Entailment-based actions for coordination

Luis Monteiro*, António Porto

Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa,
2825 Monte da Caparica, Portugal

Abstract

Coordination seems to require (at least in part) a persistent repository of information that concurrent agents can query and update. Indeed, most coordination languages are based on a shared data space model. They differ in the details of how actions and processes are defined, but most assume the data space to have a multiset structure, and actions to be rewritings. We find this view too particular and not expressive enough in many practical cases, and set out in this paper to develop a more general theory of actions, abandoning the syntactic rewriting paradigm in favour of a more abstract notion of update based on entailment. Actions may impose certain properties to be entailed or not entailed, and the corresponding update is the minimal change, possibly by removing information and adding new one, that satisfies the (dis)entailment requirements.

We work with abstract situations (standing for information states) ordered under entailment. We show that if a situation space is a coherent, prime algebraic, consistently complete poset then a suitable class of its subsets, which we call definite, corresponds to update operations with suitable generality (any situation can be obtained by an update of any other) and good compositional properties (closure under sequential and synchronous composition). These updates can be seen as unconditional determinate actions, i.e. total functions from situations to situations; these functions are always a composition of a restriction (losing information) and an expansion (adding consistent information). We show that the space of updates is itself an ordered structure similar to a situation space.

Then we consider general actions, which may be conditional and nondeterministic. They thus represent arbitrary relations between situations, but are actually more specific, coding the intensional rather than extensional behaviour, this being relevant for the synchronous composition. We formulate general actions as (suitably restricted) relations between situations and definite sets, define their synchronous, sequential and choice compositions, and show them to be fully abstract with respect to observing situation transitions under any compositional context.

The synchronous and sequential compositions give rise to an intrinsic notion of independence of actions, that reflects their ability to be truly concurrent.
0. Introduction

Software systems in many application areas are growing in size and complexity, with concurrent and distributed execution being the norm rather than the exception. The difficulties of designing and maintaining such systems have brought into focus the issue of coordination among components, prompting several proposals for coordination languages and models as a way of promoting the separation of concerns between computation and coordination.

Most coordination formalisms developed so far rely on a common notion of shared data space and generative communication. This means that a persistent repository of information can be asynchronously accessed by processes that can both query and change it, the analogue of agents sensing and acting upon a common environment. This was started by Linda [10, 7], basically a collection of primitive actions for reading, deleting and adding tuples to a shared tuple space. This simple model has been extended in Bauhaus Linda [8] to multiple nested tuple spaces. GAMMA grew out of the chemical reaction model of computation [4–6, 12], and consists basically of reaction/action rules for consuming/creating "molecules", that can fire anywhere and anytime that the reaction condition is met [5]. The LO coordination model [3, 2] introduces the notion of "forum" as a dynamically evolving set of tuple spaces (a la Linda) which can communicate via broadcast communication. The transformations inside each tuple space (consumption and production of tuples) are specified by rules, which also trigger broadcast communications and the creation of new tuple spaces by cloning; the semantics has been given in terms of proof search in Linear Logic [1].

What these formalisms have in common is twofold: they assume a multiset structure in the data space, and basic actions are given by rewrite rules. The latter is not surprising, being commonplace among models for concurrency; indeed, rewriting logic is being promoted as a general framework for concurrency [13]. The use of multisets is usually justified as providing a direct handling of resources. Both these aspects, however, can be challenged as unnecessarily restricted views. Take the direct representation of resources of the same type as multiple occurrences of a token in a multiset. This is a very low-level, inefficient representation. No one would dream of representing amounts of money in bank accounts as multisets of monetary units; the obvious choice is to use numbers. In fact, when one looks at real programs written in languages based on multiset rewriting, one finds that the token multiplicity aspect is seldom if ever used, whereas integers appear as arguments in tokens. Rewriting is also low-level, being defined in purely syntactic terms using pattern matching. Again, real languages usually extend syntactic pattern matching with some ability to establish more semantic constraints. A simple example is the use of arithmetic, to say that a certain argument is the sum of two others, or is greater than some other; in a more complex case one may wish to test whether a graph is planar, in order to invoke a specialized drawing algorithm.

The shift of perspective needed for achieving generality is that of abandoning syntactic rewriting of data in favour of querying and updating data on the basis of properties...
that it entails. This is the view behind concurrent constraint programming [15] (CCP, for short). In CCP the shared data space is a constraint store, processes may be prefixed with an ask operation checking entailment of a given constraint by the store, and tell operations update the store by consistently adding constraints to it. The monotonic growth of the store is however a major restriction. Many applications require us to deal directly with a general notion of change, whereby true properties may become false and vice versa, or information may be erased. Attempting to code the notion of "current" state in a monotonic store is hard in a concurrent and distributed setting.

In [9] CCP was modified to allow for a get operation that consumes constraints and thus provides for non-monotonicity of the store. This proposal, however, completely disrupts the entailment based nature of CCP, turning instead to the linear logic/multiset rewriting paradigm. An alternative view of the use of CCP is promoted by its timed extensions [16, 19] meant for the more restricted area of synchronous systems, in which a program runs to completion in response to a signal, but always upon a clear store. Persistence has then to be indirectly encoded by having the next program reinstate the constraints that persist, an awkward process.

A first proposal for an entailment based computation model with nonmonotonic updates was TAO [14]. In TAO a system state is composed of a task and a database. The database is a shared data space where the basic actions take place: queries (checking for entailment) and commands (imposing entailment). The task expresses a composition of processes, using several operators, ultimately built out of basic actions. The semantics for basic actions was expressed through an abstract set of situations standing for the possible information states denoted by the changing database. The situation space was assumed to be a nonempty consistently complete poset, order being read as entailment. Queries and commands were built from propositions, denoting sets of situations. A major concern was to restrict the effect of commands to definite updates. This was achieved in two steps. First one considered that command propositions are restricted to denote the upward closure of some situation; this corresponds to viewing a command as imposing a certain situation to be entailed. Then it was proved that the update of a situation by another has a unique (suitably defined) result if and only if the situation space is coherent, i.e. any pairwise consistent subset is consistent. This version of TAO is still not general enough. Although the database changes nonmonotonically in terms of entailment, it is in some sense monotonic in terms of "quantity" of information. What happens is that if a property $p$ is entailed at some point, the only way for it to cease to be entailed is by imposing its negation $\neg p$; there is no way to impose that $p$ is not entailed. This is essential if we want the ability to model loss of information, a practical necessity – data is erased on purpose from databases when no longer relevant, for example.

This paper addresses precisely the characterization of fully general updates expressing not just entailment but also disentailment of information. We perform a systematic study of their properties, something that had not been done even for the simpler more restricted case, and which is essential for a subsequent development of denotational semantics of processes for TAO. The first part of the paper deals with unconditional
determinate actions, corresponding only to commands in TAO. In the second part we study the more general notion of conditional indeterminate actions, that represent the synchronous (atomic) composition of queries and commands, and choices thereof.

1. Situation spaces

In this paper we study a model of shared data spaces based on states of partial information ordered by information content. The states of the data space are called “situations” and the information ordering the “entailment” relation. This is formalized by a set $S$ partially ordered by $\sqsubseteq$, where $x \sqsubseteq y$ means that situation $x$ has less information than situation $y$, or the information in $y$ entails the one in $x$. As a poset (abbreviation of partially ordered set), a “situation space” must satisfy the additional requirements of being consistently complete, coherent and prime algebraic. We shall introduce these requirements one by one, trying to justify its need. Apart from a finiteness condition (which plays no role in our subsequent development), situation spaces are then isomorphic to domains of configurations of event structures [20]. The reader may wish to interpret situation spaces in that way, by seeing events as propositions and conflict as inconsistency.

To motivate the formal definitions, we start with some examples. For these we use a concrete syntax that plays no role in the rest of the paper, where we deal with abstract situation spaces.

Our first example is adapted from the one in [9]. Suppose we have the predicates available($x$), booked($x, y$) and taken($x, y$), indicating respectively that theater place number $x$ is available, $x$ has been booked by person $y$, and $x$ has been taken by $y$. A situation is a set (not a multiset!) of ground atomic formulae for these predicates. On any situation we can perform queries and updates. A query like $\text{available}(x)$ checks whether place $x$ is available without changing the situation. An update of the form $!\text{booked}(x, y)$ adds to the current situation the information that $x$ has been booked by $y$. We also allow updates with negated atoms as in $!\text{available}(x)$, that removes the information that $x$ is available. Thus, queries check for entailment and updates force an entailment or a disentailment.

More complex actions can be derived by synchronous combination of queries and updates. For example, if $\text{mpc}(y)$ denotes the action (considered a fact in [9]) that $y$ makes a phone call to reserve a theater place, it can be defined by

$$\text{mpc}(y) = ?\text{available}(x) \circ !\text{available}(x) \circ !\text{booked}(x, y)$$

where we denote by $\circ$ the synchronous composition as in [14]. Thus, the effect of $\text{mpc}(y)$ is to look for an available place, make it unavailable and book it, all in one step. To actually acquire a previously booked place we define an action

$$\text{buys}(y) = ?\text{booked}(x, y) \circ !\text{booked}(x, y) \circ !\text{taken}(x, y).$$
The abstract notion of update that will be introduced later allows seemingly more general updates like \(!\{\neg \text{available}(x), \text{booked}(x, y)\}\), but prime algebraicity reduces them to synchronous compositions of updates of prime elements, as in this example.

In general, certain sets of atoms may be inconsistent, and we do not wish to consider them as (parts of) situations. This already happens in our example, where it does not make sense to have the atoms \(\text{available}(x)\) and \(\text{booked}(x, y)\) in the same situation, since clearly a booked place cannot also remain available for further booking. Such integrity constraints can be represented in the logic programming formalism we are using in this example by negative clauses like

\[
\begin{align*}
&\leftarrow \text{available}(x), \text{booked}(x, y), \\
&\leftarrow \text{booked}(x, y), \text{taken}(x, y), \\
&\leftarrow \text{available}(x), \text{taken}(x, y)
\end{align*}
\]

and the situations are required not to contain any body of an integrity constraint. An update of a nonnegated atom can no longer consist simply in adding the atom to the situation, as that could give rise to inconsistencies. Thus, while adding the atom all atoms inconsistent with it must be removed. Taking advantage of this mechanism, the actions \(\text{mpc}(y)\) and \(\text{buys}(y)\) may be redefined as

\[
\begin{align*}
\text{mpc}(y) &= \neg \text{available}(x) \circ \neg \text{booked}(x, y), \\
\text{buys}(y) &= \neg \text{booked}(x, y) \circ \neg \text{taken}(x, y)
\end{align*}
\]

since the required disentailments are consequences of the integrity constraints.

