Mapping Inferences: Constraint Propagation and Diamond Satisfaction

Gennari, R.

Publication date
2002

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: https://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
Chapter 4

Constraint Propagation Algorithms

4.1 Introduction

4.1.1 Motivations

In general, satisfying a constraint problem means computing a solution to it. CSPs can be so satisfied by a generate-and-test procedure: each possible combination of assignments is generated and tested against the given constraints. More refined strategies rely on the backtracking method: variables are instantiated sequentially; when a partial assignment is found inconsistent with a constraint, backtracking is performed to the last instantiated variable and, if possible, another value gets assigned to it. There are several variations of this basic form of backtracking, for instance see [Bac01, KvB97]. However, the run-time performance of backtracking is in general exponential for most CSPs, see [Kum92]. In [KvB97], the authors propose a theoretical comparison of backtracking algorithms for CSPs, based on a graph representation of these and on the number of visited nodes and edges.

What emerges from the analysis of the basic backtracking algorithm is that the reason for its poor performance is due to “trashing”: search keeps failing on the same type of subspace of the solution search space. Constraint propagation algorithms attempt to tackle this problem in various ways. For instance, constraint propagation algorithms, known as arc consistency algorithms, prune a CSP's domains from values that are inconsistent with the binary constraints of the CSP. Others, known as path consistency algorithms, prune inconsistent values from binary constraints of CSPs.

4.1.2 Outline

In this chapter, we present various algorithms for constraint propagation, and make use of the SGI schema (see Chapter 3) to explain these algorithms: by
representing them as instances of SGI or one of its variations; by analysing them, applying the theoretical results studied in Chapter 3.

**Instantiation.** Every time we need to represent an algorithm as an instance of SGI, we need to specify for SGI:

- an appropriate set,
- functions and a suitable partial order on the function domain, or on an equivalent set (see Subsection 3.4.3),
- the *update* operator, which is in charge of returning the necessary functions to iterate in SGI.

**Analysis.** Thus the general results obtained for SGI and its variations are used to analyse constraint propagation algorithms. For instance, inflationarity of functions is related to termination of algorithms and the absence of backtracking. Stationarity (see Definition 3.4.8) explain why some algorithms do not repeat the same kind of pruning. But these are still too general statements; let us try to specialise and clarify them in the context of the algorithms in this chapter.

**Classification.** Constraint propagation algorithms differentiate for the way and the type of pruning they perform on CSPs: i.e., in the terminology of Chapter 3, they correspond to different domain functions (see p. 23) or constraint functions (see p. 24). Thus a first broad classification separates constraint propagation algorithms according to this criterion; the sections of the present chapter correspond to the different constraint propagation classes that so emerge.

Section 4.2 is dedicated to the so-called arc and hyper-arc consistency algorithms; the SGI schema of Chapter 3 explains how those algorithms remove values from domains, and how the effects of the removals are propagated. In Section 4.3, we describe and study the so-called path consistency algorithms by means of SGI; the analysis there conducted also highlights how inconsistencies, at the level of binary constraints, are inferred and propagated along a 2-constraint ordering on the problem (see Subsection 2.5.2). Sections 4.4 and 4.5 deal with generalisations of the above algorithm classes; again, the SGI schema is sufficiently general to cover also these cases, and explain how pruning is performed (via functions) and propagation of inconsistencies is carried over (in particular, via the *update* operator).

**Separation.** A further, subtler analysis detects how each algorithm of a given class enforces and reaches its level of consistency, and so differentiates such an algorithm from the others that pertain to the same class. For instance, in Section 4.2, we describe and analyse in total four arc consistency algorithms; their
differences are easily grasped via the common language and framework of SGI function iterations (see Chapter 3). In fact, when these algorithms are shown to be all instances of the SGI schema, their differences can be explained in terms of diverse function iterations: e.g., through the use of diverse functions; through the use of some properties of functions like commutativity (see Definition 3.4.5), or stationarity (see Definition 3.4.8).

4.1.3 Structure

Section 4.2 is concerned with arc consistency and its generalisation, namely hyper-arc consistency. Then path consistency is described and studied in Section 4.3. We present an algorithm for $k$-consistency and discuss it in Section 4.4, and finally relational consistency in Section 4.5. This chapter is concluded by a summary table, see Section 4.6.

4.2 Arc and Hyper-arc Consistency

In this section, we deal with constraint propagation algorithms that only modifies domains of CSPs. We begin by introducing these notions, and then show how various algorithms for enforcing them can be recast in the framework of SGI.

First of all, let us introduce the main of those CSP properties that we deal with in this section: i.e., hyper-arc consistency as defined by Mohr and Masini, see [MM88].

**Definition 4.2.1.** Consider a CSP $P := (X, D, C)$ and a constraint $C(s)$ of $C$. Then the constraint $C(s)$ is **hyper-arc consistent** if the following condition is met:

for every $x_i \in s$ and $a \in D_i$ there exists $d \in C(s)$ such that $a = d[i]$.

We call the CSP $P$ **hyper-arc consistent** if all its constraints are hyper-arc consistent.

In other words: a CSP is hyper-arc consistent if, for each variable $x_i$ of the problem, each value in $D_i$ (the $x_i$ domain) is part of each constraint of the problem that involves $x_i$. For instance, the MAP COLOURABILITY PROBLEM in Subsection 2.3.1 is not hyper-arc consistent: in fact, the value cyan for $x_1$ is forbidden by the constraint (arc) on $x_1$ and $x_2$.

The better known notion of **arc consistency** of [Mac97] is obtained by restricting the above definition to the case of CSPs with only binary constraints; as in the case of the aforementioned MAP COLOURABILITY PROBLEM. Hence, that notion can be recast as follows: a binary constraint $C(x_i, x_j)$ is **arc consistent** if

for every $a \in D_i$ there exists $b \in D_j$ such that $(a, b) \in C(x_i, x_j)$,

for every $b \in D_j$ there exists $a \in D_i$ such that $(a, b) \in C(x_i, x_j)$. 

A CSP with only binary constraints is arc consistent if all of its constraints are.

**Example 4.2.2.** The map colourability problem \( P_{\text{map}} \) of Subsection 2.3.1 is not arc consistent, hence hyper-arc consistent. In fact, if we choose the variable \( x_1 \) and look up for its values in \( D_1 \), we see that cyan is forbidden by the constraint on \( x_1 \) and \( x_2 \); also blue is forbidden by the constraint on \( x_1 \) and \( x_3 \). Therefore, an arc consistency algorithm will remove the value cyan and blue from \( D_1 \), so obtaining a new CSP, with the same scheme and constraints as \( P_{\text{map}} \), and domain \( D_1 := \{ \text{aqua} \} \). No value will be removed from the domain of \( x_2 \), since cyan is not forbidden by any constraints that involves \( x_2 \). On the contrary, the value cyan for \( x_3 \) will be removed from the domain of \( x_3 \) because of the constraint on \( x_2 \) and \( x_3 \). Therefore, the obtained CSP, on \( x_1, x_2, x_3 \), has the same constraints as the map colourability problem \( P_{\text{map}} \) but domains which are subsets of the \( P_{\text{map}} \) ones: i.e., \( D_1 = \{ \text{aqua} \} \), \( D_2 = \{ \text{cyan} \} \), \( D_3 = \{ \text{blue} \} \).

### 4.2.1 The Basic Arc and Hyper-arc Consistency Algorithms

In this part, we describe and study the basic arc and hyper-arc consistency algorithms. First we introduce the basic algorithm for hyper-arc consistency, namely HAC-1, of which the one for arc consistency, namely AC-1, represents a special case. Then we show how GI can be instantiated to HAC-1, hence to AC-1. Finally we infer some properties concerning those basic algorithms by studying their GI instances.

**The HAC-1 and AC-1 algorithms**

The basic hyper-arc consistency algorithm HAC-1 enforces hyper-arc consistency by choosing a variable domain and iteratively enforcing hyper-arc consistency on this; AC-1 is like HAC-1 but it enforces arc consistency, i.e., only binary constraints are taken in consideration. In what follows, we first instantiate GI to both HAC-1 and AC-1, and then analyse these via the correlated instantiations of GI.

**Instantiation**

To prove that the algorithm HAC-1 for hyper-arc consistency is an instance of the GI algorithm, we need to specify, in the order, the following components:

1. a partial ordering with bottom;
2. a finite set of functions over the partial ordering;
3. the *update* operator.
Partial ordering with bottom. Consider a CSP $P := \langle X, D, C \rangle$. As partial ordering for HAC-1, we adopt the domain ordering on $P$, written as
\[ \langle \mathcal{D}(P), \sqsubseteq, D \rangle, \]
and partially ordered by the reverse of the subset relation, see Subsection 2.5.2. Therefore, given two domain sets $B := B_1, \ldots, B_n$ and $B' := B_1, \ldots, B_n$ of problems in the domain ordering, we have
\[ B \sqsubseteq B' \text{ iff } B_i \supseteq B'_i \text{ for every } i = 1, \ldots, n. \]
Remember that the input domain $D$ is the bottom of such a domain ordering.

Functions. We associate a function $\sigma(x_i; s)$ to each domain $D_i$ in $D$, and constraint $C(s)$ of the given input problem $P$ such that $x_i \in s$. Then we define the function $\sigma(x_i; s)$ as follows over the domain ordering: if $B$ is a domain set in $\mathcal{D}(P)$, then $B' := \sigma(x_i; s)(B)$ differs from $B$ at most in the domain $B'_i$ of $x_i$, this being
\[ B'_i := \Pi_i(C(s) \cap B [s]). \]
So $B'$ is always greater than $B$ with respect to $\sqsubseteq$, the domain order. Therefore each such function $\sigma(x_i; s)$ is trivially inflationary and monotone with respect to $\sqsubseteq$. Moreover, all these functions are idempotent, since intersection and projection are.

By taking into account only the binary constraints we obtain an analogous characterisation of arc consistency. The partial ordering is still the domain ordering but now on a binary CSP; see also Subsection 2.5.2. Then we associate two functions, $\sigma(x_i; x_i, x_j)$ and $\sigma(x_j; x_i, x_j)$, with, respectively, the domain $D_i$ and $D_j$, for each problem constraint $C(x_i, x_j)$ of the input problem. Thereby we define such functions over $\mathcal{D}(P)$ as follows, where $B$ is a domain set in the domain ordering on $P$:

- $B' := \sigma(x_i; x_i, x_j)(B)$ differs from $B$ at most in the domain $B'_i$ on $x_i$, since
  \[ B'_i := \Pi_i(C(x_i, x_j) \cap B_i \times B_j); \]
- $B' := \sigma(x_j; x_i, x_j)(B)$ differs from $B$ at most in the domain $B'_j$ on $x_j$, since
  \[ B'_j := \Pi_j(C(x_i, x_j) \cap B_i \times B_j). \]

Update. The update operator for HAC-1 is characterised as follows:
\[ \text{update}(G, F, \sigma(x_i; s), B) := F - G. \]
Clearly, the above characterisation of update satisfies Axiom 3.3.1. Indeed, it is not an optimal instantiation of update in terms of space and executions of
the main while loop of GI. We shall see that the HAC-3 algorithm has a better instantiation of the update operator, and hence that algorithm does not suffer from these drawbacks of HAC-1.

A similar characterisation of update can be given for AC-1, and we leave it to the reader.

**Analysis**

The use of the above defined functions is clarified by the following lemma, which also sums up the relevant properties of the $\sigma(x_i; s)$ functions.

**Lemma 4.2.3.**

(i) A CSP $P := (X, D, C)$ is hyper arc consistent iff $D$ is a common fixpoint of all the functions of the form $\sigma(x_i; s)$, hence the last one with respect to the domain order on $P$.

