furthermore, we introduce a characterization of this bottom element, showing that is can be computed in a global way even if the "more saturated than" partial order is not well-founded (which is likely to happen if the domain $\mathcal{A}$ of variables is infinite).

Definition 4.12 (Local Projection of a Boolean function)**.** *We define the* local projection *(LP) operator as follows:*

Given a Boolean function $f \in \mathcal{A}^X \to \mathbb{B}$ on variables X and a graph decomposition $D = (B, I)$ on X, we define the constraint system $\mathrm{LP}_D(f) := \mathbf{F} := (D, F)$, with $F := \{f_b\}_{b \in B}$, with $\forall b \in B$, $f_b := \exists_{|b} f$.

Definition 4.13 (Acceptable Decomposition)**.** *Let D be a graph decomposition, and f be a Boolean constraint.*

We say that D is an acceptable graph decomposition iff $[\![\mathrm{LP}(D, f)]\!] = f$.

Definition 4.14 (Global Consistency)**.** *Given a constraint system decomposition $\mathbf{F}$, we say*

that $\mathbf{F}$ *is* globally consistent *(or has global consistency) iff*

$$\forall b \in B, f_b = \exists_{|b} [\![F]\!] \ .$$

Lemma 4.13 (Basic Properties of the Local Projection of a Boolean function)**.** *Using above notations, one may prove that*

1. *for all Boolean functions* f, $f \subseteq [\![\mathrm{LP}(f)]\!]$
2. *for all Boolean functions* f, $\mathrm{LP}([\![\mathrm{LP}(f)]\!]) \equiv \mathrm{LP}(f)$
3. *for any constraint system decomposition* $\mathbf{F}$, $\mathbf{F}$ *is globally consistent iff* $\mathrm{LP}([\![\mathbf{F}]\!]) \equiv \mathbf{F}$
4. *for all Boolean functions* f, $\mathrm{LP}(f)$ *is a globally consistent constraint system decomposition.*

Proof (Proof of Inclusion)**.** *Let* $\sigma \in \mathbb{B}^X$ *a satisfying valuation of* f*, then, for each* $b \in B$*,* $\sigma_{|\mathbf{b}}$ *(the restriction of* σ *to the variables in* b*) is a solution of* $\mathrm{LP}(f)_b$*, hence* σ *is a satisfying valuation of* $\mathrm{LP}(f)$*.*

Hence, $f \subseteq \mathrm{LP}(f)$*.* □

Proof (Proof of Idempotence)**.** *Local projection is idempotent,* $\mathrm{LP}(\mathrm{LP}(f)) \equiv \mathrm{LP}(f)$*.*

First, using proof of inclusion, $f \subseteq [\![\mathrm{LP}(f)]\!] \subseteq [\![\mathrm{LP}(\mathrm{LP}(f))]\!]$*.*

Hence, we only have to prove that $[\![\mathrm{LP}([\![\mathrm{LP}(f)]\!])]\!] \subseteq [\![\mathrm{LP}(f)]\!]$*, it is enough to prove that, for each component* $b \in B$*,* $f''_b \equiv f'_b$*, i.e.,* $\exists_{|b} f' \equiv f'_b$ *(with* $f' := [\![F']\!]$*)*

Let $\sigma_b \in \mathbb{B}^X$ *a satisfying valuation of* $f''_b := \exists_{|b} f'$*, that is there exists an extension* σ *of* σ_b *such that* σ *satisfies* $\mathbf{F}'$*. However,* $\mathbf{F}' := \bigwedge_{b \in B} f'_b$*, hence,* σ *satisfies* f'_b*, hence* σ_b *satisfies* f'_b*.*

Hence, $\forall b \in B, f''_b \subseteq f'_b$*, hence,* $\mathbf{F}'' \subseteq \mathbf{F}'$*, hence,* $[\![\mathrm{LP}([\![\mathrm{LP}(f)]\!])]\!] \subseteq [\![\mathrm{LP}(f)]\!]$*, hence* $\mathrm{LP}(\mathrm{LP}(f)) \equiv \mathrm{LP}(f)$*.* □

Proof (Proof of the characterization of Global Consistency)**.** *By definition of both the local projection operator and global consistency.* □

Proof (Proof of Global Consistency)**.** *Immediate by combining properties of idempotence of local projection and the characterization of global consistency using local projection.* □

Lemma 4.14 (Locally Globally Consistent)**.** *Let* $\mathbf{F} = F$ *a constraint system decomposition, then, for any component* $b \in B$*, the following inclusion holds :*

$$\exists_{|b} [\![\mathbf{F}]\!] \subseteq f_b \ .$$

Proof. *Let σ_b a satisfying valuation of $\exists_{|b}\mathbf{F}$.*
Then, there exists σ an extension of σ_b such that σ satisfies $\mathbf{F}$.
However $[\![\mathbf{F}]\!] = \bigwedge_b f_b$, hence $\sigma_b = (\sigma)_{|b}$ satisfies f_b. □

Theorem 4.2 (Globally Consistent is Locally Consistent)**.** *Let $\mathbf{F}$ a globally consistent constraint system decomposition, then $\mathbf{F}$ is locally consistent.*

Proof. *Assuming $\mathbf{F}$ is not locally consistent, lets prove that $\mathbf{F}$ is not globally consistent.*
Then, there exists a component $b \in B$, such that $f_b \neq \bigwedge_{b' \in N_D(b)} \left(\exists_{|b} F_{b'}\right)$.
That is, there exists another component $b' \in N_D(b) \backslash \{b\}$, such that $f_b \not\subseteq \exists_{|b} F_{b'}$.
That is, there exists a satisfying valuation σ_b which cannot be extended into a satisfying valuation of $F_{b'}$, hence cannot be extended into a satisfying valuation of F.
Hence, $\mathbf{F}$ is not globally consistent, which contradicts the first hypothesis. □

Lemma 4.15 (Globally Consistent CSD Are Minimal)**.** *Let $\mathbf{F}$ a globally consistent CSD, then $\mathbf{F}$ is minimal with respect to "$\prec$". (that is, there exists no CSD $\mathbf{G} \prec \mathbf{F}$)*

Proof. *By contradiction, we assume that there exists $\mathbf{G} \prec \mathbf{F}$,*
Then there exist (at least) a component $b \in B$ such that $g_b \subsetneq f_b$,
Remembering that $\mathbf{F}$ is globally consistent, we have $f_b = \exists_{|b} G$.
Hence, using Lemma 4.14 [Locally Globally Consistent], $\exists_{|b} G \subseteq g_b \subsetneq \exists_{|b} G$. Which leads to a contradiction. □

Lemma 4.16 (Local Projection is the Bottom Element of Equivalent Constraint System Decomposition)**.** *Let f a constraint over X, and D an acceptable decomposition of f.*
We retrieve the semi-lattice $(\mathcal{F}, \prec, \wedge\text{-wise})$ as defined in Lemma 4.7.
We prove that $\mathrm{LP}(D, f)$ is the bottom element of this semi-lattice.

Proof. *Furthermore, as the lattice is finite, each semi-lattice is also finite hence have a bottom element.*
We prove that this bottom element is exactly $\mathrm{LP}([\![\mathbf{F}]\!])$

- *let $\mathbf{F}$ a constraint system decomposition, lets assume there exists a constraint system decomposition $\mathbf{G} \prec \mathrm{LP}([\![\mathbf{F}]\!])$*
- *however, according to Lemma 4.13 [Local Projection Of A Boolean Function] (4th statement), $\mathrm{LP}([\![\mathbf{F}]\!])$ is globally consistent, which according to Lemma 4.15 [Globally Consistent CSD Are Minimal] leads to a contradiction.*

□

4.2.3 Global Consistency is not Local Consistency

We exhibit a counter-example in the Boolean case (hence, we can generalize it to any finite domain case) of a formula that is locally consistent but not globally consistent.

This example relies on the presence of a loop in the decomposition graph, indeed as we shall see in Chapter 5 [Tractability Of Tree Decomposition] forbidding loops from graph decomposition allows to make both notions coincides.

Definition 4.15 (Equivalence Cycle)**.** *Let $n \geq 1$ an integer and $b \in \mathbb{B}$ a Boolean, then, we define* $\mathrm{EC}_n^b := \bigwedge_{i=0}^{n-2} (x_i \Leftrightarrow x_{i+1}) \wedge (x_{n-1} \Leftrightarrow (x_0 \oplus b))$.

We call $\mathrm{EC}_n^{\bot}$ *the Satisfiable Equivalence Cycles, which have exactly two solutions* $\vec{0}$ *and* $\vec{1}$.

We call $\mathrm{EC}_n^{\top}$ *the UnSatisfiable Equivalence Cycles, which are unsatisfiable, that is have exactly no solutions.*

Lemma 4.17 (Semantic Of Equivalence Cycle)**.** *Let $n \geq 1$ an integer and $b \in \mathbb{B}$ a Boolean, then,* $\mathrm{EC}_n^b(\sigma_X) := \neg b \wedge \left(\sigma_X = \vec{\bot} \vee \sigma_X = \vec{\top}\right)$

Proof. *Using case analysis on $b \in B$ and Boolean reasoning.* □

Definition 4.16 (Implication Cycles)**.** *Let $n \geq 1$ an integer, then, we define*

$$\mathrm{IC}_n := \bigwedge_{i=0}^{n-2} (x_i \Rightarrow x_{i+1}) \wedge (x_{n-1} \Rightarrow x_0) \quad .$$

Lemma 4.18 (Semantic Of Implication Cycles)**.** *Let $n \leq 1$ an integer, then,* $[\![\mathrm{IC}_n]\!] = \left[\!\left[\mathrm{EC}_n^{\bot}\right]\!\right]$.

Proof. *Using case analysis on $b \in B$ and implication graph resolution.* □

Example 4.1 (Example of Locally Consistent but Globally Inconsistent CSD)**.** *We shall see that local consistency does not imply global consistency :*

- *implication cycles form a class of satisfiable examples, and,*
- *unsatisfiable equivalence cycles form a class of unsatisfiable examples*

In particular,

1. IC_1 *and* $\mathrm{EC}_1^{\top}$*, no, there are globally consistent,*
2. IC_2 *and* $\mathrm{EC}_2^{\bot}$*, yes, but they are not a minimal decomposition,*
3. IC_3 *and* $\mathrm{EC}_3^{\bot}$*, yes, but their primal graph are cliques,*

4. IC_4 *and* $\mathrm{EC}_4^\top$*, yes!*

Proof. *First, we prove that the formula is locally consistent using the two following statements :*

- $(\exists_x,\, x \Rightarrow y) \equiv \top$ *(using* $x := \bot$*)*
- $(\exists_x,\, x \Rightarrow (y \oplus b)) \equiv \top$ *(using* $x := \bot$*)*
- $(\exists_y,\, x \Rightarrow y) \equiv \top$ *(using* $y := \top$*)*

From Lemma 4.18 [Semantic Of Implication Cycles], we deduce that :

- *In the satisfiable case, the local projection on each component is* $\{x_i, x_{i'}\}$ *is* $x_i \Leftrightarrow x_{i'}$*.*
- *In the unsatisfiable case, the local projection on each component is* $\bot$

Therefore, forming two classes of locally consistent but not global consistent constraint system decomposition. □

4.3 Conclusion

In order we introduced the following notions

- constraint system decomposition
- adjacent propagation
- partial order over constraint system decomposition
- local consistency
- global consistency
- local projection (which result is globally consistent)
- finally showing that local consistency and global consistency are disjoint notions

We know that any propagation strategy is confluent, however we may only exhibit a finitely terminating strategy if the domain $\mathcal{A}$ is finite itself.

Furthermore, if the domain is infinite, as soon as there is a loop in the decomposition, we may explicit an example in which no propagation strategy terminates.

Chapter 5

Tractability of Tree Decomposition

As we saw in previous chapter, adjacent propagation in constraint system decomposition has two distinct notions of consistency : local consistency and global consistency. Our example which illustrates this distinction is based upon the existence of a loop in the underlying graph decomposition.

This chapter studies the acyclical case, that is when the graph decomposition is actually a tree decomposition.

In particular, we show that in tree decomposition

- In locally consistent formula, any satisfying valuation of a component can be extended into a satisfying valuation of the whole formula
- A formula is globally consistent iff locally consistent (while the direct implication is always true, the reverse implication is gained from the restriction to tree decomposition)
- We exhibit a two-phases algorithm allowing to reach local consistency and model its cost (in term of computation time and memory) with respect to the chosen decomposition.

Definition 5.1 (Constraint System Tree Decomposition)**.** *Let* $\mathbf{F} = (D, F)$ *a constraint system decomposition (CSD), we say that* $\mathbf{F}$ *is a constraint system tree decomposition (CSTD) iff* D *is a tree decomposition (that is,* D *is a tree).*

5.1 Local Consistency Is Global Consistency

Lemma 5.1 (Characterization Of Tree Decomposition)**.** *Let* X *be a finite set of item, let* D *be a graph decomposition on* X*, then,* D *is a tree decomposition iff* D *is a forest.*

Definition 5.2 (Back-Selection)**.** *Below the description of the back-selection algorithm in pseudo-code*

```
(* [back_selection F σ^0 = σ]
* with :
```

```
*  - F a constraint system tree decomposition
*  - σ⁰ a partial satisfying valuation of F₀
*    + with F₀ the root component of F
*
* returning :
*  - σ a satisfying valuation of F extending σ⁰
*)
let rec back_selection F σ⁰ =
  let σ₀ = a completion of σ⁰ satisfying F₀ (the root of F)
  let [F₁; ···; Fₖ] = the sons of F₀
  for i = 1 to k
  do
    let σᵢ⁰ the restriction of σ₀ to the support of Fᵢ, and,
    let σᵢ = back_selection Fᵢ σᵢ⁰
  done
  return merge_valuation [σ₁; ...; σₖ]
```

Remark 5.1. *Without further assumption, this algorithm will likely fail at the* `merge_valuation` *step.*

The next lemma states that in the case of locally consistent CSTD, this algorithm always succeed.

