Chapter 3

A Schema of Function Iterations

3.1 Introduction

3.1.1 Motivations

In the remainder of the first part of our thesis, we shall gradually zoom on constraint propagation. Under this name gathers a number of mainly polynomial-time algorithms; each of these iteratively remove certain inconsistencies from CSPs, thereby attempting to limit the combinatorial explosion of the solution search space. More interestingly, all these algorithms avoid backtracking: at each iteration, a constraint propagation algorithm may remove values from CSPs, but never add them back in subsequent iterations.

Constraint propagation algorithms are known in the literature under other various names: filtering, narrowing, local consistency (which is, for some authors, a more specific notion), constraint enforcing, constraint inference, Waltz algorithms, incomplete constraint solvers, reasoners. However, here and in the remainder of this thesis, we adopt the most popular name, and always refer to them as constraint propagation algorithms.

In [Apt00a], the author states that "the attempts of finding general principles behind the constraint propagation algorithms repeatedly reoccur in the literature on constraint satisfaction problems spanning the last twenty years".

On a larger scale, the search for general principles is a common drive, shared by theoretical scientists of diverse disciplines: a series of methods to solve certain problems are devised; in turn, at a certain stage, this process calls for a uniform and general view if a common pattern can be envisaged. For instance, think of polynomial equations. Until the fifteenth century, algebra was a mere collection of stratagems for solving only numerical equations; these were expressed in words, and the account of the various solving methods was, sometimes, pure literature*. It was Viète in his *Opera Arithmetica* (1646) to introduce the use of vowels

---

* Cf. "La 'grande arte': l'algebra nel Rinascimento". U. Bottazzini, in Storia della scienza

27
for unknown values; this simplified notation paved the way to a general theory of polynomial equations and solving methods, no more restricted to numerical equations.

In this chapter, we propose a simple theory for describing constraint propagation algorithms by means of function iterations; the aim is to give a more general view on constraint propagation, based on functions and their iterations. It is well known that partial functions can be used for the semantics of deterministic programs; for instance, see [Jon97, LP81, Pap94]. The primary objective of our theorisation thus becomes that of tackling the following issues:

- abstracting which functions perform the task of pruning or propagation of inconsistencies in constraint propagation algorithms,

- describing and analysing how the pruning and propagation process is carried through by constraint propagation algorithms.

In this chapter, we mainly focus on the latter item, that is on how functions remove certain inconsistencies from CSPs and propagate the effects of this pruning. The basic theory, proposed in this chapter, will provide a uniform reading of a number of constraint propagation algorithms. Then, in Chapter 6, only after describing and analysing those algorithms in Chapters 4 and 5 via that theory, we specify which functions are traced in their study.

### 3.1.2 Outline

The topic of this chapter is a basic theory of iterations of functions for constraint propagation algorithms.

We first characterise iterations of functions (see Section 3.2) and then introduce the basic algorithm schema that iterates them by following a certain strategy (see Section 3.3). Thus, in the remainder of this chapter, we investigate some properties of the proposed algorithm schema by studying those of the iterated functions and the iterations themselves, see Section 3.4. For example, idempotency of functions will be related to fruitless loops, in terms of pruning, that can be thereby cut off. In turn, this property of functions will be traced in some specific constraint propagation algorithms in which it is used to avoid redundant computations, see Chapter 4.

On the one hand, the proposed algorithm schema is sufficient for describing many constraint propagation algorithms in terms of functions on a generic set, see Subsection 3.3.1, or on an equivalent set, see Subsection 3.4.3. On the other hand, a partial order on the function domain provides a sharper tool for analysing and studying the behaviour of these algorithms, loosely speaking. More precisely, a partial order on the function domain gives us a means to partially order the
possible computations of algorithms, see Subsection 3.3.2. Thereby, by means of
the domain order, we can pose and answer the following sort of questions about
the behaviour of constraint propagation algorithms.

- Can the order of constraint propagation affect the result?
- Or is the output problem independent of the specific order in which con-
  straint propagation is performed (see Theorem 3.3.8)?
- Do constraint propagation algorithms always terminate?
- Or what is sufficient to guarantee their termination (see Theorem 3.3.9 and
  Corollary 3.3.10)?

In all the analysed cases in Chapter 4, functions for constraint propagation al-
gorithms turn out to be inflationary with respect to a suitable partial order on
their domain, see p. 32. This property of functions also accounts for the absence
of backtracking in constraint propagation algorithms: pruning of values is never
resumed, since every execution of a constraint propagation algorithm always pro-
ceeds along an order.

Other properties of functions, related to the order, can be further used to
prune branches from the algorithm search tree; we shall study this issue in Sub-
section 3.4.2. For instance, a property that we call stationarity will be introduced
as a stronger form of idempotency; hence functions that enjoy it need to occur at
most once in any execution of the algorithm schema.

3.1.3 Structure
First, we introduce iterations of functions in Section 3.2, and the basic schema
to iterate them in Section 3.3. Variations of the basic schema, along with the
related properties of functions, are treated in details in Section 3.4. Finally, we
summarise and discuss the results of this chapter in Section 3.5.

3.2 Iterations of Functions
Given a finite set $F$ of functions $f : O \mapsto O$ over a set $O$, we define a sequence
$(o_n : n \in \mathbb{N})$ with values in $O$ as follows:

1. $o_0 := \bot$, where $\bot$ is a selected element of $O$;
2. $o_{n+1} := f(o_n)$, for some $f \in F$.

Each sequence $(o_n : n \in \mathbb{N})$ is called an iteration of $F$ functions (based on $\bot$).
An iteration of $F$ functions $(o_n : n \in \mathbb{N})$ stabilises at $o_n$ if $o_{n+k} = o_n$ for every
$k \geq 0$. 
In this chapter, we shall mainly be concerned with iterations of $F$ functions that stabilise at some specific points: in fact, we shall be interested in iterations that stabilise at a common fixpoint of all the functions: namely, an element $o \in O$ such that

$$f(o) = o \text{ for all } f \in F.$$ 

Indeed, it is not sufficient for an iteration to stabilise at $o$ for this to be a common fixpoint of all the $F$ functions, as the following simple example illustrates.