(ii) Each function $\sigma(x_i; s)$ is idempotent, monotone and inflationary with respect to the order of the domain ordering on $P$. \hfill $\Box$

Fix now a CSP $P$. By instantiating the GII algorithm with the above defined functions $\sigma(x_i; s)$, we get the basic arc consistency algorithm AC-1. So we can prove that this algorithm enjoys the following properties as a consequence of the theoretical results for SGII, hence for GII — see Theorem 3.4.4.

**Corollary 4.2.4 (HAC-1 and AC-1).** Consider a finite CSP $P := (X, D, C)$.

- Assume that $P$ is not binary. Then the hyper-arc consistency algorithm HAC-1 on $P$ terminates by computing the greatest hyper-arc consistent problem that is equivalent to $P$; that is the least common fixpoint of the $\sigma(x_i; s)$ functions, defined as above.

- Suppose that $P$ is binary. Then the arc consistency algorithm AC-1 on $P$ always terminates by computing the greatest arc consistent problem that is equivalent to $P$; that is the least common fixpoint of the $\sigma(x_i; x_i, x_j)$ and $\sigma(x_j; x_i, x_j)$ functions, defined as above.

**Proof.** We only need to observe that each $\sigma(x_i; s)$ function preserves equivalence and then apply the results of Theorem 3.4.4. As for the equivalence, it is sufficient to notice that projection does not remove solutions from a CSP; therefore, neither its composition with intersection does. \hfill $\Box$
4.2. Arc and Hyper-arc Consistency

4.2.2 The HAC-3 and AC-3 Algorithms

In the HAC-1 algorithm, each function $\sigma(x_i; s)$ is associated with a variable domain $D_i$ and a constraint $C(s)$ on a scheme to which the variable belong; each time $\sigma(x_i; s)$ is applied and modifies its arguments, all functions of type $\sigma$ that are associated with a constraint involving the variable $x_i$ are added to the set $G$ of functions to iterate. In this section we show how this information about the commutativity can be exploited to add less projection functions of the form $\sigma(x_i; s)$ to the set $G$. What follows was devised in [Apt00a].

Recall that, in Definition 3.4.5, we introduced the notion of commutativity between two functions, $f$ and $g$, on the same domain $O$ as follows:

$$fg(o) = gf(o) \text{ for all } o \in O.$$ 

First, it is worthwhile to note that not all pairs of HAC-1 functions commute. In general, functions like $\sigma(x_i; x_i, x_j)$ and $\sigma(x_k; x_i, x_k)$ do not need to commute; see [Apt00a] for an example of this phenomenon.

The following lemma clarifies which of the above functions commute.

**Lemma 4.2.5 (Commutativity).** Consider a CSP $P$ and a constraint $C(s)$ of $P$ on the scheme $s$.

- For $x_i, x_j \in s$ the functions $\sigma(x_i; s)$ and $\sigma(x_j; s)$ commute.
- If $C'(t)$ is a constraint of $P$ on a scheme $t$ and the variable $x_i$ occurs in both schemes $s$ and $t$, then $\sigma(x_i; s)$ and $\sigma(x_i; t)$ commute.

**Proof.** We only sketch how the first claim is proved, and refer the reader to [Apt00a] for a full proof.

Consider $\sigma(x_i; s)$ and $\sigma(x_j; s)$, for $i \neq j$. Let $B$ be a domain set in the domain ordering of the given problem. The former function can only modify $B_i$, the domain of $x_i$; whereas the latter function can only modify $B_j$, the domain of $x_j$. Both functions do it by looking up for $d$ in $C(s)$ such that $d \in B[s]$. Whenever $d \in B[s]$ and $d \not\in C(s)$, $\sigma(x_i; s)$ and $\sigma(x_j; s)$ remove the projections of $d$ from $B_i$ and $B_j$, respectively. Instead, if $d \in C(s)$, neither $d[i]$ nor $d[j]$ are removed from $B_i$ and $B_j$ by $\sigma(x_i; s)$ and $\sigma(x_j; s)$, respectively. Therefore these functions commute. $\square$

Fix now a CSP. We derive a modification of the hyper-arc consistency algorithm HAC-1 from Subsection 4.2.1 by instantiating, this time, the GIC algorithm schema, see Subsection 3.4.1 and Theorem 3.4.7. We use the same set of functions $\sigma(x_i; s)$ as for HAC-1. Additionally we employ the functional $\text{Comm}$ (see Definition 3.4.5) that, given a function $\sigma(x_i; s)$ from $F$, returns the set of functions that $\sigma(x_i; s)$ commute with:

$$\text{Comm}(\sigma(x_i; s), F) := \{\sigma(x_j; s), \sigma(x_i; t) : x_j \in s \text{ and } x_i \text{ is in } s \text{ and } t\}.$$
By virtue of the Commutativity Lemma 4.2.5 each set \( \text{Comm}(\sigma(x_i; s), F) \) satisfies the assumptions of Lemma 3.4.6.

By limiting oneself to the set of functions \( \sigma(x_i; x_i, x_j) \) and \( \sigma(x_j; x_i, x_j) \) associated with the binary constraints, we obtain an analogous modification of the corresponding arc consistency algorithm. Using now the commutative version \( \text{GIIIC} \) of \( \text{GII} \), we conclude that the \( \text{AC-3} \) algorithm enjoys the same properties as the \( \text{AC-1} \) algorithm. A more general conclusion holds for \( \text{HAC-1} \) and the instantiation of \( \text{GIIIC} \) with the above functions \( \sigma(x_i; s) \).

We can now state that the \( \text{HAC-3} \) and \( \text{AC-3} \) algorithms enjoy the following properties, which are immediate consequences of the Commutativity Lemma 4.2.5 and the theoretical results for \( \text{SGIIIC} \), hence for \( \text{GIIC} \): i.e., Theorems 3.4.4 and 3.4.7.

**Corollary 4.2.6 (HAC-3 and AC-3).** Consider a finite CSP \( P := (X, D, C) \).

- **Assume that** \( P \) is not binary. **Thus the hyper-arc consistency algorithm** \( \text{HAC-3} \) on \( P \), namely \( \text{GIIIC} \) with the above defined \( \sigma(x_i; s) \) functions, **always terminates** by computing the greatest hyper-arc consistent problem that is equivalent to \( P \).

- **Suppose that** \( P \) is binary. The **arc consistency algorithm** \( \text{AC-3} \) is an instance of \( \text{GIIIC} \) on \( P \) with the above defined \( \sigma(x_i; x_i, x_j) \) and \( \sigma(x_j; x_i, x_j) \) functions. **Thus it always terminates** by computing the greatest arc consistent problem that is equivalent to \( P \). \( \square \)

The difference between \( (\text{H})\text{AC-3} \) and \( (\text{H})\text{AC-1} \) relates to the different specifications of the update operator. As a consequence of this, the former will gain in execution time.

### 4.2.3 The HAC-4 and AC-4 Algorithms

In this part, we describe and study the \( \text{HAC-4} \) algorithm of [MM88] via the \( \text{SGIISE} \) — see p. 42 — and \( \text{SGIIS} \) algorithms — see p. 41. First we introduce the original algorithm; then we display the necessary set and functions for the algorithm \( \text{HAC-4} \) to become an instance of \( \text{SGIISE} \); finally, we infer properties of \( \text{HAC-4} \) by studying an equivalent \( \text{SGIIS} \) instance.

The \( \text{AC-4} \) algorithm is \( \text{HAC-4} \) for binary constraints. We limit ourselves to the description and analysis of \( \text{HAC-4} \), since \( \text{AC-4} \) is a specialisation of \( \text{HAC-4} \). For a detailed analysis of \( \text{AC-4} \) via iterations of functions, we invite the reader to consult [Gen00].

Notice that the \( \text{HAC-4} \) and \( \text{AC-4} \) algorithms assume the input CSP to be normalised, i.e. to have at most one constraint on each scheme; see Subsection 2.4.1. Otherwise, the CSP is first normalised, and then propagation takes place on its normalisation.
4.2. Arc and Hyper-arc Consistency

The original algorithm

The HAC-4 algorithm enforces hyper-arc consistency by first constructing a set of elements, called $G$, that do not participate in any consistent instantiation for any of the input problem constraints. Then the propagation phase is carried on as in Algorithm A.4, see Appendix A.

In the initial phase of HAC-4, a construction of structures and an initial pruning take place. Each $a \in D_i$, for every variable domain $x_i$ of the problem, is checked: i.e., for each constraint problem $C(s)$ such that $x_i \in s$, only all $d \in C(s)$ for which $d[x_i] = a$ holds are stored in $C(x_i, a; s)$. If one of the $C(x_i, a; s)$ is empty, then $a$ is removed from $D_i$ and the pair $(x_i, a)$ is added to $G$.

In the HAC-4 algorithm propagation phase (see Algorithm A.4, Appendix A), a pair $(x_i, a)$ is non-deterministically chosen from $G$ and the effects of the removal of $a$ from $D_i$ are propagated through all $C(x_j, b; s)$.

Instantiation

Since we want to instantiate SCISE to HAC-4, we are in need to define what follows:

- an equivalence set;
- suitable functions;
- the update operator.

Equivalence set. Assume that $P := \langle X, D, C \rangle$ is a CSP with $n$ variables. Thus consider subsets of the form $C(x_i, a; s)$ of each input constraint $C(s)$, i.e., one for each $x_i \in s$ and $a \in D_i$. Denote with $C_{hac-4}$ a set of relations and sequences, defined as follows: create a sequence of the form $(x_i, a, s)$, for each $x_i$, $a \in D_i$ and scheme $s$ of $X$; then $C_{hac-4}$ collects all pairs of the form $(C(s), (x_i, a, s))$. We usually denote each such pair as $C(x_i, a; s)$.

For instance, if $C(s) = C(x_1, x_2)$ and $D_1 = \{ a \}$ and $D_2 = \{ a, b \}$, then we have three “copies” of $C(s)$: $C(x_1, a; s)$, $C(x_2, a; s)$, $C(x_2, b; s)$.

Given $C_{hac-4}$, we can finally define $C_{hac-4}$ as follows:

- $C_{hac-4}$ belongs to $C_{hac-4}$;
- if $E$ belongs to $C_{hac-4}$, then $E$ collects pairs of the form $E(x_i, a; s) := (E(s), (x_i, a; s))$, where $E(s) \subseteq C(s)$, and $(x_i, a; s)$ is specified as for $C(x_i, a; s)$.

There is precisely one $E(x_i, a; s)$ in $E$, for each $C(x_i, a; s)$ in $C_{hac-4}$.

Given $C_{hac-4}$ as above, we define the equivalence structure $\langle P_{hac-4}, \equiv_{hac-4} \rangle$ as follows:

- its elements are subsets of tuples $\langle B, E \rangle$ in which: $B$ is a domain set in the domain ordering on $P$ (see Subsection 2.5.2); whereas $E$ is in $C_{hac-4}$.
the equivalence relation \( \langle B, E \rangle \equiv_{\text{HAC-4}} \langle B', E' \rangle \) holds iff \( B = B' \);

- the input element \( \perp_{\text{HAC-4}} \) is set equal to the pair \( \langle D, C_{\text{HAC-4}} \rangle \).

Indeed, the binary relation \( \equiv_{\text{HAC-4}} \) is an equivalence relation, since it is reflexive, symmetric and transitive.

We introduce the functions over \( P_{\text{HAC-4}} \) that we shall use to instantiate SGISE to HAC-4 as below.

**Functions.** In the following, we define two types of functions for HAC-4: one type is used for the first while loop, the other type is for the second while loop.

