Chapter 1

Definitions

1.1 Components

Definition 1.1.1 (Atomic and composite components)**.** *[26] [10][13] Fix a set C of atomic components. The set of* all *components,* Component*, is defined as the smallest set satisfying:*

1. $C \subseteq \text{Component}$.
2. *If* $c_1, \ldots, c_n \in \text{Component}$ *with* $n \geq 1$*, then*

$$\mathsf{Comp}(c_1, \ldots, c_n) \in \text{Component}.$$

No other elements are in Component*. In particular, any element of* Component *is obtained by finitely many applications of the* Comp *constructor to atomic components in* C*.*

Remark 1.1.2. *[13] The constructor* Comp *is neither assumed commutative nor associative: the tree structure and the ordered list of children in a composite are significant. In other words,*

$$\mathsf{Comp}(c_1, c_2) \quad \textit{and} \quad \mathsf{Comp}(c_2, c_1)$$

are distinct in general (ordering matters), and

$$\mathsf{Comp}\big(c_1, \mathsf{Comp}(c_2, c_3)\big) \neq \mathsf{Comp}\big(\mathsf{Comp}(c_1, c_2), c_3\big)$$

in general (parenthesization matters). Each occurrence of Comp *creates a new* composite component*; an atomic component cannot equal a composite, and two composites are equal only if they are built by the* Comp *constructor in the same tree shape with identical ordered constituents.*

Definition 1.1.3 (Immediate children)**.** *[10] The children function*

$$\text{children} : \text{Component} \longrightarrow \text{Component}^*$$

returns the (possibly empty) finite list of immediate subcomponents of a component:

$$\text{children}(c) := \begin{cases} [], & c \in C, \\ [\, c_1, \ldots, c_n \,], & c = \mathsf{Comp}(c_1, \ldots, c_n). \end{cases}$$

Thus atomic components have no children, while a composite

$$c = \mathsf{Comp}(c_1, \ldots, c_n)$$

has $\text{children}(c) = [\, c_1, \ldots, c_n \,]$*. By convention the empty list* $[]$ *has length* 0*.*

Definition 1.1.4 (Arity)**.** *For any $c \in$ Component, the* arity *of c is the number of its immediate children, i.e.*

$$\text{arity}(c) := |\text{children}(c)|.$$

An atomic component has arity 0.

Definition 1.1.5 (Subcomponent relation)**.** *The subcomponent relation*

$$\text{Sub} \subseteq \text{Component} \times \text{Component}$$

is the reflexive–transitive closure of the immediate-child relation. In particular, if

$$c = \mathsf{Comp}(c_1, \dots, c_n),$$

then $(c_j, c) \in \text{Sub}$ for each $j = 1, \dots, n$, and every component is a subcomponent of itself. For $c \in$ Component, define

$$\text{Sub}(c) := \{\, d \in \text{Component} \mid (d, c) \in \text{Sub}\},$$

the set of all (immediate or nested) subcomponents of c.

Remark 1.1.6. *[10] By a straightforward induction on structure, one can show for any $c_1, \dots, c_n \in$ Component:*

$$\text{Sub}\big(\mathsf{Comp}(c_1, \dots, c_n)\big) = \{\mathsf{Comp}(c_1, \dots, c_n)\} \cup \bigcup_{j=1}^{n} \text{Sub}(c_j),$$

and

$$\text{Sub}(c) = \{c\} \quad \textit{if } c \textit{ is atomic.}$$

In particular, Sub(c) *is a finite set for every component c.*

Proposition 1.1.7 (Unique immediate decomposition)**.** *[13] If*

$$\mathsf{Comp}(c_1, \dots, c_n) = \mathsf{Comp}(d_1, \dots, d_m)$$

as elements of Component*, then $m = n$ and $c_i = d_i$ for all $i = 1, \dots, n$.*

Proof. Assume $\mathsf{Comp}(c_1, \dots, c_n) = \mathsf{Comp}(d_1, \dots, d_m)$ in Component. Since atomic components and composites lie in disjoint parts of the inductive definition, both sides of the equality must in fact be composite. Therefore each side has a well-defined list of immediate children by Definition 1.1.3:

- $\text{children}\big(\mathsf{Comp}(c_1, \dots, c_n)\big) = [\, c_1, \dots, c_n \,]$,
- $\text{children}\big(\mathsf{Comp}(d_1, \dots, d_m)\big) = [\, d_1, \dots, d_m \,]$.