**Example 3.2.1.** Consider $O := \{0, 1, 2\}$ and the set $F$ with the following two functions:

$$f(0) := 0, f(1) := 2 \text{ and } f(2) := 2,$$

$$g(0) := 1, g(1) := 1 \text{ and } g(2) := 2.$$ 

Now, consider the iteration $(o_n : n \in \mathbb{N})$ based on $0$ such that $o_{n+1} := f(o_n)$ for every $n \in \mathbb{N}$. Indeed, the iteration stabilises at $0$; but this is not a fixpoint of $g$ since $g(0) \neq 0$.

In the above Example 3.2.1, the function $g$ is never selected. Would it be sufficient to choose $g$ after $f$ to guarantee that $o$ is a common fixpoint of the $F$ functions? Certainly not: define first $o_1 := f(o_0) = 0$, then $o_{j+1} := g(o_j)$ for $j > 0$. This iteration stabilises at $1$ and not at $2$, which is the only common fixpoint of the two functions $f$ and $g$. We can repeat the above trick infinitely many times, one for every $k > 0$: in fact, it is sufficient to set $o_{i+1} := f(o_i)$ for $0 \leq i < k$, and $o_{j+1} := g(o_j)$ for $j \geq k$; still the resulting iteration stabilises at $1$. How can we remedy this? The answer is given below, by the algorithm schema in Section 3.3: this is designed to compute a common fixpoint of finitely many functions.

### 3.3 The Basic Iteration Schema

The Structured Generic Iteration algorithm, briefly SGI, is a slightly more general version of the Generic Iteration algorithm of [Apt00a]. Both of them aim at computing a common fixpoint of finitely many functions, simply by iterating them until such a fixpoint is computed. The SGI algorithm is more general in that its first execution can start with a subset of all the given functions; then these are introduced, *only if necessary*, in subsequent iterations. So SGI covers more algorithms than the Generic Iteration algorithm does.

The SGI algorithm is displayed as Algorithm 3.3.1. Its parameters are characterised as follows.

**Convention 3.3.1 (SGI).**

- $F$ is a finite set of functions, all defined on the same set $O$;
3.3. The Basic Iteration Schema

- \( \bot \) is an element of \( O \);
- \( F_\bot \) is a subset of \( F \) that enjoys the following property: every \( F \) function \( f \) such that \( f(\bot) \neq \bot \) belongs to \( F_\bot \);
- \( G \) is a subset of \( F \) functions;
- \( \text{update} \) instantiates \( G \) to a subset of \( F \).

**Algorithm 3.3.1: SGI(\( \bot, F_\bot, F \))**

\[
\begin{align*}
o &:= \bot; \\
G &:= F_\bot; \\
\text{while } G \neq \emptyset \text{ do} & \\
& \quad \text{choose } g \in G; \\
& \quad G := G - \{g\}; \\
& \quad o' := g(o); \\
& \quad \text{if } o' \neq o \text{ then} \\
& \quad \quad G := G \cup \text{update}(G, F, g, o); \\
& \quad o := o'
\end{align*}
\]

As we shall see below, the \( \text{update} \) operator returns a subset of \( F \) according to the functions in \( G \), the current \( O \) value \( o \) and \( F \) function \( g \); its computation can be expensive, unless some information on the chosen function \( g \) and input value \( o \) is provided that can help to compute the \( F \) functions returned by \( \text{update} \), as we shall see in Chapter 4. Besides, in the SGI schema below, the function \( g \) is chosen non deterministically; no strategy for choosing \( g \) is imposed in this schema; but this is done on purpose, since SGI aims at being a general template for a number of CSP algorithms. Indeed, the complexity of the algorithm will vary according to the way in which the \( \text{update} \) operator will be specified and the function \( g \) chosen.

### 3.3.1 The basic theory of SGI

The SGI algorithm is devised to compute a common fixpoint of the \( F \) functions: i.e., an element \( o \in O \) such that \( f(o) = o \) for every \( f \in F \). Suppose that the following predicate

\[
\forall f \in F - G \ f(o) = o \quad \text{(Inv)}
\]

is an invariant of the while loop of the SGI algorithm. If \( o \) is the last input of a terminating execution of SGI, then \( G \) is the empty set and the predicate \( \text{Inv} \) above implies that \( o \) is a common fixpoint of all the \( F \) functions. We restate this as the following fact, which is used over and over in the remainder of this chapter.
FACT 3.3.1 (COMMON FIXPOINT). Suppose that the above predicate Inv is an invariant of the while loop of SGI. If \( o \) is the last input of a terminating execution of SGI, then \( o \) is a common fixpoint of the \( F \) functions. \( \square \)

Common fixpoint

The Common Fixpoint Fact 3.3.1 above suggests a simple, yet sufficient condition for SGI to compute a common fixpoint of the \( F \) functions: after an iteration of the while loop, we only need to keep, in \( G \), the functions for which the input value of the while loop is not a fixpoint. As for this, it is sufficient that the update operator in SGI satisfies the following axiom.

AXIOM 3.3.1 (COMMON FIXPOINT). Let \( o' := g(o) \) for \( g \in F \), and \( \text{Id}(g, o') := \{g\} \) if \( g(o') \neq o' \), else \( \text{Id}(g, o') \) is the empty set. If \( o' \neq o \), then

\[
\text{update}(G, F, g, o) \supseteq \{ f \in (F - G) : f(o) = o \text{ and } f(o') \neq o' \} \cup \text{Id}(g, o');
\]

otherwise \( \text{update}(G, F, g, o) \) is the empty set.

In other words, the update operator adds to \( G \) at least all the \( F \) functions, not already in \( G \), for which \( o \) is a fixpoint and the new value \( o' \) is not; besides, \( g \) has to be added back to \( G \) if \( g(o') \neq o' \).

LEMMA 3.3.2 (INVARIANCE). Let us assume the Common Fixpoint Axiom 3.3.1. Then the Inv predicate on p. 31 is an invariant of the while loop of SGI.