- Let us define a function \( \theta(x_i, a; s) \) for each input domain problem \( D_i \) and element \( a \in D_i \). Such function is the identity everywhere except, possibly, on each \( E(x_i, a; s) \) and \( B_i \):

\[
\begin{bmatrix}
E'(x_i, a; s) := \text{sel}_{i=a} C(x_i, a; s) , \\
B'_i := B_i - (\{a\} - \Pi_i( E'(x_i, a; s))) ,
\end{bmatrix}
\]

where \( \text{sel}_{i=a} \) selects all the tuples \( d \) in \( C(x_i, a; s) \) such that \( d[i] = a \). In words: \( E(x_i, a; s) \) is mapped into its subset \( E'(x_i, a; s) \) of all \( d \in C(x_i, a; s) \) — that is equal to \( C(s) \) — the \( i \)-th component of which is \( a \). Then \( B'_i \) differs in \( a \) from \( B_i \) if \( E'(x_i, a; s) \) turns out to be empty.

- We define a function \( \phi(x_i, a; s) \), for each \( C(x_i, a; s) \). If \( a \in B_i \), then \( \phi(x_i, a; s) \) is the identity. Otherwise it is the identity everywhere except on each current domain \( B_j \) and current \( E(x_j, b; s) \), for all \( x_j \in s \) different from \( i \) and \( b \in D_j \), that are mapped to \( B'_j \) and \( E'(x_j, b; s) \), respectively, as follows:

\[
\begin{bmatrix}
E'(x_j, b; s) := E(x_j, b; s) - E(x_i, a; s) , \\
B'_j := B_j - \bigcup_{b \in D_j} \Pi_j( E(x_j, b; s) - E'(x_j, b; s)) .
\end{bmatrix}
\]

So, whereas \( \theta(x_i, a; s) \) prunes the value \( a \) from the current domain \( B_i \) if it has no supports in \( C(s) \), the function \( \phi(x_i, a; s) \) takes care of propagating the effects of the removal of \( a \) from its domain. So \( \phi(x_i, a; s) \) visits each set \( E(x_j, b; s) \), for \( x_j \in s \) different from \( x_i \) and \( b \in B_j \); it removes the tuples in which \( a \) occurs from the supports of \( b \), thus determines whether \( b \) should be removed.

**Note 4.2.7.** Observe that all the functions of type \( \phi \) as described in the latter item are the identity on the input problem: in fact, for each \( \phi = \phi(x_i, a; s) \), the element \( a \) belongs to \( D_i \), the input domain of \( x_i \), by definition of \( \phi(x_i, a; s) \). So the only functions that can modify the input problem are the \( \theta \) functions defined in the former item. Besides, the \( \theta \) functions that do not modify the input problem collapse into the identity function.
4.2. Arc and Hyper-arc Consistency

The update operator. We characterise the update operator as follows.

- If \( \theta(x_i, a; s)(B, E) \neq (B, E) \), then \( \text{update}(G, F, \theta(x_i, a; s), P) \) is the set of functions \( \phi(x_i, a; t) \) from \( (F - F_L) - G \).

- Similarly, if \( \phi(x_i, a; s)(B, E) \neq (B, E) \), then the set \( \text{update}(G, F, \phi(x_i, a; s)) \) only contains all the functions \( \phi(x_j, b; t) \) of \( (F - F_L) - G \) that satisfy the following conditions: \( x_j \in s, j \neq i, b \in B_j - B'_j \).

The following result is trivial, hence we state it as a fact.

**Fact 4.2.8.** The HAC algorithm is an instance of the SGIISE with the above defined equivalence set and functions.

Analysis

Now, we also set up to study SGIISE, once this is instantiated with the above set and functions. As for that, we need some technical lemmas; they are useful to prove that each SGIISE trace with \( \theta \) and \( \phi \) functions over the equivalence set \( R_{HAC-4} \) can be mapped into an \( \equiv_{HAC-4} \)-equivalent SGIISE trace over \( F_{HAC-4} \equiv_{HAC-4} \), and that this computes the greatest hyper-arc consistent problem that is equivalent to the input problem.

We start considering the following functions that are then iterated by SGIISE. These functions are defined on the domain ordering of \( P \), see Subsection 2.5.2:

- \( \Theta(x_i, a; s) \) is the identity on each \( B_j \) with \( j \neq i \), whereas it maps \( B_i \) to \( B'_i := B_i - (\{a\} - \Pi_i(C(s))) \);

- \( \Phi(x_i, a; s) \) is the identity on \( B_k \) with \( k = i \) or \( k \not\in s \); whereas it maps every other \( B_j \) to \( B'_j := B_j - E_j \), where \( E_j \) is the set of all \( b \in B_j \) that enjoy both the following properties:

\[
\exists d \in C(s) \text{ such that } d[i] = a \text{ and } d[j] = b, \\
\forall d' \in B(s) \cap C(s) \text{ such that } d'[j] \neq b.
\]

The update operator is characterised like in Lemmas 3.4.3 and 3.4.9:

- If \( \Theta(x_i, a; s)(B) = B \), then \( \text{update}(G, F, \Theta(x_i, a; s), P) \) is the empty set; otherwise \( \text{update}(G, F, \Theta(x_i, a; s), P) \) is the set of functions \( \Phi(x_i, a; t) \) from the set \( (F - F_L) - G \);

- similarly, if \( \Phi(x_i, a; s)(B) = B \), then \( \text{update}(G, F, \Phi(x_i, a; s), P) \) is the empty set. If that is not the case, then the set \( \text{update}(G, F, \Phi(x_i, a; s)) \) only contains all the functions \( \Phi(x_j, b; t) \) of \( (F - F_L) - G \) that satisfy the following conditions: \( x_j \in s, j \neq i, b \in B_j - B'_j \).
Clearly, the following statement holds, and it is proved as Lemma 4.2.3.

**Lemma 4.2.9.**
(i) A CSP $P := (X, D, C)$ is hyper-arc consistent iff $D$ is a common fixpoint of all functions of type $\Theta$ and $\Phi$, hence the last one with respect to the domain order on $P$.
(ii) Each function of type $\Theta$ and $\Phi$ is stationary, monotone and inflationary with respect to the domain order on the input problem; besides, if a $\Theta$ function is the identity on the input problem, then it is the identity function.

At this point, we need to prove the following two lemmas, concerning executions of SGIIS with the $\phi$ and $\theta$ functions, before we can state our main equivalence results.

**Lemma 4.2.10.** Consider a CSP $P$ and the associated $PHAC-\Theta$ equivalence set. Let $E_B$ be the set of all the $\theta$ functions as above. Suppose that $B$ and $E$ are the current input in an execution of the second while loop. Then we have the following:

- if $d \in E(x_i, a; s)$ then $d[i] = a$;
- if $\phi(x_j, b; s) \in G$ then $b \not\in B_j$;
- if $\phi(x_i, a; s)$ is the chosen function and $d \in E(x_j, b; s) - E(x_i, a; s)$, then $d[i] \neq a$ or $b \not\in B_j$.

**Proof.** The first item relies on the definition of $\theta(x_i, a; s)$ and update. Similarly, the statement in the second item follows from the definition of $\theta(x_j, b; s)$ and update.

As for the last item, observe that every $d \in E(x_j, b; s)$ belongs to $C(s)$. If $d[i] = a$ then the fact that $d \not\in E(x_i, a; s)$ is not due to $\theta(x_i, a; s)$ but to some function of the form $\phi(x_k, c; s)$. If $k \neq j$, then $d$ cannot belong to $E(x_j, b; s)$ either, due to the same function $\phi(x_k, c; s)$. Hence $d[i] \neq a$. Finally, if $k = j$, then $b = d[j] = c$, hence $\phi(x_k, c; s)$ is equal to $\phi(x_j, b; s)$. Thus the second item yields that $b \not\in B_j$.

Given the above lemma, it is easy to conclude the following one concerning the equivalence of the SGIISE algorithm, with $\theta$ and $\phi$ functions, and SGIIS with $\Theta$ and $\Phi$ functions.

**Lemma 4.2.11.** The SGIISE algorithm with $PHAC-\Theta$, $\theta$ and $\phi$ functions is $\equiv_{HAC-\Theta}$-equivalent to SGIIS with $PHAC-\Theta/\equiv_{HAC-\Theta}$, $\Theta$ and $\Phi$ functions.
4.2. Arc and Hyper-arc Consistency

PROOF. Consider an execution of the SGIISE algorithm with input $\Theta$, $\Phi$ functions and $\bot := D$; notice that the domain of functions is isomorphic to $P_{HAC-4} / \equiv_{HAC-4}$. Then the equivalence of the iterations with $\theta$ and $\Theta$ functions is obvious. On the other hand, the equivalence of executions with $\phi$ and $\Phi$ functions follows from Lemma 4.2.10 and the above definition of update. In fact, if $a \not\in B_i$, then

$$E'(x_j, b; s) := E(x_j, b; s) - E(x_i, a; s),$$

$$B'_j := B_j - \bigcup_{b \in D_j} \pi_j(E(x_j, b; s) - E'(x_j, b; s)),$$

and Lemma 4.2.10 entails $B' = \Phi(B)$. A similar argument proves the opposite implication.

Therefore, we get the following result concerning HAC-4 and its specialisation AC-4 to binary CSPs.

COROLLARY 4.2.12 (HAC-4 AND AC-4).

(i). Every execution of the HAC-4 algorithm terminates by computing the least fixpoint of the above defined $\Theta$ and $\Phi$ functions; i.e., the greatest hyper-arc consistent problem equivalent to the input one.

(ii). Every execution of the AC-4 algorithm terminates by computing the least fixpoint of the above defined $\Theta$ and $\Phi$ functions on binary constraints; i.e., the greatest arc consistent problem equivalent to the input one.

PROOF. Theorem 3.4.10 and Lemma 4.2.9 imply that every execution of the SGIISE algorithm terminates, by computing the least common fixpoint of the $\Theta$ and $\Phi$ functions; this is the greatest hyper-arc consistent problem that is equivalent to the input one, due to Lemma 4.2.9 again. Thus Lemma 4.2.11 and Corollary 3.4.16 yield our corollary.

4.2.4 The HAC-5 and AC-5 Algorithms

The AC-5 algorithm of [vHDT92] is itself an algorithm schema, devised to enforce arc consistency on binary CSPs. As AC-4, the AC-5 algorithm is split in two main procedures: in the initial phase, a construction of structures takes place; then the real propagation part starts, and elements that do not participate in any consistent instantiation to some problem constraints are iteratively removed from their respective domain.

The original algorithm

The algorithm by [vHDT92] is split into two main steps. We describe the following version of AC-5, as proposed in [vHDT92].
1. In the first step, for any constraint \( C(x_i, x_j) \) of the given CSP, the procedure \textit{arc-cons} creates a subset \( \Delta(i) \) of \( D_i \), for each \( x_i \) of the problem; the set \( \Delta(i) \) collects all the elements \( a \in D_i \) for which no elements \( b \) exist in \( D_j \) such that \( (a, b) \in C(x_i, x_j) \). Then, for each \( a \in \Delta(i) \), all triples \( ((x_k, x_i), a) \) such that \( C(x_k, x_i) \) is a constraint of the problem are stored for future iterations, and the elements of the set \( \Delta(i) \) are deleted from \( D_i \).

2. In the second step, a triple \( ((x_i, x_j), b) \) is non-deterministically chosen and deleted from \( G \); if \( b \) has been removed from \( D_j \), then \textit{loc-arc-cons} updates the set \( \Delta(i) \subseteq D_i \) by adding all elements \( a \) that are no (more) supported in \( C(x_i, x_j) \) by any element of \( D_j \) (after \( b \) has been removed from \( D_j \)); then, for each \( a \in \Delta(i) \), all triples \( ((x_k, x_i), a) \) such that \( C(x_k, x_i) \) is a constraint of the problem are stored for future iterations, and the elements of \( \Delta(i) \) are removed from the domain \( D_i \).