In Chapter 6 we shall see that the CSTD being forward consistent (cf. Definition 5.6 [Forward/Backward Consistency]) is actually enough for this algorithm to succeed.

Lemma 5.2 (Back-Selection Succeeds on Locally Consistent CSTD)**.** *let* $\mathbf{F} = (D, F)$ *a constraint system decomposition, with* D *a tree decomposition, then for any component* $b \in B$*, then, for any partially satisfying valuation* σ_b *of* f_b*,* `back_selection` $\mathbf{F}$ σ^0 *succeeds and returns* σ *an extension of* σ^0 *is a satisfying valuation of* F*, that is* $\mathbf{F}$*.*

Proof. *We set* b *as the root of the tree-decomposition, then, we can reason by induction on tree-structure of the decomposition.*

Lets show by induction the following statement "Given σ_b^0 *a partial satisfying valuation of the root component* b*, any local extension* σ_b *of* σ_b^0 *can be extended into a satisfying valuation* σ *of* F*"*

Let σ_0 *a partial satisfying valuation of the root component* b*,*
Let σ_b *an extension of* σ_0 *into a satisfying valuation of the root component* b*.*
We denote $\sigma^0_{b_i}, \ldots, \sigma^0_{b_k}$ *the* k *projections of* σ_b *on the interface of* b *with its sons* b_i*.*
Then, by induction, σ_i^0 *can be extended into a solution* σ_i *of its sub constraint system decomposition.*
Using Lemma 3.1 [Separation Property Of Tree Decomposition], we have : $\forall i, j$, $\mathtt{supp}(\sigma_i) \cap \mathtt{supp}(\sigma_j) \subseteq \mathtt{supp}(\sigma_b)$*.*
Hence, σ_b *can be consistently extended using all* $(\sigma_i)_i$ *into* σ *a solution of the constraint system decomposition.* □

Lemma 5.3 (Extending Local Solution in Locally Consistent CSTD)**.** *let* $\mathbf{F} = (D, F)$ *a constraint system decomposition, with* D *a tree decomposition, then for any component* $b \in B$*, then, for any satisfying valuation* σ_b *of* f_b*, there exists an extension* σ *such that* σ *is a satisfying valuation of* F*, that is* $\mathbf{F}$*.*

Proof. *Corollary of Lemma 5.2.* □

Theorem 5.1 (Using Tree Decomposition Local Consistency Iff Global Consistency)**.** *let* $\mathbf{F} = (D, F)$ *a constraint system decomposition, with* D *a tree decomposition, then* $\mathbf{F}$ *is globally consistent iff* $\mathbf{F}$ *is locally consistent*

Proof. *First,* $\mathbf{F}$ *is globally consistent* $\Rightarrow$ $\mathbf{F}$ *is locally consistent always holds using arbitrary decomposition.*

Second, lets prove that if $\mathbf{F}$ *is locally consistent then* $\mathbf{F}$ *is globally consistent.*

That is, for any component $b \in B$*,* $f_b = \exists_{|b} F$*.*

- *Lemma 4.14 [Locally Globally Consistent] states that* $\exists_{|b} F \subseteq f_b$ *holds in arbitrary decomposition.*
- $f_b \subseteq \exists_{|b} F$ *holds using Lemma 5.3 (Extensible Local Solution in Tree Decomposition)*

□

5.2 Orientation and Global Propagation Scheme

Definition 5.3 (Orienting an Undirected Graph)**.** *Let* $U = (V, E)$ *an undirected graph and* $O = (V', E')$ *a directed graph, we say that* O *is an orientation of* U *if* $V' = V$*,* O *is acyclical, and,*

$$\forall (x, y) \in E, ((x, y) \in E') \oplus ((y, x) \in E')$$

Definition 5.4 (Rooting a Tree)**.** *Let* $T = (V, E)$ *a an undirected tree, and* $r \in V$*, we define* T_r *the rooting of* T *in* r *as an orientation of* T *where each arcs are directed toward the root.*

$$T_v := (V, \{(v, w) \in E \mid d_T(v, r) < d_T(w, r)\})$$

For the sake of self-contentness we state the commonly used graph transpose, that is the directed graph where every arc (s, d) is replaced by and (d, s).

Definition 5.5 (Graph Transposed)**.** *Let* $G = (V, E)$ *a directed graph, we define* $G^{\intercal} := (V, E^{\intercal})$ *the transposed of* G *as the graph where each edge's direction has been reversed that is* $E^{\intercal} := \{(y, x) \mid (x, y) \in E\}$*.*

Definition 5.6 ((Locally) Forward/Backward Consistency)**.** *Let* $\mathbf{F} = (D, F)$ *a constraint system decomposition and* O *an orientation of* D*, we say that* $\mathbf{F}$ *is forward consistent with respect to* O *iff*

$$\forall (s, d) \in O.E,\ f_d \subseteq \exists_{|d} f_s$$

Furthermore, we say that $\mathbf{F}$ *is backward consistent with respect to* O *iff* $\mathbf{F}$ *is forward consistent with respect to* O^{t}*.*

Lemma 5.4 (Characterization of Local Consistency as Forward and Backward Consistent)**.** *Let* $\mathbf{F} = (D, F)$ *a constraint system decomposition and* O *an orientation of* D*, then,* $\mathbf{F}$ *is both forward and backward consistent with respect to* O*, iff,* $\mathbf{F}$ *is locally consistent*

Proof. *Expand thrice definitions and rewrite.* □

Definition 5.7 (Forward Propagation)**.** *Let* $\mathbf{F} = (D, F)$ *a constraint system decomposition and* O *an orientation of* D*.*

We inductively define the forward propagation ($\mathrm{FP}(\mathbf{F}, O)$*) as follows, select a non visited component such that the source of all incoming arcs (in* O*) have been visited (the DAG configuration of* O *ensures the existence of such a component).*
Perform adjacent propagation with on all incoming arcs.
The process stops when all components have been visited.

Proof (Proof of Termination)**.** *The forward propagation process terminates because of DAG-induction.* □

Lemma 5.5 (Basic Properties Of Forward Propagation)**.** *Let* $\mathbf{F} = (D, F)$ *a constraint system decomposition and* O *an orientation of* D*.*

One may prove that the resulting constraint system decomposition $\mathbf{F}' := \mathrm{FP}(\mathbf{F}, O)$ *is:*

1. *independent of the order on which computation have been made,*
2. *more saturated than the operand :* $\mathrm{FP}(\mathbf{F}, O) \preccurlyeq \mathbf{F}$*), and,*
3. *forward consistent.*

Proof (Proof of Increased Saturation ($\mathrm{FP}(\mathbf{F}' \preccurlyeq \mathbf{F})$)**.** *Finite number of composition of adjacent propagation which each preserve semantic equivalence and decrease according to the partial order* $\preccurlyeq$*.* □

Proof (Proof of Forward Consistency)**.** *First, we compute a topological order S (that is an evaluation sequence) over the orientation O (which is a DAG).*
Prove by induction over S that all forward propagated interconnections remain forward consistent. □

Definition 5.8 (Backward Propagation)**.** *Let $\mathbf{F} = (D, F)$ a constraint system decomposition and O an orientation of D.*

We define backward propagation with respect to O as the forward propagation with respect to the transposed of $O^{\mathtt{t}}$: $\mathrm{BP}(\mathbf{F}, O) := \mathrm{FP}(\mathbf{F}, O^{\mathtt{t}})$.

Lemma 5.6 (Basic Properties Of Backward Properties)**.** *Let $\mathbf{F} = (D, F)$ a constraint system decomposition and O an orientation of D.*

One may prove that the resulting constraint system decomposition $\mathbf{F}' := \mathrm{FP}(\mathbf{F}, O)$ is:

1. *independent of the order on which computation have been made,*
2. *more saturated than the operand :* $\mathrm{BP}(\mathbf{F}, O) \preccurlyeq \mathbf{F}$*), and,*
3. *backward consistent.*

Proof. *Trivial, by unfolding definition of backward propagation and using Lemma 5.5 which states the same properties about forward propagation.* □

Definition 5.9 (Global Propagation Process)**.** *Given $\mathbf{F} = (D, F)$ a constraint system decomposition and O an orientation of D, we define the global propagation process* $\mathrm{GP}(\mathbf{F}, O) := \mathrm{BP}(\mathrm{FP}(\mathbf{F}, O), O)$.

Definition 5.10 (Natural Orientation)**.** *Let $D = (B, I)$ a tree-decomposition and $b \in B$ any of its component.*

We define the natural orientation of D toward b (denoted $\mathcal{O}(D, b)$) as the orientation where all edges are directed toward the root b. (If D is not connected, then select one root per connected component and naturally orient each of them).

Theorem 5.2 (Backward Propagation Of Forward Consistent Is Locally Consistent)**.** *Let $\mathcal{O}$ a natural orientation of D toward a given component b_0.*

Let $\mathbf{F}$ a forward consistent CSTD according to $\mathcal{O}$.

Then, $\mathrm{BP}(\mathbf{F})$ *is locally consistent.*

Proof. *We denote $\mathbf{F}^1 := \mathbf{F}$ and $\mathbf{F}^2 := \mathrm{BP}(\mathbf{F}^1, O)$ (A.2).*
In particular, $\mathbf{F}^1$ is forward consistent with respect to O (B.1), $\mathbf{F}^2$ is backward consistent with respect to O (B.2).

Furthermore, $\mathbf{F}^2 \preccurlyeq \mathbf{F}^1$ (C).

Hence, given a directed interconnection $(s, d) \in I$, we have the following statements

- $(F^2)_d \subseteq (F^1)_d$ *(C_d)*
- $(F^1)_d \subseteq \exists_{|d}(F^1)_s$ *(B.1)*
- $(F^2)_s = (F^1)_s \wedge \exists_{|s}(F^2)_d$ *(A.2) (holds as every component has at most one outgoing interconnection with respect to orientation O)*

Lets prove that the directed interconnection (s, d) is locally foward consistent, that is, $(F^2)_d \subseteq \exists_{|d}(F^2)_s$. The proof is done by using satisfying valuation level reasoning :

- *let $\sigma_d \models (F^2)_d$*
  - *hence, $\sigma_d \models \exists_{|s}(F^2)_d$ (a)*
- *hence, $\sigma_d \models (F^1)_d$ (using C_d)*
- *hence, $\sigma_d \models \exists_{|d}(F^1)_s$ (using B.1)*
- *we may extend σ_d into $\sigma_{s\cup d} \models (F^1)_s$ (b)*
- *by lattice structure, $\sigma_{s\cup d} \models \underbrace{(F^1)_s \wedge \exists_{|s}(F^2)_d}_{=(F^2)_s}$.*
- *by projection $\sigma_d = (\sigma_{s\cup d})_{|d} \models \exists_{|d}(F^2)_s$*

Therefore, the formula $\mathbf{F}^2$ is both locally forward and backward consistent, that is, using Lemma 5.4 [Characterization Of Local Consistency As Forward And Backward Consistency], the formula $\mathbf{F}^2$ is locally consistent. □

Theorem 5.3 (Global Consistency in a Single Propagation Step for Tree Decomposition)**.** *Let $\mathbf{F} = (D, F)$ a constraint system decomposition with D a connected tree-decomposition. Let $\mathcal{O}$ the natural orientation of D toward some component b.*

Then,

$$\mathrm{GP}(\mathbf{F}, O) = \mathrm{LP}(\llbracket \mathbf{F} \rrbracket) \ .$$

That is, globally (and locally) consistent.

Proof. *Using,*

- *Lemma 5.5 [Basic Properties Of Forward Propagation], and,*
- *Theorem 5.2 [Backward Propagation Of Forward Consistent Is Locally Consistent], and,*
- *Theorem 5.1 [Using Tree Decomposition Local Consistency Iff Global Consistency].*

□

5.3 Time and Space Complexity

Memory Cost On the over hand, one may estimate the memory consumption of a single component as a non-decreasing function $\beta(b)$.

Then, one may estimate the overall memory consumption as:

$$M(D) = \sum_{b \in B} \beta(b)$$

In exponential and super-exponential cases we expect α and β to coincides up to some non-significant factor, hence, we just have to consider the same function α for both costs.

However, in small-exponential ($x < 2$) and sub-exponential cases, we expect α and β to be disctinct enough for it to be taken into account.

Anyway, one should define an appropriate compromise between time and memory by balancing both time and memory costs.

Computation Cost One may estimate the computation time of adjacent propagation between two components s (as source) and d (as destination) as a non-decreasing function $\alpha(|s|)$.