PROOF. The base step follows from the definition of \( F \) (see Convention 3.3.1 above), and the induction step is easily proved by means of the Common Fixpoint Axiom 3.3.1. \( \square \)

The above Common Fixpoint Fact 3.3.1 and Invariance Lemma 3.3.2 immediately imply the following theorem.

THEOREM 3.3.3 (COMMON FIXPOINT). Let us assume the Common Fixpoint Axiom 3.3.1. If \( o \) is the last input of a terminating execution of the SGI algorithm, then \( o \) is a common fixpoint of all the \( F \) functions. \( \square \)

EXAMPLE 3.3.4. Let us consider Example 3.2.1 as input to SGI so that \( \bot := 0 \). First, assume \( F \bot := \{g\} \). In the first while loop, only \( g \) can be chosen and applied; so, after the loop, \( v \) is set equal to 1. In the same loop, update adds \( f \) to \( G \) and \( g \) is removed from \( G \). So, at the end of the first loop, \( G = \{f\} \) and \( o = 1 \). Then \( f \) is chosen and applied to 1, \( o \) is set equal to 2 and \( G \) to the empty set at the end of the loop. So SGI terminates by computing 2, a common fixpoint of the
two functions. Instead, if $F_\bot$ is instantiated to the whole set $F := \{ f, g \}$, there are two possible executions of SGI, both terminating with 2. Note that, given Convention 3.3.1 and Axiom 3.3.1, there are three different executions of the SGI algorithm with $F := \{ f, g \}$ and $\bot := 0$.

### SGI iterations

We started this chapter with generic iterations of functions, and provided a schema that computes a common fixpoint of finitely many functions. Hereby we show how function iterations and the SGI schema are related. First of all, let us denote by $id$ the identity function on the domain of the iterated functions of $F$; indeed all the common fixpoints of the $F$ functions are, trivially, fixpoint of $id$. Then every execution of SGI gives rise to an iteration of the $F \cup \{ id \}$ functions. To explain how, we first introduce traces of SGI executions.

Consider an execution of the SGI algorithm — see Algorithm 3.3.1. The SGI trace $\langle (o_n, G_n) : n \in \mathbb{N} \rangle$ of the execution is defined as follows:

- $o_0 := \bot, G_0 := F_\bot$;
- suppose that $o_n$ and $G_n$ are the input of the $n$-th while loop of SGI. If $G_n$ is the empty set, then $o_{n+1} := id(o_n)$ and $G_{n+1} := \emptyset$. Otherwise, let $g$ be the chosen function and set $o_{n+1}$ equal to $g(o_n)$. Then we define $G_{n+1}$ as the set $(G_n - \{ g \}) \cup \text{update}(G_n, F, o, o_n)$ if $o_{n+1} \neq o_n$, else as the set $G_n - \{ g \}$.

Then $\langle o_n : n \in \mathbb{N} \rangle$ is an SGI iteration of the $F$ functions.

**Example 3.3.5.** Let us revisit Example 3.3.4. There we have the following three SGI traces:

1. $(\langle 0, \{ g \} \rangle, (1, \{ f \}), (2, \emptyset), \ldots)$;
2. $(\langle 0, \{ f, g \} \rangle, (0, \{ g \}), (1, \{ f \}), (2, \emptyset), \ldots)$;
3. $(\langle 0, \{ f, g \} \rangle, (1, \{ f \}), (2, \emptyset), \ldots)$.

The first trace corresponds to the SGI instantiation with $F_\bot := \{ g \}$; whereas the SGI instantiation related to the second and third traces is for $F_\bot := \{ f, g \}$.

Traces provide another tool to formulate and study properties of SGI, like termination. We shall say that the trace $\langle (o_n, G_n) : n \in \mathbb{N} \rangle$ stabilises at $o_k$ iff the iteration $\langle o_n : n \in \mathbb{N} \rangle$ does so and $G_k = \emptyset$. Now, the termination condition for the while loop in SGI is that $G$ must be empty; the last input $o$ of a terminating execution of SGI is the value computed by SGI. Hence it is easy to check that the following statements are equivalent:

1. the SGI algorithm terminates by computing $o$;
2. the SGI trace stabilizes at $o_k = o$.

We reformulate this equivalence as the following fact, as it will allow us to switch from executions of SGI to traces, and vice versa, when dealing with the termination of SGI.

**Fact 3.3.6.** An execution of the SGI algorithm terminates by computing $o$ iff the associated trace stabilizes at $o$. 

### 3.3.2 Ordering Iterations

Suppose that we can define a partial order $\sqsubseteq$ over the set $O$. Then this can be used to order iterations.

#### Least common fixpoint

Suppose that the $F$ functions are monotone with respect to a partial order $\sqsubseteq$ on $O$; namely, for every $f \in F$,

$$ o \sqsubseteq o' \implies f(o) \sqsubseteq f(o'). $$

Then we can prove that the common fixpoints of the $F$ functions, as computed by SGI, coincide with the least fixpoint of the $F$ functions. So let us assume the following statement.

**Axiom 3.3.2 (Least Fixpoint).**

(i). The structure $\langle O, \sqsubseteq, \bot \rangle$ is a partial ordering with bottom $\bot \in O$.

(ii). The $F$ functions are all monotone with respect to $\sqsubseteq$.

Given the above axiom, we have the following lemma as in [Apt00a].

**Lemma 3.3.7 (Stabilisation).** Assume the Common Fixpoint Axiom 3.3.1 and the Least Fixpoint Axiom 3.3.2. Consider a fixpoint $w$ of the $F$ functions. Let $\langle o_i : i \in \mathbb{N} \rangle$ be a generic iteration of $F$ that satisfies the following properties:

- $o_0 := \bot$;
- $o_{i+1} := g(o_i)$, for some $g \in F$.

Then $o_i \sqsubseteq w$, for every $f \in F$ and $o_i$ in the iteration $\langle o_i : i \in \mathbb{N} \rangle$.

**Proof.** The proof is by induction on $i \in \mathbb{N}$. The base case is trivial since $\bot$ is the bottom of the ordering. As for the induction step, let us assume that $o_i \sqsubseteq w$. Thus, we invoke monotonicity (see Axiom 3.3.2) and obtain $o_{i+1} := g(o_i) \sqsubseteq w$. 