As pointed out in [vHDT92], \textit{AC-5} is a generic algorithm: in fact, it can also be instantiated to \textit{AC-4} by slightly changing the definition of the sets \( \Delta(i) \). In the latter case, the functions that we used for \textit{AC-4} are adopted. In case the definition of \( \Delta(i) \) is chosen as stated item 1 and 2 above, we need a new equivalence relation and new functions to instantiate SGI to this version of \textit{AC-5}.

\textbf{Instance}

In the following, we define the main ingredients to instantiate SGI\textsc{e} to the aforementioned version of \textit{AC-5}:

- an equivalence set;
- suitable functions;
- the \textit{update} operator.

\textbf{Equivalence set.} Consider a CSP \( P := \langle X, D, C \rangle \) and define \( P_{\text{AC-5}} \) as the set of pairs \( (B, E) \), where \( B \) and \( E \) are both domain sets of the domain ordering on \( P \), see Subsection 2.5.2. Then the binary relation \( \equiv_{\text{AC-5}} \) over \( P_{\text{AC-5}} \) is defined as follows:

\[ \langle B, E \rangle \equiv \langle B', E' \rangle \text{ iff } B = B'. \]

The element \( \bot \) is set equal to \( \langle D, D \rangle \), where \( D \) is the input domain set. Indeed, the above defined relation is of equivalence.

\textbf{Functions.} The set \( F \) contains two sorts of functions that we describe as below.
4.2. Arc and Hyper-arc Consistency

The function $\theta(x_i; x_i, x_j)$ corresponds to $\text{arc-\_consistency}(x_i, x_j, \Delta_i)$: in fact, the function $\theta(x_i; x_i, x_j)$ maps $\langle B, E \rangle$ into $\langle B', E' \rangle$ so that $B$ and $B'$ and, respectively, $E$ and $E'$ differ at most in $B_i$ and $B'_i$ as follows:

\[
\begin{align*}
E'_i & := E_i - \Pi_i(C(x_i, x_j)), \\
B'_i & := B_i - E'_i.
\end{align*}
\]

The function $\theta(x_j; x_i, x_j)$ is characterised in a similar way; we leave it to the reader.

The function $\phi(x_j, b; x_i, x_j)$ corresponds to the AC-5 procedure that is called $\text{local\_arc-\_consistency}(x_i, x_j, b; \Delta_i)$. In fact, if $b \not\in B_j$, then $\phi(x_j, b; x_i, x_j)$ maps $\langle B, E \rangle$ into $\langle B', E' \rangle$ so that $B$ and $B'$ and, respectively, $E$ and $E'$ differ at most in $B_i$ and $B'_i$ as follows:

\[
\begin{align*}
E'_i & := \{ a \in B_i : P1(a, x_j, b; a; x_i, x_j) \text{ and } P2(x_j, b; a; x_i, x_j) \text{ hold } \}, \\
B'_i & := B_i - E'_i,
\end{align*}
\]

where $P1(x_j, b; a; x_i, x_j)$ and $P2(x_j, b; a; x_i, x_j)$ are, respectively,

\[
\exists d \in C(s) \ (d[i] = a \wedge d[j] = b), \quad (P1) \quad \forall d \in C(s) \cap B(s) \ d[i] \neq a, \quad (P2)
\]

Otherwise, if $b \in B_j$, the function $\phi(x_j, b; x_i, x_j)$ is the identity function. The function $\phi(x_i, a; x_i, x_j)$ is characterised in a similar manner.

The set $F$ contains all the above defined functions. The subset $F_\perp$ contains the functions of type $\theta$, which are the only $F$ functions that can modify the input value $\langle D, D \rangle$. So $F - F_\perp$ contains all the remaining functions of type $\phi$.

The update operator. The first part of the proposed version of the algorithm AC-5 is encoded in the actions of inspecting and deleting all functions like $\theta(x_i; x_i, x_j)$ or $\theta(x_j; x_i, x_j)$ from $G$ in SGIISE; when, for instance, $\theta(x_i; x_i, x_j)$ is chosen and applied, the operator update propagates the effects of the eventual reduction of $B_i$ by adding the suitable functions $\phi(x_j, b; x_i, x_j)$ to $G$. Besides we want to instantiate SGIISE to the second part of the algorithm AC-5 by means of the functions $\phi(x_j, b; x_i, x_j)$ of $F$. Therefore we define update as follows:

- if $\theta(x_i; x_i, x_j)(B, E) \neq (B, E)$, then $\text{update}(G, F, \theta(x_i; x_i, x_j), (B, E))$ is the subset of $F - G$ functions $\phi(x_i, a; x_i, x_k)$ or $\phi(x_i, a; x_k, x_i)$ such that $a \in E'_i$, the function $\theta(x_j, (x_i, x_j))$ is characterised analogously;

- if $\phi(x_j, b; x_i, x_j)(B, E) \neq (B, E)$, then $\text{update}(G, F, \phi(x_j, b; x_i, x_j), (B, E))$ is the subset of $F - G$ functions $\phi(x_i, a; x_i, x_k)$ or $\phi(x_i, a; x_k, x_i)$ of $F - G$ such that $a \in E'_i$; otherwise it is the empty set. An analogous characterisation can be given for $\phi(x_i, a; x_j, x_j)$. 
Now it is trivial to check that AC-5 becomes an instance of SGIISE by means of the above $\theta$ and $\phi$ functions.

**FACT 4.2.13.** The AC-5 algorithm is an instance of SGIISE. \qed

**Analysis**

It is as well easy to check that SGIISE with the above $\theta$ and $\phi$ functions is $\equiv_{AC-5}$ equivalent to SGIS with the functions $\Theta$ and $\Phi$ for AC-4, as described in Subsection 4.2.3.

**Lemma 4.2.14.** The SGIISE algorithm with the above $\theta$ and $\phi$ functions is $\equiv_{AC-5}$-equivalent to the SGIS algorithm with the $\Theta$ and $\Phi$ functions in Subsection 4.2.3, for binary constraints, and defined on the 2 domain ordering on $P$. \qed

We have now all we need to prove the following results concerning AC-5.

**Corollary 4.2.15 (AC-5).** Given a finite CSP $P$, the AC-5 algorithm always terminates by computing the greatest arc consistent problem equivalent to $P$; that is, the least common fixpoint of the functions of type $\Theta$ and $\Phi$ defined as above.

**Proof.** Our thesis follows from Lemma 4.2.9 and Theorem 3.4.10, concerning the $\Theta$ and $\Psi$ functions, via Lemma 4.2.14 and Corollary 3.4.16. \qed

**A hyper-arc consistency version of AC-5**

By exploiting the generality of SGIISE, we can extend AC-5 to an algorithm schema that enforces hyper-arc consistency like AC-5 enforces arc consistency. Indeed, it is sufficient to recast the above functions $\theta$ and $\sigma$ for AC-5 as follows.

(\(\theta\)). The function $\theta(x_i; s)$, where $x_i \in s$ and $C(s)$ is a constraint of the input problem, maps $\langle B, E \rangle$ into $\langle B', E' \rangle$ so that $B$ and $B'$, and respectively $E$ and $E'$ differ at most in their $i$-th components as follows:

\[
\begin{bmatrix}
E'_i := E_i - \Pi_i(C(s)), \\
B'_i := B_i - E'_i.
\end{bmatrix}
\]

(\(\phi\)). The function $\phi(x_j, b; s)$ maps $\langle B, E \rangle$ into $\langle B', E' \rangle$ so that $B$ and $B'$, and respectively $E$ and $E'$ differ at most in their $i$-th components as follows, for every $x_i \in s$ different from $x_j$:

\[
\begin{bmatrix}
E'_i := \{ a \in B_i : P1(x_j, b; a; x_i, x_j) \text{ and } P2(x_j, b; a; x_i, x_j) \text{ hold } \}, \\
B'_i := B_i - E'_i.
\end{bmatrix}
\]
4.3 Path Consistency

where $P1(x_j, b; a; x_i, x_j)$ and $P2(x_j, b; a; x_i, x_j)$ are, respectively,

$$\exists d \in C(s) \ (d[i] = a \land d[j] = b), \quad (P1) \quad \forall d' \in B(s) \cap C(s) \ d'[j] \neq b, \quad (P2)$$

Otherwise, i.e. if $b \in B_j$, the function $\phi(x_j, b; s)$ behaves like the identity function.

As for AC-5, we can derive the following result by Theorem 3.4.10, via Lemma 4.2.9, and Corollary 3.4.16.

**Corollary 4.2.16 (HAC-5).** Given a finite CSP $P$, the SGIIIS algorithm with the above defined functions $\theta$ and $\phi$ always terminates, by computing the greatest hyper-arc consistent problem equivalent to $P$. $\square$

### 4.3 Path Consistency

The notion of path consistency was introduced in [Mon74]. It is defined for a special type of CSPs. For simplicity we limit ourselves to binary CSPs: i.e., their constraints are only binary.

In Subsection 2.5.1, we introduced the join operation on constraints. In case of binary relations like $R \subseteq D_i \times D_j$ and $S \subseteq D_j \times D_k$, the composition of $R$ and $S$, which is defined as follows

$$R \cdot S := \{(a, b) : (a, c) \in R \land (c, b) \in S\},$$

amounts to a sequential application of join and projection to $R$ and $S$. Note that, if $C(x_i, x_j)$ is a constraint on the variables $x_i$ and $x_j$, and $C(x_j, x_k)$ is a constraint on the variables $x_j$ and $x_k$, then $\Pi_{x_i, x_j}(C(x_i, x_j) \cdot C(x_j, x_k))$ is a constraint on the variables $x_i$ and $x_k$. Whereas, if $k < j$, then the composition of $C(x_i, x_j)$ with $C(x_k, x_j)$ is not defined; yet, their join is. This is due to the commutativity of join, as defined in Subsection 2.5.1.

We first introduce the standard notion of path consistency, and then see how we can recast it through the join operation. In the following definition, instead of schemes, we have sets of variables.

**Definition 4.3.1.** We call a CSP $P := \langle X, D, C \rangle$ path consistent if its 2 completion $\bar{P}^2$ enjoys the following property: for each set of distinct variables $\{x_i, x_j, x_k\}$ of $P$, we have

$$C(x_i, x_k) = C(x_i, x_k) \cap (C(x_i, x_j) \cdot C(x_j, x_k)).$$

In other words, a CSP is path consistent if, for each triple of its variables, $x_i$, $x_j$ and $x_k$, the following holds: if $(a, c) \in C(x_i, x_k)$, then there exists $b \in D_j$ such that $(a, b) \in C(x_i, x_j)$ and $(b, c) \in C(x_j, x_k)$. 
We provide an alternative characterisation of path consistency. In fact, in the above definition, relations of the form \( C(x, x') \) are used, for any subset \( \{x, x'\} \) of the considered sequence of variables. If \( \{x, x'\} \) is not a scheme of the given CSP scheme of variables, then \( C(x, x') \) is a supplementary relation that is not a constraint of the original CSP. At the expense of some redundancy we can rewrite the above definition so that only the constraints of the considered CSP are involved. This is the contents of the following characterisation, whose proof follows from the definition of join, projection, intersection and composition.