It is important to notice that we only allow binary integrity constraints (two atoms in the body) in order to ensure that updates are deterministic (this entails no loss of generality as we consider later a more general notion of action that is nondeterministic). For example, if we had an integrity constraint \(\leftarrow p, q, r\), to perform the update \(\neg p\) in the situation \(\{q, r\}\) we could nondeterministically remove either \(q\) or \(r\) before adding \(p\).

In abstract terms, this implies that the space of situations is coherent, in the sense that a set of situations is inconsistent (has an inconsistent union) only if two of them are mutually inconsistent. Though our example is close to the logic programming paradigm, the abstract model studied in this paper applies to other cases where binary integrity constraints arise naturally, like functional dependencies in relational data bases or single values assigned to variables in imperative programming.

Coherence is a very strong requirement. Even such a simple system as the Herbrand constraint system gives rise to a noncoherent space. For example, the set \(\{x = a, x = y, y = b\}\) is inconsistent if \(x\) and \(y\) are variables and \(a\) and \(b\) are distinct constants, but any two constraints in the set are consistent. It is possible to relax the coherence requirement and consider general constraint systems, provided the domains of application of updates are restricted to situations where they are deterministic. This is precisely the case in CCP (concurrent constraint programming [19]), with the even stronger restriction that the situations must be consistent with the update. But the general case is a topic for
future research, and in this paper we assume that situation spaces are coherent, because that allows a uniform and simplified treatment of updates. Even respecting coherence, a limited form of constraints can sometimes be used, as the following example shows. The example also illustrates the use of an entailment relation along with integrity constraints to define the allowed situations.

The atomic formulae have the form $x \leq n$, $x \geq n$ or $x = n$ where $x$ is a variable and $n$ is a natural (or real) number. The integrity constraints are

\[
\neg x \leq m, \quad x = n,
\]

\[
\neg x \leq m, \quad x \geq n,
\]

\[
\neg x = m, \quad x \geq n
\]

for all $x$ and all $m < n$. The entailment relation is characterized by obvious implications, among which are

\[
x \leq n, \quad x \geq n \rightarrow x = n,
\]

\[
x \leq m \rightarrow x \leq n
\]

for all $x$ and all $m < n$. A situation is a consistent and entailment-closed set of atoms. (Intuitively, a situation associates with each variable an interval, possibly reduced to a single point.) It is easy to see that if a finite set $A$ of atoms is inconsistent then two atoms in $A$ are already inconsistent. Indeed, there must exist a variable $x$ such that the atoms in $A$ involving $x$ are inconsistent. Then let $m$ be the least number such that $x \leq m$ or $x = m$ is in $A$, and $n$ the greatest number such that $x \geq n$ or $x = n$ is in $A$. We have $m < n$, otherwise the atoms involving $x$ would be consistent. It follows that $x \leq m$ or $x = m$ together with $x \geq n$ or $x = n$, as the case may be, is an inconsistent pair in $A$.

According to the definitions that will be presented later, the space of situations in this example (and the previous ones) is consistently complete, coherent and prime algebraic. Consistency completeness is a consequence of the fact that if a set of situations has a consistent union then its entailment closure is a situation. Coherence is just the property noted above. Prime algebraicity means here that the atoms of the form $x \leq m$ and $x \geq n$ are sufficient to uniquely determine any situation (the atoms $x = n$ are entailed by the joint presence of $x \leq n$ and $x \geq n$), but are also necessary (the precise information conveyed by $x \leq m$ or $x \geq n$ cannot be independently obtained with other atoms). Note that an update like $\neg(x = n)$ is equivalent to the synchronous composition $\neg(x \leq n) \circ (x \geq n)$.

In general, a situation space may be defined, just like in this example, by giving a set of elements (here the atoms of the form $x \leq m$ and $x \geq n$), a symmetric and irreflexive incompatibility relation between pairs of elements (the binary integrity constraints), and a pre-order on the elements (the entailment relation) such that elements entailed by compatible elements are still compatible. The situations are then the down-closed sets of pairwise compatible elements. Thus, apart from a finiteness condition, situation spaces are just domains of configurations of event structures [20], as noted before.
We now turn to the formal development of the theory. The most basic assumption that can be made about a situation space $S$ is that it is consistently complete. Intuitively, this means that a consistent set of situations can always be combined into a new situation that comprises exactly the information contained in the given situations. Recall that a subset $X$ of a poset $S$ is consistent if every finite subset of $X$ has an upper bound in $S$. Note that the empty set is consistent if and only if $S$ is not empty. A poset is consistently complete, and we call it a cc-poset, if it is not empty and every consistent subset has a least upper bound. The least upper bound of the empty set is the least element of $S$, written $\bot$. We denote the least upper bound of $X$ by $\bigcup X$, with the usual notational variants for two-element sets and families of elements.

We say $x, y \in S$ are compatible if $\{x, y\}$ is consistent, and call $x \sqcup y$ the join of $x$ and $y$.

Coherence of the situation space is justified by the kind of operations we wish to perform on the data space. The most basic operations are to query if a given piece of information is entailed by the current situation, and to update the situation with new information. A query does not change the current situation, and is entirely described by the entailment relation. In an update, if the new information is inconsistent with the existing one, to preserve consistency part of the existing information has to be removed while adding the new one. This is in contrast to CCP, where information is never removed from the store. In CCP the store grows monotonically, but in our model the data space changes in a nonmonotonic fashion. The monotonic growth is semantically captured in the CCP model by closure operators [18], while in our model an update is the composition of a projection (the dual of a closure) to remove information with a closure to add information (see [11] to see the application of projections and closures in denotational semantics). We now make these ideas precise.

Removal of information is accomplished by a restriction – a special kind of projection. We call a function $r : S \rightarrow S$ a restriction if it is

- monotone: $x \subseteq y$ implies $r(x) \subseteq r(y)$;
- decreasing: $r(x) \sqsubseteq x$; and
- eradicating: $y \sqsubseteq r(x)$ implies $r(y) = y$.

All conditions are implicitly universally quantified. Intuitively, $r(x)$ is the situation obtained after removing a given piece of information from $x$. The information removed is intended to be the same for all situations, so if $x$ has less information than $y$, it remains so after the removal. Clearly, after the removal we obtain a situation with less information. Finally, if a situation has less information than one with the piece of information already removed then there is nothing left to remove (in other words, the information is eradicated once and for all). This explains the conditions of the definition of restriction. In the theater example, to make place number $n$ unavailable is a restriction that maps any situation $x$ to $x - \{\text{available}(n)\}$.

Note that a restriction is idempotent, because $r(r(x)) = r(x)$ is a consequence of $r(x) \subseteq r(x)$ by eradication, hence it is also a projection. The next result shows that a restriction $r$ (actually any projection) is uniquely determined by its image $r(S) = \{r(x) \mid x \in S\}$, and characterizes those subsets of $S$ that are images of restrictions.
Let $S$ be a cc-poset. If $X$ is a subset of $S$, let $\downarrow X$ and $\uparrow X$ be, respectively, the down-closure and the up-closure of $X$. Thus, $x \in \downarrow X$ if there is $y \in X$ such that $x \leq y$, and $z \in \uparrow X$ if there is $y \in X$ such that $y \leq z$. As a matter of notation, we abbreviate $\downarrow \{x\}$ to $\downarrow x$ and $\uparrow \{x\}$ to $\uparrow x$. Recall that $X$ is down-closed, resp. up-closed, if $X = \downarrow X$, resp. $X = \uparrow X$. We say $X$ is $\downarrow$-closed if $X$ contains the least upper bound of every nonempty consistent subset of $X$.

**Lemma 1.1.** Let $S$ be a cc-poset and $r : S \to S$ a restriction.

(i) If $r' : S \to S$ is a restriction with the same image as $r$ then $r = r'$.

(ii) The image $r(S)$ of $S$ under $r$ is down-closed and $\downarrow$-closed.

(iii) Every nonempty down-closed and $\downarrow$-closed subset $X$ of $S$ determines a restriction $r : S \to S$ with image $X$ by $r(x) = \bigcup (\downarrow x \cap X)$.

**Proof.** (i) For every $x \in S$, there is $x' \in S$ such that $r(x) = r'(x')$. But $r'(x') = r'(r'(x')) = r'(r(x)) \subseteq r'(x)$, where the last inequality is a consequence of $r(x) \subseteq x$. It follows that $r(x) \subseteq r'(x)$ for all $x \in S$. By a symmetric argument, $r'(x) \subseteq r(x)$ for all $x \in S$, therefore $r = r'$.

(ii) Suppose $y \subseteq z$ and $z \in r(S)$. There is $x \in S$ such that $z = r(x)$, so $r(y) = y$ by eradication. This shows that $y \in r(S)$, hence $r(S)$ is down-closed. Now suppose $X \subseteq r(S)$ is nonempty and consistent, with least upper bound $y$. We must show that $y \in r(S)$. Every $x \in X$ satisfies $x = r(x) \subseteq r(y)$, the equality because $x \in r(S)$ and the inequality because $x \subseteq y$. It follows that $y = \bigcup X \subseteq r(y) \subseteq y$, so $y = r(y)$ and therefore is in $r(S)$.

(iii) The function is well-defined because the set $\downarrow x \cap X$ is not empty (contains $\bot$) and consistent (is bounded by $x$). Furthermore, $r(x) \in X$ ($X$ is $\downarrow$-closed) and $r(x) = x$ if $x \in X$ (because in that case $\downarrow x \cap X = \downarrow x$), hence $r(S) = X$. Finally, $r$ is monotone ($x \subseteq y$ implies $\downarrow x \cap X \subseteq \downarrow y \cap X$), decreasing ($\downarrow x \cap X \subseteq \downarrow x$) and eradicating ($y \subseteq r(x) \in X$ implies $y \in X$, hence $r(y) = y$).

The restriction determined by $X$ associates with every $x \in S$ the greatest element below $x$ that is in $X$. It corresponds intuitively to remove the least possible information from $x$ so that the resulting situation is in $X$.

Addition of information is the dual of removal and is performed by an expansion, which is a particular type of closure. A function $e : S \to S$ is an expansion if it is

* monotone: $x \subseteq y$ implies $e(x) \subseteq e(y)$;
* increasing: $x \subseteq e(x)$; and
* saturating: $e(x) \subseteq y$ implies $e(y) = y$.

The intuitive reading of these conditions is similar to the one for restrictions. Note that expansions are closures, because they are idempotent. In the theater example, if we consider the cc-poset of all situations where place number $n$ is not available, the operation whereby $n$ is booked by $y$ is an expansion that maps any such situation $x$ to $x \cup \{booked(n, y)\}$.
Lemma 1.2. Let $S$ be a cc-poset and $e: S \to S$ an expansion. For all $x \in S$ we have $e(x) = x \cup e(\bot)$, so that the image of $e$ is $\uparrow e(\bot)$ and $e$ is uniquely determined by it. Furthermore, if $u \in S$ is compatible with every $x \in S$, the mapping $x \mapsto x \cup u$ is an expansion with image $\uparrow u$.