Applying the children function to the assumed equality gives

$$[\, c_1, \dots, c_n \,] = [\, d_1, \dots, d_m \,].$$

Two finite lists are equal iff they have the same length and corresponding entries are equal. Hence $m = n$ and $c_i = d_i$ for each $i = 1, \dots, n$, as required. □

1.2 Literals and Ticked Literals

Definition 1.2.1 (Literal)**.** *[17] Let C be the set of atomic components from Section 1.1. A* literal *L represents a choice of which atomic components are present in the system at a given state. Formally, a literal is any subset $L \subseteq C$. (Intuitively, $c \in L$ means the atomic component c is present or active, while $c \notin L$ means c is absent or inactive in that state.) We denote the set of all literals by*

$$\text{Literal} \;:=\; \mathcal{P}(C),$$

the power set of C.

Definition 1.2.2 (Ticked literal)**.** *[24] [17] A* ticked literal *is a pair (t, L) consisting of a discrete time value $t \in \mathbb{Z}$ (the global clock tick) together with a literal $L \in$* Literal *holding at that time. We may write $L(t)$ to denote the literal L at time t.*

Remark 1.2.3. *[24] In this model, time is an integer-valued global parameter ticking forward in uniform steps. We will often depict a state of the system as a time-indexed literal $L(t)$, meaning that exactly the atomic components in subset L are present at tick t.*

1.3 Phenomena and Events

Definition 1.3.1 (Transition)**.** *[36] [38] A* transition *represents the change of a particular component from one tick to the next. Formally, a transition is a 4-tuple*

$$(c_{src}, c_{tgt}, t, t+1)$$

consisting of an initial component $c_{src} \in$ Component *(at time t), a resulting component $c_{tgt} \in$* Component *(at time $t+1$), and consecutive time steps $t, t+1 \in \mathbb{Z}$. We write*

$$\mathrm{Transition} := \{(c_{src}, c_{tgt}, t, t+1) \mid c_{src}, c_{tgt} \in \mathrm{Component},\ t \in \mathbb{Z}\},$$

for the set of all transitions, including a distinguished no-change transition *$\epsilon_{src \to src}$. (Intuitively, $\epsilon_{src \to src}$ denotes an "idle" transition that leaves all components unchanged from t to $t+1$.)*

Definition 1.3.2 (Phenomenon)**.** *[38] A* phenomenon *is any subset $\Phi \subseteq$* Transition *of transitions. In other words, a phenomenon is a (possibly empty) collection of component transitions happening over a single tick interval. We denote by*

$$\mathrm{Phen} := \mathcal{P}(\mathrm{Transition})$$

the set of all phenomena. Two special cases are worth noting: the empty phenomenon *$\Phi = \emptyset$ (nothing changes during the tick), and the* null phenomenon *$\Phi = \{\epsilon_{src \to src}\}$ (the only "transition" is the idle transition).*

Definition 1.3.3 (Domain of a phenomenon)**.** *[38] For a phenomenon $\Phi \in$* Phen*, its* domain *is the set of components that undergo a (nontrivial) transition in Φ. Formally,*

$$\mathrm{dom}(\Phi) := \{\, c \in \mathrm{Component} \mid \exists (c_{src}, c_{tgt}, t, t+1) \in \Phi \textit{ with } c \in \{c_{src}, c_{tgt}\}\}.$$

In words, dom(Φ) *is the set of all components appearing as a source or target in at least one transition of Φ. (By convention, the idle transition $\epsilon_{src \to src}$ does not contribute any component to the domain.)*

Definition 1.3.4 (Union of disjoint phenomena)**.** *[20] [38] If $\Phi_1, \Phi_2 \in$* Phen *are phenomena with disjoint domains* dom$(\Phi_1) \cap$ dom$(\Phi_2) = \emptyset$*, their* union *(disjoint union) is defined as*

$$\Phi_1 \oplus \Phi_2 := \Phi_1 \ \cup \ \Phi_2.$$

In this case their domains are also disjoint in the union:

$$\mathrm{dom}(\Phi_1 \oplus \Phi_2) = \mathrm{dom}(\Phi_1) \cup \mathrm{dom}(\Phi_2) \quad (\textit{a disjoint union of sets}).$$

