Journal of Modern Technology and Engineering
Vol.6, No.2, 2021, pp.112-123

# FOUNDATIONS FOR THE FORMAL ANALYSIS OF TECHNOLOGY

**Arturo Geigel***

Guaynabo, Puerto Rico

**Abstract.** This paper presents a formalism for the analysis of technology. This formalism uses as a foundation the notion of a component that contains interfaces. With these interfaces, the component is capable of performing couplings which are described by basic operators that dictate the behavior of the coupling or lack thereof. Within the formalism, a particular arrangement of components and their interfaces form states. These states can be further manipulated by permutations until they form a complete set of coupled components that are identified as assemblies. The formalism uses the concept of the technological State machine to bring together the descriptions states and their transitions through permutations, and the couplings generated under such rearrangements. The formalism also introduces the concept of restrictions on permutations to further constrain the transitions between states. The technological state machine and the concept of restriction provides the necessary elements to introduce the concept of assembly constructions. Using these constructions allows the formalism to express the complexities of assembly constructions. The complexity of these constructions highlight the need for heuristics that can find feasible construction for assemblies. Finally, future directions of research are outlined and the conclusion is given.

**Keywords**: Technology, assembly, component, interface.
**AMS Subject Classification:** 00B99.
**Corresponding author:** Arturo Geigel, Guaynabo, Puerto Rico 00966, e-mail: *ageigel@airobotiks.com*
*Received: 21 April 2021; Revised: 28 May 2021; Accepted: 11 June 2021; Published: 31 August 2021.*

## 1 Introduction

The work presented in Geigel (2021) discusses the problem that technology lacks any mathematical treatment as a subject; and states the need for a formalization of technology. The proposal focused on the notion of a component as a core concept to develop the notion of technology. These components have interfaces that when two components are joined by their interfaces they form a coupling. The process of joining components through couplings produces states that form a group of permutations on the set of components. These processes of performing permutations on the set of components to test if they produce couplings is described as a technological state machine. The outcome of the state machine can be one of the possible permutations on the set of components that successfully couples all components or fails to do so. The end state, where there is a set of components that successfully couples all components, is then defined as an assembly of component. Technology is then defined in the paper as the assemblies that arise from the state changes in the state machine. This then becomes TA Interpretation of Technology. An argument presented in Geigel (2021) is that this interpretation of technology groups together seemingly unrelated and incompatible categories of conceptions of technology. While the work focused on the introduction of the concepts and its interpretation, it did not address the subject in a formal rigorous way.

This paper presents the derivation of the formalism introduced in Geigel (2021). The work presented here introduces additional concepts and properties that are necessary to understand technological assemblies. The paper extends the notion of state machine to better model coupling phenomena. The paper also introduces the concept of assembly construction to arrive at the notion of optimal construction. Finally, the paper discusses how to overcome the complexity of constructing assemblies within the formalism. This provides a pillar on which further analysis of the process of assemblies as a tool to model technological development can take place.

## 2 Components, Interfaces, and Their Operations

Assume a set of objects[1] $OBJ = \{obj_1, \ldots, obj_n\}$ and a set of interfaces $INT = \{int_{11}, \ldots, int_{nm}\}$, where the first subscript of the interface identifies its relationship to an object $obj_i$. The subscript of the object $obj_i$ identifies the object uniquely.

A relation *is component* between an object $obj_i$and an interface $int_j$, is given by:

$$is\,component_k = (obj_i, int_j)\,, \tag{1}$$

where such relation happens on a object $obj_i \in OBJ$.

**Definition:** An object $obj_i \in OBJ$ and $int_j \in INT$, given by the pair $(obj_i, int_j)$, are defined as a **component** *comp*, if:

$$P_{int}\,(is\,component_k) = 1. \tag{2}$$

We can identify[2] that a component $comp_i$ is equal to $comp_j$, if all $int_{i1}, \ldots, int_{im} = int_{j1}, \ldots, int_{jm}$.

There is a set $U_{comp}$ that has all member components $comp_1 \ldots comp_n$, and is known as the universe of components. It is assumed in the formalism, that $U_{comp}$ is not a member of itself.

**Definition:** $IND_s$ is an index set of components where $IND_s \subset U_{comp}$ and $IND_s \neq U_{comp}$.

The components $comp_1 \ldots comp_n \in IND_s$ use a second subscript $m \in \mathbb{N}$, such that $comp_{1,1} \ldots comp_{n,m}$, that will not be used unless it is not clear from context.

### 2.1 Couplings and Interactions

The interfaces $int_{i1}$ of a component $comp_i$ provide the means for $comp_i$ to carry out[3] a binary interaction *interact* between $int_{i1}$ of $comp_i$ and an interface $int_{j1}$ of $comp_j$, where $comp_i \neq comp_j$ and $int_{i1} \neq int_{j1}$. The interfaces have specific constraints or requirements for a particular interaction that are evaluated through the requirement predicate *Req*. *Req* predicate variable $x_1 \ldots x_n$ values can be specified as optional[4] or as compulsory. A *compulsory Req* predicate variable is a single predicate in a conjunction formula[5]. An *optional* requirement uses more than

[1]Set theory symbol convention used here follows the interpretation as that given in (Suppes, 2021).