Proof. We have $x \cup e(\bot) \subseteq e(x)$ because $x \subseteq e(x)$ and $e(\bot) \subseteq e(x)$. This gives $e(x) \subseteq e(x \cup e(\bot)) \subseteq e(e(x)) = e(x)$, hence $e(x \cup e(\bot)) = e(x)$. But $e(\bot) \subseteq x \cup e(\bot)$, so saturation gives $e(x \cup e(\bot)) = x \cup e(\bot)$, and we conclude that $e(x) = x \cup e(\bot)$. The remaining statements are immediate. 

This result suggests that the intuitive reading of an expansion $e$ is that it adds to every situation $x$ the information $e(\bot)$. We can now define update.

Definition 1.3 (Update). An update on a cc-poset $S$ is a function $t: S \to S$ for which there exist a restriction $r: S \to S$ and an expansion $e: r(S) \to r(S)$ such that $t(x) = e(r(x))$ for every $x \in S$.

For example, the update $\text{booked}(n, y)$ consists of the restriction that makes $n$ unavailable, followed by the expansion that actually adds the information that $n$ was booked by $y$. As for restrictions and expansions, our immediate purpose is to show that updates are uniquely determined by their images, and to characterize the subsets of $S$ that are images of updates. We say a subset $X$ of $S$ is convex if $X = \downarrow X \cap \uparrow X$, or, equivalently, if $x, z \in X$ and $x \subseteq y \subseteq z$ imply $y \in X$. Clearly, down-closed sets and up-closed sets are convex.

Definition 1.4 (Definite set). A subset $U$ of $S$ is called definite if it has a least element, is convex, and is $\downarrow$-closed.

Being convex, a definite set $U$ always satisfies $U = \uparrow U \cap \downarrow U$. The set $\uparrow U$ can also be written $\uparrow u_0$, where $u_0$ is the least element of $U$, and is easily seen to be definite. The set $\downarrow U$ can be described as the set of all elements compatible with $u_0$ whose join with $u_0$ is in $U$. Indeed, all such elements are clearly in $\downarrow U$. Conversely, if $x \in \downarrow U$ then there is $y \in U$ such that $x \subseteq y$. As $u_0 \subseteq y$, we have $u_0 \subseteq x \cup u_0 \subseteq y$, so $x \cup u_0 \in U$ by the convexity of $U$. The set $\downarrow U$ has least element $\bot$ and is convex, but is not necessarily $\downarrow$-closed. It is interesting to note that the property that $\downarrow U$ is $\downarrow$-closed (that is, definite) for every definite set $U$ is equivalent to $S$ being coherent, as we shall see below.

Proposition 1.5. Let $S$ be a cc-poset and $t: S \to S$ an update with associated restriction $r: S \to S$ and expansion $e: r(S) \to r(S)$.

(i) If $t': S \to S$ is an update with the same image as $t$ then $t = t'$.

(ii) The image $t(S)$ of $S$ under $t$ is a definite set.

(iii) Every definite set $U$ such that $\downarrow U$ is $\downarrow$-closed determines an update $t: S \to S$ with image $U$ by $t(x) = \downarrow(\downarrow x \cap \downarrow U) \cup u_0$, where $u_0$ is the least element of $U$. 
Proof. (i) First note that the down-closure of $e(r(S))$ is $r(S)$. Indeed, as $e(r(S)) \subseteq r(S)$ and $r(S)$ is down-closed, $\downarrow e(r(S)) \subseteq r(S)$. On the other hand, if $x \in r(S)$ then $x \subseteq e(x) \in e(r(S))$, hence $x \in \downarrow e(r(S))$. Thus, if $t'$ is another update with associated restriction $r'$ and expansion $e'$, with the same image $e'(r'(S)) = e(r(S))$, we must have $r'(S) = r(S)$. By the previous lemmas, we first conclude that $r = r'$ and then that $e = e'$, so that finally $t = t'$.

(ii) The least element of $t(S)$ is $t(\bot) = e(r(\bot)) - e(\bot)$. Next suppose $x, z \in t(S)$ and $x \subseteq y \subseteq z$. As $t(S) \subseteq r(S)$ and $r(S)$ is down-closed, $y \subseteq z$ implies $y \in r(S)$. But $t(S)$ is up-closed as a subset of $r(S)$, so $x \subseteq y$ implies $y \in t(S)$. Finally, it is immediate that $t(S)$ is $\sqcup$-closed.

(iii) By the previous lemmas, $U$ determines a restriction $x \mapsto \downarrow \left(\downarrow x \cap \downarrow U\right)$ with image $\downarrow U$. As $U$ is convex, it is up-closed in $\downarrow U$, so determines an expansion $y \mapsto y \sqcup u_0$ from $\downarrow U$ to $\downarrow U$ with image $U$. These two mappings determine the required update. □

The update determined by a definite set $U$, with least element $u_0$, first computes the greatest element below $x$ whose join with $u_0$ exists and is in $U$ (that is, the element $\downarrow \left(\downarrow x \cap \downarrow U\right)$), and then actually computes the join. This corresponds to the intuitive idea that $x$ is updated with the information in $u_0$, subject to the constraint that the updated situation is in $U$. Of course, we may not add the information $u_0$ directly to $x$, either because $x$ and $u_0$ are not compatible, or because $x \sqcup u_0$ is not in $U$. So before adding $u_0$ we remove from $x$ the least possible information to make this operation possible. The definite set $U$ can be understood as specifying the least change to the situation $x$ such that the information in $u_0$ is entailed in the new situation and the information in $\uparrow u_0 - U$ is disentailed.

As we have seen, to define an update a definite set $U$ must satisfy the additional condition that $\downarrow U$ is $\sqcup$-closed. We next show that this condition holds for every definite subset if and only if the cc-poset is coherent. A subset $X$ of a poset $S$ is pairwise consistent if every two-element subset $\{x, y\}$ of $X$ has an upper bound in $S$, that is $x$ and $y$ are compatible. A poset $S$ is coherent if every pairwise consistent subset of $S$ is consistent.

Proposition 1.6. A cc-poset $S$ is coherent if and only if the down-closure of every definite subset of $S$ is $\sqcup$-closed.

Proof. Assume $S$ is coherent and let $U$ be a definite set. Let $X \subseteq \downarrow U$ be nonempty and consistent. If $u_0$ is the least element of $U$ then $x \sqcup u_0 \in U$ for every $x \in X$. This implies that $X \cup \{u_0\}$ is pairwise consistent, hence consistent, so has a least upper bound $\sqcup X \cup u_0$. But this is also the least upper bound of the set $\{x \sqcup u_0 \mid x \in X\} \subseteq U$, so $\sqcup X \cup u_0 \in U$ as $U$ is $\sqcup$-closed. We conclude that $\sqcup X \in \downarrow U$.

Now suppose $\downarrow U$ is $\sqcup$-closed for every definite subset $U$ of $S$. We show every finite pairwise consistent set is consistent, so the result holds for arbitrary sets by definition of consistency. For sets with cardinality up to two there is nothing to prove. Now consider
a pairwise consistent set $X$ with cardinality greater than two, and assume the result for all sets with smaller cardinality. Put $X = W \cup \{x\}$ with $x \not\in W$. The set $\uparrow x$ is definite, so $\downarrow(\uparrow x)$ is $\sqsubseteq$-closed, by hypothesis. Now $\downarrow(\uparrow x)$ is the set of all elements compatible with $x$, hence $W \subseteq \downarrow(\uparrow x)$, as $X$ is pairwise consistent. By induction hypothesis and because $\downarrow(\uparrow x)$ is $\sqsubseteq$-closed, $\sqcup W \in \downarrow(\uparrow x)$. But this means that $\sqcup W$ and $x$ are compatible, so $X$ is consistent. \(\square\)

This result shows that in coherent cc-posets we can identify updates with definite sets. In practice it is much more convenient to work with definite sets than with updates, and this is the reason why a situation space is required to be coherent.

A typical illustration of this relation between coherence, updates and definite sets is depicted in Fig. 1.

There we have a non-coherent cc-poset, as the set $\{x', x'', u\}$ is pairwise consistent but not consistent. (In terms of the example in the beginning of this section, this space is isomorphic to the set of all situations on the set of atoms $\{p, q, r\}$ with integrity constraint $\leftarrow p, q, r$.) The set $U = \{u, u', u''\}$ is a definite set; however, we can see that it does not correspond to an update function. If we try to update $x$ with $U$, we have to lose information because $x$ is not compatible with $U$, but there are two equally likely candidates for the needed restriction, $x'$ and $x''$, resulting in two equally likely candidates $u'$ and $u''$ for the update. The reason for this is that $\sqcup U$ is not $\sqsubseteq$-closed, as $x' \sqcup x'' \not\in \sqcup U$.

In the sequel we shall define two operations on updates, namely the sequential and synchronous compositions. In order that these operations enjoy “good” properties, additional requirements on the situation space must be made, like distributivity. It turns out that prime algebraicity (that implies distributivity) satisfies all our needs, and since anyway it is a very “natural” condition, we shall make it part of the definition of situation space. Intuitively, prime algebraicity means that every situation is composed by “indivisible” pieces of information, which are formalized in our setting by the complete primes of a cc-poset.
A complete prime of a cc-poset is an element \( p \) such that, whenever \( p \subseteq \bigcup X \) for some consistent set \( X \), then \( p \subseteq x \) for some \( x \in X \). A cc-poset is prime algebraic if every element is the join of the complete primes below it (that is, entailed by it). Let us consider the intervals example, with the atoms \( w \leq n \), \( w \geq n \) and \( w = n \) for variables \( w \) and numbers \( n \). The complete primes of the corresponding situation space are the entailment closures of the sets \( \{w \leq n\} \) and \( \{w \geq n\} \) for some \( w \) and \( n \). Note that the entailment closure of \( \{w = n\} \) is not a complete prime, since it is the join of the two previously mentioned situations, but not entailed by any of them. It is clear that any situation is the least upper bound of the set of all complete prime situations below it.

The definition of prime algebraicity provides a very powerful proof technique for showing that two situations are equal - just show that they entail the same complete primes. More generally, \( x \) is below \( y \) if and only if every complete prime below \( x \) is below \( y \). We illustrate the technique by showing that a prime algebraic cc-poset is distributive, that is, \( x \cap \bigcup X = \bigcup \{x \cap y \mid y \in X\} \) for any element \( x \) and consistent set \( X \). Indeed, we always have \( \bigcup \{x \cap y \mid y \in X\} \subseteq x \cap \bigcup X \). Conversely, if \( p \) is prime and \( p \subseteq x \cap \bigcup X \) then \( p \subseteq x \) and \( p \subseteq \bigcup X \), so that \( p \subseteq y \) for some \( y \in X \); it follows that \( p \subseteq x \cap y \), hence \( p \subseteq \bigcup \{x \cap y \mid y \in X\} \). As the cc-poset is prime algebraic, we conclude that \( x \cap \bigcup X = \bigcup \{x \cap y \mid y \in X\} \).