Intuitively, $\Phi_1 \oplus \Phi_2$ represents two independent sub-phenomena occurring in parallel (since they affect completely separate sets of components). If dom(Φ_1) *and* dom(Φ_2) *are* not *disjoint, then $\Phi_1 \oplus \Phi_2$ is* not *defined (one cannot form a union of overlapping phenomena without violating the uniqueness of transitions for components).*

Remark 1.3.5 (Concurrency of independent events)**.** *[20] The notion of disjoint union formalizes how multiple independent events may occur* concurrently*. If two phenomena involve disjoint sets of components, they represent independent events that do not interfere and can be treated as occurring in parallel during the same tick. In our model, events are not globally exclusive: it is possible for several disjoint sub-phenomena to happen simultaneously. Later, we will ensure that no component can undergo two different changes in the same tick (so events with overlapping domains are ruled out). Thus, concurrency is captured by combining phenomena using $\oplus$, while conflict is avoided by requiring disjoint domains.*

1.4 Actions and Processes

In order to specify which phenomena (state transitions) are possible in the system, we introduce the notion of *actions* that cause changes. Atomic actions will be the primitive steps that produce transitions, and more complex processes will be built by composing atomic actions in sequence or in parallel.

1.4.1 Atomic actions and the free monoid

Definition 1.4.1 (Alphabet of atomic actions)**.** *[4] Let $\mathcal{A}$ be a (possibly infinite) set of* atomic actions*. We think of each $a \in \mathcal{A}$ as a fundamental operation that the system can perform in one tick. We denote by $\mathcal{A}^*$ the set of all finite sequences (words) over the alphabet $\mathcal{A}$, including the empty sequence (denoted by* 1 *or ε). This $\mathcal{A}^*$ is the free monoid generated by $\mathcal{A}$ under the operation of sequence concatenation. We will use letters like $u, v, w \in \mathcal{A}^*$ to range over action sequences (also called* processes *when viewed semantically).*

1.4.2 Atomic-action semantics

Definition 1.4.2 (Atomic-action semantics)**.** *[36] Each atomic action $a \in \mathcal{A}$ is equipped with two semantic maps describing its effect on the state:*

$$[a]_{\text{world}} : \mathbb{Z} \times \text{Literal} \longrightarrow \mathbb{Z} \times \text{Literal}, \qquad [a]_{\text{phen}} : \text{Literal} \longrightarrow \text{Phen}.$$

These must satisfy, for every $t \in \mathbb{Z}$ and every literal $L \in$ Literal:

(1) World update:

$$[a]_{\text{world}}(t, L) = (\, t + 1,\ a(L)\,),$$

where we write $a(L)$ for the resulting literal at time $t+1$. In other words, performing action a advances the global clock by one tick and transforms state L into the new state $L' = a(L)$.

(2) Literal projection:

$$\Phi \;=\; [a]_{\text{phen}}(L) \quad \Longrightarrow \quad \text{proj}(\Phi) = a(L).$$

In words, the net effect of the transitions in Φ is exactly to change the state from L to $a(L)$ (with no other side-effects). This ensures consistency between the two semantic maps.

(3) Locality & preservation: *The phenomenon $\Phi = [a]_{\text{phen}}(L)$ satisfies the constraints given by Axiom 1.4.4 (atomic preservation) and Axiom 1.4.5 (locality), below.*

Axiom 1.4.3 (Totality)**.** *[36] Each atomic action $a \in \mathcal{A}$ induces a total function on literals,*

$$a : \text{Literal} \longrightarrow \text{Literal}.$$

That is, for every literal $L \in$ Literal, the result $a(L)$ is defined (there are no preconditions for applying any atomic action). In particular, if an action has no real effect in a given situation, it may simply leave the state unchanged.

Axiom 1.4.4 (Atomic preservation)**.** *(Conservation of components.) For every literal $L \in$ Literal and every atomic action $a \in \mathcal{A}$, if*

$$a(L) = L',$$

then

$$L' \subseteq L.$$

In other words, a does not change the underlying set of atomic components present in the system: atomic components cannot be created or destroyed by any single-tick action; they can only be reorganized or change roles. Equivalently, $a(L)$ and L contain the same atomic elements, possibly arranged differently or endowed with different attributes.