[2]The subscript of the object becomes that of the component.

[3]Components assembled into themselves are divided into two parts to meet with the requirements of a binary operation, (this will later be possible by the introduction of subcomponents). The requirement $comp_i \neq comp_j$ is to constrain any analysis of a component assembled into itself, and specifying an interface into itself. The requirement $int_{i1} \neq int_{j1}$ is to constrain the identity of the interfaces when they couple, (even if the parts are equal, their function must differ). By way of example, take a splice of two wires that conduct electricity. Even if they have all the physical properties of being made of copper, have a sheath, etc., the flow of electricity at the coupling has one interface with electricity flowing in the outward direction, and the other interface has electricity going in the inward direction.

[4]Optional here refers as to the particular value that can be "swapped" for another value, including the $\emptyset$ value. Leaving the optional value out of the formula leads to an underspecified predicate *Req*.

[5]An illustration of requirements can be given, using the example of the nail that is composed of a point shank and head from Geigel (2021). The predicate *Req* from the shank can be specified as $Req_{shank\,1}\,(Union\,(P_{surface}, x))$. The $P_{surface\,1}$ of $Req_{shank\,1}$ is the first of the shortest sides from left to right of a shank that is laid horizontal, resting on its longest side. The $x$ implies the surface of another component that will be used to form the union required for *Req* to be evaluated as true. The requirements for the other two parts of the nail can be similarly defined.

one *Req* predicate specified as a conjunction of predicate terms within a conjunction formula.

The notion of optional requirement can be interpreted as two possibilities. The first interpretation is that of a compulsory element that cannot be removed and an optional requirement is added, that if missing, still meets the disjunction requirement. The other interpretation is one of a compulsory requirement that can be swapped for the other and still meet the requirement of the disjunction.

Both compulsory and optional requirements can be given at the same time as a conjunctive normal form (CNF) formula, as stated in the definition of coupling below.

**Definition:** A **coupling** $Coupling_{ij}$ is created when[6]:

$$\begin{gathered} Coupling_{ij} = interact(comp_i, comp_j) \\ = (Req_{i1}(x) \vee \ldots \vee Req_{il}(x)) \wedge (Req_{il+1}(x) \vee \ldots \vee Req_{im}(x)) \ldots \\ (Req_{j1}(x) \vee \ldots \vee Req_{jn}(x)) \wedge (Req_{jn+1}(x) \vee \ldots \vee Req_{jo}(x)) = 1 \end{gathered} \tag{3}$$

The definition of coupling as a CNF is an ideal of the coupling $Coupling_{ij}$ that covers all possibilities of a coupling between $comp_i$ and $comp_j$[7].

To simplify the analysis of couplings, we can restrict it to just the conjunction of the formula, and from there give the following Lemma:

**Lemma 1.** *If a Coupling $Coupling_{ij}$ has only conjunctions of predicates Req to test, then $Coupling_{ij}$ has $2^m$possible results.*

*Proof.* Follows from elementary combinatorics analysis. □

**Definition:** A component $comp_i$ is similar to component $comp_j$, if both have the same compulsory *Req* predicates, but different *Req* predicates in the optional requirements.

## 2.2 Operations

The coupling operation allows for the interaction of $comp_i$ and $comp_j$ to achieve a coupling. This part contemplates extending the analysis to $comp_1 \ldots comp_n \in IND_s$ by the use of binary operators on $comp_i$ and $comp_j$ using set operations of union and intersection, (in the form of the difference), with that of the interaction to achieve couplings.

Several components can be put together using the union of components:

$$comp_1 \cup \ldots \cup comp_n = comp_1 + \ldots + comp_n$$

The union of components, together with the coupling operation, make the **coupling addition** $CA$ in the case of positive coupling:

$$CA = comp_i \diamondsuit\, comp_j = (comp_i + comp_j) \wedge (Coupling_{ij} = 1) \tag{4}$$

There is also the case of a **failed coupling** $FC$:

$$FC = comp_i \overline{\diamondsuit}\, compj = (comp_i + comp_j) \wedge (Coupling_{ij} = 0) \tag{5}$$

Similarly, one can subtract components using:

$$comp_n \setminus \ldots \setminus comp_1 = comp_n - \ldots - comp_1,$$

with a decoupling operation $Coupling = 0$, to make the **decoupling** or coupling subtraction $CS$ as:

$$CS = comp_i \triangle\, comp_j = (comp_i - comp_j) \wedge (Coupling_{ij} = 0)\,. \tag{6}$$

[6]First order logic symbols used are those as in **?**.