**Definition 1.7 (Situation space).** A situation space is a consistently complete, coherent and prime algebraic poset.

We now proceed to show that, with an appropriate order, the set of all definite subsets of a situation space is itself a situation space. Let \( \mathcal{S} \) be a situation space and \( \mathcal{U} \) the set of all definite subsets of \( \mathcal{S} \). If \( U \) and \( V \) are definite subsets of \( \mathcal{S} \), we write \( U \subseteq V \) if \( V \subseteq U \). When definite sets are viewed as updates, this order means intuitively that \( U \) constrains the updated situations less than \( V \). (This interpretation is especially useful with respect to the synchronous composition of updates.) As \( \mathcal{U} \) is also ordered by inclusion, when there is risk of confusion we shall make explicit the order we have in mind. We say two definite sets are compatible if they have a nonempty intersection. Equivalently, \( U \) and \( V \) are compatible if their least elements \( u_0 \) and \( v_0 \) are compatible in \( \mathcal{S} \) and their join \( u_0 \sqcup v_0 \) is in \( U \cap V \). (Indeed, if this property holds then \( U \cap V \neq \emptyset \); conversely, if \( x \in U \cap V \) then \( u_0 \) and \( v_0 \) are compatible as they are bound by \( x \) and \( u_0 \sqcup v_0 \) is in \( U \cap V \) by convexity.)

**Lemma 1.8.** Let \( \{U_i \mid i \in I\} \) be a nonempty set of definite subsets of \( \mathcal{S} \). If \( U_i \) and \( U_j \) are compatible for all \( i, j \in I \) then \( \bigcap_{i \in I} U_i \) is a definite set.

**Proof.** For every \( i \in I \), let \( u_0 \) be the least element of \( U_i \). By hypothesis, the set \( \{u_0 \mid i \in I\} \) is pairwise consistent, hence consistent. It follows that, for every \( i \in I \), the set \( \{u_0 \sqcup u_j \mid j \in I\} \subseteq U_i \) is also consistent, so its least upper bound is in \( U_i \). But the least upper bound is \( \bigcup_{j \in I} u_0 \), which therefore is in \( \bigcap_{i \in I} U_i \), and is clearly its least element. Convexity and \( \sqcup \)-closure are immediate. \( \square \)
Together with the fact that $\mathcal{S}$ is the $\subseteq$-least definite set, this lemma shows that $(\mathcal{U}, \subseteq)$ is a coherent cc-poset, where the least upper bound of a family of definite sets is its intersection. (Note that it also shows that compatibility of definite sets is in fact compatibility with respect to $\subseteq$, because a nonempty intersection of definite sets is a definite set.) We now proceed to prove that $(\mathcal{U}, \subseteq)$ is prime algebraic.

Let us denote the complement $\mathcal{S} - \uparrow x$ by $\uparrow x$. Clearly, we have $X \subseteq \uparrow x$ if and only if $X \cap \uparrow x = \emptyset$. Note that if $p \in \mathcal{S}$ is a complete prime then $\uparrow p$ is a definite set. Indeed, its least element is $\bot$ and it is clearly convex; on the other hand, if $X \subseteq \uparrow p$ is consistent and nonempty and $\bigcup X \not\subseteq \uparrow p$ then $p \subseteq \bigcup X$; but then $p \subseteq x$ for some $x \in X$, so $x \not\in \uparrow p$, contradicting the assumption that $X$ is contained in $\uparrow p$.

**Proposition 1.9.** Let $\mathcal{S}$ be a situation space, $p$ a complete prime of $\mathcal{S}$ and $U$ a definite subset of $\mathcal{S}$.

(i) $\uparrow p$ and $\uparrow p$ are complete primes of $(\mathcal{U}, \subseteq)$.

(ii) $U = \bigcap \{\uparrow p \mid U \subseteq \uparrow p\} \cap \bigcap \{\uparrow p \mid U \subseteq \uparrow p\}$, where each $p$ is a complete prime of $\mathcal{S}$.

(iii) Every complete prime of $(\mathcal{U}, \subseteq)$ has the form $\uparrow p$ or $\uparrow p$ for a complete prime $p$ of $\mathcal{S}$.

Therefore, $(\mathcal{U}, \subseteq)$ is prime algebraic, hence a situation space.

**Proof.** (i) Let $\{U_i \mid i \in I\}$ be a consistent set of definite subsets of $\mathcal{S}$ such that $\bigcap_i U_i \subseteq \uparrow p$. If $u_{i_0}$ is the least element of $U_i$ for each $i \in I$ then $\bigcup_i u_{i_0} \in \uparrow p$, that is, $p \subseteq \bigcup_i u_{i_0}$. As $p$ is a complete prime we have $p \subseteq u_{i_0}$ for some $i$, hence $U_i \subseteq \uparrow p$. This shows $\uparrow p$ is a complete prime of $(\mathcal{U}, \subseteq)$. Next consider a set $\{U_i \mid i \in I\}$ as before, but this time suppose that $\bigcap_i U_i \subseteq \uparrow p$. If $U_i \not\subseteq \uparrow p$ for every $i$, the set $U_i \cap \uparrow p$ is not empty, so $p \subseteq u_{i_0} \in U_i$ (because $U_i$ is convex). The set $\{u_{i_0} \mid i \in I\} \cup \{p\}$ is consistent because it is pairwise consistent, and so is $\{p \cup u_{i_0}, \bigcup_i u_{i_0}\} \subseteq U_i$. It follows that $p \subseteq \bigcup_i u_{i_0} \subseteq U_i$. As this is true for every $i \in I$, we conclude that $p \subseteq \bigcup_i u_{i_0} \in \bigcap_i U_i$, contradicting the assumption that $\bigcap_i U_i \subseteq \uparrow p$.

(ii) The set $\bigcap \{\uparrow p \mid U \subseteq \uparrow p\}$ is up-closed and, as in the proof of the previous lemma, its least element $u$ is the join of the complete primes $p$ such that $U \subseteq \uparrow p$. But $U \subseteq \uparrow p$ if and only if $p \subseteq u_0$, where $u_0$ is the least element of $U$, so $u$ and $u_0$ entail the same complete primes. It follows that $u = u_0$ and therefore $\bigcap \{\uparrow p \mid U \subseteq \uparrow p\} = \uparrow u_0 = \uparrow U$. We next show that $\bigcap \{\uparrow p \mid U \subseteq \uparrow p = \emptyset\} = \downarrow U$. The set $\bigcap \{\uparrow p \mid U \subseteq \uparrow p = \emptyset\}$ is obviously down-closed and contains $U$, so it contains $\downarrow U$. If it contained an element $x$ not in $\downarrow U$, it contained some complete prime $p \subseteq x$ not in $\downarrow U$ (indeed, if $\downarrow U$ contained every complete prime below $x$ then it would also contain $x$, as it is $\downarrow$-closed). But then we would have $U \subseteq \uparrow p = \emptyset$ and $x \not\in \uparrow p$, contradicting the assumption that $x \in \bigcap \{\uparrow p \mid U \subseteq \uparrow p \} = \emptyset$.

(iii) If $U$ is a complete prime in $(\mathcal{U}, \subseteq)$ then, by the previous statement (ii), $\uparrow p \subseteq U$ or $\uparrow p \subseteq U$ for some complete prime $p$ in $S$ such that $U \subseteq \uparrow p$ or $U \subseteq \uparrow p$, respectively. Thus we have either $U = \uparrow p$ or $U = \uparrow p$. □
This result has a very intuitive interpretation. The first and third statements together indicate that the indivisible updates (identified here with definite sets) are those that force the entailment or disentailment of an indivisible piece of information in $S$. The middle statement shows that any update is completely characterized by the indivisible pieces of information which it forces a situation to entail or disentail.

It is useful to think of the complete primes as atomic propositions, with $p \sqsubseteq q$ meaning that $q$ entails $p$, and with $\{p, q\}$ inconsistent if $p$ and $q$ have no upper bound in $S$. The space $S$ is then isomorphic to the set of all consistent and entailment-closed sets of complete primes, and we can take advantage of the intuition provided by the examples at the beginning of this section. Using the notation of the examples, the updates $!p$ and $!\neg p$ correspond to the definite sets $\uparrow p$ and $\uparrow \neg p$. The update associated with $U$ is then the synchronous composition (to be precisely defined below) of the $!p$ such that $U \subseteq \uparrow p$ with the $!\neg q$ such that $U \subseteq \uparrow \neg q$.

2. Operations on updates

For the rest of the paper, and unless explicitly stated otherwise, we shall assume that $S$ is a situation space. We know that definite subsets $U$ of $S$ can be identified with updates, so we often call $U$ itself an update and denote by $x \triangleright U$ the update of $x \in S$ by $U$. The associated restriction with image $\downarrow U$ will be denoted $x \upharpoonright U$, and called the restriction of $x$ to $U$, so that $x \upharpoonright U = \bigsqcup (\downarrow \cap \downarrow U)$. The update can then be written

$$x \triangleright U = (x \upharpoonright U) \cup u_0,$$

where $u_0$ is the least element of $U$. It is interesting to note that the restriction can be defined in terms of the update by

$$x \upharpoonright U = x \cap (x \triangleright U).$$

Indeed, $x \upharpoonright U \subseteq x$ and $x \upharpoonright U \subseteq x \triangleright U$, so $x \upharpoonright U \subseteq x \cap (x \triangleright U)$. But $x \cap (x \triangleright U) \subseteq x$ and $x \cap (x \triangleright U) \subseteq \downarrow U$, hence $x \cap (x \triangleright U) \subseteq x \upharpoonright U$.

Our immediate purpose is to define a sequential composition of updates, but first we need an auxiliary result.

**Lemma 2.1.** Let $U$ be a definite subset of $S$. For all $x, w, y \in S$:

(i) If $x \subseteq y$ then $x \upharpoonright U = x \cap y \upharpoonright U$.

(ii) If $x \subseteq y$ and $x \upharpoonright U \subseteq w \subseteq y \upharpoonright U$ then $w = v \upharpoonright U$ for some $v \in S$ such that $x \subseteq v \subseteq y$.

(iii) If $X \subseteq S$ is consistent then $(\bigsqcup X) \upharpoonright U = \bigsqcup \{x \upharpoonright U \mid x \in X\}$.