**FACT 4.3.2 (ALTERNATIVE PATH CONSISTENCY).** A 2 complete CSP is path consistent iff the three following relations hold all true:

\[
\begin{align*}
C(x_i, x_j) & := C(x_i, x_j) \cap \Pi_{x_i, x_j}(C(x_i, x_j) \bowtie C(x_j, x_k)), \\
C(x_i, x_k) & := C(x_i, x_k) \cap \Pi_{x_i, x_k}(C(x_i, x_k) \bowtie C(x_j, x_k)), \\
C(x_j, x_k) & := C(x_j, x_k) \cap \Pi_{x_j, x_k}(C(x_i, x_j) \bowtie C(x_i, x_k)),
\end{align*}
\]

for each scheme \( x_i, x_j, x_k \) of the CSP variable.

**EXAMPLE 4.3.3.** The Temporal CSP in Subsection 2.3.4 is not path consistent: in fact, the relation *follows* in \( C(x_1, x_4) \) is not consistent with \( \Pi_{x_2, x_4}(C(x_1, x_2) \bowtie C(x_2, x_4)) \). Thereby, path consistency algorithms will remove *follow*, and so reduce \( C(x_1, x_4) \) to the singleton relation *precedes*.

### 4.3.1 The PC-1 Algorithm

The PC-1 is the basic algorithm for path consistency; it is presented in Appendix A. In the present subsection, we show that PC-1 is an instance of GI, and then we analyse it through GI iterations.

**Instantiation**

To instantiate the GII algorithm to PC-1 we need to specify the following components:

- a partial ordering,
- finitely many function on this,
- and the *update* operator.

We do it as below.

**Partial ordering.** To study path consistency, given a 2 complete CSP \( P := (X, D, C) \), we consider the 2-constraint ordering defined in Subsection 2.5.2.
4.3. Path Consistency

Functions. Next, given a scheme \( x_i, x_j, x_k \) of the variables of \( P \) we introduce three functions on the partial ordering on \( P \) as below. Denote an element of the partial ordering with \( B \).

- The function \( \sigma(x_i, x_k; x_j) \) only modifies the binary constraint \( B(x_i, x_k) \) of \( B \), by returning the following set \( B'(x_i, x_k) \):
  \[
  B'(x_i, x_k) := B(x_i, x_k) \cap \Pi_{x_i, x_k}(B(x_i, x_j) \bowtie B(x_j, x_k)).
  \]

- The function \( \sigma(x_i, x_j; x_k) \) only modifies the binary constraint \( B(x_i, x_j) \) of \( B \), by returning the following set \( B'(x_i, x_j) \):
  \[
  B'(x_i, x_j) := B(x_i, x_j) \cap \Pi_{x_i, x_j}(B(x_i, x_k) \bowtie B(x_j, x_k)).
  \]

- The function \( \sigma(x_j, x_k; x_i) \) only modifies the binary constraint \( B(x_j, x_k) \) of \( B \), by returning the following set \( B'(x_j, x_k) \):
  \[
  B'(x_j, x_k) := B(x_j, x_k) \cap \Pi_{x_j, x_k}(B(x_i, x_j) \bowtie B(x_i, x_k)).
  \]

In what follows, when using a function \( \sigma(x_i, x_k; x_j) \), we implicitly assume that the variables \( x_i, x_j, x_k \) are pairwise different and that \( j < k \).

Finally, the notion of path consistency is clearly related to the common fixed-points of the above defined functions, and these are idempotent, monotone and inflationary over the constraint ordering. We collect these properties as in the following lemma, whose proof is just a consequence of the given characterisation of the functions \( \sigma \) as above.

**Lemma 4.3.4.**

(i). A CSP is path consistent if it a common fixpoint of the functions \( \sigma(x_i, x_k; x_j) \) defined as above.

(ii). The functions \( \sigma(x_i, x_k; x_j) \) are idempotent, monotone and inflationary over the 2-constraint ordering of the given CSP.

(iii). The functions \( \sigma(x_i, x_k; x_j) \) do not remove solutions from the given CSP.

The update operator. The update operator is specified as follows:

\[
\text{update}(\sigma(x_i, x_k; x_j), F, G, B) := \{ \sigma(x_i, x_m; x_n) : |\{x_i, x_k\} \cap \{x_l, x_m, x_n\}| \geq 2 \}.
\]

In other words: each time a function \( \sigma(x_i, x_k; x_j) \) modifies \( B \), all functions that involve at least two of the variables \( x_i \) and \( x_k \) are added to \( G \). Indeed, this is not an optimal instantiation of update. We shall see how it can be optimised by resorting to commutativity again, as in the case of the AC-3 algorithm in Subsection 4.2.2.
Analysis

Given the above lemma, it is now easy to prove the following result, as a consequence of Corollary 3.3.12: it is sufficient to proceed as for HAC-1 and AC-1. The reader is invited to consult [Apt99a, Apt00a] for a more detailed analysis.

**Corollary 4.3.5 (PC-1).** Consider a 2 complete CSP $P := \langle X, D, C \rangle$, such that all constraints in $C$ are finite. Let $P$ be the input of PC-1. Then every execution of PC-1 terminates, by computing the greatest path consistent problem, equivalent to $P$; i.e., the least common fixpoint of all the functions $\sigma(x_i, x_k; x_j)$ defined above.

### 4.3.2 The PC-2 Algorithm

In Section 4.2, we illustrated how the AC-3 constitutes an improvement of AC-1, by a clever instantiation of the update operator. The PC-2 algorithm is an improvement of PC-1 much in the same spirit as AC-3 is of AC-1.

In the PC-1 algorithm, each time a function $\sigma(x_i, x_k; x_j)$ is applied and modifies its arguments, all functions associated with a triplet of variables including $x_i$ and $x_k$ are added to the set $G$ of functions to iterate. This is not, indeed, an optimal choice of update.

In [Apt00a], the author proves how fewer functions can be added via update, by taking into account commutativity. To this end, the following lemma is proved. Its proof is as for the case of AC-3, thus we invite the reader to consult the latter or [ib].

**Lemma 4.3.6.** Consider a 2 complete CSP, involving among others the variables $x_i, x_j, x_k$ and $x_l$. Then the functions $\sigma(x_i, x_k; x_j)$ and $\sigma(x_i, x_k; x_l)$ commute. □

In other words, each pair of functions of the form $\sigma(x_i, x_k; \ldots)$ commute; this for every variable scheme $(x_i, x_k)$ of the problem. The functional $\text{Comm}(\sigma(x_i, x_k; x_j))$ is then defined as follows, for each variable $x_j$ such that $j \neq i, \neq k$ — consult also Definition 3.4.5:

$$\text{Comm}(\sigma(x_i, x_k; x_j), F) = \{\sigma(x_i, x_k; x_l) : x_l \text{ is different from } x_i, x_k\}.$$  

Thus, for each function of type $\sigma$, the set $\text{Comm}(\sigma, F)$ contains precisely $n - 3$ elements, where $n$ is the number of variables of the considered CSP. This quantifies the maximal gain obtained by using the commutativity information, loosely speaking: more precisely, update will need less functions at each iteration of the instance of GIIC for PC-2, than in the correlated instance of GI for PC-1.

By virtue of the above lemma and Theorem 3.4.7, the following result is easily proved.
4.3. Path Consistency

Corollary 4.3.7 (PC-2). Consider a 2 complete CSP \( P := (X, D, C) \), such that all constraints in \( C \) are finite. Let \( P \) be the input of PC-2. Then every execution of PC-2 terminates, by computing the greatest path consistency problem, equivalent to \( P \); i.e., the least common fixpoint of all the functions \( \sigma(x_i, x_k; x_j) \) defined above.

4.3.3 The PC-4 Algorithm

The PC-3 algorithm was devised in [MH86]. However here we refer to its corrected version, named PC-4, presented in [HL88]. This algorithm enforces path consistency on binary CSPs, by exploiting additional structures in the same fashion as (H)AC-4 and AC-5. So, in the following, we shall restrict our attention to binary CSPs, and prove that PC-4 is an instance of the SGIiSE algorithm schema.

Notice that we assume that the input problem \( P \) is 2 complete — see Subsection 2.4.2. This will help us to reduce the overload of notations, and it is a minor change, since the initialisation phase of PC-4 reduces the input problem to its 2 completion. Therefore, constraint propagation takes place on a 2 complete problem.

The algorithm

The PC-4 algorithm is split into two parts:

- the first part of the algorithm consists in an initialisation of structures such that, at the end of it, the following properties hold: if \( E \) is the current constraint set and \( C \) the input one, then

  1. \((x_k : d, x_l : e) \in G \iff (x_k : d, x_l : e) \in C(x_k, x_l) - E(x_k, x_l), \)
  2. each \( C(x_k : d, x_j : c; x_l) \) is a subset of \( D_l \), and \( e \in C(x_k : d, x_j : c; x_l) \) iff \((d, e) \in C(x_k, x_l) \) and \((e, c) \in C(x_l, x_j)\);

- in the second part, the real propagation phase takes place, see Algorithm A.5 in Appendix A. In fact, every tuple \((x_k : d, x_l : e) \in G \) is chosen and removed from \( G \), each only once; then the pairs, affected by the removal of \((d, e) \) from \( C(x_k, x_l) \), are all inspected. So, if one of them has no more supporting pairs in \( E(x_k, x_l) \), it gets removed from its corresponding binary constraint; then it is added to \( G \) in order to propagate the effect of its removal.

We have slightly changed PC-4 (see Algorithm A.5 in Appendix A) for this makes it easier to instantiate SGIiSE to PC-4. Notice that this new version is equivalent to the original one by [HL88] and retains its worst time and space complexity; the difference is in the use of structures of the form \( C(x_k : d, x_j : c; x_l) \) instead of so-called counters.
Chapter 4. Constraint Propagation Algorithms

Instantiation

Equivalence set. We can assume that the input CSP $P := \langle X, D, C \rangle$ is 2 complete, see Subsection 2.4.2; else we add the necessary constraints to it.

While arc consistency algorithms remove elements from domains, path consistency algorithms such as PC-1 and PC-2 (see Subsections 4.3.1 and 4.3.2, respectively) propagate constraints by modifying binary constraints; so does PC-4. However, contrary to PC-2 and PC-1, the PC-4 algorithm has also support structures, like AC-4, AC-5 and their respective hyper-arc versions. Those structures are used to avoid checking, more than once, that a given pair is consistent with the input constraints in $P := \langle X, D, C \rangle$.

In order to define the equivalence set, for instantiating SGIIIE to PC-4, we introduce supplementary structures as follows. For each scheme $\langle x_i, x_j \rangle$ of the problem, variable $x_k$ such that $i, j \neq k$, we put, for each $a \in D_i$, $b \in D_j$, and $c \in D_k$:

\[
E(x_i : a, x_k : c; x_j) := \begin{cases} 
E_j(i, a, k, c, j) & \text{if } i < k \text{ and } E_j \subseteq D_j, \quad (IK) \\
E_k(j, k, i, a, j) & \text{if } k < i \text{ and } E_k \subseteq D_k, \quad (KI) \\
E_i(k, c, j, b) & \text{if } k < j \text{ and } E_i \subseteq D_i, \quad (KJ) \\
E_i(j, b, k, c, i) & \text{if } j < k \text{ and } E_i \subseteq D_i. \quad (JK)
\end{cases}
\]

Let $E$ denote the set that contains all such structures: precisely, either (IK) if $i < k$ or (KI) otherwise, and (KJ) if $k < j$ or (JK) otherwise; this for each scheme $\langle x_i, x_j \rangle$, $x_k$ such that $k \neq i$, $a \in D_i$, $b \in D_j$ and $c \in D_k$. Thus $C_{PC-4}$ collects all sets like $E$, whose elements are structures as above.

Finally, we can define the equivalence set $P_{HAC-4}$, that contains all pairs $\langle B, E \rangle$ defined as follows:

- $B$ belongs to the 2-constraint ordering on the problem $P$;
- $E$ belongs to $C_{PC-4}$, i.e., is a collection of structures as above.