[7]Following the previous example in footnote 5, if all requirements are met, then all conjuncts will be evaluated true and two couplings will be made.

As with the failed addition, there is also a **failed decoupling** $FD$[8]:

$$FD = comp_i \overline{\triangle} comp_j = (comp_i - compt_j) \wedge (Coupling_{ij} = 1)\,, \tag{7}$$

whenever $comp_n \setminus \ldots \setminus comp_1 \neq \emptyset$.

When referring to any of the operations: $\diamondsuit$ , $\overline{\diamondsuit}$, $\triangle$, or $\overline{\triangle}$, the generic operation symbol $comp_i \odot compj$ is used.

## 2.3 Properties of the CA Operation

To cover particular properties of the $CA$ operation, it is necessary to enter into a peculiarity of interfaces which can restrict the operations carried out on components. The peculiarity of interfaces is that most interfaces are usually unique, making the analysis of the operations carried on them non trivial in nature.

**Definition:** An interface $int_{ij}$ of object $comp_i$ that couples with the interface $int_{kl}$ of $comp_k$ is **unique,** given that only $int_{ij}$ of object $comp_i$ can produce a coupling with $int_{kl}$ of $comp_k$. Otherwise, an interface $int_{ij}$ of object $comp_i$ is **not unique** if there exists an interface $int_{mn}$ that creates the coupling with $int_{kl}$ of an object $comp_l$. [9]

**Theorem 1.** *If $comp_i$ has a unique interface $int_{i1}$ and unique interface $int_{i2}$, where $int_{i1} \neq int_{i2}$ , then the $CA$ operation is **irreflexive**.*

*Proof.* It is just a matter of performing an operation between $comp_i \odot comp_i$, using the interfaces $int_{i1}$ and $int_{i2}$, since both $int_{i1}$ and $int_{i2}$ are unique, this produces $comp_i \odot comp_i = comp_i \overline{\diamondsuit} comp_j = FC$ between $comp_i$ and $comp_j$, therefore the $comp_i \odot comp_i$ operation is irreflexive. □

This notion of having unique components restricts the possibilities of the $CA$ operation substantially, since it cannot be taken for granted that there are non unique components when carrying a $CA$ operation.

**Theorem 2.** *If $comp_i$ does have a unique interface $int_{i1}$, and $comp_j$ also does have a unique $int_{j1}$, then the CA operation is **not symmetric**.*

*Proof.* Assume that $comp_i$ has interfaces $int_{i1}$ and $int_{i2}$, and $comp_j$ also has an $int_{j1}$ and an $int_{j2}$. An operation between $comp_i \odot comp_j$ is done between $int_{i1}$ and $int_{j1}$, where both are unique, such that operation $comp_i \odot comp_j = comp_i \diamondsuit comp_j = CA$ is achieved. If an operation between $comp_i \odot comp_j$ is carried out using $int_{i1}$ and $int_{j2}$, or $int_{i2}$ and $int_{j1}$,the constraint that both $int_{i1}$ and $int_{j1}$ are unique produces $comp_i \odot comp_j = comp_i \overline{\diamondsuit} comp_j = FC$, therefore the $CA$ operations is irreflexive. □

## 3 The Technology State Machine

### 3.1 States, Permutations Triggers, and Transitions

To have a complete analysis of components, it is not enough to just specify operations $comp_i \odot comp_j$ on $comp_1 \ldots comp_n \in IND_s$. Components need to be repositioned so that $comp_i \odot comp_j$ operations are tried on different arrangements of components $comp_1 \ldots comp_n \in IND_s$. The analysis here is extended to include the notion of a state, where components have a particular ordering along with their respective $comp_i \odot comp_j$ operations. The analysis also includes the

[8]This is the state where $(comp_1 - comp_2)$ is tried but never achieved, since the requirement that $Coupling_{12} = 0$ is never achieved.

[9]$m$ does need to be equal to $i$, since the definition of uniqueness is with respect to the interface, and not the component.

transition from one state to the other, where new orderings and new $comp_i \odot comp_j$ operations are carried out in the new ordering of all components $comp_1 \ldots comp_n \in IND_s$.

The definition of state is given as follows:

**Definition:** Given a set $\{comp_1, comp_2, \ldots, comp_n\}$, where $comp_1 \ldots comp_n \in IND_s$, and an operation $comp_i \odot comp_j$ between all members of $IND_s$, a state is defined as:

$$State_i = comp_1 \odot comp_2 \odot \ldots \odot comp_n \tag{8}$$

Given the indexed set $IND_S$ with its state $State_i$, then any rearrangements on $State_i$ correspond to a permutation $\sigma : S \to S$, whereby using the index for the mapping can be represented as:

$$\sigma = \begin{pmatrix} comp_1 & \odot & comp_2 & \odot & \ldots & \odot & comp_n \\ f(comp_1) & \odot & f(comp_2) & \odot & \ldots & \odot & f(comp_n) \end{pmatrix}$$

**Definition:** The permutation function $\sigma$ on $State_i$ is a **permutation trigger**[10] $PT$ from a $State_i$ to a $State_j$, where $i$ and $j$ are indexes to the set of all possible orderings on states.