The above lemma shows how a partial ordering can be used to compare computations of SGI with functions on the ordering, and highlights the role of monotonicity in the following result, which follows from the lemma and Theorem 3.3.3.
3.3. The Basic Iteration Schema

**Theorem 3.3.8 (Least Fixpoint).** Let $F$ be a finite set of functions over $O$, and assume the Common Fixpoint Axiom 3.3.1 and the Least Fixpoint Axiom 3.3.2. Then all the terminating executions of SGI compute the same value: i.e., the least common fixpoint of all the $F$ functions with respect to the partial order on $O$. □

**Termination**

From this point onwards, let us write $o \sqsubseteq o'$ whenever $o \sqsubseteq o'$ and $o \neq o'$. A $\sqsubseteq$-chain in $O$ is any subset of $O$, that is totally ordered by $\sqsubseteq$.

In order to ensure the termination of SGI, for any input, we must ascertain that every SGI iteration stabilises. If all the $F$ functions are computable, every SGI iteration is totally ordered by $\sqsubseteq$, and all $\sqsubseteq$-chains are finite, then we can guarantee termination. The following axiom formalises these ideas.

**Axiom 3.3.3 (Termination).**
- Each $f \in F$ is a computable function over a partial ordering with bottom $O := (O, \sqsubseteq, \bot)$.
- The $F$ functions are inflationary with respect to the partial order: namely, $o \sqsubseteq f(o)$ for every $o \in O$ and $f \in F$.
- The ordering $(O, \sqsubseteq)$ satisfies the ascending chain condition (ACC), i.e. each $\sqsubseteq$-chain in $O$ is finite.

Now, given the above axiom, we can prove the following termination result.

**Theorem 3.3.9 (Termination 1).** Assume the Common Fixpoint Axiom 3.3.1 and the Termination Axiom 3.3.3. Then SGI always terminates, by computing a common fixpoint of the $F$ function.

**Proof.** At each iteration of the while loop, either $o \sqsubseteq o'$ — due to inflationarity, see Axiom 3.3.3 — or $g$ is removed from $G$. Axiom 3.3.3 yields that all $\sqsubseteq$-chains are finite; since $G \sqsubseteq F$ is finite too, the algorithm terminates. □

Notice that every finite partial ordering satisfies the ACC in Axiom 3.3.3; moreover, every function on a finite set is computable. Thus we draw the following conclusion as a corollary of Theorem 3.3.9.

**Corollary 3.3.10 (Termination 2).** Let us assume the Common Fixpoint Axiom 3.3.1 and that the $F$ functions are defined on a finite partial ordering with bottom $(O, \sqsubseteq, \bot)$. Suppose that the $F$ functions are inflationary with respect to $\sqsubseteq$. Then every execution of SGI terminates. □

**Note 3.3.11.** Many algorithms for CSPs deal with finite domains. Whenever those algorithms are instances of SGI, the above Corollary 3.3.10 will ensure that
a simple condition on the iterated functions is sufficient to guarantee the termination of the instance algorithms. However, functions for non-standard CSPs as in Chapter 5, often, have infinite domains; then Corollary 3.3.10 will not be of help, and we shall need to resort to the above Theorem 3.3.9.

### 3.3.3 Finale

The main results concerning the basic SGI algorithm schema are collected in the following corollary; this gathers Theorems 3.3.3, 3.3.8, 3.3.9 and Corollary 3.3.10. Figure 3.3.3 depicts a search tree of SGI, under the assumptions of either one of the statements in the following corollary.

**Corollary 3.3.12.**

(i). Assume the Common Fixpoint Axiom 3.3.1, the Least Fixpoint Axiom 3.3.2 and the Termination Axiom 3.3.3. Then every execution of SGI terminates by computing the least common fixpoint o of the iterated functions.

(ii). Assume the Common Fixpoint Axiom 3.3.1 and that the F functions are defined on a finite partial ordering with bottom \( (O, \sqsubseteq, \perp) \). Suppose that the F functions are all monotone and inflationary with respect to \( \sqsubseteq \). Then every execution of SGI terminates by computing the least common fixpoint o of the iterated functions. \( \square \)

---

**Figure 3.1:** SGI search tree.
3.4 Variations of the Basic Schema

3.4.1 The Generic Iteration Schema

We started Section 3.3 by claiming that SGI is a slightly more general version of the Generic Iteration (GI) algorithm of [Apt00a]. We shall also need to refer to the latter schema in Chapter 4, and hence we explain it in more details as below.

The difference between the two basic schemas is that the set of functions $G$ in GI is initialised to the whole set of functions $F$: i.e., $F_{1}$ is the whole set $F$. Therefore, we have the following results for GI as a consequence of Corollary 3.3.12.

**Theorem 3.4.1.**
(i) Assume the Common Fixpoint Axiom 3.3.1, the Least Fixpoint Axiom 3.3.2 and the Termination Axiom 3.3.3. Then GI always terminates by computing the least common fixpoint of the iterated functions.

(ii) Assume the Common Fixpoint Axiom 3.3.1 on update and that the $F$ functions are defined on a finite partial ordering with bottom $(O, \subseteq, \bot)$. Suppose that the $F$ functions are monotone and inflationary with respect to $\subseteq$. Then GI always terminates, computing the least common fixpoint of the iterated functions.$\Box$

3.4.2 Iterations Modulo Function Properties

The GI algorithm is a variation of SGI in that $G$ is differently initialised. Other variations of the basic schema SGI are obtained by optimising the instantiation of $G$ in the while loop via update: in SGI, all the functions for which the new computed value $d' = g(o)$ is not a fixpoint are added to $G$. Indeed, more functions are added to $G$, more executions of the while loop are needed. In the following, we study some properties of functions that allow us to reduce the number of executions of the while loop by an efficient instantiation of $G$ via update. Each property is studied separately and gives rise to a different version of the SGI schema; all these or their combinations will be used in Chapter 4.

**Idempotent functions**

Notice that the chosen function $g$ is removed from the set $G$ of iterated functions in SGI if the test $gg(o) = g(o)$ returns true. This is always true, independently of the specific value $o$, if $g$ is idempotent, as specified below.