Axiom 1.4.5 (Locality)**.** *Each atomic action $a \in \mathcal{A}$ has a finite* static scope $\operatorname{scope}(a) \subseteq C$*. If a component $c \notin \operatorname{scope}(a)$ lies outside this scope, then c remains unchanged by a. Equivalently, $[a]_{\text{phen}}(L)$ contains no transition affecting c (as source or target). Thus, an atomic action can only directly cause transitions for components within its scope. By this axiom and the finiteness of* $\operatorname{scope}(a)$*, every atomic action affects at most finitely many components in any given state.*

(Together, Axioms 1.4.4 and 1.4.5 enforce physical constraints on atomic actions: no new fundamental components appear or disappear in one tick, and actions have only local impact on a bounded part of the system.)

1.4.3 Composition of actions

We now describe how the semantics of atomic actions extend to sequences of actions. Sequential composition corresponds to performing actions one after another over multiple ticks.

Definition 1.4.6 (Action of a sequence)**.** *[4] Let $\tilde{w} = a_1 a_2 \cdots a_k \in \mathcal{A}^*$ be a finite sequence of atomic actions (with $k \geq 0$). For any literal $L \in$* Literal*, we define the* iterated action *of $\tilde{w}$ on L recursively as:*

$$\tilde{w}(L) := \begin{cases} L, & \text{if } \tilde{w} \text{ is the empty sequence,} \\ a_1\big(\,(a_2 \cdots a_k)(L)\big), & \text{if } k \geq 1. \end{cases}$$

That is, $\tilde{w}(L)$ is obtained by applying the actions $a_1, a_2, \ldots, a_k$ in order to the state L. We also extend the world-transformer semantics: given (t, L), we define

$$[\tilde{w}]_{\text{world}}(t, L) := (\, t + k,\ \tilde{w}(L)\,),$$

by applying each a_i in turn via $[a_i]_{\text{world}}$, so that after the full sequence the time has advanced by k ticks and the final state is $\tilde{w}(L)$. Similarly, the overall phenomenon $[\tilde{w}]_{\text{phen}}(L)$ is defined as the union of each intermediate action's transitions in sequence. (More formally, one can prove by induction that

$$[\tilde{w}]_{\text{phen}}(L) = \bigcup_{i=1}^{k} [a_i]_{\text{phen}}\big(a_{i+1} \cdots a_k(L)\big),$$

which is a union of transitions occurring at successive ticks.)

Axiom 1.4.7 (Semantic injectivity)**.** *[31] Let $u, v \in \mathcal{A}^*$ be two (finite) sequences of atomic actions. If*

$$u(L) = v(L) \quad \text{for every } L \in \text{Literal } (\textit{i.e. } u \textit{ and } v \textit{ have exactly the same net effect on all possible states}),$$

then $u = v$ as sequences. In other words, no two distinct action sequences induce identical behavior in all circumstances.

1.4.4 Process semantics

We now introduce more general *processes*, which may involve parallel (concurrent) execution as well as sequential execution of actions. This will allow us to describe phenomena where multiple disjoint component changes happen in the same tick.

Definition 1.4.8 (Process syntax)**.** *[4] Processes are built from atomic actions using sequential and parallel composition. Formally, the class of processes X is generated by the grammar:*

$$\mathsf{X} ::= a \mid \mathsf{X}_1 ; \mathsf{X}_2 \mid \mathsf{X}_1 \oplus \mathsf{X}_2 \quad \textit{where } a \in \mathcal{A}.$$

Here $X_1; X_2$ denotes the sequential execution of process X_1 followed by process X_2, and $X_1 \oplus X_2$ denotes the parallel (concurrent) execution of X_1 and X_2 in the same time interval. We impose the restriction that the parallel composition $X_1 \oplus X_2$ is permitted only if $\operatorname{scope}(X_1) \cap \operatorname{scope}(X_2) = \emptyset$ *— in words, the two processes* *must act on disjoint sets of components. (This condition prevents conflicting parallel actions on the same component.) We allow parallel composition of more than two processes by associating $\oplus$ to the left (for example, $X_1 \oplus X_2 \oplus X_3$ is interpreted as $(X_1 \oplus X_2) \oplus X_3$), which is defined only if* $\mathrm{scope}(X_1) \cap \mathrm{scope}(X_2) \cap \mathrm{scope}(X_3)$ *are pairwise disjoint. The unit process (empty sequence) is denoted by* 1 *and acts as an identity for sequential composition (i.e.* $1; X = X; 1$*).*