Note that state changes from $State_i$ to a $State_j$ imply changes in the interaction between components due to their repositioning[11].

To simplify the discussion on the permutations of components, the analysis will assume that all components have two interfaces, and that the ordering is limited to a single dimension. Under these simplifying assumptions, we can associate a vector $v_\sigma(state_i)$, such that:

$$v_\sigma(state_i) = \begin{pmatrix} comp_1 \\ comp_2 \\ \vdots \\ comp_n \end{pmatrix},$$

and a transformation matrix $M_{ij}$, where the entries are $ij = 1$, if $f(i) = j$.

This allows matrix multiplication of the form:

$$M_{ij} v_\sigma(state_i) = \begin{pmatrix} f(comp_1) \\ f(comp_2) \\ \vdots \\ f(comp_n) \end{pmatrix} \tag{9}$$

The rearrangement of states allows for quantification of the number of possible orderings on the set of components $IND_S$. This is given in the following basic lemma.

**Lemma 2.** *The permutation allows $n * n - 1 * n - 1 \ldots 2, 1 = n!$ alternative orderings on the indexed set $IND_S$.*

*Proof.* Follows from basic combinatorics. □

## 3.2 Permutation Process

As stated above, it is the permutations trigger $PT$ that produces the transition from $State_i$ to a $State_j$. The permutation function $\sigma$ on $State_i$ carries some additional interpretations on the relation between states. These notions are consolidated under the permutation process.

[10]It is irrelevant for the purpose of the analysis where this trigger comes from. It is enough to state its existence for the purpose of the analysis.

[11]Continuing the example from footnote 5, the interaction $state_i = point \odot head \odot shank$ does not provide a coupling, as opposed to $state_j = head \odot shank \odot point$.

**Definition:** A **Permutation Process** ($PP$) is a 3-tuple of the form:

$$Proc_{ij} = (PT_{ij}, State_i, State_j) \tag{10}$$

The Process 3-tuple structure constrains any assertion to:

$$\sim (State_i \wedge \sim State_j) \tag{11}$$

Given equation 11, the $PP$ assertion as a 3-tuple of $PT_{ij}$, $State_i$, and $State_j$, binds a specific permutation trigger $PT_{ij}$ on $State_i$ and $State_j$. This is done by having bound specifically the values of $State_i$ and $State_j$ to true, such that the permutation trigger is also true, making the implication a necessary one. Any other interpretation of $State_i$ and $State_j$ as not being true is irrelevant, since it does not satisfy the constraint of having a $PP$ assertion.

## 3.3 The Technology State Machine, the Stable and NOOP State

While the technology state machine can be defined using the standard quintuple $A = (Q, \Sigma, \eta, s_0, s)$ of a normal state machine, it will be convenient to frame the machine within the formal analysis undertaken here. This will be done by adding the permutation trigger, as well as two additional states, as part of the specification. The introduction of these three new tuples highlights the unique properties of the state machine under consideration. The state machine is now extended to form a **technology state machine** that is of the form of a 7-tuple $AT = (Q, \Sigma, PP, s_0, s, s_{stable}, NOOP)$ where:

- $Q$ is the set of **states**.
- $\Sigma$ represents the **alphabet**.
- $PP$ is the **Permutation Process**.
- $state_0 \in Q$ identifies the **start state**.
- $state_f \subset Q$ is the **final state**.
- $state_{stable} \subset Q$ is the **Stable State**.
- $NOOP \subset Q$ is the **NOOP state**.

Within this new technology state machine, we now define the stable state as:

**Definition:** A $State_m$ under a permutation ordering through the mapping $\sigma : State_i \rightarrow State_j$ in the $PP$ is said to be **stable or complete** if there is a state such that $comp_i \odot compi + 1 = comp_i \diamondsuit comp_{i+1}$ for all of the component interfaces. The stable state is usually assigned the end state[12].

In addition to the stable state, there is a NOOP state, defined as follows:

**Definition:** A NOOP state[13] is a state that does not change the couplings of the components or the ordering on the indexed set $IND_S$.

From the notion of NOOP, we derive the following result:

**Theorem 3.** If a Permutation Process $PP$ from $State_i$ to a $State_j$, where $State_j = State_i$, then the transition from $State_i$ to a $State_j$ is a **NOOP**.

*Proof.* Since $State_j = State_i$, there is no change in the ordering on the indexed set $IND_S$. It also implies $State_i = comp_1 \odot comp_2 \odot \ldots \odot comp_n$ is equal to $State_j = comp_1 \odot comp_2 \odot \ldots \odot comp_n$, therefore all the operations from state $i$ will be the same as those of state $j$. □

[12]The Assembly definition given below will state that $stable\, state = end\, state$, the case where the stable state is not the end state is when considering the analysis of a subcomponent within the construction of an assembly.