Then, one may estimate the overall cost of the computation as:

$$T(D) = \sum_{(s,d)\in O} \alpha(|s|) + \sum_{(s,d)\in O} \alpha(|d|) = \sum_{\substack{(x,\,y)\,\in\, I \\ x\,<\,y}} (\alpha(|x|) + \alpha(|y|)) = \sum_{b\in B} \alpha(|b|) \cdot b^{\circ} \ ,$$

One may essentially distinguish between three cases

- α is polynomial (or sub-polynomial, or pseudo-polynomial)
  - e.g. system of (real or rationnal) linear equations
  - $\alpha(n) := n^k$
  - the highest degree term is lower than 2, one should not use CSTD as it likely will increase computation time
  - otherwise, in expected structured case, one may expect some saving in computation time and memory consumption.
    - warning : computing a good tree decomposition may actually be longer than the actual computation
  - α is an exponential in x
    - e.g., SAT, MaxSAT, integer linear programming, CSP
    - $\alpha(n) := x^n$
    - if provided a good tree-decomposition, we are expecting significant savings in term of computation time and memory consumption.
    - However, if $x < 2$, the computation time of the good tree decomposition may outweight its benefits.
    - Even if $x \geq 2$ using appropriate heuristics to find "good enough" tree decomposition may be appropriate
  - α is super exponential
    - e.g. system of linear inequation, system of polynomial equations
    - if provided a good tree-decomposition, we are expecting significant savings in term of computation time and memory consumption.
    - We conjecture that, one can fall back to an exponential-ish case by lexicographically minimizing $\alpha(n) := X^n$ with X an abstract polynomial variable.
    - Taking time to compute the "best" tree decomposition (or at least a very good one) is likely a good idea, as the complexity of finding the optimal tree decomposition is at most $\mathcal{O}\left(2^{|X|^2}\right)$

Note : if the domain $\mathcal{A}$ is finite, then, α is at most exponential in $|D|$.

5.4 Conclusion

One may notice that we have exhibited a process which terminates independently of finiteness of the domain of the variables. Furthermore, we computed an upper bound on the computation cost (time and memory) associated with a chosen tree-decomposition, hence, envisonning a way of selecting a "good" one.

Chapter 6

Introducing Tree Based Reduction

Chapters 4 and 5 use the Boolean projection operator $\exists$ to deal with constraint system satisfaction using a two-phase propagation strategy in the case of tree-decomposition.

This chapter widen the scope of this propagation strategy to deal with optimizing constraint systems (e.g., $(\mathbb{N}, \max, +)$ which has been used in a effective implementation)

The main change is the necessity to introduce a co-projection operator Π and replace the "forward propagation process" by a "forward reduction process"

6.1 Extending Parameter Scope To Optimizing Constraints

Firstly, we extend the co-domain of constraints from Boolean to an arbitrary set $\mathcal{B}$. We assume that,

- $\mathcal{B}$ contains (at least) two elements denoted $\bot$ (false) and $\top$ (true)
- $\mathcal{B}$ has an operator $\barwedge$, which is associative and commutative.
- We assume the following basic properties
  - For any elements $c, d \in \mathcal{B}$,
  - $c \barwedge d = \bot \Leftrightarrow c = \bot \vee d = \bot$, in particular $\bot \barwedge c = \bot$.
  - $\top \barwedge c = c$.
- For any element $b \in \mathcal{B}$, we denote $\mathbb{B}(c) := "c \neq \bot" \in \mathbb{B} \subseteq \mathcal{B}$ its Boolean projection.
  - that is, $\mathbb{B}(\bot) = \bot$, and,
  - $\forall c \neq \bot, \mathbb{B}(c) = \top$.
- For interpretation sake, we may assume that some elements are better than others (lets say a partial pre-order), in particular all non-Boolean elements are better than $\top$ and $\top$ is better than $\bot$.
- This notion of "better" is only explicited for interpretation as it does not appear explicitly in the upcoming manipulations.

Then, we may extend the definition of our basic objects to take into account the extension of the co-domain

- a domain $\mathcal{A}$ in which lives variables
- a finite set X of n variables,
  - Variables are denoted $X = \{x_1, \ldots, x_n\}$.
  - We call valuation a mapping from X to $\mathcal{A}$, generally denoted $\sigma \in \mathcal{A}^X$

- a co-domain $\mathcal{B}$ with the above mentioned assumptions.
- a finite set F of m optimizing constraint, that is, functions $f_i \in \mathcal{A}^X \to \mathcal{B}$.
  - The optimizing constraint f is unsatisfied iff the function returns $\bot$
  - We define the Boolean projection of f as $\mathbb{B}(f) := \mathbb{B} \circ f = x \mapsto \mathbb{B}(f(x))$
  - We denote $\sigma \models f :=$ "$f(\sigma) \neq \bot$"
- a conjunction operator $\bigcirc\!\!\!\!\wedge$ on constraints
  - we lift the operator $\bigcirc\!\!\!\!\wedge$ defined on the co-domain $\mathcal{B}$ to constraints $\mathcal{A}^X \to \mathcal{B}$
- we define the semantics of F as $[\![F]\!] = \bigcirc\!\!\!\!\bigwedge_i f_i$.

As we are studying optimization, we need to extend the existential projection to take into account the optimizing component.

First, we define projection and co-projection operators

Axiom 6.1 ((Optimizing) Projection Operator)**.** *We assume a projection operator $\exists : F \to \mathcal{B}$, such that,*

- $\exists f = \bot \iff f = \bot$
- $\forall b \in \mathcal{B}, \exists f = b \implies \exists \sigma \in \mathcal{A}^X, f(\sigma) = b$

Remark 6.1 (Interpretation As An Optimal Value)**.** *$\exists f$ can be interpreted as returning the optimal value of f for a given notion of optimality (min or max for example).*

Definition 6.1 (Partial Projection Operator)**.** *We extend $\exists$ to deal with partial projection as follows:*

- *let $Y \subseteq X$ a subset of variables,*
- *we split variables in X into Y and $Y^{\complement}$,*
- *for any function f, we define $\exists_Y f := y' \in Y^{\complement} \mapsto \exists f(\cdot, y')$*
- *to avoid cumbersome support casting, we assume that missing variables in Y are reintroduced in $\exists_Y f$ as useless ones.*

Lemma 6.1 (Projection and Evaluation)**.** *let $Y \uplus Z = X$ be a 2-partition of the set of variables X, then,*

$$\forall z \in \mathcal{A}^Z, (\exists_Y f)[Z \leftarrow z] = \exists_Y (f[Z \leftarrow z]) \ .$$

Proof. *Implied by Definition 6.1 of the [Partial Projection Operator].* □

Lemma 6.2 (Projection and Partial Evaluation)**.** *let Y and Z be two non intersecting sets of variables (that is, $Y \bigcap Z = \emptyset$), then,*

$$\forall z \in \mathcal{A}^Z, (\exists_Y f)[Z \leftarrow z] = \exists_Y (f[Z \leftarrow z]) \ .$$

Proof. *We denote $R := X \backslash (Y \uplus Z)$, such that $Y \uplus Z \uplus R = X$ is a 3-partition of X.*
We rewrite the goal statement using classical functional extensionality, that is:

$$\forall (y, z, r) \in \mathcal{A}^{Y+Z+R},\ ((\exists_Y f)[Z \leftarrow z])\,[Y \leftarrow y, R \leftarrow r] = (\exists_Y (f[Z \leftarrow z]))\,[Y \leftarrow y, R \leftarrow r]\ .$$

That is, for any valuation $(y, z, r) \in \mathcal{A}^{Y+Z+R}$,

$$(\exists_Y f)[Y \leftarrow y, Z \leftarrow z, R \leftarrow r] = (\exists_Y (f[Z \leftarrow z]))\,[Y \leftarrow y, R \leftarrow r]\ .$$

By unfolding Definition 6.1, we have that $y \in \mathcal{A}^Y$ is useless, hence can be simplified.

$$(\exists_Y f)[Z \leftarrow z, R \leftarrow r] = (\exists_Y (f[Z \leftarrow z]))\,[R \leftarrow r]\ .$$

We use Lemma 6.1 twice
- *on the left hand side with $f := f$, $Y := Y$ and $Z := Z \uplus R$,*
- *on the right hand side with $f := f[Z \leftarrow z]$, $Y := Y$ and $Z := R$.*

Hence,

$$(\exists_Y f)[Z \leftarrow z, R \leftarrow r] = \exists_Y\,(f[Z \leftarrow z,\ R \leftarrow r]) = (\exists_Y (f[Z \leftarrow z]))\,[R \leftarrow r]\ .$$

□

Lemma 6.3 (Support Of Partial Projection)**.** *Let f be an optimizing constraint, and $Y \subseteq X$ a set of variables, then,*

$$\mathtt{supp}\,(\exists_Y f) \subseteq \mathtt{supp}\,(f) - Y\ .$$

Proof. *Let $x \in X$ be a variable, then, lets show that,*

$$x \in \mathtt{supp}\,(\exists_Y f) \subseteq x \in (\mathtt{supp}\,(f) - Y)\ .$$

- *if $x \in Y$, by Definition 6.1 Of Partial Projection Operator, $x \notin \mathtt{supp}\,(\exists_Y f)$,* □
- *otherwise, $x \notin Y$,*
  - *hence, $x \in U(f) = \mathtt{supp}\,(f)^{\mathtt{c}}$ (we denote $U(f)$ the set of useless variables in f)*
  - *let $\sigma_x \in D$, and, $\sigma'_x \in D$ two (possibly distinct) valuation of x,*
  - *lets show that, $x \in U(\exists_Y f)$, that is, $(\exists_Y f)\,(\sigma_x) = (\exists_Y f)\,(\sigma'_x)$*
  - *using Lemma 6.2 Projection And Partial Evaluation, we rewrite as, $(\exists_Y f(\sigma_x)) = (\exists_Y f(\sigma'_x))$,*
  - *using $x \in U(f)$, we have $f(\sigma_x) = f(\sigma'_x)$, hence, the equality holds.*

□

Axiom 6.2 (Composing Partial Projections)**.** *Let Y and Z two subsets of variables in X, then, we assume that,*

$$\exists_Y \exists_Z f = \exists_{Y \cup Z} f\ .$$

Axiom 6.3 (Composing Projection and Partial Evaluation)**.** *let f an optimizing constraint (with $f \neq \bot$), and $Y \uplus Z = X$ a 2-partition of X, and $(y, z) \in \mathcal{A}^Y \times \mathcal{A}^Z$ such that $f(y, z) = \exists f$.*

Then,

$$f(y, z) = (\exists_Y f)(z) = \exists f \quad .$$

Remark 6.2 (Independence Of Axiom 6.2 and Axiom 6.3)**.** *Both axioms are independent, Axiom 6.2 speaks about co-domain while Axiom 6.3 speaks about domain.*

Definition 6.2 (Co-Projection Operator)**.** *We define the* co-projection *operator* Π *for any $f \in \mathcal{A}^X \to \mathcal{B}$, we define $\Pi f \in \mathcal{A}^X \to \mathcal{B}$ as :*

$$\Pi f := x \in \mathcal{A}^X \mapsto \mathbb{B}(f(x)) \wedge \underbrace{(f(x) = \exists f)}_{\in \mathbb{B}} \quad .$$

We extend Π to deal with partial co-projection

- *for any function $f(y, z)$, we define $\Pi_Y f := z \mapsto \Pi f(\cdot, z)$*

Lemma 6.4 (Evaluation Of Co-Projection)**.** *Let $Y \uplus Z = X$ a 2-partition of X and f an optimizing constraint, then,*

$$\forall (y, z) \in \mathcal{A}^{Y+Z}, (\Pi_Y f)(y, z) = (\mathbb{B}(f(y, z)) \wedge (f(x, y) = (\exists_Y f)(z))) \quad .$$

Proof. *Unfold Definition 6.2 [Co-Projection Operator], then, Lemma 6.2 [Projection and Partial Evaluation].* □

Lemma 6.5 (Boolean Projection and Co-Projection)**.** *Let f an optimizing constraint, then,*

$$\Pi f \subseteq \mathbb{B}(f)$$

Proof. *Using functional extensionality, and, unfolding the Definition 6.2 [Co-Projection Operator], and, basic Boolean reasoning.* □

Lemma 6.6 (Decomposition Of Co-Projection)**.** *Let f an optimizing constraint, and, $Y \uplus Z$ a 2-partition of X, then,*

$$\Pi f = \Pi_Y(\exists_Z f) \wedge \Pi_Z f \quad .$$

Proof. *First, using functional extensionality, we turn the first statement into,*

$$\forall (y,z) \in \mathcal{A}^X,\ (\Pi f)(y,z) = (\Pi_Y(\exists_Z f) \wedge \Pi_Z f)(y,z)$$

Let $(y,z) \in \mathcal{A}^X$, that is, if $f(y,z) = \bot$, the statement holds.
Otherwise, $f(y,z) \neq \bot$, and,

$$\text{"}f(y,z) = \exists f\text{"} \iff \text{"}\Pi_Y(\exists_Z f)(y) = \top\text{"} \wedge \text{"}f(y,z) = (\exists_Z f)(y)\text{"}$$

That is,

$$\text{"}f(y,z) = \exists f\text{"} \iff \text{"}(\exists_Z f)(y) \neq \bot\text{"} \wedge \text{"}(\exists_Z f)(y) = \exists_Y(\exists_Z f)\text{"} \wedge \text{"}f(y,z) = (\exists_Z f)(y)\text{"}$$

However, "$(\exists_Z f)(y) \neq \bot$" holds, and $\exists_Y(\exists_Z f) = \exists f$, hence,

$$\text{"}f(y,z) = \exists f\text{"} \iff \text{"}(\exists_Z f)(y) = \exists f\text{"} \wedge \text{"}f(y,z) = (\exists_Z f)(y)\text{"}$$

We split the equivalence into implications and discard both

- "$f(y,z) = \exists f$" $\implies$ "$(\exists_Z f)(y) = \exists f$" $\wedge$ "$f(y,z) = (\exists_Z f)(y)$"
  - *Using Axiom 6.3 [Composing Projection And Partial Evaluation]*
- "$(\exists_Z f)(y) = \exists f$" $\wedge$ "$f(y,z) = (\exists_Z f)(y)$ $\implies$ "$f(y,z) = \exists f$""
  - *Using transitivity of equality, $f(y,z) = (\exists_Z f)(y) = \exists f$.*

□

Theorem 6.1 (Decomposition Of Partial Co-Projection)**.** *Let f an optimizing constraint, and, Y and Z two non-intersecting sets of variables, then,*

$$\Pi_{Y \uplus Z} f = \Pi_Y(\exists_Z f) \wedge \Pi_Z f \ .$$

Proof. *Using classical functional extensionality and Lemma 6.6 [Decomposition Of Co-Projection].* □

Axiom 6.4 (Disjoint Support Separation)**.** *Let f and g two optimizing constraints such that* $\mathtt{supp}(f) \cap \mathtt{supp}(g) = \emptyset$*, then, we assume that :*

- $\exists (f \mathbin{\textcircled{\wedge}} g) = (\exists f) \mathbin{\textcircled{\wedge}} (\exists g)$
- $\Pi (f \mathbin{\textcircled{\wedge}} g) = (\Pi f) \wedge (\Pi g)$

Example 6.1 (Example Of Parameter Instances)**.** *We give examples of how each parameter can be instantiated*

- $\mathcal{A} \in \{\mathbb{B}, \mathbb{N}, \mathbb{Q}, \mathbb{R}\}$
- $\mathcal{B} \in \{\mathbb{N}, \mathbb{Q}, \mathbb{R}\}$
- *f can be restriction of the full function space, e.g., systems of linear, affine, or, polynomial equations.*

- $\bigcirc\!\!\!\!\wedge \in \{+, \times\}$
- $\exists \in \{\min, \max\}$ *(then,* $\Pi \in \{\text{argmin}, \text{argmax}\}$*, respectively)*

Definition 6.3 (Semantics Of Optimizing Constraints)**.** *Let f an optimizing constraint, we define $[\![f]\!]^{\mathcal{O}}$ the semantic of f as $= [\![f]\!]^{\mathcal{O}} := (\exists f, \Pi f)$*

Definition 6.4 (Equivalence)**.** *Lets f and g two constraints, we say that $f \equiv g$ iff $[\![f]\!]^{\mathcal{O}} = [\![g]\!]^{\mathcal{O}}$.*
That is f and g have the same optimal value and share the same set of optimal valuations.