**Definition 3.4.2.** A function $g : O \rightarrow O$ is idempotent if $gg(o) = g(o)$ for every $o \in O$. 

As suggested above, an idempotent function can always be removed after being chosen. The following diagram shows what happens otherwise.

\[ o \rightarrow g(o) \rightarrow gg(o) \]

So any iteration as above can be equivalently reduced to one in which the second application of \( g \) is removed, if this function is idempotent; i.e., \( Id(g, o') \), as in the Common Fixpoint Algorithm ref axiom:sgi:1, is always set to the empty set for \( g \) idempotent. The following lemma is an immediate consequence of that axiom and Definition 3.4.2 above.

**Lemma 3.4.3 (Idempotency).** Consider a finite set \( F \) of idempotent functions on \( O \). Then

\[
\text{update}(G, F, g, o) \supseteq G \cup \{ f \in F - G : f(o) = o \text{ and } f(o') \neq o' \}
\]

satisfies the Common Fixpoint Axiom 3.3.1. \( \square \)

Let us call SGII the version of SGI with the update operator as in Lemma 3.4.3, where the second I stands in for Idempotent. Then the following theorem is a trivial consequence of that lemma and Corollary 3.3.12.

**Theorem 3.4.4.**

(i). Assume the Least Fixpoint Axiom 3.3.2 and the Termination Axiom 3.3.3. Then every execution of SGII terminates by computing the least common fixpoint of the iterated functions, if these are all idempotent.

(ii). Assume that the \( F \) functions are defined on a finite partial ordering with bottom \( (O, \sqsubseteq, \bot) \). Suppose that the \( F \) functions are all monotone and inflationary with respect to \( \sqsubseteq \). Then every execution of SGII terminates, computing the least common fixpoint of the iterated functions, if these are all idempotent. \( \square \)

**Commutative functions**

Commutativity of an operation is a useful property in computations: it means that the operation provides the same result regardless of permutations of the combined elements. Function composition is not, in general, a commutative operation. However there are classes of functions on which composition is commutative; thereby the order in which these functions are composed is irrelevant. The following definition aims at capturing precisely this, and it is a special case of the notion of centraliser of an element with respect to a given operation, like in group theory; see for instance [Her75].
3.4. Variations of the Basic Schema

**Definition 3.4.5.** Let $F$ be a set of functions over a set $O$, consider a function $g : O \mapsto O$, and the subset $\text{Comm}(F, g)$ of $F$ of all functions $f$ such that

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

Then the set $\text{Comm}(F, g)$ is the set of $F$ functions that *commute* with $g$.

As stated and proved in [Apt00a], commutativity can be exploited to reduce executions in the GI algorithm. This carries over to the SGI schema in the same manner.

**Lemma 3.4.6.** If the update operator satisfies the Common Fixpoint Axiom 3.3.1, then so does $\text{update}(G, F, g, o) - \text{Comm}(F, g)$.

**Proof.** Suppose that $fg(o) = gf(o)$; then $f(o) = o$ implies $gf(o) = g(o)$; thus $\text{update} - \text{Comm}$ satisfies the Common Fixpoint Axiom 3.3.1 if $\text{update}$ does. □

Let us rename SGI with Commutativity, briefly SGIc, the SGI algorithm with $\text{update} - \text{Comm}$ in place of $\text{update}$. The above Lemma 3.4.6 allows us to transfer, to SGIc, all the results concerning SGI as summarised in Corollary 3.3.12.

**Theorem 3.4.7.**
(i). Assume Assume the Common Fixpoint Axiom 3.3.1, the Least Fixpoint Axiom 3.3.2 and the Termination Axiom 3.3.3. Then SGIc always terminates by computing the least common fixpoint of the iterated functions.
(ii). Assume the Common Fixpoint Axiom 3.3.1 on update and that the $F$ functions are defined on a finite partial ordering with bottom $(O, \subseteq, \bot)$. Suppose that the $F$ functions are all monotone and inflationary with respect to $\subseteq$. Then SGIc always terminates, computing the least common fixpoint of the iterated functions. □

**Stationary functions**

While the properties of idempotency and commutativity do not rely on any partial order on the given set $O$, the following property does.

**Definition 3.4.8.** Let $f$ be an inflationary function, defined over a partial ordering $(O, \subseteq)$. Then the function $f$ is *stationary from* $o \in O$ and $o'$ if it enjoys the following property:

$$\text{if } f(o) \neq o, \; o \subseteq o' \text{ and } f(o') \subseteq o'' \text{ then } f(o'') = o''.$$ 

More in general, an inflationary function $f$ is *stationary* if there exist $o, o' \in O$ such that $f$ is stationary from them.
In other words: consider a totally ordered iteration and suppose that a stationary function $f$ is known to affect a value $o$ in it; after the first application of $f$ to $o$ or a subsequent value $o'$ in the iteration, $f$ does not change any value that follows in the iteration. The following diagram shows schematically what happens whenever a stationary function $f$ is applied again, after $f$ modifies a value in the iteration.

In brief, the iteration in the above diagram can equivalently be reduced to one in which the second application of $f$ is removed, if this function is stationary. The below lemma states precisely that stationary functions can be added at most once to $G$, namely the set of functions to iterate.

In order to formulate the lemma properly, we resort to traces and state the following axiom.

**Axiom 3.4.1 (stationarity).**

(i) The $F$ functions are all stationary on $\langle O, \sqsubseteq, \bot \rangle$ and, if $f \in F_\bot$ and $f(\bot) = \bot$, then $f$ is the identity on $O$.

(ii) If $\langle (o_n, G_n) : n \in \mathbb{N} \rangle$ is the trace of an execution of $SGI$, $G_n$ denotes the set of $G$ functions at the $n$-th while loop of $SGI$; then put

$$update(G_n, F, g, o_n) := \left\{ f \in F - \bigcup_{k \leq n} G_k : f(o_n) = o_n, f(o_{n+1}) \neq o_{n+1} \right\};$$

this for every $n$.