[13]From the point of view of the state transitions, there is no need to specify the NOOP until the states are placed in a time interval, where the NOOP becomes relevant to the analysis of states.

In addition to the above comments on the technological state machine, it is worth noting that the start state can be a configuration where all components are uncoupled. It can also be a random configuration where some or all components are coupled. The particular specification will determine the initial configuration of components as $State_0$.

### 3.4 Generalizations

Note that, thus far there are two simplifications done to the analysis made on components. The first is that the number of elements on a state remain fixed for the analysis. This is only true under the most simplistic of analysis. The removal of the simplification would further complicate the calculation carried out in lemma 2.

A further generalization comes with the lifting of the limitation of components with only two interfaces. To generalize to $k$ interfaces on more than one dimension. To expand the analysis, a bisection to a coordinate space where index $m \to \Re^n$ is introduced the left side of equation 9, where the components are placed on specific coordinates. With the introduction of this coordinate space, equation 9 becomes:

$$M_{ij} v_{St}\left(m\right), \tag{12}$$

where $M_{ij}$ is a transformation matrix, and $v_{St}\left(m\right)$ is a vector of coordinates of the $m$ component.

## 4 Assemblies

The objective of the TA Interpretation of Technology is to give an account of technology in terms of assemblies. The definition of technology given in Geigel (2021) is that:

> "***Technology are assemblies that arise from within a set of components through state changes.***"

The definition of assembly is as follows:

**Definition:** An assembly $A$ is a set $IND_S$ of components that *has reached a stable state,* such that $stable\, state = end\, state$.

The following theorem states the upper bound on the complexity of going from $stable\, state$ to $end\, state$:

**Theorem 4.** *An assembly A that has Couplings with only conjunctions of predicates Req under permutation operations has a complexity of* $2^m * n!$.

*Proof.* The result follows from lemma 1 and lemma 2, noticing that under each permutation the analysis of $2^m$ possible results is carried out. □

This result is not a positive result; and further explanations are needed to understand how technological development takes place, and this is explored in section 5. To simplify the exposition and handle some of the complexity of the topic, the rest of the exposition will assume a 1-dimensional assembly, giving way to the following definition of 1 dimensional chain:

**Definition:** A component chain $CH$ is a 1 dimensional state $State_h = comp_1 \odot comp_2 \odot \ldots \odot comp_n$, that can include the start state, where length[14] of $CH$ is denoted as$|L|$. If the chain is the end state, then it is an assembly chain $ACH$.

[14]The notion of length can then be extended for an arbitrary number of components in a stable state as a path in a graph that is on a multi dimensional space.

### 4.1 Subcomponents of Assemblies and Atomic Components

The analysis of components have thus far guided the analysis of assemblies. Another way to go about the analysis of assemblies is by using subcomponents of the assembly. This notion is now defined.

**Definition:** A component $comp_i$ is a **subcomponent**[15] of $comp_k$, if $comp_i \subseteq comp_k$, and there is a $comp_j \subseteq comp_k$ from which a $CS$ operation is carried out to obtain $comp_i$.

To delimit the number of decompositions into subcomponents[16], there is a need for a stopping criterion. This is achieved by introducing the definition of atomic component as follows:

**Definition:** A component $comp_i$ is **atomic** if there is no $CS$ operation that can be carried out to obtain $comp_j$ and $comp_k$ from $comp_i$.

Thus, the number of possible applications of the CS operation to obtain subcomponents ends when components are atomic. The determination of when a component is atomic depends on the particular definition of components chosen for the analysis of the assembly $A$. The notion of subcomponents also allows for the following transitive relation:

**Theorem 5.** *For any assembly $A$, having subcomponents $subcomp_i$, $subcomp_j$,$subcomp_k \subseteq Assembly\,A$, if $subcomp_i \subset subcomp_j$ and $subcomp_j \subset subcomp_k$, then $subcomp_i \subset subcomp_k$.*

*Proof.* Follows directly from set theory. □

**Theorem 6.** *A component that contains subcomponents is not atomic (i.e.* ***non-atomic****).*

*Proof.* If an a component $comp_k$ contains subcomponent $comp_i$ and $comp_j$, it is of the form $comp_i \subseteq comp_k$ and $comp_j \subseteq comp_k$. In addition, the CA operation $comp_k$ is obtained using subcomponents $comp_i$ and $comp_j$. Since there is a $CS$ operation that can be carried out to obtain $comp_j$ and $comp_k$ from $comp_i$, therefore it is not atomic. □

### 4.2 Operations and Properties of Assemblies

With the definition of components, assembly subcomponents, and atomic components, the following theorem can be stated as a closure property.