6.2 Functional Partial Extensionality

Lemma 6.7 (Equality Of Partial (Co-)Projection Implies Equality Of Total (Co-Projection))**.** *Let f and g two constraints, and $Y \subseteq X$ a subset of variables, then,*

- *On the projection side :*

$$(\exists_Y f = \exists_Y g) \implies \exists f = \exists g \ ,$$

- *and, on the co-projection side :*

$$[(\exists_Y f = \exists_Y g) \wedge (\Pi_Y f = \Pi_Y g)] \implies \Pi f = \Pi g \ .$$

Proof. *Lets prove both equality separately:*

- *lets prove $\exists f = \exists g$,*
  - *iff $\exists_{Y^c} \exists_Y f = \exists_{Y^c} \underbrace{\exists_Y g}_{=\exists_Y f}$ (using Axiom 6.2 [Composing Partial Projections]).*
  - *hence, the equality holds.*
- *lets prove $\Pi f = \Pi g$,*
  - *iff $\Pi_{Y^c}(\exists_Y f) \wedge \Pi_Y f = \Pi_{Y^c}(\underbrace{\exists_Y g}_{=\exists_Y f}) \wedge \underbrace{\Pi_Y g}_{=\Pi_Y f}$ (using Lemma 6.1 [Decomposition Of Co-Projection])*
  - *hence, the equality holds.*

□

Lemma 6.8 (Rephrasing Partial Equivalence as Partial (Co-)Projection)**.** *Let f and g two constraints, and $Y \subseteq X$ a subset of variables, then, we assume that,*

$$\left(\forall y \in \mathcal{A}^Y,\ f[Y \leftarrow y] \equiv g[Y \leftarrow y]\right) \iff (\exists_{Y^c} f = \exists_{Y^c} g) \wedge (\Pi_{Y^c} f = \Pi_{Y^c} g) \ .$$

Proof. *By Definition of* $\exists_{Y^c}$, *and,* Π_{Y^c}, $\equiv$. □

Theorem 6.2 (Functional Partial Extensionality)**.** *Let* f *and* g *two constraints, and* $Y \subseteq X$ *a subset of variables, then, we assume that,*

$$\left(\forall y \in \mathcal{A}^Y,\, f[Y \leftarrow y] \equiv g[Y \leftarrow y]\right) \Rightarrow f \equiv g \ .$$

Proof. *First, we use Lemma 6.8 to rephrase hypothesis.*
Then, we unfold the Definition 6.4 of Equivalence ($\equiv$).
Finally, we use Lemma 6.7 (with $Y := Y^C$*) to complete the proof.*
□

Remark 6.3 (Functional Partial Extensionality should not be an Equivalence)**.** *One should not turn the above axiom into an equivalence, otherwise, one may deduce that* $f \equiv g \implies f = g$, *which we dot not want.*

6.3 Semantic Separation and Boolean Projection

Lemma 6.9 (Projection Of Constant)**.** *Let* $b \in \mathcal{B}$, *then, (1)* $\exists b = b$ *and (2)* $\Pi b = \mathbb{B}(b)$.

Proof. *Using Axiom 6.1 [Projection Operator], along with case distinction on "*$f = \bot$*".* □

Definition 6.5 (Boolean Constraint)**.** *Let* f *an optimizing constraint, we say that* f *is a Boolean constraint iff* $\forall \sigma \in \mathcal{A}^X, f(\sigma) \in \{\bot, \top\}$

Lemma 6.10 (Projecting Boolean Constraints)**.** *Let* f *a Boolean constraint, then,*

- $\exists f \in \mathbb{B}$
- $f \neq \bot \Rightarrow \exists f = \top$
- $\Pi f = f$

Proof. *Using Axiom 6.1 [Projection Operator], along with case distinction on "*$f = \bot$*".* □

Lemma 6.11 (Projection of Boolean Constraints with Constants)**.** *Let f a Boolean constraint (with $f \neq \bot$), and, $b \in \mathcal{B}\backslash\{\bot\}$, then,*
- $\exists(f \owedge b) = b$, *and,*
- $\Pi(f \owedge b) = f$

Proof. *Using Axiom 6.4 on f and c, as* $\texttt{supp}(f) \cap \underbrace{\texttt{supp}(c)}_{=\emptyset} = \emptyset$.

Along above Lemmas 6.9, and 6.10.
Hence,
- $\exists(f \owedge c) = (\exists f) \owedge (\exists c) = \top \owedge c = c$
- $\Pi(f \owedge c) = (\Pi f) \owedge (\Pi c) = f \owedge \top = f$

□

Lemma 6.12 (Semantic Separation)**.** *For any constraint f, we have $f \equiv \Pi f \owedge \exists f$*

Proof. *We may assume $f \neq \bot$, (otherwise, $f = \bot$, then $\exists f = \bot$, then $\Pi f \owedge \exists f = \bot$. □)*
That is,

$$(\exists f = \exists\,(\Pi f \owedge \exists f)) \wedge (\Pi f = \Pi\,(\Pi f \owedge \exists f)) \quad .$$

Then, Πf is a Boolean constraint (and $\Pi f \neq \bot$) and $\exists f \in \mathcal{B}$ is a constant (and $c \neq \bot$).
Hence, using Lemma 6.11 [Projection Of Boolean Constraint With Constants], we deduce that,
- $\exists\,(\Pi f \owedge \exists f) = \exists(\exists f) = \exists f$
- $\Pi\,(\Pi f \owedge \exists f) = \Pi(\Pi f) = \Pi f$

Which allows to prove that the above property holds. □

Theorem 6.3 (Parametric Semantic Separation)**.** *Let f be a constraint and Y be a subset of variables, then, $f \equiv \Pi_Y f \owedge \exists_Y f$.*

Proof. *We split X into Y and Y^C.*
Using Theorem 6.2 [Functional Partial Extensionality] with Y^C, it is enough to prove that $\forall z \in \mathcal{A}^{Y^C}$, $f(\cdot, z) \equiv \Pi_Y f(\cdot, z) \owedge \exists_Y f(\cdot, z)$.
Which simplifies into, $\forall z \in \mathcal{A}^{Y^C}$, $f(\cdot, z) \equiv \Pi f(\cdot, z) \owedge \exists f(\cdot, z)$.
Which holds using Lemma 6.12[Semantic Separation]. □

6.4 Adjacent Reduction And Semantic Preservation

Lemma 6.13 (Characterization Of Boolean Saturation)**.** *Let f be an optimizing constraint and g a Boolean constraints, then,*

$$f = f \mathbin{⩓} g \iff \mathbb{B}(f) \subseteq g \ .$$

Proof. *First, $\mathbb{B}(f) \subseteq g \implies f = f \mathbin{⩓} g$*

- *Using functional extensionality, let $\sigma \in \mathcal{A}^X$, lets prove $f(\sigma) = f(\sigma) \mathbin{⩓} g(\sigma)$.*
- *g is a Boolean constraint, hence, $g(\sigma) \in \{\bot, \top\}$.*
- *if $g(\sigma) = \top$, then, the equality holds using basic assumptions on the co-domain.*
- *otherwise, one may assume $g(\sigma) = \bot$, hence, using primiary hypothesis, $f(\sigma) = \bot$.*
- *Hence, both terms are equal to $\bot$, hence the equality holds.*

Second, $f = f \mathbin{⩓} g \implies \mathbb{B}(f) \subseteq g$

- *that is, using functional extensionality, let $\sigma \in \mathcal{A}^X$,*

$$f(\sigma) = f(\sigma) \wedge g(\sigma) \implies \mathbb{B}(f)(\sigma) = \top \implies g(\sigma) = \top \ .$$

- *by contradiction, we assume that $g(\sigma) = \bot$, then, $f(\sigma) = \bot$ in primary hypothesis, using transitivity it leads to the contradiction $\top = \bot$.*

□

Lemma 6.14 (Disjoint Support Separation Of Equivalence)**.** *Let $A \uplus B = X$ a 2-partition of X, and f', f', g, g' four optimizing constraints, such that*

- $\mathtt{supp}\,(f), \mathtt{supp}\,(f') \subseteq A$*, and,*
- $\mathtt{supp}\,(g), \mathtt{supp}\,(g') \subseteq B$

Then,

$$[f \equiv f' \wedge g \equiv g'] \implies f \mathbin{⩓} g \equiv f' \mathbin{⩓} g' \ .$$

Proof. *Using Axiom 6.4 we are left with proving*

- $\exists f \mathbin{⩓} \exists g = \exists f' \mathbin{⩓} \exists g'$*, and,*
- $\Pi f \wedge \Pi g = \Pi f' \wedge \Pi g'$.

Which hold by unfolding Definition 6.4 of Equivalence,

- $f \equiv f' \iff \exists f = \exists f' \wedge \Pi f = \Pi f'$
- $g \equiv g' \iff \exists g = \exists g' \wedge \Pi g = \Pi g'$

□

Theorem 6.4 (Core Semantic Preservation)**.** *Let $A \uplus B \uplus C \uplus D = X$ a 4-partition of X. Let f, g, h three optimizing constraints such that*

- $\mathtt{supp}\,(f) \subseteq A \uplus B$*, and,*
- $\mathtt{supp}\,(g) \subseteq B \uplus C$*, and,*
- $\mathtt{supp}\,(h) \subseteq B \uplus C \uplus D$*, and,*
- *h is a Boolean constraint, and,*
- $\exists_D h \subseteq \mathbb{B}(g)$*, and,*
- *furthermore, we define $g' := \exists_C g \mathbin{⩓} \Pi_C g$.*

$$f \owedge g \owedge h \equiv f \owedge g' \owedge h \ .$$

Proof. *For ease of use, we denote $f(a,b)$, $g(b,c)$ and $h(b,c,d)$.*

Using Theorem 6.2 [Functional Partial Extensionality] we are left with proving that, for any $(a,b,s \in \mathcal{A}^{A+B+D}$,

$$f(a,b) \owedge g(b,\cdot) \owedge h(b,\cdot,d) \equiv f(a,b) \owedge g'(b,\cdot) \owedge h(b,\cdot,d)$$

One may notice that, in the above equation, the only free variables are the one in C.

If either $f(a,b) = \bot$, $g(b,\cdot) = \bot$ or $h(b,\cdot,d) = \bot$, then the equation holds using basic rewriting on $\owedge$.

Hence, we may assume $f(a,b) \neq \bot$, $g(b,\cdot) \neq \bot$, and, $h(b,\cdot,d) \neq \bot$.

Furthermore,

- *lets prove $g(b,\cdot) \owedge h(b,\cdot,d) = g(b,\cdot)$*
  - *Using Lemma 6.13 [Characterization Of Boolean Saturation], it simplifies into*

$$\mathbb{B}(g(b,\cdot)) \subseteq h(b,\cdot,d)$$

  - *Which holds by partial evaluation of $\mathbb{B}(g) \subseteq \exists_D h$.*
- *lets prove $g'(b,\cdot) \owedge h(b,\cdot,d) = g'(b,\cdot)$*
  - *Using Lemma 6.13 [Characterization Of Boolean Saturation], it simplifies into*

$$\mathbb{B}(g'(b,\cdot)) \subseteq h(b,\cdot,d)$$

  - *which holds as*

$$\mathbb{B}(\Pi_C g(b,\cdot)) \subseteq \mathbb{B}(g(b,\cdot)) \subseteq h(b,\cdot,d)$$

Hence, the goal property simplifies into proving:

$$f(a,b) \owedge g(b,\cdot) \equiv f(a,b) \owedge g'(b,\cdot)$$

This property holds using Lemma 6.14 [Disjoint Support Separation Of Equivalence] and Theorem 6.3 [Parametric Semantic Separation]. □

6.5 Optimizing Constraint System Tree Decomposition

Theorem 6.4 paves the way toward a tree-decomposition-based compositional resolution of optimizing constraint systems.