Now we can formulate the following Stationarity Lemma: there we assume that the $F$ functions are idempotent, since this simplifies the proof, even tough the extension to the non-idempotent case is possible.

**Lemma 3.4.9 (stationarity).** Assume the Stationarity Axiom 3.4.1 and that all the $F$ functions are idempotent. Then $update$ satisfies the Common Fixpoint Axiom 3.3.1.

**Proof.** We only need to prove that, if $f \in \bigcup_{k<n} G_k$ and $f \notin G_n$, then $f(o_{n+1}) = o_{n+1}$. If $f \in F_\bot$ and $f(\bot) = \bot$, then $f(o_{n+1}) = o_{n+1}$ by Axiom 3.4.1. Else, there must be $o_i$ in the iteration such that $i < n$ and $f(o_i) \neq o_i$, due to Axiom 3.4.1 again. Since $f \notin G_n$, then there exists $i \leq j < n$ and $o_j$ in the iteration such that $o_{j+1} = f(o_j)$. Therefore, inflationarity yields the following:

$$o_i \sqsubseteq o_j \text{ and } f(o_j) = o_{j+1} \sqsubseteq o_n \sqsubset o_{n+1}.$$
3.4. Variations of the Basic Schema

Thus we can conclude that \( o_i \neq f(o_i), o_i \sqsubseteq o_j \) and \( f(o_j) \sqsubseteq o_{n+1} \) hold. Then Definition 3.4.8 yields \( f(o_{n+1}) = o_{n+1} \).

In Chapter 4, we make an extensive use of the variation of the SGI algorithm with stationary and idempotent functions. So we rewrite SGI with Stationary and Idempotent functions as the SGIIS Algorithm 3.4.1.

Algorithm 3.4.1: SGIIS(\( \perp, F_\perp, F \))

\[
\begin{align*}
v & := \perp; \\
g & := \emptyset; \\
\text{while } F_\perp \neq \emptyset \text{ do} \\
& \quad \text{choose } g \in F_\perp; \\
& \quad F_\perp := F_\perp - \{g\}; \\
& \quad o' := g(o); \\
& \quad \text{if } o' \neq o \text{ then} \\
& \quad \quad G := G \cup \text{update}(G, F, g, o); \\
& \quad \quad F := F - \text{update}(G, F, g, o); \\
& \quad \quad o := o'; \\
& \quad \text{while } G \neq \emptyset \text{ do} \\
& \quad \quad \text{choose } g \in G; \\
& \quad \quad G := G - \{g\}; \\
& \quad \quad o' := g(o); \\
& \quad \quad \text{if } o' \neq o \text{ then} \\
& \quad \quad \quad G := G \cup \text{update}(G, F, g, o); \\
& \quad \quad \quad F := F - \text{update}(G, F, g, o); \\
& \quad \quad \quad o := o'; 
\end{align*}
\]

The Stationarity Lemma 3.4.9 and the Idempotency Lemma 3.4.3 allow us to transfer, to SGIIS, all the results concerning SGI as summarised in Corollary 3.3.12.

**Theorem 3.4.10.**

(i). Assume the Stationarity Axiom 3.4.1, the Least Fixpoint Axiom 3.3.2 and the Termination Axiom 3.3.3. Then every execution of SGIS terminates by computing the least common fixpoint of the iterated functions.

(ii). Assume the Stationarity Axiom 3.4.1, the Least Fixpoint Axiom 3.3.2, and that the \( F \) functions are defined on a finite partial ordering with bottom \( \{O, \sqsubseteq, \perp\} \). Then every execution of SGIS terminates by computing the least common fixpoint of the iterated functions. \( \square \)
3.4.3 Iterations Modulo Equivalence

The SGI algorithm with Equivalence (SGIE) is SGI with functions defined on an equivalence structure \((V, \equiv)\), and such that the if conditional depends on the equivalence of the input and output values, and not necessarily their identity; see Algorithm 3.4.2. Like for SGI, the update operator selects and returns functions of \(F\). Thus this algorithm iterates functions from a set \(F\) until a value \(v\) is found for which \(g^V(v) \equiv v\). Indeed, if the equivalence relation \(\equiv\) collapses into the identity, we have the SGI algorithm back.

Algorithm 3.4.2: SGIE(\(\bot, \equiv, F_\bot, F\))

\[
v := \bot^V; \\
G^V := F^V; \\
\textbf{while } G^V \neq \emptyset \textbf{ do} \\
\quad \text{choose } g^V \in G^V; \\
\quad G^V := G^V - \{g^V\}; \\
\quad v' := g^V(v); \\
\quad \textbf{if } v' \neq v \textbf{ then } G^V := G^V \cup \text{update}(F^V, g^V, v); \\
\quad v := v'; \\
\textbf{end while}
\]

**Note 3.4.11.** We let \(\text{update}(F^V, g^V, v)\) be an unspecified subset of \(G\). In fact, in the case of the SGIE algorithm schema, the update operator varies according to the instance CSP algorithms. However, as for the results in this part, we do not need to assume anything more of update that it is returns a subset of \(F\) functions.

**SGIE iterations**

As in the case of SGI, we associate SGIE traces with executions of an SGIE algorithm. Again, notice that the identity function \(id^V\) on \(V\) does not affect any value in any computation of SGIE; i.e., \(id^V(v) \equiv v\) for every \(v \in V\).

The SGIE traces \((v_n, G^V_n) : n \in \mathbb{N}\) of executions of the SGIE algorithm are defined like the SGI traces:

- \(v_0 := \bot_V, G^V_0 := F^V_\bot\):

- suppose that \(v_n\) and \(G^V_n\) are the input of the \(n\)-th while loop of SGIE. If \(G^V_n\) is the empty set, then \(v_{n+1} := v_n\) and \(G^V_{n+1} := \emptyset\). Otherwise, let \(g^V\) be the chosen function and set \(v_{n+1}\) equal to \(g^V(v_n)\). Then the set \(G^V_{n+1}\) is defined as \(G^V_n - \{g^V\} \cup \text{update}(F^V, G^V_n, g^V, v_n)\) if \(v_{n+1} \neq v_n\), otherwise as the set \(G^V_n - \{g^V\}\).
3.4. Variations of the Basic Schema

The iteration \( \langle v_n : n \in \mathbb{N} \rangle \) is called an SGI \( \text{iteration} \) of SGI.