The equivalence relation $\equiv_{HAC-4}$ we need to define is, clearly, the following one: $\langle B, E \rangle \equiv \langle B', E' \rangle$ iff $B = B'$.

Functions. As in the cases of (H)AC-4 and (H)AC-5, there are two types of functions: the former, denoted by $\theta$, is used in the first while loop of SGIIIE; the latter, denoted by $\phi$, is used in the second while loop of SGIIIE. We describe them as below.

The $\theta$ functions. For each $(a, c) \in C(x_i, x_k)$ and $j = 1, \ldots, n$ different from $i$ and $k$, we define a function $\theta(x_i : a, x_k : c; x_j)$ that is the identity everywhere except, possibly, on the following sets:
4.3. Path Consistency

- the subset $E(x_i : a, x_k : c; x_j)$ of $D_j$ is mapped into the subset $E'(x_i : a, x_k : c; x_j)$ of $D_j$, such that $b \in E'(x_i : a, x_k : c; x_j)$ if both the following properties hold:

\[
\begin{align*}
(a, b) & \in C(x_i, x_j) & \text{if } i < j, \text{ otherwise } (b, a) & \in C(x_j, x_i), \\
(b, c) & \in C(x_j, x_k) & \text{if } j < k, \text{ otherwise } (c, b) & \in C(x_j, x_k);
\end{align*}
\]

- the subset $B(x_i, x_k)$ of the input $C(x_i, x_k)$ is mapped into its subset $B'(x_i, x_k)$ so that

\[
B'(x_i, x_k) := \begin{cases} 
B(x_i, x_k) - \{(a, c)\} & \text{if } E'(x_i : a, x_k : c; x_j) = \emptyset, \\
B(x_i, x_k) & \text{else.}
\end{cases}
\]

The $\phi$ functions. For each $(a, c) \in C(x_i, x_k)$ and $j = 1, \ldots, n$ different from $i$ and $k$, we define a function $\phi(x_i : a, x_k : c; x_j)$ that is the identity if $(a, c) \in B(x_i, x_k)$; else, it is the identity almost everywhere except, possibly, on the following sets — where $b$ ranges over $D_j$:

- If $i < j$ each subset $E'(x_i : a, x_j : b; x_k)$ of $D_k$ is mapped into its subset

\[
E'(x_i : a, x_j : b; x_k) := E(x_i : a, x_j : b; x_k) - \{c\},
\]

else each $E'(x_j : b, x_i : a; x_k)$ of $D_k$ is mapped into its subset

\[
E'(x_j : b, x_i : a; x_k) := E(x_j : b, x_i : a; x_k) - \{c\}.
\]

Similarly, if $j < k$ each $E'(x_j : b, x_k : c; x_i)$ is mapped into

\[
E'(x_j : b, x_k : c; x_i) := E(x_j : b, x_k : c; x_i) - \{a\},
\]

else each $E'(x_k : c, x_j : b; x_i)$ is mapped into

\[
E'(x_k : c, x_j : b; x_i) := E(x_k : c, x_j : b; x_i) - \{a\}.
\]

- Then the set $B(x_i, x_j)$ is mapped into

\[
B'(x_i, x_j) := \begin{cases} 
B(x_i, x_j) - \{(a, b)\} & \text{if } E'(x_i : a, x_j : b; x_k) = \emptyset, \\
B(x_i, x_j) & \text{else,}
\end{cases}
\]

if $i < j$, else the set $B(x_j, x_i)$ is mapped into

\[
B'(x_j, x_i) := \begin{cases} 
B(x_j, x_i) - \{(b, a)\} & \text{if } E'(x_j : b, x_i : a; x_k) = \emptyset, \\
B(x_j, x_i) & \text{else.}
\end{cases}
\]
Similarly, the set $B(x_j, x_k)$ is mapped into
\[
B'(x_j, x_k) := \begin{cases} 
B(x_j, x_k) - \{(b, c)\} & \text{if } E'(x_j : b, x_k : c; x_i) = \emptyset, \\
B(x_j, x_k) & \text{else.}
\end{cases}
\]

if $j < k$, else the set $B(x_k, x_j)$ is mapped into
\[
B'(x_k, x_j) := \begin{cases} 
B(x_k, x_j) - \{(c, b)\} & \text{if } E'(x_k : c, x_j : b; x_i) = \emptyset, \\
B(x_k, x_j) & \text{else.}
\end{cases}
\]

The update operator Given the above functions of type $\theta$ and $\phi$, we can define the update operator as follows:

(\theta). if $\Pi_1 \theta(x_i : a, x_k : c; x_j)(B, E) \neq B$, then update adds to $G$ all the $\phi$ functions of the following form: if $i < j$, all functions $\phi(x_i : a, x_k : c; x_l)$ for $(a, b) \in E(x_i, x_j) - E'(x_i, x_j)$ and $l \neq i, j, k$; else all functions $\phi(x_j : b, x_i : a; x_k)$ for $(b, a) \in E(x_j, x_i) - E'(x_j, x_i)$ and $l \neq i, j, k$; if $j < k$, all functions $\phi(x_j : b, x_k : c; x_l)$ for $(b, c) \in E(x_j, x_k) - E'(x_j, x_k)$ and $l \neq i, j, k$; else all functions $\phi(x_j : b, x_l : c; x_i)$ for $(b, a) \in E(x_j, x_i) - E'(x_j, x_i)$ and $l \neq i, j, k$.

Now it is easy to check that the following statement holds.

**FACT 4.3.8.** The PC-4 algorithm is an instance of SGIISE whenever this algorithm iterates the above defined functions of type $\theta$ and $\psi$.

**Analysis**

As in the case of (H)AC-4 and (H)AC-5, we need to define functions on the quotient set $P_{\equiv_{\text{HAC-4}}} / \equiv_{\text{HAC-4}}$ to study PC-4 by means of SGIIS. Hence we define functions of two types, $\Theta$ and $\Phi$, as below.

In order to obtain a more compact notation, we introduce the following shorthand in the remaining of this subsection.

**CONVENTION 4.3.1.** For every pair of distinct $i, j = 1, \ldots, n$, let $\overrightarrow{B(x_i, x_j)}$ denote $B(x_i, x_j)$ if $i < j$, else $B(x_j, x_i)$.

Thus $\overrightarrow{B(x_i, x_j)} = \overrightarrow{B(x_j, x_i)}$, and this will simplify the presentation of the functions for SGIIS as below.

The $\Theta$ functions. For each pair $(a, c) \in C(x_i, x_k)$ in the 2 completion of $P$, and $j = 1, \ldots, n$ different from $i$ and $k$, we define a function $\theta(x_i : a, x_k : c; x_j)$
4.3. Path Consistency

that is the identity if \((a, c)\) belongs to \(\Pi_{x_i, x_k}(\overline{C(x_i, x_j)} \bowtie \overline{C(x_j, x_k)})\); else it is the identity almost everywhere except, possibly, on the set \(B(x_i, x_k)\) that is mapped into \(B'(x_i, x_k)\), defined as follows:

\[
B'(x_i, x_k) := B(x_i, x_k) - \{(a, c)\}.
\]

The \(\Phi\) functions. For each pair \((a, c)\) \(\in C(x_i, x_k)\) in the 2 completion of \(P\), and \(j \neq i, k\), we define a function \(\Phi(x_i : a, x_k : c ; x_j)\) that is the identity if \((a, c) \not\in B(x_i, x_k)\); else it is the identity almost everywhere except, possibly, on the constraint on the variables \(x_i\) and \(x_j\), and on the constraint on the variables \(x_j\) and \(x_k\):

\[
\begin{align*}
B'(x_i, x_j) &:= B(x_i, x_j) \cap \Pi_{x_i, x_j}(B(x_i, x_k) \bowtie_{k \neq c} B(x_k, x_j)) & \text{if } i < j, \\
B'(x_j, x_i) &:= B(x_j, x_i) \cap \Pi_{x_j, x_i}(B(x_i, x_k) \bowtie_{k \neq c} B(x_k, x_j)) & \text{else},
\end{align*}
\]

where \(\bowtie_{k \neq c}\) results from the composition of first \(\bowtie\) and then \(sel_{k \neq c}\):

\[
\begin{align*}
B'(x_j, x_k) &:= B'(x_j, x_k) \cap \Pi_{x_j, x_k}(B(x_j, x_k) \bowtie_{i \neq a} B(x_i, x_k)) & \text{if } j < k, \\
B'(x_k, x_j) &:= B'(x_k, x_j) \cap \Pi_{x_k, x_j}(B(x_k, x_j) \bowtie_{i \neq a} B(x_i, x_k)) & \text{else},
\end{align*}
\]

where \(\bowtie_{i \neq a}\) results from the composition of first \(\bowtie\) and then \(sel_{i \neq a}\).

Consider now the 2-constraint ordering on \(P\), see Subsection 2.5.2. We summarise the main properties of the above functions \(\Theta\) and \(\Sigma\) as in the following lemma.

**Lemma 4.3.9.** The above defined \(\Theta\) and \(\Phi\) are monotone, stationary and inflationary on the 2-constraint order on the input problem; besides, if a \(\Theta\) function is the identity on the input problem, then it is the identity function. \(\square\)

Given Lemma 4.3.9 we can infer the following result as a corollary of Theorem 3.4.10.

**Corollary 4.3.10.** Every execution of \(\text{SGIS}\), with input a finite CSP \(P\) and the above defined functions of type \(\Theta\) and \(\Phi\), always terminates by computing the greatest path consistent problem that is equivalent to \(P\). \(\square\)

The following lemma can be proved by an argument similar to that used in Lemma 4.2.11 for \(\text{AC-4}\).

**Lemma 4.3.11.** Every \(\text{SGISE}\) execution with \(\theta\) and \(\phi\) functions is \(\equiv_{\text{PC-4}}\)-equivalent to an \(\text{SGIS}\) iteration with \(\Theta\) and \(\Phi\) functions. \(\square\)
Now we have all the technical results for proving the following statement.

**Corollary 4.3.12 (PC-4).** The PC-4 algorithm, with input a finite CSP \( P \), always terminates by computing the least common fixpoint of the above defined \( \Theta \) and \( \Phi \) functions, that is the greatest path consistent problem, equivalent to the input problem \( P \).

**Proof.** The claim follows by Lemma 4.3.11, Corollaries 3.4.16 and 4.3.10. \( \square \)

**Note 4.3.13.** Note the difference in the termination conditions for PC-1 or PC-2 versus PC-4. In the former two cases, we only need that the constraints are finite to ensure termination; hence, those algorithms can be applied and terminate in the case of the Temporal CSP in Subsection 2.3.4. However, this is not the case for PC-4: in fact the termination condition in Corollary 4.3.12 assumes that also domains are finite. The reason is easy to explain via the functions used for PC-4 in SGIIISE or SGIIIS: these functions are parameterised by domain elements, hence we have a finite number of \( F \) functions only if the input CSP domains are finite.

### 4.4 Local Consistency

#### 4.4.1 Local Consistency as \( k \) Consistency

In Section 4.2 and 4.3, we defined two properties of CSPs that gave rise to a number of constraint propagation algorithms: arc and path consistency. Freuder generalised both those properties to a general form of *local consistency* in [Fre78]. There are two versions of this notion, a weak and a strong one, defined as below.