Definition 6.6 (Optimizing Constraint System Tree Decomposition (OCSTD))**.** *We define an optimizing constraint system tree decomposition as an constraint system decomposition, which decomposition is a tree decomposition, and which constraints are optimizing constraints.*

Definition 6.7 (Semantics Of OCSTD)**.** *Let $\mathbf{F} = (F, D)$ an OCSTD, we define*

- *$[\![\mathbf{F}]\!] := [\![F]\!]$, and,*

- $[\![\mathbf{F}]\!]^{\mathcal{O}} := [\![[\![\mathbf{F}]\!]]\!]^{\mathcal{O}} = (\exists[\![\mathbf{F}]\!], \Pi([\![\mathbf{F}]\!]))$

Definition 6.8 (Equivalence Of OCSTD)**.** *Furthermore, we say that two optimizing constraint systems* $\mathbf{F}$ *and* $\mathbf{G}$ *are equivalent iff* $[\![\mathbf{F}]\!] \equiv [\![\mathbf{G}]\!]$*, that is,* $[\![\mathbf{F}]\!]^{\mathcal{O}} = [\![\mathbf{F}]\!]^{\mathcal{O}}$*.*

Definition 6.9 (Local Projection (Optimization Case))**.** *We denote* $\mathrm{LP}^{\mathbb{B}}$ *the local projection as defined in Chapter 4. We extend the definition of* LP *as follows:*

Let D *a tree decomposition with* $\emptyset \in D$*, and* f *a constraint, we denote* $\mathrm{LP}(D, f) = \mathbf{F}$ *where*

- $\mathbf{F} = \mathrm{LP}^{\mathbb{B}}(D, \Pi f)$
- *except,* $f_\emptyset = \exists f$

Definition 6.10 (Acceptable Decomposition)**.** *Let* D *be a graph decomposition, and* f *be an optimizing constraint.*

We say that D *is an acceptable graph decomposition iff* $[\![\mathrm{LP}(D, f)]\!]^{\mathcal{O}} = [\![f]\!]^{\mathcal{O}}$*.*

Definition 6.11 (Globally Consistent (Optimization Case))**.** *Let* $\mathbf{F}$ *a constraint system decomposition, we say that* $\mathbf{F}$ *is globally consistent iff* $\mathrm{LP}([\![\mathbf{F}]\!]) = \mathbf{F}$

6.6 From Forward Propagation to Forward Reduction

Definition 6.12 (Forward Reduced)**.** *Let* $\mathbf{F}$ *an OCSTD oriented with respect to* $\mathcal{O}$*, let* $(s, d) \in O$ *two adjacent components, then* (s, d) *is forward reduced iff*

- f_s *is a Boolean constraint, and,*
- $\mathbb{B}(f_d) \subseteq \exists_{|d} f_s$

Furthermore, we say that $\mathbf{F}$ *is forward reduced iff, for every interconnection* $(s, d) \in \mathcal{O}$*,* (s, d) *is forward reduced in* $\mathbf{F}$*.*

Definition 6.13 (Forward Reduction)**.** *Let* $\mathbf{F}$ *an OCSTD oriented with respect to* $\mathcal{O}$*, let* $(s, d) \in \mathcal{O}$ *an interconnection, such that* $\mathbf{F}_s$ *(the sub-OCSTD rooted in* s*) is forward reduced, then, we define the forward reduction operator* $\mathbf{F}[s \rightsquigarrow d] := \mathbf{F}'$*, such that*

- $f'_s := \Pi_{(s-d)} f_s$*, and,*
- $f'_d := f_d \mathbin{⩓} \exists_{(s-d)} f_s$*, and,*
- *otherwise,* $\forall b \in B \backslash \{s, d\}$*,* $f'_b := f_b$

Lemma 6.15 (Extending Root Partial Solution in Forward Consistent OCSTD)**.** *Let* $\mathbf{F}$ *a forward consistent rooted OCSTD and* σ^0 *a partial solution of its root component, then, there exists* σ *an extension of* σ^0 *such that* σ *satisfies* $\mathbf{F}$*.*

In particular, denoting b_0 the root component of the decomposition, we have,

$$f_{b_0} = \exists_{|b_0}[\![\mathbf{F}]\!] \ .$$

Proof. *The main statement is proved by induction on the rooted tree decomposition structure. We denote b the root component of the tree decomposition.*
Let σ_b an extension of σ_0 such that σ_b satisfies f_b.
We denote $\sigma^0_{b_i}, \dots, \sigma^0_{b_k}$ the k projections of σ_b on the interface of b with its sons b_i.
Then, by induction, σ^0_i can be extended into a solution σ_i of its sub constraint system decomposition.
Using Lemma 3.1 [Separation Property Of Tree Decomposition], we have : $\forall i, j$, $\mathtt{supp}(\sigma_i) \cap \mathtt{supp}(\sigma_j) \subseteq \mathtt{supp}(\sigma_b)$.
Hence, σ_b can be consistently extended using all $(\sigma_i)_i$ into σ a solution of the constraint system decomposition.

The corollary statement is proved by combining the main statement with Lemma 4.14 [Locally Globally Consistent]. □

Theorem 6.5 (Forward Reduction Semantic Preservation)**.** *Let $\mathbf{F}$ an OCSTD oriented with respect to $\mathcal{O}$, let $(s,d) \in \mathcal{O}$ an interconnection, such that $\mathbf{F}_s$ (the sub-OCSTD rooted in s) is forward reduced, then*

1. *$\mathbf{F}[s \rightsquigarrow d] \equiv \mathbf{F}$, and,*
2. *(s,d) is forward reduced, and,*
3. *$\mathbf{F}[s \rightsquigarrow d]_s$ (the sub-OCSTD of $\mathbf{F}[s \rightsquigarrow d]$ rooted in s) is forward reduced*

Proof. *First, lets prove $\mathbf{F}[s \rightsquigarrow d] \equiv \mathbf{F}$*

- *Then,*
  1. *we have $f_s = \exists_{|s}[\![\mathbf{F}]\!]$ (using Lemma 6.15)*
  2. *in $\mathbf{F}_s$ all component but b_0 the root one are Boolean constraints.*
- *Thus, we may use Theorem 6.4 [Core Semantic Preservation] with*
  - *f defined as $\bigwedge_{f \in \mathbf{F} \setminus \mathbf{F}_s} f$, and,*
  - *g defined as f_s, and,*
  - *h defined as $\bigwedge_{f \in \mathbf{F}_s \setminus \{f_s\}} f$.*
- *Hence, $\mathbf{F}[s \rightsquigarrow d] \equiv \mathbf{F}$.*

Second, one may prove that (s,d) using Theorem 6.3 [Parametric Semantic Separation].

Finally, we prove that $\mathbf{F}[s \rightsquigarrow d]_s$ is forward reduced using transitivity of set inclusion and Lemma 6.5 [Boolean Projection And Co-Projection] on root-adjacent arcs, all other interconnection being left untouched by the transformation. □

Definition 6.14 (Forward Reduction Process)**.** *Let $\mathbf{F} = (D, F)$ an OCSTD and O an orientation of D. We inductively define the forward reduction process (*$\mathrm{FRP}(\mathbf{F}, O)$*) as follows, select a non visited component such that the source of all incoming arcs (in O) have been visited (the tree configuration of O ensures the existence of such a component).*
Perform forward reduction on all incoming arcs.

The process stops when all components have been visited.

Lemma 6.16 (Forward Reduction Process Forward Reduces)**.** *Let* $\mathbf{F}$ *an OCSTD and* $\mathcal{O}$ *an orientation of* D*. Then* $\mathrm{FRP}(\mathbf{F})$ *is forward reduced according to* $\mathcal{O}$*.*

Proof. *By induction on the tree-structure of the process and according to Theorem 6.5* □

Lemma 6.17 (Forward Reduced implies Forward Consistent)**.** *Let* $\mathbf{F}$ *a forward reduced OCSTD, then* $\mathbf{F}' := \mathbf{F} \backslash \{f_\emptyset\}$ *is*

- *exclusively composed of Boolean constraints,*
- *forward consistent (see Definition 5.6)*

Definition 6.15 (Backward Propagation Process)**.** *Let* $\mathbf{F}$ *a forward reduced OCSTD and* $\mathcal{O}$ *an orientation of* D*.*

We denote $\mathbf{F}' := \mathbf{F}$ *with* $f_\emptyset := \mathbb{B}(f_\emptyset)$ *(which according to Lemma 6.17 [Forward Reduced implies Forward Consistent]) is actually a (forward consistent) CSTD.*

We denote $\mathbf{F}''$ *the CSTD obtained by applying the backward propagation process (see Definition 5.7) according to* $\mathcal{O}$*.*

We define the backward propagation process (BPP) of the OCSTD $\mathbf{F}$ *as* $\mathbf{F}$ *where all components of* $\mathbf{F}'$ *but* $f_\emptyset$ *has been substituted by* $\mathbf{F}''$*.*

Lemma 6.18 (Backward Propagation Process Of Forward Reduced Is Local Projection)**.** *Let* $\mathbf{F}$ *a forward reduced OCSTD and* $\mathcal{O}$ *an orientation of* D*.*

Then, $\mathrm{BPP}(\mathbf{F}) = \mathrm{LP}_D([\![\mathbf{F}]\!])$*.*

Proof. *Using Lemma 6.17, we have that* $\mathbf{F}'$ *is forward consistent.*

Hence, according to Theorem 5.2, we have that, $\mathbf{F}''$ *is locally consistent according to Definition 4.11.*

One may conclude using Theorem 5.1 [Using Tree Decomposition Consistency Iff Global Consistent]. □

Definition 6.16 (Reduction Process)**.** *We define the reduction process as the composition of the forward reduction process (FRP) and the backward propagation process (BPP)*

$$\mathrm{RP}(\mathbf{F}) := \mathrm{BPP}(\mathrm{FRP}(\mathbf{F})) \ .$$

Theorem 6.6 (Reduction Process Is Local Projection)**.** *Let* $\mathbf{F}$ *an OCSTD, then,*

$$\mathrm{RP}(\mathbf{F}) = \mathrm{LP}(\llbracket \mathbf{F} \rrbracket) \ .$$

Proof. *The proposition holds by combining*

- *Lemma 6.16 [Forward Reduction Process Forward Reduces], and,*
- *Lemma 6.18 [Backward Propagation Process Of Forward Reduced Is Local Projection].*

□

Chapter 7

Cartesian Operators

As mentioned in Section 3.1 [Constraint System Algebra] (CSA) we introduced how to use the $\texttt{CSA} = (D, \mathbb{N}, +, \times, 0, 1)$ to compute the number of satisfying valuation.

In this section, we want to show that one can relate back this number to input valuation, by introducing a method with given a CSTD $\mathbf{F}$ allows to construct a function (in the form of a circuit) which establish an isomorphism between $\underbrace{\{\sigma \in D^X \mid [\![\mathbf{F}]\!](\sigma)\}}_{=\{D^X \mid [\![\mathbf{F}]\!]\}}$ and $[\![0, \#[\![\mathbf{F}]\!] - 1]\!]$ (with the convention than $[\![0, -1]\!] = \emptyset$).

7.1 Cartesian Algebra

To further formalize this notion, we introduce $\ell(n) := [\![0, n-1]\!] = \{0, \ldots, n-1\}$ i.e. the set containing the n first natural integers (in particular $\ell(0) = \emptyset$, and $\#\ell(n) = n$), along with $\mathcal{L} := \{\ell(n) \mid n \in \mathbb{N}\}$.

We attach a structure of commutative semi-ring (CSR, see Section 3.1 [Constraint System Algebra]) to it :

- $\ell(x) +_{\mathcal{L}} \ell(y) := \ell(x + y)$, and,
- $\ell(x) \times_{\mathcal{L}} \ell(y) := \ell(x \times y)$.

We explicit the isomorphism (denoted $\overrightarrow{?}$) and reverse-isomorphism (denoted $\overleftarrow{?}$) between the left-hand side and right-hand side of the above set-operators

- $\overrightarrow{+} \in \ell(x) +_{\mathcal{L}} \ell(y) \to \ell(x + y)$
  - $\overrightarrow{+}(\text{Left}(n_x)) = n_x \in \ell(x + y)$
  - $\overrightarrow{+}(\text{Right}(n_y)) = x + n_y \in \ell(x + y)$
- $\overleftarrow{+} \in \ell(x + y) \to \ell(x) + \ell(y)$
  - $\overleftarrow{+}(n') = \begin{cases} \text{Left}\, n' & \text{if } n' < x \\ \text{Right}(n' - x) & \text{otherwise} \end{cases} \in \ell(x) + \ell(y)$
- $\overrightarrow{\times} \in \ell(x) \times \ell(y) \to \ell(x \times y)$
  - $\overrightarrow{\times}(n_x, n_y) = n_x \times y + n_y$
- $\overleftarrow{\times} \in \ell(x \times y) \to \ell(x) \times \ell(y)$
  - $\overleftarrow{\times}(n') = (n' \div y, n \mod y)$

In the sequel, we shall explicit both the isomorphism (and reverse-isomorphism) between $\{D^X \mid [\![\mathbf{F}]\!]\}$ and $\ell(\#[\![\mathbf{F}]\!])$.

Remark 7.1. *Cartesian Algebra Isomorphism as Rebase Algorithm By building a circuit for both the isomorphism and reverse-isomorphism, one defines a basic yet cardinal-optimal rebase algorithm.*

By defining these isomorphisms we can use them as a basic yet cardinal-optimal rebase algorithm.