**Proof.** (i) We have $x \upharpoonright U = \bigsqcup \{v \mid v \subseteq x, v \subseteq \downarrow U\}$ and $x \cap y \upharpoonright U = \bigsqcup \{x \cap w \mid w \subseteq y, w \subseteq \downarrow U\}$, by definition of restriction and distributivity. The desired equality follows if we show that the sets under the sign $\bigsqcup$ are equal. But if $v$ is in the first set, it is also
in the second, since it has the form \( x \sqcap w \) with \( w = v \). Conversely, if \( x \sqcap w \) is in the second set then it is in the first, since \( \downarrow U \) is down-closed.

(ii) Clearly, \( x \sqsubseteq x \sqcup w \sqsubseteq y \). Put \( v = x \sqcup w \). By the previous result, \( v \uparrow U = v \sqcap y \uparrow U = (x \sqcap y) \uparrow U \sqcup (w \sqcap y) \uparrow U = x \uparrow U \sqcup w = w \).

(iii) Put \( z = \bigcup X \). As \( \bigcup X \subseteq z \) we have, by the first statement, \( (\bigcup X) \uparrow U = (\bigcup X) \sqcap z \uparrow U = (\bigcup \{x \uparrow U \mid x \in X\}) \sqcup (z \uparrow U) = \bigcup \{x \uparrow U \mid x \in X\} \).

The key result to define a sequential composition of updates is the following.

Lemma 2.2. If \( U \) and \( V \) are definite subsets of \( \mathcal{S} \) then \( W = \{ x \triangleright V \mid x \in U \} \) is also definite.

Proof. Let \( u_0 \) and \( v_0 \) be the least elements of \( U \) and \( V \), respectively. Clearly, the least element of \( W \) is \( w_0 = u_0 \triangleright V = u_0 \sqcup V \).

To see that \( W \) is convex, we assume that \( w_0 \subseteq z \subseteq x \triangleright V \) for some \( z \in \mathcal{S} \) and \( x \in U \), and show that \( z = y \triangleright V \) for some \( y \in U \). As \( u_0 \sqcup V \subseteq w_0 \subseteq z \), we conclude that \( u_0 \sqcup V \subseteq z \subseteq x \triangleright V \subseteq x \sqcup V \). By Lemma 2.1, there is \( y \in \mathcal{S} \) such that \( u_0 \subseteq y \subseteq x \) and \( y \sqcup V = z \sqcup x \sqcup V \). As \( U \) is convex, \( y \in U \), so we only have to show that \( y \triangleright V = z \), that is, \( (z \sqcup x \sqcup V) \sqcup v_0 = z \).

We now show that \( W \) contains the least upper bounds of its nonempty subsets that are consistent in \( \mathcal{S} \). Take one such subset, which may be given as the set of all \( x \triangleright V \) with \( x \in X \), for an appropriate subset \( X \) of \( U \). We must show that \( \bigcup \{x \triangleright V \mid x \in X\} \) is in \( W \).

First note that we may assume that each \( x \in X \) satisfies \( x = x \sqcup V \sqcup u_0 \). Indeed, put \( x' = x \sqcup V \sqcup u_0 \). We have \( u_0 \subseteq x' \subseteq x \), hence \( x' \in U \) because \( U \) is convex. But we also have \( x \sqcup V \subseteq x' \subseteq x \), hence \( x' \triangleright V = x \sqcup V \), so in particular, \( x \triangleright V = x' \triangleright V \). Thus, if we have \( x \neq x' \), we just replace \( x \) with \( x' \).

Let \( s \) be the least upper bound of the \( x \triangleright V \) for \( x \in X \), that must be shown to be in \( W \). As each \( x \sqcup V \) is bounded by \( x \triangleright V \), the set of the \( x \sqcup V \) for \( x \in X \) is consistent, and so has a least upper bound \( t \subseteq s \). We have \( t \sqcup u_0 = s \), as \( t \sqcup v_0 \) is the least upper bound of the \( x \sqcup V \sqcup v_0 = x \triangleright V \) for \( x \in X \). To end the proof we only need to find \( u \in U \) such that \( u \sqcup V = t \).

We first show that \( X \) is consistent. Indeed, let \( x, y \in X \). The elements \( u_0, x \sqcup V \) and \( y \sqcup V \) are pairwise compatible, as \( u_0 \) and \( x \sqcup V \) are bounded by \( x, u_0 \) and \( y \sqcup V \) by \( y \), and \( x \sqcup V \) and \( y \sqcup V \) by \( t \). As \( \mathcal{S} \) is coherent, \( \{u_0, x \sqcup V, y \sqcup V\} \) is consistent, and so is \( \{x, y\} \), because \( x = x \sqcup V \sqcup u_0 \) and \( y = y \sqcup V \sqcup u_0 \), by our hypothesis on the elements of \( X \). Since every pair of elements of \( X \) is consistent, \( X \) is consistent, once again by the coherence of \( \mathcal{S} \).

Let \( u \) be the least upper bound of \( X \), which is in \( U \) because \( U \) is definite. As every element \( x \) of \( X \) has the form \( x \sqcup V \sqcup u_0 \), we conclude that \( u = t \sqcup u_0 \). Since \( t \) is compatible with \( v_0 \), because \( t \sqcup u_0 = s \), we have \( t \sqcup V = t \). By the third statement of Lemma 2.1, \( u \sqcup V = (t \sqcup V) \sqcup (u_0 \sqcup V) = t \sqcup (u_0 \sqcup V) = t \), as required. \( \Box \)
The definite set $W$ defined in this lemma is the image of $U$ by the mapping $x \mapsto x \triangleright V$. In accordance with a well-established usage, we denote this set by $U \triangleright V$ and call it the sequential composition of $U$ and $V$.

**Definition 2.3 (Sequential composition).** The sequential composition of definite sets (or updates) $U$ and $V$ is the definite set (update) $U \triangleright V$.

This operation satisfies several useful properties. The hardest to prove is associativity, for which we first state and prove an auxiliary lemma.

**Lemma 2.4.** Let $U$ and $V$ be definite subsets of $S$ and $x \in S$. We have

(i) $(x[U])[V = (x[U]) \cap (x[V]) \subseteq x[U \triangleright V) \subseteq x[V]$. 

(ii) $x \triangleright (U \triangleright V) = (x \triangleright U) \triangleright V$.

**Proof.** (i) As $x[U] \subseteq x$ we have, by Lemma 2.1, $(x[U])[V = (x[U]) \cap (x[V])$.

Let $u_0$ and $v_0$ be the least elements of $U$ and $V$, respectively, so that $u_0[V \sqcup v_0$ is the least element of $U \triangleright V$. By Lemma 2.1,

$$x \triangleright U) \triangleright V = (x \triangleright U \sqcup u_0)[V \sqcup v_0 = (x[U])[V \sqcup u_0[V \sqcup v_0.$$

Thus, $(x[U])[V$ is below $x$ and its join with $u_0[V \sqcup v_0$ is in $U \triangleright V$, hence $(x[U])[V \subseteq x[U \triangleright V]$. 

From

$$v_0 \subseteq x(U \triangleright V) \sqcup v_0 \subseteq x(U \triangleright V) \sqcup u_0[V \sqcup v_0 \subseteq U \triangleright V \subseteq V$$

we conclude that $x(U \triangleright V) \sqcup v_0 \subseteq V$, hence $x(U \triangleright V) \subseteq x[V]$. 

(ii) As in the proof of (i), we may write

$$(x \triangleright U) \triangleright V = (x[U])[V \sqcup u_0[V \sqcup v_0,$$

$$x \triangleright (U \triangleright V) = x[(U \triangleright V) \sqcup u_0[V \sqcup v_0.$$

Again by (i), we conclude that $(x \triangleright U) \triangleright V \subseteq x \triangleright (U \triangleright V)$. We prove the converse inequality by showing that every complete prime $p \subseteq x \triangleright (U \triangleright V)$ satisfies $p \subseteq (x \triangleright U) \triangleright V$. As $p$ is a complete prime, we have $p \subseteq x[(U \triangleright V)$ or $p \subseteq u_0[V \sqcup v_0$. Clearly, we may assume that $p \subseteq u_0[V \sqcup v_0$. As $u_0[V \sqcup v_0 \subseteq p \sqcup u_0[V \sqcup v_0 \subseteq x \triangleright (U \triangleright V)$, we conclude that $p \sqcup u_0[V \sqcup v_0$ is in $U \triangleright V$. Thus, there is $y \in U$ such that $p \sqcup u_0[V \sqcup v_0 = y[V \sqcup v_0$, and, in particular, $p \subseteq y[V \sqcup v_0$. But $p \not\subseteq v_0$, so we must have $p \subseteq y[V$, hence $p$ is compatible with $u_0$. As $p \subseteq x[(U \triangleright V) \subseteq x$, it follows that $p \subseteq x[U$. On the other hand, by (i), $p \subseteq x[(U \triangleright V) \subseteq x[V$, so that finally $p \subseteq (x[U]) \cap (x[V) \subseteq (x \triangleright U) \triangleright V$, again by (i). $\square$
The following are some simple properties satisfied by the sequential composition. A more interesting property appears in Proposition 2.8.

**Proposition 2.5.** Sequential composition of updates satisfies the following properties:

(i) **Associativity:** \( U \circ (V \circ W) = (U \circ V) \circ W. \)

(ii) **Identity law:** \( \emptyset \circ U = U \circ \emptyset = U. \)

(iii) **Idempotency:** \( U \circ U = U. \)

**Proof.** Only associativity needs to be considered. We calculate

\[
U \circ (V \circ W) = \{ x \in \emptyset \circ (V \circ W) \mid x \in U \} \\
= \{ (x \in V) \circ W \mid x \in U \} \\
= \{ y \circ W \mid y \in U \circ V \} \\
= (U \circ V) \circ W,
\]

by Lemma 2.4. \( \square \)

The next result shows that for any two situations there is a \( \subseteq \)-least update that transforms one in the other.