**Weak consistency**

Consider a CSP \( P := (X, D, C) \) with \( n \) variables in \( X \), and a scheme \( s \) of \( X \) of length \( k \). An assignment \( d \) for \( s \) is \( k \) consistent iff it satisfies every constraint \( C(s') \) of \( P \) over a scheme \( s' \) of \( s \). So, different levels, hence notions of local consistency can be defined for the same problem \( P := (X, D, C) \):

- the problem \( P \) is 1 consistent, or node consistent, iff, for every variable \( x_i \) of \( P \), the unary constraint \( C(x_i) \) on \( x_i \) is contained in the domain \( D_i \) of \( x_i \);
- given \( 1 < k \leq |X| \), \( P \) is \( k \) consistent iff every \((k - 1)\) consistent assignment \( d \) for \( P \) on \( s \) can be extended to a \( k \) consistent instantiation, for every possible extension \( s' = s \cup x_j \) of \( s \).

Suppose that \( k \) is equal to 2. Then arc consistency on binary CSPs, that are node consistent, clearly coincide with 2 consistency. The same holds for path consistency: a binary CSP, node consistent, is path consistent iff it is 3 consistent. We summarise these properties as follows.
**Fact 4.4.1.**

(i) A binary CSP, 1 consistent, is arc consistent iff it is 2 consistent.

(ii) A binary CSP, 1 consistent and 2 complete, is path consistent iff it is 3 consistent.

**Strong local consistency**

A stronger notion of local consistency can be defined as follows: a problem $P := (X, D, C)$ is strongly $k$-consistent, for $1 \leq k < |X|$, iff it is $j$-consistent for every $j \leq k$.

Notice that requiring a problem to be strongly $k$ consistent is by far more demanding than requiring it to be $k$ consistent. A problem can be trivially $k+1$ consistent, if there are not $k$ consistent instantiations, which is not the case for strong $k$ consistency. Consider the following example.

**Example 4.4.2.** Let $P$ be the CSP on two variables, $x_1$ and $x_2$, and $x_3$, domains equal to $\{0, 1\}$, and constraints on each scheme of two variables, forbidding that these are equal (see Example 2.5.4). This problem is clearly arc consistent, hence 2 consistent: in fact, for each variable, there is an instantiation for it that satisfies all binary constraints of $P$. However, $P$ is not strongly 2 consistent, nor consistent.

As the above toy example suggests, if a problem is “sufficiently” strong consistent, then it is also consistent. We make the above claim precise as follows. The proof is easy, and the requirement that at least one domain should not be empty is fundamental to guarantee the existence of a 1 consistent instantiation; see also [Fre78].

**Fact 4.4.3.** Consider a CSP $P$ with $k \geq 1$ variables. If $P$ is strongly $k$ consistent, with at least one not-empty domain, then $P$ is consistent.

Thus, a strongly $n$ consistent CSP is globally consistent, i.e., any consistent instantiation of a scheme of the variables can be extended to a consistent instantiation of all of the variables without backtracking.

**4.4.2 The KS Algorithm**

The KS algorithm by Cooper [Coo89] is an optimisation of the synthesis algorithm by Freuder [Fre78]. Both algorithms enforce strong $k$ consistency over a CSP; the former can enforce $k$ consistency with a minor simplification. We shall account for the whole strong $k$ consistency algorithm by Cooper here, and explain how the GIISE schema can be instantiated to it.

In the following part, we need to extend a scheme or remove a variable from it, i.e. to make use of both projection and join, see Subsection 2.5.1. Hereby, we
remind the following abbreviations: if $s$ is a scheme of variables and $x_j \notin s$, then $t := s \cup x_j$ will denote the scheme on the set of variables of $s$ plus $x_j$. Similarly, if $x_i$ is one of the variables in $s$, then $r := s - \{x_i\}$ will stand in for the scheme on the variables of $s$ minus $x_i$. Finally, if $d \in C(s)$, the variable $x_j$ does not occur in $s$ and $a \in D_j$, then $e := d \uplus a$ is the tuple of $D[s \cup \{x_j\}]$ such that $e[s] = d$ and $e[j] = a$.

The original algorithm

As in the case of PC-4, also the Cooper algorithm enforces propagation at the level of constraints by exploiting additional structures for efficiency reasons; namely to store already checked values.

The algorithm by Cooper is split in two main sub-programs: the initialisation process takes place in the first step; then propagation is achieved by iteratively pruning $i$ inconsistent values, for all $i \leq k$.

However, the first sub-program of KS is only meant to construct structures, there is no pruning of values. There, the KS algorithm reduces the input problem $P$ to a $k$ strong complete one, for some $k$ not greater than the number of variables in $P$; so to an equivalent problem that has precisely one constraint per scheme, the length of which is not greater than $k$, see also Subsection 2.4.2. In Appendix A, we present the initialisation and propagation phases in Algorithm A.1; these are slightly modified versions of the original, in which counters (e.g. $Counter[d,r,j]$) are used to store the number of support values in place of the support values themselves, as we instead do (e.g. via $C(d,r,j)$). In the second sub-program, a pruned tuple $d$ is chosen and the effects of its removal from $C(t)$ are propagated in two stages. Let $i$ be the length of $t$. If $i < k$, first all $(i + 1)$ consistent instantiations $d'$ such that $d'[t] = d$ are considered; then, if $i > 1$, all $(i - 1)$ consistent instantiations $d'$ on schemes $s$ of $t$ are checked for supports if $d' = d[s]$.

Therefore we shall overlook the initialisation phase of KS when defining functions to instantiate SGIISE. Nevertheless, we are able to devise functions for SGIE that account for the optimal behaviour of the propagation phase of KS, compared to the synthesis algorithm by Freuder; in fact, those functions are defined on a set that exploits further structures than constraint orderings. Hence we shall define an appropriate equivalence relation on the domain of those functions, and show that we can devise a constraint ordering on the quotient set. Then we shall study this instance of SGIISE by passing to the quotient set and functions on this for SGIS.

Instantiation

The equivalence structure. Consider a CSP $P := \langle X, D, C \rangle$ and assume that it is strongly $k$ complete. Remind, from Subsection 2.4.2, that the process of strong $k$ completion amounts to constructing an equivalent problem that has
4.4. Local Consistency

the same variable scheme and domain set as the original one, but precisely one
constraint over each scheme.

Now, let $P_{ks}$ be the class of pairs $\langle B, E \rangle$ in which $B$ is a constraint set
in the strong $k$-constraint ordering on $P$ (see Subsection 2.5.2), and $E$ is a family
of structures, defined as follows: for each scheme $s$ on $X$, $x_j \not\in s$ and $d \in C(s)$,
precisely one structure of the form

$$E(d, s, x_j) := \langle E_j, \langle d, s, x_j \rangle \rangle,$$

for $E_j$ a subset of $D_j$, belongs to $E$.

The binary relation $\equiv_{ks}$ on $P_{ks}$ is defined as follows:

$$\langle B, E \rangle \equiv_{ks} \langle B', E' \rangle \iff B = B'.$$

Indeed, $\equiv_{ks}$ is an equivalence relation and the quotient set $P_{ks}/\equiv_{ks}$ is isomorphic
to the universe $O_k$ of the strong $k$-constraint ordering on $P$.

Functions. For each scheme $s$ of $X$ of length $i$ and $d \in D[s]$, we define a
function $\phi(d, s, i)$ that propagates the $i$ inconsistent instantiation $d$ to all the
$i - 1$ and $i + 1$ instantiations. Namely, if $\langle B, E \rangle$ is the input to $\phi(d, s, i)$, then the
output value $\langle B', E' \rangle$ differs from $\langle B, E \rangle$ at most in the following components if
$d \not\in B(s)$, else $\phi(d, s, i)$ is the identity everywhere:

- in case $i$, the length of $s$, is less than $k$, then $\phi(d, s, i)$ considers every $x_k \not\in s$
and the resulting join scheme $t := s \cup \{x_k\}$, and maps each such $B(t)$ into

$$B'(t) := B(t) - \bigcup_{a \in D_k} \{d \upharpoonright a\};$$

- if $i$, the length of $s$, is greater than 1, then $\phi(d, s, i)$ considers every $x_j \in s$,
the resulting projection scheme $r := s - \{x_j\}$, and modifies the sets $E(r, e, j)$,
for each $e \in D(r)$, and $B(r)$ as follows:

$$\begin{bmatrix}
E'(r, e, j) & := & E(r, e, j) - \{d[j]\}, \\
B'(r) & := & B(r) - \{e : E'(r, e, j) = \emptyset\}.
\end{bmatrix}$$

The update operator. At this point, we are left to characterise the update
operator: if $\Pi_1 \phi(e, s, i)(B) \neq B$, then update adds to $G$ all the remaining $F$
functions $\phi(e, r, i - 1)$ or $\phi(e', t, i + 1)$ such that $e \in B(r) - B'(r)$ and $e' \in
B(t) - B'(t)$.

Given the above definitions of the functions $\phi$ and update, we can state the
following result as fact.

Fact 4.4.4. The propagation phase of Algorithm A.1 in Appendix A is an in-
stance of SGIISE: $\bot$ is the input $P_{ks}$; the set $F$ collects all the $\phi$ functions
as defined above; the set $F_\bot$ collects all the $\phi(d, s, i)$ functions such that $d \in
D[s] - C(s)$; the update operator is instantiated as above.
Analysis

Given the CSP $P$, we have to define functions, say $\Phi$, over the strong $k$-constraint ordering on $P$. If we let $O_k$ denote the universe of such constraint ordering on $P$, then, clearly, $O_k$ is isomorphic to $P_{ks}/\equiv_{ks}$. Moreover, we want each GIIS iteration with the functions $\phi$ over $P_{ks}$ to be $\equiv_{ks}$-equivalent to an GIIS iteration with the functions $\Phi$ over $O_k$. Therefore, we define such functions $\Phi$ as below.

Functions over $O_k$. We define a set of functions $\Phi$ over the strong $k$-constraint ordering on $P$. For each scheme $s$ of length $i$ of $X$ and $d \in C(s)$, let $\Phi(s, d, i)$ be the identity if $d \in B(s)$, else it modifies only the following subsets of the input $B$:

- if $i$, the length of $s$, is strictly less than $k$, then $\Phi(s, d, i)$ considers every $x_k \notin s$ and the resulting join scheme $t := s \cup \{x_k\}$, and maps $B(t)$ into
  \[ B'(t) := B(t) - \bigcup_{a \in D_k} \{d \land a\} \]

- if $i$, the length of $s$, is strictly greater than $1$, then $\Phi(d, s, i)$ considers every $x_j \in s$, the resulting projection scheme $r := s - \{x_j\}$, and modifies $B(r)$ as follows:
  \[ B'(r) := \begin{cases} 
  B(r) - \{d[r]\} & \text{if, for all } d' \in B(s), d'[r] = e \\
  B(r) & \text{yields } d'[x_j] = d[x_j],
  \end{cases} \]

Clearly, the $\Phi$ functions are monotone, inflationary and stationary with respect to the strong $k$-constraint ordering on $P$. The following lemma collects the main properties of these functions that will be used to study SGIIS for the Cooper algorithm.

**Corollary 4.4.5.**

(i) A CSP $P$ is strongly $k$ consistent iff it is a common fixpoint of the $\Phi$ functions, hence their least one with respect to the strong $k$-constraint order on $P$.

(ii) The $\Phi$ functions are monotone, stationary and inflationary with respect to the strong $k$-constraint order on $P$; besides, each $F_\perp$ function that does not modify the input problem is the identity. $\square$

The proof of the equivalence of the considered instantiations of SGIISE and SGIIS is analogous to that for HAC-4.
4.5. Relational Consistency

**Lemma 4.4.6.** SGIISE with the $\phi$ functions over $P_{KS}$ is $\equiv_{KS}$-equivalent to SGIIS with the $\Phi$ functions over $O_k \cong P_{KS}' \equiv_{KS}$. □

Therefore, we infer the following result from Corollary 4.4.5, Theorem 3.4.10, the above Lemma 4.4.6 and Corollary 3.4.16.