We can generalize these binary operators into n-ary operators as follows :

- for any constraint $f \in D^X \to \mathcal{L}$, $\sum_{\mathcal{L}} f = \ell\left(\sum_{\sigma \in D^X} \#f(\sigma)\right)$, and,
- for any set of intervals of $L = (\ell_1, \dots, \ell_k) \subseteq \mathcal{L}$, $\prod_{\mathcal{L}} L = \ell\left(\prod_{\ell \in L} \#L\right)$.

To explicit the isomorphism for the n-ary sum operator, we first introduce the $\int f$ notation defined as, given a function $f \in D^X \to \mathcal{L}$, we define $\int f \in D^X \to \mathbb{N}$ such that

$$\left(\int f\right)(\sigma) := \sum_{\sigma' \leq \sigma} \#f(\sigma') \ .$$

We also define :

$$\left(\overrightarrow{\int} f\right)(\sigma) := \sum_{\sigma' < \sigma} \#f(\sigma') \ .$$

And :

$$\left(\overleftarrow{\int} f\right)\left(n \in \ell\left(\sum_{\sigma} \#f(\sigma)\right)\right) := \sigma \text{ such that } \left(\overrightarrow{\int} f\right)(\sigma) \leq n < \left(\int f\right)(\sigma) \ .$$

We show in Section 7.2 [Implementing and Representing Cartesian Operators] how to efficiently implement and represent these operators.

One may chose any order $\leq$ on valuation for this definition to be valid, however, in the sequel we assume a lexicographic order with respect to a well-chosen order of the variables of X (and arbitrary order over D).

Then, we may explicit $\overrightarrow{\sum}$ and $\overleftarrow{\sum}$

- $\overrightarrow{\sum} \in \left\{(\sigma, n_\sigma) | \sigma \in D^X \wedge n_\sigma \in \ell\left(f(\sigma)\right)\right\} \to \ell\left(\sum_{\sigma \in D^X} \#f(\sigma)\right)$
  - $\overrightarrow{\sum}(\sigma, n_\sigma) := \left(\overrightarrow{\int} f\right)(\sigma) + n_\sigma$
- $\overleftarrow{\sum} \in \ell\left(\sum_{\sigma \in D^X} \#f(\sigma)\right) \to \left\{(\sigma, n_\sigma) | \sigma \in D^X \wedge n_\sigma \in \ell\left(f(\sigma)\right)\right\}$
  - $\overleftarrow{\sum}(n') :=$ let $\sigma := \left(\overleftarrow{\int} f\right)(n')$ in $\left(\sigma, n' - \left(\overrightarrow{\int} f\right)(\sigma)\right)$

We define $\prod_{\mathcal{L}}$ as the recursive application of the $\times_{\mathcal{L}}$ operator, hence, the definition of $\overrightarrow{\prod}$ and $\overleftarrow{\prod}$ naturally follows.

As one may see $\sum$ acts like a projection operator with respect to the constraint's domain on which it is applied. We may generalize its definition to take into account partial projection, let

$Y \uplus Z = X$ be a partition of Z, then $\sum_Y f = \sigma_Z \mapsto \sum_{\mathcal{L}} f(\cdot, \sigma_Z)$. Contrary to earlier chapters we cannot plunge back the result into $D^X \to \mathcal{L}$ as it would change its semantics.

Hence, we define the Cartesian CSA as $\texttt{CSA} = (D, \mathcal{L}, +, \times, \emptyset, \{0\})$.

Interestingly, as we have already made explicit the isomorphism and reverse-isomorphism between the input and output of both operators ($+$ and $\times$) along with their n-ary counterparts ($\sum_{\mathcal{L}}$ and $\prod_{\mathcal{L}}$), using the classical reduction algorithm (see Section 3.1) while storing intermediate results, allows to make explicit the isomorphism and reverse-isomorphism between the domain space $\{D^X \mid [\![\mathbf{F}]\!]\}$ and the rebased-space $\ell(\#f)$.

Hence, simplifying the theoretical work. We refer the reader to the following section to read more details about the actual implementation of the method.

7.2 Implementing and Representing Cartesian Operators

Definition 7.1. *Valued Valuation In the current context, we define a valued valuation $\sigma = (\sigma, \omega)$ as a pair, where $\sigma \in D^X$ and $[\![\mathbf{F}]\!](\sigma) = \top$ and $\omega : f \in F \mapsto n_f \in f(\sigma)$.*

Considering that usually all constraints are binary ones, their only valid value is $0 \in \ell(1)$.

Definition 7.2. *Global Isomorphism This inductive algorithms takes rCSTD $\mathbf{F}$ and a valued valuation ω of $[\![\mathbf{F}]\!]$ and compute the unique related element $\vec{[\![\mathbf{F}]\!]}(\sigma) \in \ell(\#f)$.*

let rec *global_isomorphism* $\mathbf{F}$ σ^0 σ =
σ_{00} := $\sigma \cap \mathrm{supp}(F_0)$
σ_0 := $\sigma^0 \cup \sigma_{00}$
n_0 := $\sigma(F_0)$
$\forall i$, N_i := $f_i(\sigma_0)$
$[\mathbf{F}_1; \ldots; \mathbf{F}_k]$:= *the sons* ***of*** F_0 ***in*** $\mathbf{F}$
for $i = 1$ ***to*** n
do
  σ_i^0 := *the restriction* ***of*** σ_0 ***to*** $\mathrm{supp}(\mathbf{F}_i)$
  σ_i := *the restriction* ***of*** σ ***to*** $\mathrm{supp}_{\sigma\omega}(\mathbf{F}_i) - \mathrm{supp}_{\sigma\omega}(\mathbf{F}_0)$
  n_i := *global_isomorphism* $\mathbf{F}_i$ σ_i^0 σ_i
done
return $\vec{\sum}_{F_0(\sigma^0,\cdot)} \left(\sigma_{00}, \vec{\prod}_{(N_0,N_1,\ldots,N_k)}(n_0, n_1, \ldots, n_k) \right)$

We explicit the following notations :

f_0 the primary valued constraint, that is, from the initial problem. If several valued constraints are related to the same component we assume that the compression/expansion is performed separately from the present induction.

- *f_i is the secondary valued constraint induced by the forward reduction of $(\mathbf{F}_i)_0$ on the current root.*
- $\mathrm{supp}(\mathbf{F})$ *returns the set of variables appearing in* $\mathbf{F}$
- $\mathrm{supp}_{\sigma\omega}(\mathbf{F})$ *returns the set of variables and constraint identifiers in* $\mathbf{F}$

In the term return we extend the valued valuation ω with $\{f_i \mapsto n_i\}$ although it may seem strange at first glance, one must remember that forward reduction created additional constraint which do not belong to the initial problem and are used to allow components to communicate.

In essence what the `global_isormophism` *inductive function does is first cast the valuation on each sub-rCSTD, retrieve the matching integer for each sub-system, annotate each secondary constraint (the one computed during forward reduction) with its corresponding integer, and locally apply both isomorphism operators* $\overrightarrow{\sum}$ *and* $\overrightarrow{\prod}$.

Definition 7.3. *Global Reverse-Isomorphism This inductive algorithms takes rCSTD* $\mathbf{F}$ *and an element* $n' \in \ell(\#f)$ *and compute the unique related valued valuation* $\omega := \overleftarrow{[\![\mathbf{F}]\!]}(n')$.

```
let rec global_reverse_isomorphism F σ⁰ n' =
  σ₀₀, n'' := ←∑_{F₀(σ⁰,·)}(n')
  σ₀ := σ⁰ ⊎ σ₀₀
  ∀i, Nᵢ := fᵢ(σ₀)
  (n₀, n'₁, ..., n'ₖ) := ←∏_{(N₀,N₁,...,Nₖ)}(n'')
  [F₁; ...; Fₖ] := the sons of F₀ in F
  for i = 1 to n
  do
    σᵢ⁰ := the restriction of σ₀ to supp(Fᵢ)
    σᵢ := global_reverse_isomorphism Fᵢ σᵢ⁰ n'ᵢ
  done
  return σ₀₀ ⊎ {f₀ ↦ n₀} ⊎ᵢ σᵢ
```

So far we have implement two algorithm to respectively compute the isomorphism and reverse-isomorphism relating $\{D^X \mid \xi\}$ with $\ell(\#f)$, the final step is to convert this algorithms as symbolic program in order to use them as transfer function in a rebase algorithm.

Essentially transforming these inductive algorithms into circuit is quite straight forward, the main difficulty is to express each operators as a symbolic program. In our usual case of a Boolean domain, digital circuits are a simple yet concise kind of symbolic programs.

Remark 7.2. *The following circuit is not intended to be compiled into a canonical representation but used as is, although it should work for CSD with a small number of solution (around less that* 2^{30}*) that is, either a small or very constrained one (or a combination of both factor).*

Definition 7.4. *Circuit of* qprod

```
let qprod q b r = q × b + r
```

Definition 7.5. *Circuit of* qmod *A simple way to either compute non-negative integer quotient or modulo in term of Boolean circuit is to inductively compute both in parallel. Hence, we define* $\mathrm{qmod}(a, b) = (q, r)$ *with*

a, b, q, r *being binary encoded non-negative integers, and,*

- $q = a \div b$ *and* $r = a \mod b$.

For any non-negative binary encoded integer i*, we denote*

$|i|$ *the number of bit used to represent* $|i|$

- i_k *the* k*-th bit of* i*, if* i *is encoded in big-endian then* i_0 *represents its strongest otherwise (in small-endian) it represent its weakest bit. In the following, we assume a big-endian representation.*
- i_+ *denotes* i *deprived of its first bit* i_0*.*
- i `lsl` k *is* i *with its last* k *bits filled with zeros. (*$[\![i \texttt{ lsl } k]\!] := [\![i]\!] \times 2^k$*)*
- *the* normalize *operator iteratively removes the strongest bit if its isomorphic to the constant circuit* 0

```
let qmod a b =
  R := a
  q := 0^|a|
  for k = |a| -1 downto 0
  do
    q_k := (b lsl k) ≥ R
    R ← R - q_k × (b lsl k)
  done;
  return (normalize(q), R)
```

One may notice that this circuit also work if "$b = 0$*" by returning "*$q = 0$*" and "*$R = a$*". Hence, the property "*$a = b.q + R$*" holds even if "*$b = 0$*". However, to assume that "*$0 \leq R < b$*" one must ensure that "*$b > 0$*".*

Definition 7.6. *Circuit of* $\overrightarrow{\times}$

```
let times_isomorphism NX NY nX nY =
  cast (ℓ(NX × NY)) (qprod nX NY nY)
```

Definition 7.7. *Circuit of* $\overleftarrow{\times}$

```
let times_reverse_isomorphism NX NY n' =
  cast (ℓ(NX) × ℓ(NY)) (qprod n' NY)
```

The circuit of $\overrightarrow{\prod}$ (resp. $\overleftarrow{\prod}$) is made by composing $\overrightarrow{\times}$ (resp. $\overleftarrow{\times}$) with it self $(k-1)$-times.

Lemma 7.1. *Characterization of* $\overrightarrow{\int}$ *Let* f *be constraint, then either*

$f \in D^{\emptyset} \to \mathcal{L}$*, then,* $\overrightarrow{\int} f = \varepsilon \mapsto 0$

- *otherwise,* $\overrightarrow{\int} f = \sigma \mapsto \left(\overrightarrow{\int} f_0\right) \star \left(\#f_0 + \left(\overrightarrow{\int} f_1\right)\right)$

With $\star$ *being the Shannon operator.*

Remark 7.3. *One could refine refine the definition of* $\int$ *and related properties to take into account some primitives, for examples useless or 0-canalizing variables (either, ordered, uniform or linear).*

The core requirement being that $\left(\overleftarrow{\int} f\right)$ *and* $\left(\overrightarrow{\int} f\right)$ *keep the same type remain each other's inverse :*

$$\left(\overleftarrow{\int} f\right) = \left(\overrightarrow{\int} f\right)^{-1} .$$

Definition 7.8. *Circuit of* $\overrightarrow{\int}$ *We define* $\overrightarrow{\int}$ *by induction on the arity of its input function using the above characterization.*

Remark 7.4. *One may notice that assuming that* f *is encoded using an MDD, then, the circuit of* $\overrightarrow{\int} f$ *as described above, has a size bounded by* $\mathcal{O}\left(|f| \times |X|\right)$ *and a depth bounded by* $\mathcal{O}(|X|^2)$.

In practice, we encode f *using a array of BDD but we can easily emulate this behavior using memoization technique.*

Lemma 7.2. *Characterization of* $\int$ *using* $\overrightarrow{\int}$ *For any constraint* $f \in D^X \to \mathcal{L}$,

$$\int f = \left(\overrightarrow{\int} f\right) + f \ .$$

Definition 7.9. *Circuit of* $\overleftarrow{\int}$ *The circuit of* $\overleftarrow{\int} f$ *is derived from the circuit of* $\overrightarrow{\int} f$ *copied* $|X|$ *times in order to perform a binary search in* $\overrightarrow{\int} f$.

```
let integration_reverse_isomorphism f n' =
  compute ∫→f and wrap it as a module
  σ := 0^|X|
  for i = 0 to |X| -1
  do
    σ' := σ with σ'_i ← 1
    σ_i ← (∫→f)(σ') ≤ n'
  done;
  σ
```

Remark 7.5. *Assuming that* $\overrightarrow{\int} f$ *has a space complexity of* $\mathcal{O}(|f| \times |X|)$ *and a depth of*

$\mathcal{O}(|X|^2)$, then, $\overleftarrow{\int} f$ has a space complexity of $\mathcal{O}(|f| \times |X|^2)$ and a depth of $\mathcal{O}(|X|^3)$.