**Theorem 7.** *For any Permutation Process $PP$, having permutation trigger that operates on any given state, the changes in the interaction between components of $IND_S$, we obtain either an assembly, subcomponents, or atomic components.*

*Proof.* The proof consists of five parts:

1. Start with a component that is atomic. Since no $CS$ operation can be done, by definition, the process of applying operations $\odot$ is done.

2. Start with a component that is non-atomic. Apply a single $CS$ operation to obtain (a) both atomic elements, or (b) obtain an atomic element with a component chain $|L-1|$. If the result is that both are atomic elements, then stop. If the result is an atomic element with a component chain $|L-1|$, continue to apply the $CS$ operation on the component chain $|L-1|$ to obtain an atomic element and a component chain $|L-2|$. If the chain length $|L| = 1$, then stop; else repeat until the component chain length $|L| = 1$ and all components are atomic elements.

[15] The use of component *comp* and *subcomp* will be used interchangeably unless it is not clear by context.

[16] As noted in Geigel (2021) the notion of subcomponent can be used to define technology using the notion of component, (i.e. subcomponent as defined here), from the following passage "What if we looked inside technologies?... If you open up a jet engine (or aircraft gas turbine power plant, to give it its professional name), you find components inside" (Arthur, 2009, pp.18-19) , but this would allow for infinite recursion.

3. Start with an assembly $A$. Apply a single $CS$ operation to obtain (a) both atomic elements or (b) obtain an atomic element with a component chain $|L-1|$. If the result is that both are atomic elements, then stop. If the result is an atomic element with a component chain $|L-1|$, continue to apply the $CS$ operation on the component chain $|L-1|$ to obtain an atomic element and a component chain $|L-2|$. If the chain length $|L|=1$, then stop; else repeat until the component chain length $|L|=1$ and all components are atomic elements.

4. Start with a component chain $CH$ of length $|L|$, having at one end of the chain a component $comp_i$ and a separate component $comp_j$. Apply $comp_i \odot comp_j$ operation on $comp_i$ with $comp_j$, which will be either $\odot = \diamondsuit$ or $\odot = \overline{\diamondsuit}$ . If $\odot = \diamondsuit$, then a component chain $CH$ of length $|L+1|$ is obtained. If component chain $CH$ is the end state, then an assembly $A$ is obtained.

5. Apply a permutation on a component chain $CH$ of length $|L|$, obtained in steps 1 through 4. Then this will take the component chain $CH$ from a $State_i$ to a $State_j$, where $State_j = State_i$. At this point, one can apply operation on $State_j = comp_1 \odot comp_2 \odot \ldots \odot comp_m$, where each $\odot$ operation is a $NOOP$, therefore the same component chain $CH$. This also applies to the component chain $CH$ that is equal to the end state.

$\square$

While all operations are either an assembly, subcomponents or atomic components, some applications can break an assembly. For example, there are components that cannot be removed from the assembly because it will break the assembly, (the chain cannot be put back together to form Assembly $A$). This can be specified by identifying the component $comp_i$ of an assembly $A$ as a critical assembly component, defined as:

**Definition:** A component $comp_i$ is a **critical assembly component,** if $comp_i$ is part of a component chain $CH_A$, where couplings cannot be achieved without the inclusion of $comp_i$ to carry out operations $comp_{i-1} \diamondsuit comp_i$ and $comp_i \diamondsuit comp_{i+1}$.

The two operations utilizing $comp_i$ allow for the inclusion of $comp_{i-1}$ and $comp_{i+1}$ into component chain $CH_A$.

**Theorem 8.** *If assembly A that contains critical components $comp_i$, then assembly A cannot have a $CS$ operation on $comp_i$.*

*Proof.* Given assembly A with a component $comp_i$ such that:

$State_f = comp_1 \odot comp_2 \odot \ldots comp_{i-1} \odot comp_i \odot comp_{i+1} \odot comp_n$ ,

apply a $\odot = \triangle$ operation, such that:

$State_h = comp_1 \odot comp_2 \odot \ldots comp_{i-1} \triangle comp_i \triangle comp_{i+1} \odot comp_n$,

resulting in two component chains. Given that the assembly A does not have critical component $comp_i$, we have no coupling between $comp_{i-1}$ and $comp_{i+1}$ , therefore Assembly A cannot have a $CS$ operation. $\square$

Given the definition of an assembly that contains critical components, we can have two possible types of assemblies[17]. The first is that of an **essential assembly** that only contains critical components whose assembly chain $ACH$ length $|L| = n$. The second is a **mixed assembly** that contains critical and non critical components.

**Definition:** Two components $comp_i$ and $comp_j$ are **interchangeable** if one can be exchanged with the other with respect to assemblies $A$.

[17]The argument that there exists a third classification that is a **non essential assembly,** which is one where all components are not considered critical is considered an identity problem. An assembly that has no essential components does not fall under any category of technology. This is an instance, that while possible in logical terms, it does not have any significance to technology, since an assembly must have at least one critical assembly from which to derive its identity.