As in the case of SGI, traces are useful to formulate and study termination conditions on SGI. So we shall say that the SGI \( \text{trace} \) stabilises at \( v_n \) if the iteration \( \langle v_n : n \in \mathbb{N} \rangle \) does so and \( G_n^V = \emptyset \). The following equivalence will be useful in the below part.

**Fact 3.4.12.** An iteration of an SGI algorithm terminates by computing \( v \) iff the associated trace stabilises at \( v \).

The least \( \equiv \)-class and termination

Suppose that we can devise a partial order \( \sqsubseteq \) on a quotient set \( O \) isomorphic to \( V/\equiv \), such that \( \langle O, \sqsubseteq, \bot^O \rangle \) turns out to be a partial ordering with bottom. Then we can try to transfer the analysis and results concerning SGI, over the partial ordering \( \langle O, \sqsubseteq, \bot^O \rangle \), to SGI over the equivalence structure \( \langle V, \equiv \rangle \).

Let \( F^V \) and \( F^O \) be, respectively, a finite set of functions over \( V \) and \( O \). Consider a bijective map \( f : F^O \rightarrow F^V \) that maps the identity of \( F^O \) to the identity function of \( F^V \). Let us denote

\[
f^V := f(f^O),
\]

for each \( f^O \in F^O \). Now, suppose that an SGI trace \( \langle (v_n, G_n^V) : n \in \mathbb{N} \rangle \) of \( F^V \) functions can be associated with an SGI trace \( \langle (o_n, G_n^O) : n \in \mathbb{N} \rangle \) of \( F^O \) functions via \( f \), and that such traces enjoy the following property:

1. \( v_0 \in o_0 \);

2. for every \( n \geq 0 \), if \( v_{n+1} = f^V(v_n) \) for \( f^V \in G_n^V \) then \( o_{n+1} = f^O(o_n) \) for \( f^O \in G_n^O \), and the following property holds:

   there exists \( m \geq n + 1 \) such that \( v_m \in o_m \).

Then the two traces \( \langle (v_n, G_n^V) : n \in \mathbb{N} \rangle \) and \( \langle (o_n, G_n^O) : n \in \mathbb{N} \rangle \) are called \( \equiv \)-equivalent via \( f \).

The characterisation of \( \equiv \)-equivalence, via a function \( f \), is sufficient to obtain the following result.

**Lemma 3.4.13.** Consider an SGI trace \( o := \langle o_n : n \in \mathbb{N} \rangle \) and an \( \equiv \)-equivalent SGI trace \( v := \langle v_n : n \in \mathbb{N} \rangle \).

- The trace \( v \) stabilises at a value \( v \in V \) if the trace \( o \) does so at a value \( o \in O \);

- furthermore, the value \( o \in O \) (where \( O \) is isomorphic to \( V/\equiv \)) corresponds to the \( \equiv \)-class of \( v \).
PROOF. Suppose that \( \langle o_n : n \in \mathbb{N} \rangle \) stabilises at \( o_n \). Then \( G_n^O = \emptyset \), hence \( G_n^V \) is empty due to the definition of equivalent traces above. Then \( v_n \in o_n \) follows from the above definition. Therefore \( \langle (v_n, G_n^V) : n \in \mathbb{N} \rangle \) stabilises at \( v_n \in o_n \). \( \square \)

The following definition extends the notion of \( \equiv \)-equivalence between traces to an analogous between algorithm executions.

**Definition 3.4.14.** If there exists a map \( f \) such that every SGIE trace with \( F^V \) is \( \equiv \)-equivalent to an SGI trace with \( F^O \) via \( f \), then SGIE with \( F^V \) is called \( \equiv \)-equivalent to SGI with \( F^O \).

The following theorem is a consequence of Facts 3.3.6 and 3.4.12, and the above Lemma 3.4.13.

**Theorem 3.4.15.** Suppose that every execution of SGI terminates by computing the least common fixpoint \( o \) of the \( F_o \) functions. If SGIE with \( F^V \) functions over \( \langle V, \equiv \rangle \) is \( \equiv \)-equivalent to SGI, then every execution of SGIE terminates by computing \( \equiv \)-equivalent values; i.e. values \( v \) in the \( \equiv \)-class that corresponds to \( o \). \( \square \)

Theorem 3.4.15 above implies that we can study instances of the SGIE schema — that takes in input a set \( V \) with an equivalence relation — if we can provide, for them, equivalent instances of the SGI algorithm schema:

1. we devise a partial ordering with bottom on a set \( O \), isomorphic to the quotient set \( V/\equiv \);

2. then we check whether the given instance of SGIE on \( V \) is \( \equiv \)-equivalent to an instance of SGI on \( O \), with suitable functions on \( O \);

3. if this equivalence holds, then Theorem 3.4.15 implies that, if SGI terminates by always computing the same value, then SGIE terminates by always computing values which belong to the same equivalence class.

These transfer results, summarised as in the below corollary, are consequences of Theorem 3.4.15, and Corollary 3.3.12 for SGI.

**Corollary 3.4.16.** Consider an instance of SGI with \( F^O \) functions on a finite partial ordering \( O := (O, \sqsubseteq, \perp^O) \). Let SGIE be instantiated with \( F^V \) functions on an equivalence structure \( V/\equiv \) that is isomorphic to \( O \). Furthermore, suppose that this instance of SGIE is \( \equiv \)-equivalent to the instance of SGI with the \( F^O \) functions on the partial ordering \( O \). Thus we have the following results:

- if the \( F^O \) functions are inflationary, then every execution of SGIE with the \( F^V \) functions on \( V \) terminates;
3.5. Conclusions

- if the $F^O$ functions are also monotone, then every execution of SGIE with the $F^V$ functions and $V$ terminates, by computing values which are all in the $\equiv$-class of the least common fixpoint of the $F^O$ functions.