**Corollary 4.4.7 (KS).** The $KS$ algorithm over a finite problem $P$ terminates by computing the least fixpoint of the functions $\Phi$ as above defined; i.e., the greatest strongly $k$ consistent problem that is equivalent to $P$. □

4.5 Relational Consistency

Whereas in $k$ and strong $k$ consistency, variables and their instantiations are the key notions, in the definition of relational consistency as below, relations rather than variables are under analysis. In this section we define this new notion of consistency, as in [DvB97], and prove that the basic algorithm schema for enforcing it is an instance of GI.

**Definition 4.5.1.** Consider a CSP with constraint set $C$, and scheme $X$. Assume that $C' := \{C(s_1), \ldots, C(s_n)\}$ is a subset of distinct constraints in $C$, and $s$ is the join scheme of $s_1, \ldots, s_n$.

- Let $t$ be a scheme of $s$. Then $C'$ is *relationally $m$ consistent relative to $t$* if any $t$ consistent instantiation can be extended to an $s$ instantiation that satisfies $\bowtie_m R(s_i)$.

- The set $C'$ is *relationally $(i, m)$ consistent* if it is relationally $m$ consistent relative to each scheme $t$ of $s$, that has length $i$. If $C'$ is relationally $(i, m)$ consistent for every $i \leq m$, then it is *relationally $m$ consistent*.

- A CSP is *relationally $(i, m)$ consistent* if every subset of $m$ constraints in $C$ is such. The characterisation of *relational $m$ consistency* is analogous.

- A CSP is *strongly relational $(i, m)$ consistent* if it is $(i, k)$ relational consistent for each $k \leq m$. The characterisation of *strong relationally $m$ consistency* is analogous.

To illustrate the above defined notions, we copy the following example directly from [DvB97].
EXAMPLE 4.5.2. Consider the CSP over the scheme \( X := x_1, x_2, x_3, x_4, x_5 \), where the domains of the variables are all \( D = \{a, b, c\} \) and the relations are given by,

\[
\begin{align*}
C(x_2, x_3, x_4, x_5) &:= \{(a, a, a, a), (a, b, a, a), (a, a, b, a), (a, a, a, b)\} \\
C(x_1, x_2, x_5) &:= \{(b, a, b), (c, b, c), (b, a, c)\}.
\end{align*}
\]

The constraints are not relationally 2 consistent. For example, the instantiation \( x_2 = a, x_3 = b, x_4 = b \) is a consistent instantiation as it trivially satisfies all the applicable constraints. Similarly, the constraints are not relationally 3 consistent. For example, the instantiation \( x_1 = c, x_2 = b, x_3 = a, x_4 = a \) is, trivially, a consistent instantiation, but it does not have an extension to \( x_5 \) that satisfies \( C(x_2, x_3, x_4, x_5) \) and \( C(x_1, x_2, x_5) \) simultaneously. If we add the constraints \( C(x_2) = C(x_3) = C(x_4) = \{a\} \) and \( C(x_1) = C(x_5) = \{b\} \), the set of solutions of the CSP does not change, and it can be verified that the CSP is both relationally 2 and 3 consistent.

As the authors of [DvB97] remark, when all the problem constraints are binary, relational \( m \) consistency is identical (up to minor preprocessing) to variable-based \( m \) consistency. The virtue in their notion of relational \( m \) consistency is that it can be embedded, naturally, into algorithms for enforcing desired levels of relational \( m \) consistency, and it allows a simple generalisation of \( k \) consistency.

However, as for \( k \) consistency and hyper-arc consistency, verifying relational \( m \) consistency can be exponential even for relational 2 consistency, if the arity of the constraints is not bound.

4.5.1 The \( \text{RC}_{(i,m)} \) Algorithm

The original algorithm for relational \( m \) consistency, called \( \text{RC}_m \), is, in the authors’ words, “a brute-force algorithm for enforcing strong relational \( m \) consistency on a CSP”. In the remainder of the present subsection, we use \( \text{GI} \) to enforce relational \((i,m)\) consistency in the spirit of \( \text{RC}_m \) (cf. [DvB97]), and then analyse this is instance of \( \text{GI} \).

Instantiation

Consider a CSP \( P := \langle X, D, C \rangle \). To instantiate the \( \text{GI} \) algorithm to \( \text{RC}_{(i,m)} \) on \( P \), we need to provide the following components:

1. a partial ordering on \( P \);
2. functions on the chosen partial ordering;
3. the specification of the \textit{update} operator.
Partial ordering. To enforce relational \((i, m)\) consistency, we employ the \(i\) constraint ordering on \(P\) as partial ordering. Thus we assume \(P := \langle X, D, C \rangle\) to be \(i\) complete; else we complete it as in Subsection 2.5.2.

Functions. Consider \(B\) in the \(i\) constraint ordering on \(P\). Assume a scheme \(s\) and a scheme \(t\) of \(s\) such that the length of \(t\) is \(i\). Thus consider \(m\) constraints \(B(s_1), \ldots, B(s_m)\) of \(B\) such that \(s = \bigcup^n_{i=1} s_i\), and denote
\[
\mathbf{s} := (s_1, \ldots, s_m).
\]

Finally, define the function \(\sigma(t, s, s)\) as follows on \(B\): if \(B' := \sigma(t, s, s)(B)\), then \(B'\) differs from \(B\) for the constraint on \(t\), this being
\[
B'(t) := B(t) \cap \prod_t (\forall s_i \in s_i. B(s_i)).
\]

If \(B'(t)\) is the empty set, then \(\sigma(t, s, s)\) sets the whole \(B\) to the empty set. Else all the other constraints in \(B\) are unaffected by \(\sigma(t, s, s)\).

The update operator. The update operator returns the empty set if \(B'(t)\) is empty. Else, it adds all the constraints of the problem to \(G\) for further inspection. Clearly, this choice of the update operator could be optimised in a number of ways; for instance, by requiring that update should only add the relations which are affected by the change of \(B(t)\).

Analysis

At this point, it is routine to check that the functions \(\sigma(t, s, s)\) are idempotent, monotone and inflationary over the \(i\) constraint ordering.

**Lemma 4.5.3.**

(i) A CSP \(P := \langle X, D, C \rangle\) is relational \((i, m)\) consistent iff \(C\) is a common fixpoint of all the functions of the form \(\sigma(t, s, s)\), hence the last one with respect to the \(i\) constraint ordering on \(P\).

(ii) Each function \(\sigma(t, s, s)\) is idempotent, monotone and inflationary with respect to the \(i\) constraint ordering on \(P\). \qed

Fix now a CSP \(P\). By instantiating the GII algorithm with the above defined functions \(\sigma(t, s, s)\), we get the algorithm \(\text{RC}_{(i,m)}\). Thus we can prove that this algorithm enjoys the following properties as a consequence of Theorem 3.4.4.

**Corollary 4.5.4 (\(\text{RC}_{(i,m)}\)).** Consider a CSP \(P := \langle X, D, C \rangle\) with finite constraints. Suppose that \(P\) is binary. Then \(\text{RC}_{(i,m)}\) on \(P\) always terminates by computing the greatest relational \((i, m)\) consistent problem that is equivalent to \(P\); that is the least common fixpoint of the \(\sigma(t, s, s)\) functions, defined as above. \(\Box\)
The above result completes our analysis of quite a number of constraint propagation algorithms. In the following section, we summarise what we have learnt from this analysis, and so conclude the present chapter.

4.6 Conclusions

4.6.1 Synopsis

In this chapter, we describe and analyse a series of constraint propagation algorithms through the unifying framework of SGI iterations, see Chapter 3. Properties of the algorithms are interpreted as properties of functions, so that the verification of the algorithms becomes a straightforward application of the theoretical results obtained for SGI function iterations.

Thus these algorithms are separated into classes, according to the space the correlated functions prune of inconsistencies: see the rightmost column in Table 4.1. Then a more refined analysis differentiates between the algorithms that pertain to the same class. In Table 4.1, such differences are expressed in the following terms: functions, whether these are related to sets (i.e., domains or constraints) or points (i.e., values in domains or tuples in constraints); properties of functions, (i.e. commutativity, inflationarity, stationarity and idempotency) used to avoid fruitless while loops, via an efficient instantiation of \textit{update}.

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Functions</th>
<th>update</th>
<th>Search Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC-1</td>
<td>set</td>
<td>Axiom 3.3.1</td>
<td>domain ordering</td>
</tr>
<tr>
<td>AC-3</td>
<td>set</td>
<td>commutativity and idempotency</td>
<td>domain ordering</td>
</tr>
<tr>
<td>AC-4</td>
<td>point</td>
<td>stationarity, inflationarity and idempotency</td>
<td>domain ordering</td>
</tr>
<tr>
<td>PC-1</td>
<td>set</td>
<td>Axiom 3.3.1</td>
<td>2-constraint ordering</td>
</tr>
<tr>
<td>PC-2</td>
<td>set</td>
<td>commutativity and idempotency</td>
<td>2-constraint ordering</td>
</tr>
<tr>
<td>PC-4</td>
<td>point</td>
<td>stationarity, inflationarity and idempotency</td>
<td>2-constraint ordering</td>
</tr>
<tr>
<td>KC</td>
<td>point</td>
<td>stationarity, inflationarity and idempotency</td>
<td>(strong) $k$-constraint ordering</td>
</tr>
<tr>
<td>RC$_{(i,m)}$</td>
<td>set</td>
<td>idempotency</td>
<td>$i$-constraint ordering</td>
</tr>
</tbody>
</table>

Table 4.1: Constraint propagation algorithms and their comparison through SGI.
4.6. Conclusions

4.6.2 Discussion

The framework of SGI iterations allows us to verify various constraint propagation algorithms in a systematic and uniform way, and compare them as above. The more traditional ways of verifying the correctness of those algorithms are complicated by the diversity of structures or representation of CSPs adopted in them; cf. the algorithms in Appendix A. Besides, this heterogeneity complicates their direct comparison, which is usually done only by examples.

In the cases of (H)AC-4, (H)AC-5, PC-4 and KS, CSP constraints and domains are pruned of inconsistencies by means of additional support structures, used for time efficiency reason; these have to be “scraped away” through an equivalence relation (i.e., passing from SGISE to SGIIIS) to obtain functions that modify domains or constraints in a monotonic manner. This complicates the verification proofs and the simplicity of GI gets slightly lost. Nevertheless, at the level of domains and constraints, also those algorithms work in a monotone and inflationary manner: i.e., they all iterate functions that satisfy our Axioms 3.3.1 and 3.3.2. This results from the analysis conducted in this chapter.

This same analysis can be extended to algorithms like bound-consistency (see [Apt99a]) that iterate functions defined on specific domains: intervals of real numbers. Other algorithms for arc consistency such as AC-6 can be studied through SGI as well.

The AC-6 algorithm exploits the same intuitions behind AC-4: functions are set based, in that each function $f$ is associated with a specific value $c$ in a domain, and searches whether $c$ is consistent with a given constraint of the problem. The main difference is that AC-6 functions searches for only a single value $b$ such that the pair of $b$ and $c$ belongs to the given constraint; whereas AC-4 searches for all such $b$. Therefore, functions for AC-6 do not perform a universal selection but a sort of existential selection: i.e., they stop their search when a value that satisfies certain properties is found. In AC-6, this is achieved by imposing a total order on each domain and selecting always the minimum value in the ordering that satisfies the required properties.

We shall clarify this difference, between the so-called universal and existential selection in Chapter 6. First, we enrich our class of constraint propagation algorithms by studying non-standard constraints. The new algorithms are briefly presented in the following chapter, where they are still described and analysed via SGI iterations.