**Theorem 9.** *$Subcomp_i$ of Assembly $A$ is interchangeable with $comp_j$.*

*Proof.* It is given from the definition of assembly, that Assembly $A$ has a stable state: $State = comp_1 \odot comp_2 \odot \ldots comp_{i-1} \triangle comp_i \triangle comp_{i+1} \odot comp_n$;

and that interfaces $int_{i,i-1}$ and $int_{i,i+1}$ of object $obj_i$ couple to interfaces $int_{i-1,i}$ and $int_{i+1,i}$. Using definition 2.3 and assuming $int_{i-1,i}$ and $int_{i+1,i}$ of object $obj_i$ that are not unique with respect to $int_{i-1,i}$ and $int_{i+1,i}$. Using again the definition 2.3, there is a $comp_j$ with $int_{j,i-1}$ and $int_{j,i+1}$ that are not unique with respect to $int_{i-1,i}$ and $int_{i+1,i}$, such that when $Sucomp_i$ is replaced with $comp_j$ the final state of Assembly $A$ is:

$$\begin{aligned} State &= comp_1 \odot comp_2 \odot \ldots comp_{i-1} \triangle comp_i \triangle comp_{i+1} \odot comp_n \\ &= comp_1 \odot comp_2 \odot \ldots comp_{j-1} \triangle comp_j \triangle comp_{j+1} \odot comp_n \end{aligned}$$ □

**Remark:** The importance of theorem 9 is that interchangeability does for replacement of components what the theorem 3 does for operations.

The notion of interchangeability can then be used to define when an assembly $A$ is equal to Assembly $B$. The definition is given as:

**Definition:** Assembly $A$ is equal to Assembly $B$, if: all subcomponents of assembly $A$ are interchangeable with assembly $B$, and all subcomponent of assembly $B$ are interchangeable with assembly $A$.

The notion of interchangeability can also be used to define when an Assembly $A$ is similar to Assembly $B$, as follows:

**Definition:** Assembly $A$ is similar to Assembly $B$, if: all subcomponents of assembly $A$ are interchangeable with assembly $B$, all subcomponents of assembly $B$ are interchangeable with assembly $A$, and there is one subcomponent $Sucomp_i$ from assembly $A$ that is similar to $Sucomp_j$ from assembly $B$.

## 5 Construction of Assemblies

To introduce constructions of assemblies, we first note that states can be unbounded. An unbounded state is one that contains transition triggers to all other states in the technological state machine $AT$. A technological state machine $AT$ having all states as unbounded is a complete technological state machine. A complete technological state machine is a trivial one, since all states point to the end state, including the start state.

**Definition:** A State $state_i$ of a technological State machine $TSM$ having states $states_0 \ldots State_i \ldots State_j \ldots state_f$ has a restriction, if $state_i$ cannot reach state $state_j$ through a permutation trigger $PT$.

All permutations triggers $PT$ from $State_i$ *of a* $TSM$ *to any* $State_j$ of the $TSM$ form the neighborhood of $State_i$.

**Definition:** A **construction** is a sequence of permutation triggers $PT$ from any $State_i$ *to any* $State_j$ on a technological state machine $AT$, such that $state_0 \ldots end\,state \subseteq AT$.

We can say that a set $IND_S$ is **implementable,** if there is a construction assembly $A$; otherwise the set $IND_S$ is not implementable, if there does not exist a construction to obtain assembly $A$.

All constructions would be limited if only assembly $A$ is possible. Therefore, the construction must allow for addition of components to assembly as part of the sequence of states, and is given by the following definition.

**Definition:** An assembly $A$ is extendable if within a construction there is a subcomponent $subcomp_i \in Assembly\,A$ that has a free interface and another component $comp_j \notin Assembly\,A$ that also has a free coupling.

Upon a permutation trigger, the assembly changes such that $\diamondsuit$ can be carried out, resulting in $comp_j \in Assembly\,A$.

We can have that an assembly $A$ is **compatible** with assembly $B$, if assembly $A$ extends assembly $B$ as shown in the following theorem:

**Theorem 10.** *Assembly $A$ is compatible with assembly $B$, if assembly $B$ extends assembly $A$.*

*Proof.* Assume assembly $A$ is extendible, having a subcomponent $subcomp_i \in Assembly\,A$ that has a free interface. Further, assume that assembly $B$ is extendible having a subcomponent $subcomp_j \in Assembly\,B$ that has a free interface. The assembly $B$ is added as a component of assembly $IND_S$, such that upon a permutation trigger, the assembly $IND_S$ changes, allowing the operation $\diamondsuit$ to be carried out, resulting in a coupling that takes place between $subcomp_i$ and $subcomp_j$; and $IND_S$ becomes a stable state as part of a construction of assembly $A$. □

## 5.1 Relative Optimality of Constructions

**Definition:** The construction length $|EL|$ is the number of permutation triggers that are present on a construction.

The notion of equivalence does not restrict the construction to be of the type $|EL|_1 = |EL|_2$, where the subscript is the particular construction. This allows for the definition of equivalent as:

**Definition:** Two constructions are equivalent, if they have the same start state and end state.

The equivalence defined above is to be differentiated from strict equivalence which is the case when $|EL|_1 = |EL|_2$.