Variations of SGIE

All versions of SGIE can be modified similarly and so generate a corresponding version of SGIE. However, in Chapter 4, we only deal with the following variations of SGIE:

- the GI algorithm (see Subsection 3.4.1) with equivalence, namely GIE;
- the SGIIIS algorithm (see Algorithm 3.4.1) with equivalence, denoted by SGIISE.

All these algorithms share the same parameters, which are specified as follows:

- an equivalence structure, namely a set $V$ and an equivalence binary relation $\equiv$ on it;
- $\bot^V$, an element of $V$;
- a finite set $F^V$ of functions $f^V : V \rightarrow V$;
- a subset $F_\bot^V$ of $F^V$ that contains every $F^V$ function $f^V$ for which $f^V(\bot) \neq \bot$;
- the update operator that selects and returns a subset of $F$ functions.

The definitions and results given above for SGIE are easily extended to the cases of GIE and SGIISE. We leave the task to fill in the details to the reader.

3.5 Conclusions

3.5.1 Synopsis

This chapter presents a basic algorithm schema, SGIE, and some of its variations. The SGIE schema iteratively applies functions until a common fixpoint of theirs is found: the Common Fixpoint Axiom 3.3.1 provides a sufficient property for this, and characterises the basic strategy of SGIE. Then Axioms 3.3.1 and 3.3.2 state sufficient properties for SGIE to find the least common fixpoint of the functions and terminate, respectively. Notice that all those properties are encountered in most constraint propagation algorithms, see Chapter 4 below; there, the SGIE schema or its variations are used as “templates” to explain and differentiate those algorithms.

Variations of the basic schema are thus studied in Section 3.4: they are differentiated in terms of update and properties of functions; these differences account
for different strategies of the algorithms in Chapter 4. Besides, some of those algorithms use additional support structures, so to speak: i.e., they remove values from the given CSP domains or constraints by storing information in other structures. In those cases the SGIE, namely SGI on an set equipped with an equivalence relation, proves useful: first the algorithms are instantiated to SGIE; then the additional structures are "scraped away" through the adopted equivalence relation, so that SGI can be used to analyse those algorithms too. These instances of SGI iterate functions that only remove values from domains or constraints, and do it in a monotone and inflationary manner; thus we are able to transfer the results obtained for SGI instances to SGIE instances.

Some of the main variations of SGI and SGIE are summarised in the following table that contains in each cell, from left to right:

- a variation of SGI or SGIE;
- the related properties of functions;
- where the related update operator is characterised;
- where a variation is applied in Chapter 4, which deals with constraint propagation algorithms for CSPs — these are introduced in Chapter 2.

<table>
<thead>
<tr>
<th>SGI &amp; SGIE Variations</th>
<th>Properties of Functions</th>
<th>The update Operator</th>
<th>Where in Chapter 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGIIS</td>
<td>idempotency, stationarity</td>
<td>Idempotency Lemma 3.4.3, Stationarity Lemma 3.4.9</td>
<td>(H)AC–4, (H)AC–5, PC–4</td>
</tr>
<tr>
<td>SGISE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GI</td>
<td></td>
<td>Common Fixpoint Axiom 3.3.1</td>
<td>(H)AC–1, PC–1, RC_{(i,m)}</td>
</tr>
<tr>
<td>GIC</td>
<td>commutativity</td>
<td>Commutativity Lemma 3.4.6</td>
<td>AC–3</td>
</tr>
<tr>
<td>GIIS</td>
<td>idempotency, stationarity</td>
<td>Idempotency Lemma 3.4.3, Stationarity Lemma 3.4.9</td>
<td>KC</td>
</tr>
<tr>
<td>GIISE</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Chapter 5 concerns itself with non-standard CSPs that allow to obtain optimal partial solutions, according to certain criteria: the original algorithm schema for constraint propagation is extended via SGI. In Chapter 2 and Sections 5.4, 5.5 of Chapter 5, we also apply the results of the present chapter as displayed in the following table: this shows how properties of functions or update are correlated to properties of algorithms in both Chapters 2 and 5.
3.5. Conclusions

<table>
<thead>
<tr>
<th>Properties of update or Functions</th>
<th>Properties of Algorithms</th>
<th>Where in Chapter 4</th>
<th>Where in Chapter 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common Fixpoint Axiom 3.3.1</td>
<td>partial correctness</td>
<td>Corollaries 4.2.4, 4.2.6, 4.2.12, 4.2.15, 4.2.16, 4.3.5, 4.3.7, 4.3.12, 4.4.7, 4.5.4</td>
<td>Corollary 5.4.4</td>
</tr>
<tr>
<td>Monotonicity Axiom 3.3.2</td>
<td>confluence</td>
<td>as above</td>
<td>Corollaries 5.4.5 and 5.4.6</td>
</tr>
<tr>
<td>Inflationarity Axiom 3.3.2</td>
<td>termination</td>
<td>as above</td>
<td>Corollaries 5.4.7, 5.4.14 and 5.4.15</td>
</tr>
</tbody>
</table>

3.5.2 Discussion

Using a single framework for presenting constraint propagation algorithms makes it easier to verify and compare these algorithms. Again from a theoretical viewpoint, this approach allows us to separate the properties that concur in the definition of a constraint propagation algorithm: e.g., inflationarity is related to termination and absence of backtracking; monotonicity to confluence; stationarity, commutativity and idempotence explain optimised strategies for various constraint propagation algorithms. Preserving equivalence is another important property of those algorithms: in such a general setting, we cannot tackle it, since we study functions on “generic” sets, i.e., not on CSPs. Nonetheless, in Chapter 4, it is always easy to prove, by means of the adopted functions, that constraint propagation algorithms maintain equivalence.

From an applicative viewpoint, this approach allows us to parallelise constraint propagation algorithms in a simple and uniform way and result in a general framework for distributed constraint propagation algorithms; see [Mon00]. This shows that constraint propagation can be viewed as the coordination of cooperative agents. Additionally, such a general framework facilitates the combination of these algorithms, a property often referred to as solver cooperation or combination. Finally, the generic iteration algorithm SGI and its specializations can be used as a template for deriving specific constraint propagation algorithms in which specific scheduling strategies are employed.