We directly encode $\overrightarrow{\sum}$ and $\overleftarrow{\sum}$ as circuits using their definitions : Then, we may explicit $\overrightarrow{\sum}$ and $\overleftarrow{\sum}$

- $\overrightarrow{\sum}(\sigma, n_\sigma) := \left(\overrightarrow{\int} f\right)(\sigma) + n_\sigma$
- $\overleftarrow{\sum}(n') :=$ let $\sigma := \left(\overleftarrow{\int} f\right)(n')$ in $\left(\sigma, n' - \left(\overrightarrow{\int} f\right)(\sigma)\right)$

Chapter 8

Parametric Reduction

In essence, extending tree-based reduction with parameter variables is covered by Theorem 6.2 [Functional Partial Extensionality].

8.1 Parametric Reduction

Let $P \subseteq X$ a subset of variables, we call parameter variables.

Then given an optimizing constraint set F, our objective is to compute $\exists_{|P} f$ and $\Pi_{|P} f$. That is for every valuation of parameter variables σ_P, the "optimal" value $(\exists_{|P} f)(\sigma_P)$ of $f(\sigma_P)$, along with the "arg-optimal" set $(\Pi_{|P} f)(\sigma_P)$ of $f(\sigma_P)$.

We generalize Definition 4.2 [Graph Decomposition] into parametric graph decomposition and forward parametric graph decomposition.

Definition 8.1 (Parametric Graph Decomposition)**.** *Let $D = (B, I)$ a graph decomposition, we say that D is a parametric graph decomposition (with respect to P) iff*

- $\forall b \in B, P \subseteq b$*, and,*
- $P \in B$*.*

Remark 8.1 (Why $P \in B$)**.** *We use P as the root component of the tree-decomposition when orienting.*

Furthermore, in the Boolean case, the component f_P is used to represent the parametric satisfiability (that is, the "set" of valuation of parameter variables for which the constraint system has a solution).

Finally, in the quantitative case, the component f_P is used to represent the parametric "optimal" value.

Remark 8.2. *In the Boolean case, parametric local projection is just local projection with respect to a parametric graph decomposition.*

Definition 8.2 (Parametric Local Projection of Parametric Optimizing Constraint System)**.** *Let D a parametric graph decomposition with respect to P and f a parametric constraint.*

We denote $\mathrm{LP}(f, D, P) := \mathbf{F}$*, where :*

- $\mathbf{F} := \mathrm{LP}^{\mathbb{B}}(\Pi_{|P} F, D \backslash \{P\})$ *extended with*
- $f_P := \exists_{|P} f$*.*

Remark 8.3. *One should make the difference between*

- $D\backslash\{P\}$*, that is, the decomposition D where the block P (which exists by Definition 8.1 [Parametric Graph Decompostiotn]) is removed.*
- $D\backslash P$*, that is the decomposition $D' = (\{b\backslash P \mid b \in B\}, I')$ with I' the corresponding mapping of edges.*

Lemma 8.1 (Parameter Evaluation)**.** *Let f be a parametric (optimizing) constraint, and D be a parametric graph decomposition with respect to P.*

Assuming D is an acceptable decomposition of f (see Definition 4.10), then,

$$\forall \sigma_p \in D^P, \mathrm{LP}(f, D, P)[P \leftarrow \sigma_p] = \mathrm{LP}(f[P \leftarrow \sigma_P], D\backslash P) \ .$$

That is,

Let $\sigma_P \in D^P$ be a valuation of parameter variables, then, $LP(f, D, P)$, evaluated in σ_P is equal to the local projection of f with parameter variables set to σ_P.

Proof. *By definition parametric graph decomposition, for any component $b \in B$, we have $P \subseteq b$.*

In the Boolean case,

Hence, using Lemma 6.2 [Projection and Partial Evaluation], we prove the required statement by unfolding the definition of LP *operator.* □

Remark 8.4 (Orientation in the Parametric Case)**.** *Let D be a parametric graph decomposition.*

In the parametric case, we assume that any orientation $\mathcal{O}$ as component P at its root.

Remark 8.5 (Parametric Forward/Backward Propagation)**.** *Assuming*

- *usage of a parametric graph decomposition, and,*
- *proper orientation (as stated in Remark 8.4 [Orientation in the Parametric Case])*

Parametric forward/backward propagation is no different from classical forward/backward propagation.

Remark 8.6 (Parametric Forward Reduction)**.** *In a parametric graph decomposition, the component P acts as root of the decomposition (instead of $\emptyset$ in the non-parametric version).*
Hence, f_P acts as the root of the parametric OCSTD instead of $f_\emptyset$.
Hence, no algorithmic change is required.

Remark 8.7 (Parametric Reduction)**.** *We defined parametric reduction similarly to non-parametric reduction by composing, parametric forward reduction and parametric backward propagation.*

Theorem 8.1 (Parametric Global Propagation is Local Projection)**.** *Let P a set of parameter variables.*
Let D a parametric graph decomposition with respect to P.
Let $\mathbf{F}$ a parametric CSTD with respect to D.
Then,

$$\mathrm{GP}(\mathbf{F}) = \mathrm{LP}(\llbracket F \rrbracket, D, P) \ .$$

Proof. *Corollary of Theorem 5.3.* □

Theorem 8.2 (Parametric Reduction is Local Projection)**.** *Let P a set of parameter variables.*
Let D a parametric graph decomposition with respect to P.
Let $\mathbf{F}$ a parametric OCSTD with respect to D.
Then,

$$\mathrm{RP}(\mathbf{F}) = \mathrm{LP}(\llbracket \mathbf{F} \rrbracket, D, P) \ .$$

Proof. *By first adapting proofs of Lemma 6.18 [Backward Propagation Process of Forward Reduced is Local Projection] and Lemma 6.6 [Reduction Process is Local Projection].* □

8.2 Forward Parametric Graph Decomposition

The parametric graph decomposition deals with mode variables in a monolithic way, in this section, we introduce *forward parametric graph decomposition* a refinement of this decomposition which takes advantage of the locality of mode variables within the primal graph.

In particular, in the admittedly strange case that a parameter variables is not used it still appears everywhere in the support.

While this issue is limited if the underlying representation can efficiently deal with useless variables, it still hinders the cost estimation related to the computation time and memory usage of the method.

This decomposition allows to represent more precisely how does parameter variables do propagate during forward reduction.

Definition 8.3 (Forward Parametric Graph Decomposition)**.** *Let $D = (B, I)$ a graph decomposition and $\mathcal{O}$ an orientation of D, then $\mathcal{D}$ is forward parametric iff*

- *for every connected component D_i of $D \backslash P$, there exists a unique element $b_{D_i} \subseteq P$ which has no outgoing arc in $\mathcal{O}$.*
- *for every oriented interconnection $(s, d) \in \mathcal{O}$, $(s \cap P) \subseteq (d \cap P)$.*

Remark 8.8 (Casting Forward Parametric to Parametric Graph Decomposition)**.** *Let $D = (B, I)$ a forward parametric graph decomposition and $\mathcal{O}$ an orientation of D, we define $D' = (B', I')$ the graph decomposition where for each $b \in B$, we define $b' := b \cup P$ in B'. Interconnections in D are mapped accordingly into D'.*

One may show that D' is a parametric graph decomposition.

Remark 8.9. *Parametric graph decomposition are trivial case of forward parametric graph decomposition.*

Remark 8.10. *adapt global propagation process and reduction process to take into account for forward parametric graph decomposition.*

Remark 8.11 (Cost Estimation)**.** .

Joan: Add discussion on cost estimation.

8.3 Casting Down Parametric Tree Decomposition

This section aims at

1. defining parametric tree decomposition as an extension of the classical tree-decomposition when dealing with parametrised formula (with either propagation or reduction), and,
2. reducing the problem of finding a "good" parametric tree decomposition to more classical problem of finding a "good" tree-decomposition by adapting the graph on which it is computed.

Remark 8.12. *Admitted definitions on graphs : paths, distance. Variation of admitted definition on graphs : vertex-contraction, vertex-minor.*

Definition 8.4 (Path In Graph)**.** *Let $G = (V, E)$ a graph, we call $p = (p_0, \cdots, p_k) \in V^*$ a path in G iff,*

$$\forall i,\ (p_i, p_{i+1}) \in E \ .$$

We denote $|p| = k$.

Definition 8.5 (End Point Path In Graph)**.** *Let $G = (V, E)$ a graph, and, $u, v \in V$ two vertices, and p a path in G.*

We say that p is a path from u to v (denoted $p : u \to v$) iff $p_0 = u$ and $p_{|p|} = v$.

We say that p is a path between u and v (denoted $p : u \leftrightarrow v$) iff p is either a path from u to v or a path from v to u.

Definition 8.6 (Distance In Graph)**.** *Let $G = (V, E)$ a graph, and, $u, v \in V$ two vertices.*

We define the distance in G between u and v as

$$d_G(u, v) = \min_{p:u\leftrightarrow v} |p| \ .$$

If there exists no path between u and v, then, we define the distance between them to be $+\infty$

That is, $d_G(u, v) = +\infty \iff \neg(\exists p : u \leftrightarrow v)$.

We extend the definition of distance from vertices to sets of vertices as follows:

Given two non-empty sets $Y, Z \subseteq V$, we denote

$$d_G(Y, Z) := \min_{(y,z)\in Y\times Z} d_G(y, z) \ .$$

That is, the distance between two sets of vertices is the distance between their closest vertices.

Definition 8.7 (Neighbourhood In Graph)**.** *Let $G = (V, E)$, and $Y \subseteq V$, we denote*

- *$N_G(Y) := \{v \in V \mid d_G(v, Y) \leq 1\}$, the neighbourhood of Y in G.*
- *$N_G[Y] := \{v \in V \mid d_G(v, Y) = 1\} = N_G(Y) \backslash Y$, the strict-neighbourhood of Y in G.*

Definition 8.8 (Induced Sub-Graph)**.** *Let $G = (V, E)$ be a graph an $Y \subseteq V$ be a subset of vertices.*

We denote $G(Y) := (Y, E \bigcap Y^2)$ the sub-graph of G induced by Y.

Definition 8.9 (Vertex Minor)**.** *Let $G = (V, E)$ be a graph an $Y \subseteq V$ be a subset of vertices.*

We denote $G[Y] := (Y, E[Y])$ the vertex minor of G induced by Y, defined as :

$$E[Y] := \left\{(u, v) \in E \mid (u, v \in Y) \wedge \left(\exists p : u \leftrightarrow v \in G\left(\{u, v\} \uplus Y^{\complement}\right)\right)\right\} \ .$$

We call vertex contraction of G with respect to v the graph $G[\{v\}^{\complement}]$ which is G with v being removed and a clique on $N_G[v]$ being added.

Remark 8.13 (Distinction Between Vertex Minor and Edge Minor)**.** *The notion of edge contraction was popularized by Robertson and Seymour [10]. The notions of edge minor and vertex minor are similar, in the sense, that they both preserve reachability while removing an basic element of the graph.*

- *vertex contraction preserves reachability while removing a vertex by creating a clique on its neighborhood.*
- *edge contraction preserves reachability while removing an edge by merging incident vertices.*

Definition 8.10 (Mapping Existential Projection As Graph Operators)**.** *Let $G = (V, E)$ be a graph and $Y \subseteq V$ a subset of vertices.*

We define existential projection as $\exists_Y G := G[Y^{\complement}]$.

Definition 8.11 (Connected Component Of A Graph)**.** *Let $G = (V, E)$ a graph, and $Y \subseteq V$ a subset of vertices.*

Then Y is said connected in G iff

$$\forall u, v \in V^2, \exists p : u \leftrightarrow v \in G \ .$$

We say that Y is a connected component of G iff Y is connected in G and Y is maximal with respect to the inclusion $\subseteq$.

We define a connected component decomposition of G as a partition $\mathbf{Y} = (Y_i)_i$ of X, that is $\forall i$, $Y_i \neq \emptyset$ and $Y_1 \uplus \cdots \uplus Y_k = V$, such that for every index i, Y_i is a connected component of G.

We define the connected decomposition of Y in G as a partition $\mathbf{Y} = (Y_i)_i$ of Y, such that $\mathbf{Y}$ is a connected component decomposition of $G(Y)$.

Lemma 8.2 (Linear-Time Existential Projection)**.** *Let $G = (V, E)$ be a graph and $Y \subseteq V$ a subset of vertices.*

Assuming that Y is connected in G (that is, $G(Y)$ is a connected component), then,

We define the existential projection $\exists_Y G$ of G with respect to Y, as :

- $\exists_Y G := (V', E')$, *with,*
- $V' := V \backslash Y$, *and,*
- $E' := \left(E \bigcup N_G(Y)^2\right)_{|(V')^2}$.

Otherwise, we compute a connected decomposition $\mathbf{Y} = (Y_i)_i$ of Y in G.

Then, $\exists_Y G = \exists_{Y_k} \cdots \exists_{Y_1} G$ (one may notice that the result is independent of the projection order of the components of $\mathbf{Y}$).

Hence, one can compute vertex-minors in linear-time as follows $G[Y] = \exists_{Y^{\complement}} G$.

Remark 8.14 (Existential Projection)**.** *$\exists_v G$ can be interpreted as a local transitive elimination of v in G.*

Then, $\exists_Y G$ can be also interpreted as a local transitive elimination of every vertices $v \in Y$ in G, notice that the resulting graph is independent of the order in which vertices are eliminated.