The notion of equivalence then allows us to compare constructions that produce the same end state from a single start state. The most relevant concept in this comparison process is that of the optimal construction.

**Definition:** A construction $|EL|_1$ is optimal with respect to $|EL|_2$, if $|EL|_1 < |EL|_2$ and there is no $|EL|_0$, such that $|EL|_0 < |EL|_1 < |EL|_2$.

**Theorem 11.** *An execution $|EL|_1$ is optimal with respect to $|EL|_2$, if it contains a minimal subset of Permutation Triggers PT under the Permutation Process PP.*

*Proof.* Assume that $|EL|_1$ does not contain a minimal subset of Permutation Triggers $PT$. Then there exists an $|EL|_0$ that has $PT$ -1 permutation triggers, such that $|EL|_0 < |EL|_1$ , then $|EL|_0 < |EL|_2$ and therefore $|EL|_0$ would be optimal with respect to $|EL|_2$. Therefore, $|EL|_1$ contains a minimal subset of Permutation Triggers PT. □

## 5.2 Comments on Practical Approaches to Obtain Relatively Optimal Constructions

While the theory covered so far contains the mathematical aspects of the theory, it is still not implementable. For the theory to be implementable, there must be a a temporal index $\tau$ and a function that maps states and permutation triggers to a temporal interval $\Delta\tau$. It is on this temporal construction where a technology embodied as a technological assembly can be judged as feasible or not. Given theorem 4, it is clear that there are some problems with assembly construction. This points to the main problem within the analysis of technology and it is presented here in the following conjecture:

**Conjecture:** There is no direct algorithm that shows a procedure of going from a particular $IND_S$ to the construction of an assembly.

**Remark:** The problem is based on the assumption apriori, that there is no information about restrictions and that any algorithm would have to go through validation of all conjuncts and all permutations in theorem 4. The best case scenario is to do a stochastic search to find out parts of states and transitions that correspond to the technological state machine.

When having this hurdle, then the only reasonable solution is that the algorithm utilized to perform a construction that goes from $IND_S$ to a construction of assembly $A$ is by stochastic search. This is also not realizable except for the simplest of assemblies when constraints are not known. To address these problems, there are several strategies that can be implemented[18].

The strategies to deal with the complexity of technological assemblies are:

1. Implement a memory device for assemblies subcomponents, component interfaces, requirements, their constraints, and their construction.
2. Component and subcomponent reuse.
3. Assume that there exist reasonable similarity relationships between unknown to known: assembly subcomponents, component interfaces, requirements and constraints.
4. For unknown assemblies and subcomponents, limit $m$, $n$, or both.

The first strategy allows for the accumulation of knowledge of the construction and the elements, (subcomponents, etc.), contained in such constructions. The second allows the memory device to engage in other constructions of assemblies once the construction is established. The third allows for a mechanism of generalization on which to bundle unknown constructions and make them possible. This comes with the caveat of possibly a not implementable construction that leads to failure. Lastly, when all three previous strategies have been implemented, the unknowns are handled in a limited problem space.

## 6 Conclusion and Future Work

This work has presented the the foundations of the formalism for the analysis of technology. The formalism derived uses the notion of technological state machine to explain the permutations carried out on components and the interaction between components that can give rise to couplings. Successful couplings on a set of components with their couplings then becomes a stable state. The stable state of the set of components is then defined as an assembly. The assembly, under this interpretation, is the basic unit of technology. The work has focused mainly on how these technology assemblies are constructed and the complexity in achieving them.

The work points to future effort to solve how to provide details to further formalize the strategies to deal with the complexity of technological assemblies. Some of the work that remains is the concept of technological description which is important in explaining further complexities that arise from the construction of assemblies under practical scenarios. Another area to further research on is the handling of the four strategies on a limited problem space to create new assemblies. On these, there is a question of novelty involved and how it is determined. The question also remains as to what are the bounds of new technological development of assemblies given the limitations imposed by the strategies to deal with the complexity of assembly constructions.

## References

Arthur, W.B., (2009). *The Nature of Technology: What It Is and How It Evolves.* 1st ed., London: Free Press.

Geigel, A., (2021). Formalizing Technology. Manuscript submitted for publication.

Hamilton, A.G., (1988). *Logic for Mathematicians.* Cambridge: Cambridge University Press.

Suppes, P., (1960). *Axiomatic Set Theory.* Series, New York: Dover Publications.

[18]The strategies are implemented in technological research and construction. They simply have not been formalized in a manner as proposed here.