**Proposition 2.6.** If \( s, t \in \mathcal{F} \), there is a \( \subseteq \)-least definite set \( U \) such that \( s \circ U = t \), given by

\[
U = \{ x \in \mathcal{F} \mid x \cap s \subseteq t \subseteq x \cup (s \cap t) \}
\]

with least element

\[
u_0 = \bigsqcup \{ p \in \mathcal{F} \mid p \text{ complete prime, } p \subseteq t, p \nsubseteq s \}.
\]

**Proof.** Let us check that \( u_0 \) is the least element of \( U \). First we show that \( u_0 \in U \), that is, \( u_0 \cap s \subseteq t \subseteq u_0 \cup (s \cap t) \). Any complete prime \( p \) below \( u_0 \cap s \) is below \( u_0 \). By definition of \( u_0 \), and because \( p \) is a complete prime, we conclude that \( p \) is below some complete prime below \( t \), so \( p \) is itself below \( t \). This proves the first inequality. Next, if \( p \) is a complete prime below \( t \) then either \( p \nsubseteq s \) or \( p \nsubseteq s \). In the first case, \( p \) is below \( s \cap t \), and in the second, below \( u_0 \), so in either case it is below \( u_0 \cup (s \cap t) \) and the second inequality is proved. Finally, let \( x \in U \). If \( p \) is a complete prime such that \( p \subseteq t \) and \( p \nsubseteq s \) then \( p \subseteq x \cup (s \cap t) \) and \( p \nsubseteq s \), so we must have \( p \subseteq x \). This shows \( u_0 \subseteq x \).

To see \( U \) is convex suppose \( u_0 \subseteq x \subseteq y \in U \). We have

\[
x \cap s \subseteq y \cap s \subseteq t \subseteq u_0 \cup (s \cap t) \subseteq x \cup (s \cap t),
\]

hence \( x \in U \). Now suppose \( X \subseteq U \) is nonempty and consistent. Again,

\[
(\bigsqcup X) \cap s = \bigsqcup \{ x \cap s \mid x \in X \} \subseteq u_0 \cup (s \cap t) \subseteq (\bigsqcup X) \cup (s \cap t),
\]

so \( \bigsqcup X \in U \). We conclude that \( U \) is a definite set.
We now show that \( s \triangleright U = t \). Clearly, \( t \in U \). We have \((s \cap t) \cup u_0 = t\) because, for every complete prime \( p \), \( p \in t \) if and only if \( p \not\in t \) and \( p \in s \), or \( p \not\in t \) and \( p \not\in s \), if and only if \( p \not\in (s \cap t) \cup u_0 \). As \( s \cap t \subseteq s \), we conclude that \( s \cap t \subseteq s[U \). Conversely, \( s \triangleright U \in U \) implies \( s[U \subseteq s \cap U \subseteq t \), hence \( s[U \subseteq s \cap t \). We thus have \( s[U = s \cap t \), so that \( s \triangleright U = s[U \cup u_0 = (s \cap t) \cup u_0 = t \).

Finally, suppose \( V \) is another definite set such that \( s \triangleright V = t \). Note that, by the previous lemma, \( s[V = s \cap t \). We prove that every \( x \in V \) is in \( U \) by showing that \( x \cap s \subseteq t \cap (s \cap t) \). Since \( x \cap s \subseteq s \subseteq x \in V \), we conclude that \( x \cap s \subseteq s[V \), hence \( x \cap s \subseteq t \). On the other hand, if \( v_0 \) is the least element of \( V \), \( t = s[V \cup v_0 = (s \cap t) \cup v_0 \subseteq x \cup (s \cap t) \). Thus, \( V \subseteq U \), hence \( U \subseteq V \), as required. \( \square \)

The next result shows that if a definite set has the same image by two updates then the updates have the same restriction to the definite set – a property not shared by arbitrary functions.

**Lemma 2.7.** If \( U \), \( V \) and \( W \) are definite sets such that \( U \triangleright V = U \triangleright W \) then \( x \triangleright V = x \triangleright W \) for every \( x \in U \).

**Proof.** If \( x \in U \) then \( x = x \triangleright U, \) hence \( x \triangleright V = (x \triangleright U) \triangleright V = x \triangleright (U \triangleright V). \) Similarly, \( x \triangleright W = x \triangleright (U \triangleright W). \) \( \square \)

The following property of sequential composition is somewhat unexpected. It basically states that from \( U \) and \( U \triangleright V \) we cannot recover \( V \), but we can find a \( \leq \)-least \( W \) that has the same effect on \( U \) and all \( \subseteq \)-greater definite sets.

**Proposition 2.8.** If \( U \) and \( V \) are definite sets, there is a \( \leq \)-least definite set \( W \) such that \( U \triangleright W = U \triangleright V \). For that \( W \) we have \( Z \triangleright W = Z \triangleright V \) for every definite set \( Z \subseteq U \).

**Proof.** We show that there is a least definite set \( W \) having the same restriction to \( U \) as \( V \). Both statements then follow from the previous lemma. For every \( x \in U \), let \( W_x \) be the least definite set such that \( x \triangleright W_x = x \triangleright V \), which exists by Proposition 2.6. In particular, \( V \subseteq W_x \), so \( W = \bigcap_{x \in U} W_x \) is not empty. If a definite set \( Z \) satisfies \( x \triangleright Z = x \triangleright V \) for every \( x \in U \) then \( Z \subseteq W_x \) for every \( x \in U \), hence \( Z \subseteq W \). This shows that if \( W \) has the same restriction to \( U \) as \( V \) then it is the least such set.

Let us show that \( x \triangleright W = x \triangleright V \) for every \( x \in U \). Let \( w_{x_0} \) be the least element of \( W_x \) and \( w_0 \) the least element of \( W \), which is the join of the \( w_{x_0} \). By Proposition 2.6, \( x[W_x = x \cap x \triangleright V \) and \( x \triangleright V = x \triangleright W_x \triangleright w_{x_0} \cup (x \cap x \triangleright V) \subseteq w_0 \cup (x \cap x \triangleright V). \) On the other hand, \( x \triangleright V \subseteq W_x \), so \( w_0 \subseteq x \triangleright V \) and \( w_0 \cup (x \cap x \triangleright V) \subseteq x \triangleright V \). It follows that \( x \triangleright V = w_0 \cup (x \cap x \triangleright V), \) which also shows that \( x \cap x \triangleright V \subseteq x[W \). To conclude the proof we show that \( x[W \subseteq x \cap x \triangleright V \). But we have \( x[W \subseteq x \) and \( x[W \cup w_0 \subseteq W \subseteq W_x \), hence \( x[W \subseteq x \cap x \triangleright V \), as required. \( \square \)
We now define the synchronous composition of two updates, which combines in one the effect of the two updates together. In order to do this, however, the updates must be compatible.

**Definition 2.9 (Synchronous composition).** The synchronous composition of two compatible definite sets (or updates) $U$ and $V$ is their intersection $U \cap V$.

The synchronous composition satisfies all properties of (nonempty) intersection. What is not so obvious is that it coincides with sequential composition of compatible definite sets (in either order).

**Lemma 2.10.** Let $U$ and $V$ compatible definite subsets of $\mathcal{S}$. For every $x \in \mathcal{S}$,

- $x \triangleright (U \cap V) = (x \triangleright U) \cap (x \triangleright V) \cup u_0 \cup v_0$,
- $x \triangleright (U \cap V) = x \triangleright (U \triangleright V) = x \triangleright (V \triangleright U) = (x \triangleright U) \cap (x \triangleright V)$,
- $x \triangleright (U \cap V) = x \triangleright (U \triangleright V) = x \triangleright (V \triangleright U)$.

**Proof.** The least element of $U \cap V$ is $u_0 \cup v_0$, so the first equality follows if we show that $x \triangleright (U \cap V) = (x \triangleright U) \cap (x \triangleright V)$. If $y \subseteq x \triangleright (U \cap V)$ then $y$ is below $x$ and the join $y \cup u_0 \cup v_0$ exists and is in $U \cap V$. It follows that $y \cup u_0$ is in $U$ and $y \cup v_0$ is in $V$, so that $y \subseteq x \triangleright U$ and $y \subseteq x \triangleright V$. Conversely, suppose $y$ is below $x$, $y \cup u_0 \in U$ and $y \cup v_0 \in V$. The set $\{y, u_0, v_0\}$ is pairwise consistent, so is consistent. As $u_0 \cup v_0$ is in $U$ and in $V$, and these sets are closed for joins of consistent sets, we conclude that $y \cup u_0 \cup v_0$ is in $U \cap V$. This shows that $y \subseteq x \triangleright (U \cap V)$.

To prove the second equalities we only need to show, by Lemma 2.4, that $x \triangleright (U \triangleright V) \subseteq x \triangleright U$. Suppose $p$ is a complete prime and $p \subseteq x \triangleright (U \triangleright V)$. Then $p$ is below $x$ and $p \cup u_0 \cup v_0$ is in $U \triangleright V$ (note that $u_0 \triangleright V = u_0$, as $u_0 \in \downarrow V$). To conclude that $p \subseteq x \triangleright U$ we must show that $p \cup u_0 \in U$. By definition of $U \triangleright V$, there is $y \in U$ such that $y \triangleright V \cup v_0 = p \cup u_0 \cup v_0$, so that $p \subseteq y \triangleright V \cup v_0$. As $p$ is a complete prime, we have $p \subseteq y \triangleright V$ or $p \subseteq v_0$. In the first case, $u_0 \subseteq p \cup u_0 \subseteq y$, so $p \cup u_0 \in U$. In the second case, $u_0 \subseteq p \cup u_0 \subseteq u_0 \cup v_0$, and the conclusion is the same.

The remaining statements are immediate. □

**Proposition 2.11.** If $U$ and $V$ are compatible definite subsets of $\mathcal{S}$, then $U \cap V = U \triangleright V = V \triangleright U$.

**Proof.** This is a direct consequence of the previous lemma. □

3. Actions

Our methodological program is to consider that processes are built by suitable composition operations from a basic layer of *actions*, these being the source of atomic
steps in the operational semantics, corresponding to transitions between situations. We must therefore ask ourselves what is a suitable general notion of action, capable of serving as the basis for constructing general processes.

The updates, as characterized in the previous sections, correspond to all possible unconditional determinate actions, i.e. those that are applicable in any situation and have a unique result. In fact, we know that any transition between two situations can be achieved by an update. But to the same transition may correspond several updates, the possible functions combining restrictions and expansions that capture the transition. This is an important aspect of actions, intensionality, which makes them more discriminating than their extensional manifestation as situation transformers. We have seen that updates can be composed synchronously, thereby exposing their intensional character, and this turns out to be a useful operation to have.

But we also know that for processes we need to consider other things beyond updates, to express what they cannot: conditionality and nondeterminism.

Conditional actions are essential for giving an agent the ability to react to its environment in ways that depend on the state of that same environment, and thus indirectly on the actions of other agents sharing the environment. The simplest form of conditional action is a query, whose execution has no effect on the environment but can only take place if the environment satisfies some condition. In our setting a query action corresponds to specifying the set of situations where it can execute. Such an action is useful by virtue of its possible composition with other actions, either sequentially or synchronously. Sequential composition allows an agent to delay starting a certain process until some condition is met. The synchronous composition of a query and an update is a well-known device for concurrent control of critical resources. For example, in the theater scenario introduced earlier, a ticket booking involves synchronizing a query on available with an update on booked.