Definition 8.12 (Simplification Of Parametric Graph)**.** *Let* $\mathbf{G} = (P \uplus V, E)$ *a parametric graph.*

Let $\mathbf{V} = (V_i)_i$ *the connected decomposition of* V *in* G*.*

Then, we define the simplification $\text{simple}(\mathbf{G})$ *of* G *as*

- $\text{simple}(\mathbf{G}) = (G'_i)_i$,
- *with* $G'_i := G(N_G(V_i)) \cup K_{N_G[V_i]}$,
- *with, for any set of vertices* Y, $K_Y = \left(Y, Y^2\right)$ *the complete sub-graph (clique) on* Y*.*

Definition 8.13 (Concatenation Of Parametric Graphs)**.** *Let* $G_1 = (P_1 \uplus V_1, E_1)$ *and* $G_2 = (P_2 \uplus V_2, E_2)$ *be two parametric graphs.*

Such that

- $V_1 \cap V_2 = \emptyset$*, and,*
- $V_1 \cap P_2 = \emptyset$*, and,*
- $V_2 \cap P_1 = \emptyset$*.*

We denote $G_1 || \ G_2$ *their concatenation, defined as*

- $G_1 || \ G_2 := (P_{12} \uplus V_{12}, E_{12})$*, with*
- $P_{12} = P_1 \cup P_2$*, and,*
- $V_{12} = V_1 \uplus V_2$*, and,*
- $E_{12} = E_1 \cup E_2$*.*

Definition 8.14 (Concatenation Of Parametric Tree-Decomposition)**.** *Let* $G_1 = (P_1 \uplus V_1, E_1)$ *and* $G_2 = (P_2 \uplus V_2, E_2)$ *be two parametric graphs which can be concatenated.*

Let T_1 *be a parametric tree-decomposition of* G_1 *and* T_2 *be a parametric tree-decomposition of* G_2*.*

Then, we define $T_1 || \ T_2$ *their concatenation (which is a parametric tree-decomposition of* $G_1 || \ G_2$*) as follows*

- *let* $(B_1, I_1) := T_1$ *with* $b_1 \in B$ *its root, and,*
- *let* $(B_2, I_2) := T_2$ *with* $b_2 \in B$ *its root, and,*
- *we denote* $B_{12} := B_1 \cup B_2 \cup P$ *with* $P_{12} = P_1 \cup P_2$*, and,*
- *we denote* $I_{12} := I_1 \cup I_2 \cup \{(b_1, P), (b_2, P)\}$*.*
- *we define* $T_1 || \ T_2 := T_{12} := (B_{12}, I_{12})$*.*

Definition 8.15 (Simple Parametric Graph)**.** *Let* $G = (P \uplus V, E)$ *a parametric graph.*

We say that G *is simple iff*

- $G(V)$ *is connected, and,*
- $P = N_G[V]$*, and,*
- $G(P)$ *is a clique.*

Theorem 8.3 (Parametric Tree-Decomposition Of Simple Parametric Graph)**.** *Let* $G = (P \uplus V, E)$ *a parametric graph.*

Then,

$$\text{TD}^{\mathcal{P}}(G) = \text{TD}(G) \ .$$

Furthermore,

$$\mathrm{tw}^{\mathcal{P}}(G) = \mathrm{tw}(G) \ .$$

Theorem 8.4 (Simplification Preserves Tree-Decomposition)**.** *Let $G = (P \uplus V, E)$ be a parametric graph, then, parametric tree-decompositions on G are exactly the concatenation of tree-decomposition on each graph in* $\mathrm{simple}(G)$.

That is, denoting $(G_i)_i := \mathrm{simple}(G)$,

$$\mathrm{TD}^{\mathcal{P}}(G) = \mathrm{TD}^{\mathcal{P}}(G_1)||\cdots||\,\mathrm{TD}^{\mathcal{P}}(G_k) \ .$$

Lemma 8.3 (Treewidth Of Concatenation Of Parametric Graph)**.** *Let G_1 and G_2 two parametric graph, such that G_1 and G_2 can be concatenated.*

Then,

$$\mathrm{tw}^{\mathcal{P}}(G_1||G_2) = \max(\mathrm{tw}^{\mathcal{P}}(G_1),\, \mathrm{tw}^{\mathcal{P}}(G_2)) \ .$$

Theorem 8.5 (Parametric Treewidth Reduced To Classical Treewidth)**.** *Let $\mathbf{G} = (P \uplus V, E)$ be a parametric graph, then,*

$$\mathrm{tw}^{\mathcal{P}}(\mathbf{G}) = \max_{G \in \mathrm{simple}(\mathbf{G})} \mathrm{tw}(G) \ .$$

Proof. *Using*

- *Theorem 8.4 [Simplification Preserves Tree-Decomposition], and,*
- *Lemma 8.3 [Treewidth Of Concatenation Of Parametric Graph], and,*
- *Theorem 8.3 [Parametric Tree-Decomposition Of Simple Parametric Graph].*

□

8.4 Parametric Back-Selection

Dealing with the parametric extension is rather straight-forward from a semantic point-of-view by extensively using functional extensionality, along with

- altering the definition of tree-decomposition to forward-parametric-tree-decomposition, ensuring that
  - the root component only contains parameter variables, and,
  - for any component, is set of parameter variables, contains the set of parameter variables in any of its children.
- adapting the algorithmic to efficiently deal with parameters, in particular, defining a parametric back-selection based on symbolic computation.

Definition 8.16 (Parametric Solution Selection)**.** *In order to define parametric symbolic back-selection, we first need to implement parametric (solution) selection.*

We denote `parametric_solution_selection(ξ, Y, Z) = S`

- *given $\xi \in \mathcal{A}^X \to \mathbb{B}$ a Boolean constraint,*

- *given two non-intersecting sets Y and Z, such that* $\texttt{supp}\,(\xi) \subseteq Y \uplus Z$
- *returns a function $S \in \mathcal{A}^Y \to \mathcal{A}^Z \cup \{\bot\}$ such that*
  - *for every valuation $\sigma_Y \in \mathcal{A}^Y$, either,*
    - *$\xi(\sigma_Y, \cdot) = \bot$ (the sub-constraint is unsatisfiable), then $S(\sigma_Y) = \bot$,*
    - *otherwise, $S(\sigma_Y)$ is solution of $\xi(\sigma_Y, \cdot)$, that is, $\xi(\sigma_Y, S(\sigma_Y)) = \top$.*

This function is actually implemented in two inductive steps,

- *first, we inductively (on the structure of BDD) compute a maximal (number of solution wise) sub-constraint ξ', such that $\xi' \subseteq \xi$ and $\forall \sigma_Y \in \mathcal{A}^Y, \sharp(\xi'(\sigma_Y, \cdot)) \leq 1$*
  - *for any constraint f, $\sharp f$ returns the number of solution of f.*
- *then, we compute for each variable $z \in Z$, $\xi'_z := \exists_{Z \setminus \{z\}} \xi'$, this step can also be performed by induction over the structure of BDD.*
- *finally, every $\{\xi'_z\}_z$ are packed together together with $\exists_{|Y} \xi$ to form the correct object $S \in \mathcal{A}^Y \to \mathcal{A}^Z \cup \{\bot\}$*

Definition 8.17 (Multimode Back-Selection)**.** *We define parametric back-selection in a parametric CSTD in three steps,*

- *first we describe the algorithm for each step of back-selection,*
- *then, we apply this single step inductively on the tree structure of the CSTD,*
- *finally, we merge all partial solution together (this merging is consistent by hypothesis).*

First, we describe a single back-selection step *:* `backproj_step(s, S_s, d, ξ_d)` *= S_d where*

- *s is a set of regular (i.e. non-mode) variables denoted by a component in the tree decomposition*
- *$S_s \in \mathcal{A}^M \to (\mathcal{A}^s \cup \{\bot\})$ is an affectation of the variables in s,*
- *d is a set of regular (i.e. non-mode) variables denoted by a component in the tree decomposition*
- *ξ_d is a Boolean constraint, such that*
  - *$\texttt{supp}\,(\xi_d) \subseteq d \cup M$, and,*
  - *(consistency hypothesis) S_s is a (parametric) solution to $\exists_{(d-s)} \xi_d$*
- *$S_d \in \mathcal{A}^M \to (\mathcal{A}^d \cup \{\bot\})$ is a (parametric) solution to ξ_d extending S_s while restricted to d.*
  - *first, we inject S_s into ξ_d defining a new formula ξ'_d such that all solution of ξ'_d extend S_s.*
  - *dues to consistent hypothesis, $\exists_{|M} \xi_d = \exists_{|M} \xi'_d$*
  - *using parametric solution selection on ξ'_d we retrieve a maximal affection S_d of ξ'_s,*
  - *therefore, S_d extends S_s while being restricted to d and satisfying ξ_d.*

Second, we inductively use this single step of back-selection *We leverage the fact that the input formula is forward consistent to ensure that the consistency hypothesis is met at each step of the algorithm.*

Finally, we merge (in linear time) all partial solution*, due to the separation property of parametric tree decomposition and by correction of this method, the merging leads to a consistent affection $S \in \mathcal{A}^M \to (\mathcal{A}^X \cup \{\bot\})$.*

Lemma 8.4 (Consistency of Multimode Back-Selection with Single-Mode Back-Selection)**.**

Let σ_M be a valuation of mode variables (that is, a mode), then applying the parametric back-selection on f and evaluating in σ coincides with a possible execution of the single-mode back-selection on $f(\sigma_M, \cdot)$

Conclusion

In this work, we introduce the of Constraint System Algebra (CSA) which allows to state an essential difference between queries on constraint system which are purely based on the result of some well-defined computation and these which have to exhibit some kind of relation between the returned result and the variables used to define the constraint system.

We showcase several common problems (SAT, ILP, etc.) in term of a well chosen CSA, then split these problems into three categories (1) double idempotent CSA, that is both operators are idempotent (e.g. $\wedge/\vee$ or $\max / \min$), which is covered by Chapter 4 [Propagation and Consistency] and Chapter 5 [Tractability of Tree Decomposition], (2) idempotent/monotonic CSA, that is the addition operator is idempotent and the multiplication operator is monotonic (w.r.t some well-chosen order) (e.g. $\max /+$) which is covered by Chapter 6 [Tree-Based Reduction] and finally double monotonic CSA, that is, both operators are monotonic with respect to some well-chosen order (e.g. $+/\times$).

Orthogonally, in addition to these classes of CSA, we introduce their parametric extension along with how to adapt previous algorithm to the parametric case.

Future work include

- benchmark the implementation of these methods as implemented in Snowflake [13] on several large scale examples,
- extend the methodology for totally ordered co-domains to partially ordered co-domains,
- improve the computation method of parametric problem using result on the Fixed-Parameter-Tractable (FPT) algorithm for Quantified Boolean Formula (QBF)[1]
- extend the parametric to quantifier alternation case, and, later, to several-players games.

Bibliography

[1] Florent Capelli and Stefan Mengel. "Knowledge Compilation, Width and Quantification". In: *CoRR* abs/1807.04263 (2018). arXiv: 1807.04263. URL: http://arxiv.org/abs/1807.04263.

[2] Diego Cifuentes and Pablo A. Parrilo. "Chordal Networks of Polynomial Ideals". In: *SIAM Journal on Applied Algebra and Geometry* 1.1 (2017), pp. 73–110. DOI: 10.1137/16m106995x. URL: https://doi.org/10.1137%2F16m106995x.

[3] Diego Cifuentes and Pablo A. Parrilo. "Exploiting Chordal Structure in Polynomial Ideals: A Gröbner Bases Approach". In: *SIAM Journal on Discrete Mathematics* 30.3 (2016), pp. 1534–1570. DOI: 10.1137/151002666. URL: https://doi.org/10.1137%2F151002666.

[4] A. Darwiche and P. Marquis. "A Knowledge Compilation Map". en. In: *1* 17 (Sept. 2002), pp. 229–264. ISSN: 1076-9757. DOI: 10.1613/jair.989. URL: https://jair.org/index.php/jair/article/view/10311 (visited on 05/15/2019).

[5] Rina Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

[6] Katherine Heinrich. "Path decomposition". In: *Le Matematiche; Vol 47, No 2 (1992); 241-258* 47 (Jan. 1993).

[7] Philippe Jégou, Hélène Kanso, and Cyril Terrioux. "On the Relevance of Optimal Tree Decompositions for Constraint Networks". In: *2018 IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI)*. 2018, pp. 738–743. DOI: 10.1109/ICTAI.2018.00116.

[8] Jan-Hendrik Lange and Paul Swoboda. *Efficient Message Passing for 0-1 ILPs with Binary Decision Diagrams*. 2020. DOI: 10.48550/ARXIV.2009.00481. URL: https://arxiv.org/abs/2009.00481.

[9] S. Parter. "The Use of Linear Graphs in Gauss Elimination". In: *SIAM Review* 3.2 (1961), pp. 119–130. DOI: 10.1137/1003021. eprint: https://doi.org/10.1137/1003021. URL: https://doi.org/10.1137/1003021.

[10] Neil Robertson and Paul D. Seymour. "Graph Minors. II. Algorithmic Aspects of Tree-Width." In: *J. Algorithms* 7.3 (1986), pp. 309–322. URL: http://dblp.uni-trier.de/db/journals/jal/jal7.html#RobertsonS86.

[11] T. Schiex, Hélène Fargier, and Gérard Verfaillie. "Weighted Constraint Satisfaction Problems: Hard and Easy Problems". In: *IJCAI*. 1995.

[12] Stefan Szeider. "On Fixed-Parameter Tractable Parameterizations of SAT". In: *Theory and Applications of Satisfiability Testing*. Ed. by Enrico Giunchiglia and Armando Tacchella. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 188–202. ISBN: 978-3-540-24605-3.

[13] Joan Thibault. *Snowflake : Generic Symbolic Dynamic Programming framework*. URL: `https://gitlab.com/boreal-ldd/snowflake`.

RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399