The other important feature we want to capture in actions is nondeterminism, in its external and internal varieties. External nondeterminism corresponds to the ability to take different courses of action depending on the state of the environment. Internal nondeterminism allows for different courses of action to be taken from one given state. Thus we are led to formalize an action as a relation between situations and updates, i.e. a relation on $\mathcal{S} \times \mathcal{U}$. External nondeterminism is reflected in the fact that different situations can be associated with different updates, and internal nondeterminism in that distinct updates may be associated with the same situation. For example, the action $mpc(y)$ in the theater example contains all pairs $(s, U)$ where $available(x) \in s$ and $U = \{ t \mid available(x) \not\in t, booked(x, y) \in t \}$ for some $x$. Note that this action is conditional (not executable if no places are available), has external nondeterminism (cannot book a place that is not available) and internal nondeterminism (can book any place available).

Are all relations on $\mathcal{S} \times \mathcal{U}$ to be considered distinct actions? The criterion should be one of full abstractness relative to observable behaviour under all contexts. We must, of course, define in precise terms the notions of observability and context.
An action is operationally associated with a transition from one situation to another (possibly the same). The existence of a changing situation whose transitions are the byproduct of the execution of actions (inside processes) give us a very natural notion of observability, which is truly detached from the source of causality: what we observe of an action are the possible situation transitions caused by its execution. This characterization of observability through an entity which is external to the processes is a defining feature of our framework, and more generally of the shared data space model of coordination, in contrast to approaches such as the $\Pi$-calculus.

What contexts for actions must we consider? In general they should be the processes, but it turns out that we can stay wholly within the realm of actions, by defining suitably general ways of combining actions to form other actions. If two actions are distinguishable by observing their behaviours in the context of a given process, there is also an action composition context that distinguishes them. In other words, all the compositional potential of processes emerges from the compositional power of actions themselves.

So what are the action composition operations? One that we have already pointed out is the synchronous composition. Intuitively, this means combining compatible updates for the same situation into a single update subsuming both. Another possible combined effect of actions is through sequential composition. If we execute two actions in sequence, we may think about how to achieve the same effects with a single action. From a process perspective this is equivalent to defining a sequential transaction, i.e. a process achieving in a single step the cumulative effects (without external interference) of a sequential process. We also need to build an action out of two others by simply adding together their individual possible behaviours. This form of composition is called choice, as the new action’s behaviours are chosen from those of the given actions. It turns out that these three forms of composition are all we need. For example, parallel composition (again in the transaction sense) can indeed be expressed through sequentiality and choice; this will become clearer towards the end of this section when we discuss action independence and true concurrency.

We are now in a position to define actions, knowing that we want full abstractness regarding the observation of transitions under all possible synchronous, sequential and choice compositions. It turns out that indeed not all relations on $\mathcal{S} \times \mathcal{U}$ can be distinguished as actions. To see why, consider a very simple example of situation space $\mathcal{S} = \{ \perp, \text{on}, \text{off} \}$ with information about a switch, $\{ \text{on}, \text{off} \}$ being inconsistent and $\text{on} \perp \text{off} = \perp$. Now consider the two "actions" $a = \{ (\text{on}, \{\text{on}\}), (\text{on}, \{\text{off}\}) \}$ and $a' = a \cup \{ (\text{on}, \mathcal{S}) \}$. Both give rise to the two transitions $\text{on} \rightarrow \text{on}$ and $\text{on} \rightarrow \text{off}$, so by themselves or by sequencing with other actions they are indistinguishable. Can a synchronous composition with some action $b$ make the distinction? This must involve a

\footnote{We could consider as observable only the transitions where there is a change of situation, i.e. excluding stuttering. The results for our characterization of actions would be the same, but the formal treatment a bit more cumbersome.}
pair \((on, U) \in b\) that, when composed with \((on, \mathcal{S})\) in \(a'\), originates a transition it does not when composing with \(a\). But this cannot be: if \(U\) contains \(on\), the composition with \((on, \mathcal{S})\) in \(a'\) gives rise to the single transition \(on \rightarrow on\), which appears also when composing \((on, U)\) with \((on, \{on\})\) in \(a\); if \(U\) contains \(off\), the composition with \((on, \mathcal{S})\) originates instead the transition \(on \rightarrow off\), but so does the composition with \((on, \{off\})\) in \(a\); finally, if \(U = \{
abla\}\), the transition \(on \rightarrow \nabla\) is generated in both compositions with \(a\) and \(a'\).

So we need to place some restriction on \(\mathcal{S} \times \mathcal{U}\) relations for them to be considered actions. The simplest solution is to opt for a closure condition, which roughly states that if any possible effect of an update \(U\) on a situation \(s\), in any synchronous composition context, can be similarly obtained from within an action \(a\), then \((s, U)\) must also be in \(a\). In the previous example \(a'\) is an action whereas \(a\) is not, given that the effects of \(\mathcal{S}\) on \(on\), in any synchronous context, are also obtainable from \(a\) (via \((on, \{on\})\)), but \((on, \mathcal{S})\) is not in \(a\).

**Definition 3.1 (Action).** Let \(a\) be a subset of \(\mathcal{S} \times \mathcal{U}\), and write \(a(s)\) for the set \(\{U | (s, U) \in a\}\), for any \(s \in \mathcal{S}\). A pair \((s, U) \in \mathcal{S} \times \mathcal{U}\) is covered by \(a\) if for every \(V \in \mathcal{U}\) compatible with \(U\) there is \(W \in a(s)\) compatible with \(V\) such that \(s \rightarrow (V \cap U) = s \rightarrow (V \cap W)\). If every \((s, U)\) covered by \(a\) exists in \(a\), then \(a\) is called an action. We denote by \(\mathcal{A}\) the set of all actions.

Being sets, actions are naturally ordered by inclusion and we may perform on them set-theoretic operations.

**Lemma 3.2.** If \((a_i)_{i \in I}\) is a nonempty family of actions, the intersection \(\bigcap_{i \in I} a_i\) is an action.

**Proof.** If \(a = \bigcap_{i \in I} a_i\) is not an action, there is a pair \((s, U)\) covered by \(a\) that is not in \(a\). That pair must be missing from some action \(a_j\), but being covered by \(a \subseteq a_j\) it is also covered by \(a_j\), a contradiction. \(\square\)

We now formalize observability.

**Definition 3.3 (Observables).** If \(a\) is a subset of \(\mathcal{S} \times \mathcal{U}\), not necessarily an action, we define

\[ O(a) = \{(s, s \triangleright U) : (s, U) \in a\} \]

to be the set of observables of \(a\).

We have claimed that actions have sufficient generality, and the following two lemmas establish results in that direction. The first shows that actions can capture any conditional nondeterministic behaviour.

**Lemma 3.4.** If \(A \subseteq \mathcal{S} \times \mathcal{S}\) then \(a = \{(s, U) | (s, s \triangleright U) \in A\}\) is an action, and \(O(a) = A\).
Proof. Suppose \((s, U)\) is covered by \(a\). In particular, \(\mathcal{S}\) being compatible with \(U\), there is \(W \subset a(s)\) such that \(s \triangleright (\mathcal{S} \cap U) = s \triangleright (\mathcal{S} \cap W)\), that is, \(s \triangleright U = s \triangleright W\). We deduce that \((s, s \triangleright U) \in A\), so \((s, U) \in a\), i.e. \(a\) is an action. By construction, \(\mathcal{O}(a) \subseteq A\). To show that \(A \subseteq \mathcal{O}(a)\), consider any \((s, t) \in A\). By Proposition 2.6 there is an update \(U\) such that \(t = s \triangleright U\), so \((s, U) \in a\) and therefore \((s, s \triangleright U) = (s, t) \in \mathcal{O}(a)\).  

The next lemma shows that by restricting only certain relations on \(\mathcal{S} \times \mathcal{U}\) to be actions we do not leave out any relation whose observable behaviour is not captured by an action.

Lemma 3.5. For every \(a \subseteq \mathcal{S} \times \mathcal{U}\), there is a least action \(\bar{a} \supseteq a\) with respect to set inclusion such that \(\mathcal{O}(\bar{a}) = \mathcal{O}(a)\).

Proof. The set of actions that include \(a\) and have the same observables is not empty, by Lemma 3.4, so the required action is the intersection of them all (Lemma 3.2).

The following lemma will be useful for our full abstractness proof. It shows the kind of components of the least action corresponding to a single update applied in a single situation.

Lemma 3.6. If \(s \in \mathcal{S}\) and \(Z \in \mathcal{U}\), then \(\{(s, Z)\} \subseteq \{(s, W) \mid W \in \mathcal{U}, W \subseteq Z\}\).

Proof. By definition of \(\{(s, Z)\}\) (Lemma 3.5), we only need to prove that \(a = \{(s, W) \mid W \in \mathcal{U}, W \subseteq Z\}\) is an action. We show that if \(U \not\subseteq Z\) then \((s, U)\) is not covered by \(a\). More specifically, we prove that there exists \(V \in \mathcal{U}\) compatible with \(U\) such that \(V \cap Z = \emptyset\). Suppose first that \(z_0 \not\subseteq u_0\), where \(z_0\) and \(u_0\) are the least elements of \(U\) and \(Z\), respectively. There is a complete prime \(p\) such that \(p \not\sqsubset z_0\) and \(p \not\sqsubset u_0\). In that case we may take \(V = \uparrow p\). Assume next that \(z_0 \subseteq u_0\). As \(U \not\subseteq Z\), there is \(x \in U - Z\), and we may take \(V = \uparrow x\). We have clearly \(U \cap V \neq \emptyset\), and \(Z \cap V = \emptyset\) follows from the convexity of \(Z\). Indeed, if \(y \in Z \cap V\) then \(z_0 \sqsubseteq x \sqsubseteq y\), which implies \(x \in Z\), a contradiction.

It is perhaps interesting to see some concrete cases of the above lemma. Keeping to the switch example, consider \(a = \{(on, \{\bot, on\})\}\). We have \(\mathcal{O}(a) = \{(on, on)\}\), so adding \((on, \{on\})\) to \(a\) does not change its observables. In fact \((on, \{on\})\) is covered by \(a\), because its observable effect under any synchronous composition (just \((on, on)\)) is also obtained by \((on, \{\bot, on\})\) in \(a\). So \(a\) is not an action, and it turns out that \(\bar{a} = a \cup \{(on, \{on\})\}\). Notice, however, that \(b = \{(on, \{on\})\} = \bar{b}\). In this case \((on, \{\bot, on\})\) is not covered by \(b\), and the reason is that it can synchronize with the compatible pair \((on, \{\bot\})\) to generate the observable \((on, \bot)\), whereas \(b\) is not compatible with \((on, \{off\})\) and their synchronous composition is the "impossible" empty action with no observables.

We now define the three composition operations on the set of actions.
Definition 3.7. For all \( a, b \in \mathcal{A} \),

(Sequential) \( a \circ b = \{(x, U \rightarrow V) \mid (x, U) \in a, (x \rightarrow U, V) \in b\} \),

(Synchronous) \( a \otimes b = \{(x, U \cap V) \mid (x, U) \in a, (x, V) \in b, U \cap V \neq \emptyset\} \),

(Choice) \( a \oplus b = a \cup b \).

We are finally ready for the full abstractness result. It is worth noticing that we do not need to consider all contexts under the operators, one synchronous composition being enough.

**Proposition 3.8** (Full abstractness of actions). For any two actions \( a \) and \( b \), we have \( a = b \) if for any action \( c \) we have \( \mathcal{C}(a \otimes c) = \mathcal{C}(b \otimes c) \).

**Proof.** Suppose \( a \neq b \). Assume, without loss of generality, that \( (s, U) \in a - b \). Then \( (s, U) \) is not covered by \( b \), so there is \( V \in \mathcal{U} \) compatible with \( U \) such that \( s \triangleright (V \cap U) \neq s \triangleright (V \cap W) \) for every \( W \in b(s) \) compatible with \( V \). Let \( c = \{(s, V')\} \). For \( t = s \triangleright (V \cap U) \) we have \( (s, t) \notin \mathcal{C}(b \otimes c) \). We now prove that \( (s, t) \notin \mathcal{C}(a \otimes c) \). For any \( (s, t') \in \mathcal{C}(b \otimes c) \) we must have \( t' = s \triangleright (V' \cap U) \) such that \( V' \in c(s) \), \( W \in b(s) \) and \( V' \) and \( W \) are compatible. As \( V' \subseteq V \), by Lemma 3.6, \( V \) and \( W \) are also compatible. We have \( \mathcal{C}(c) = \{(s, s \triangleright V)\} \), so \( s \triangleright V' = s \triangleright V \). We can then establish that \( t' = s \triangleright (V' \cap W) = (s \triangleright V') \triangleright W = (s \triangleright V) \triangleright W = s \triangleright (V \cap W) \neq t \). \( \square \)

The possibility of synchronous composition of actions justifies their intensionality, leading us to distinguish between actions that have the same observables when executed on their own. As an example, let us look again at the switch, and consider the actions \( a = \{(on, S')\} = \{(on, S), (on, \{on\})\} \) and \( b = \{(on, \{on\})\} = \{(on, \{on\})\} \). They have the same observables \( \mathcal{C}(a) = \mathcal{C}(b) = \{(on, on)\} \), but, being different actions, they must be distinguishable by a synchronous context. Indeed, consider the action \( c = \{(on, \{\bot, off\})\} = \{(on, \{\bot, off\}), (on, \{\bot\})\} \). We can check that \( a \otimes c = c \) whereas \( b \otimes c = \emptyset \).

We now look at the issue of concurrency of actions. One of the interesting aspects of defining actions independently of processes is that actions implicitly encode their potential for true concurrency.

If one specifies a collection of actions to be executed "in parallel", i.e. with no a priori constraints on their execution, can we arbitrarily synchronize some of them to execute in a single step? For some incompatible actions this is obviously not possible, say switching to \( on \) and to \( off \) at the same time. But even if the synchronous composition of two actions is not empty, should we allow the synchronous execution when just the parallel execution was specified? Our answer is that it depends on the actions. We consider two actions to be independent when their executions in no way affect each other, and assume that independent actions may execute synchronously, i.e. in truly concurrent fashion, as this cannot affect the overall behaviour. Dependent actions, on
the contrary, are not allowed to execute synchronously by their own free will, as it were. This at the root of understanding resource handling.

The intuitive understanding of independence is that the combined effect of two independent actions does not depend on whether we execute one before the other, or vice versa, or synchronously.

**Definition 3.9.** Two actions \( a, b \in \mathcal{A} \) are independent if \( a \otimes b = a \odot b = b \odot a \).

Let us look at some examples.

Consider the switch situation space and the actions \( \text{TurnOn} = \mathcal{S} \times \{ \{\text{on}\} \} \) and \( \text{OnOff} = \{ (\text{on}, \{\text{off}\}) \} \). Let us first look at self-independence. We can check that \( \text{TurnOn} = \text{TurnOn} \otimes \text{TurnOn} = \text{TurnOn} \odot \text{TurnOn} \), i.e. \( \text{TurnOn} \) is independent of itself. This means that executing the action twice in sequence or just once (synchronously with itself) is exactly the same. This does not always happen, for example \( \text{OnOff} \) is not independent of itself. The reason is that \( \text{OnOff} \odot \text{OnOff} \) is the empty action, different from \( \text{OnOff} \odot \text{OnOff} = \text{OnOff} \). Executing \( \text{OnOff} \) twice in sequence (without interference) is impossible, whereas executing it once is not.

The question of self-independence sheds light on the issue of resource handling. In many coordination languages such as Linda, LO and GAMMA the shared data space is a multiset of items which are viewed as resources, actions being based on explicit addition and removal of items. In our setting, the prime elements of a situation are not viewed as resources, and it is the actions which can be resourceful or not. \( \text{OnOff} \), being self-dependent, is a resourceful action, i.e. doing it twice is not the same as doing it once. By contrast, \( \text{TurnOn} \) is not resourceful as it is self-independent, so executing it any number of times is the same as executing it once. In short, it is not the switch (or its states) which are resources, but the actions upon it that can be said to be resourceful or not. It is interesting to notice that even in multiset based languages one can have nonresourceful actions, for example \( \text{read} \) in Linda or the corresponding action in LO that atomically removes and adds the same item.

Looking now at independence between different actions, we can see that \( \text{TurnOn} \) and \( \text{OnOff} \) are not independent of one another. Indeed, \( \text{TurnOn} \odot \text{OnOff} = \mathcal{S} \times \{ \{\text{off}\} \} \) (what we might call \( \text{TurnOff} \)), \( \text{OnOff} \odot \text{TurnOn} = \{ (\text{on}, \{\text{on}\}) \} \) (say, \( \text{StayOn} \)) and \( \text{TurnOn} \odot \text{OnOff} = \emptyset \), the impossible action. We see that the parallel execution of \( \text{TurnOn} \) and \( \text{OnOff} \) implicitly forces sequentiality, and amounts to a choice between \( \text{TurnOff} \) and \( \text{StayOn} \), corresponding to the nondeterministic sequential ordering of the two actions.

**Definition 3.10.** The *parallel* composition of two actions \( a, b \in \mathcal{A} \) is

\[
a \oplus b = (a \odot b) \odot (b \odot a).
\]

The above equation defines parallel composition as a derived notion, and is usually regarded as implying a nontruly concurrent semantics. First, let us stress once again that the composition operators we are dealing with here are not those for processes,
but those for (atomic) actions. Secondly, the equation holds regardless of there being a
notion of true concurrency entailed by action independence. If \(a\) and \(b\) are independent,
then we know that \(a \mathbin{\odot} b = b \mathbin{\odot} a = a \mathbin{\odot} b\) and therefore \(a \mathbin{\oplus} b = a \mathbin{\odot} b\), i.e. the
parallel composition is akin to the synchronous composition (true concurrency). If,
on the other hand, \(a\) and \(b\) are not independent, then they cannot be "accidentally"
synchronized, i.e. their parallel execution always involves sequentiality.

Let us present yet another example that further clarifies the last remark. Imagine
that our situation space deals with two independent switches. The prime elements are
\(\text{on}_1, \text{off}_1, \text{on}_2, \text{off}_2\), and we can have nonprime situations such as \(\text{on}_1 \sqcup \text{off}_2\). Consider now
the two actions \(\text{OnOff}^{12} = \{(s, \text{\text{off}}_2) \mid \text{on}_1 \sqsubseteq s\}\) and \(\text{OnOff}^{21} = \{(s, \text{\text{off}}_1) \mid \text{on}_2 \sqsubseteq s\}\). Clearly \(\text{OnOff}^{12} \odot \text{OnOff}^{21} = \text{OnOff}^{21} \odot \text{OnOff}^{12} = \emptyset\), as the precondition of ei-
ther action is inconsistent with the postcondition of the other. However, \(\text{OnOff}^{12} \mathbin{\otimes} \text{OnOff}^{21} = \{(\text{on}_1 \sqcup \text{on}_2, \{\text{off}_1 \sqcup \text{off}_2\})\}\), i.e. the actions’ synchronous execution does
have an observable behaviour. When composing these two dependent actions in paral-
lel, it seems therefore natural to exclude the possibility of synchronous execution, as
we would get a behaviour that cannot be obtained by any sequential execution.

4. Conclusions and further work

In this paper we developed a theory of actions which are based on semantic entail-
ment of situations rather than syntactic pattern matching of data. These actions capture
a very general notion of change, including information removal. The main restriction
on the situation space is that it must be coherent, thus precluding the use of arbitrary
constraint systems. However, this is for guaranteeing determinism of all updates in all
situations. It may be worth exploring how to relax this universality condition, as it
is not required in practice, i.e. we may work within a noncoherent space and only
perform updates which are deterministic. We should investigate a general theory of
updates and actions in this more general setting.

Actions are intended as building blocks for defining concurrent processes, whose
coordinated behaviour emerges from the combined effect of their actions in a shared
data space. In fact the results presented here, while interesting on their own, will gain
in importance in the light of the construction of processes from actions. An important
further work, then, is the development of a theory of processes based on these actions.
Such processes should give denotational semantics to languages along the lines of
TAO [14], whose operational semantics are defined through steps which correspond to
actions as presented here.

The synchronous composition of actions should not be confused with their parallel
composition when viewed as processes. Synchronicity has to be enforced by suitable
operators at the level of processes. TAO, for example, has a synchronous operator
behaving exactly as described here as it applies only to single action processes. We
can generalize it to apply to general processes, with the intended effect of synchronizing
either their first or their last step.
The sequential composition was defined here for actions, not processes. When considering processes in general, the action resulting from the sequential composition of other actions corresponds to applying an atomization operation to a process consisting of a sequence of actions, that effectively turns the complex processes into an atomic action. This is very useful, being a form of transaction as performed in database systems. So our future theory of processes should incorporate the atomisation operation, in whose definition the sequential action composition should be relevant.

One of the interesting aspects of our theory is the notion of independence of actions, that we define without any mention of processes. This means that when considering "true concurrency" semantics for processes we should distinguish the two levels of actions and processes. Some actions by themselves are already intrinsically dependent, and no true concurrent execution is possible; any synchronous execution, if possible, has to be forced upon them. When (occurrences of) actions are combined to form processes, further dependence constraints may arise, through the use of sequential composition, on occurrences of independent actions, thus adding to the basic level of dependence.

References