Definition 1.4.9 (Scope of a process)**.** *[4] [38] The* scope *of a process X, denoted* $\mathrm{scope}(X) \subseteq C$, *is defined inductively:*

- *If $X = a$ is an atomic action, then* $\mathrm{scope}(X) := \mathrm{scope}(a)$ *(the static scope of a, see Axiom 1.4.5).*
- *If $X = X_1; X_2$ is a sequential composition, then* $\mathrm{scope}(X) := \mathrm{scope}(X_1) \cup \mathrm{scope}(X_2)$.
- *If $X = X_1 \oplus X_2$ is a parallel composition, then* $\mathrm{scope}(X) := \mathrm{scope}(X_1) \cup \mathrm{scope}(X_2)$.

Thus, $\mathrm{scope}(X)$ *is the set of all components that process X might affect. Note that if $X_1 \oplus X_2$ is well-formed (disjoint parallel), then* $\mathrm{scope}(X_1)$ *and* $\mathrm{scope}(X_2)$ *are disjoint, so* $\mathrm{scope}(X)$ *has a finite size [20]. In particular, every process X has a finite scope (by finite union of finite sets), and it affects no components outside its scope.*

Definition 1.4.10 (World-state semantics of processes)**.** *[4], [31] The world-transformer semantics* $[X]_{\mathrm{world}}$ *extends to composite processes as follows:*

- *If $X = a$ (atomic), then*

$$[X]_{\mathrm{world}}(t, L) := (\, t+1,\ a(L)\,) \quad (\textit{as in Definition 1.4.2}).$$

- *If $X = X_1\ ;\ X_2$ (sequential), then*

$$[X_1; X_2]_{\mathrm{world}}(t, L) := \textbf{\textit{let}}\ (t', L') := [X_1]_{\mathrm{world}}(t, L)\ \textbf{\textit{in}}\ [X_2]_{\mathrm{world}}(t', L').$$

  In other words, first execute X_1 starting from (t, L) to reach an intermediate state (t', L') at time t', then execute X_2 from there. The final time and state reflect the consecutive execution of X_1 then X_2.

- *If $X = X_1 \oplus X_2$ (parallel), then*

$$[X_1 \oplus X_2]_{\mathrm{world}}(t, L) := \textbf{\textit{let}}\ (t_1, L_1) := [X_1]_{\mathrm{world}}(t, L),\ (t_2, L_2) := [X_2]_{\mathrm{world}}(t, L)\ \textbf{\textit{in}}\ (\, \max(t_1, t_2),\ L_1 \cup L_2\,).$$

Here both X_1 and X_2 are started in parallel *from the* same *initial time t and state L. Since they operate on disjoint scopes, they do not interfere with each other's progression. The process X_1 finishes at time t_1 with state L_1, and X_2 finishes at time t_2 with state L_2. Without loss of generality assume $t_1 \geq t_2$ (otherwise relabel). This means X_2 finished earlier, at tick t_2, while X_1 took longer (until t_1). During $[t_2, t_1]$, X_2 simply remains idle. The combined parallel process thus completes at time* $\max(t_1, t_2)$ *with resulting state obtained by* merging *the contributions of L_1 and L_2. We write*

$$L_1 \uplus L_2$$

for this union of state literals*—defined precisely in Definition 1.4.13—which, intuitively, coincides with $L_1 \cup L_2$ since X_1 and X_2 act on disjoint components. (If $t_1 \neq t_2$, one process idles waiting for the other to finish.)*

Definition 1.4.11 (Phenomenon semantics of processes)**.** *[31] The phenomenon-generation semantics* $[X]_{\mathrm{phen}}$ *of a process X is defined inductively on the process structure:*

1. *If $X = a$ is atomic, then*

$$[X]_{\mathrm{phen}}(L) := [a]_{\mathrm{phen}}(L),$$

   using the atomic-action semantics of Definition 1.4.2.

2. *If $X = X_1; X_2$ is sequential, let*

$$\Phi_1 := [X_1]_{\text{phen}}(L), \qquad L' := X_1(L)$$

(so L' is the intermediate literal after executing X_1). Then

$$[X_1; X_2]_{\text{phen}}(L) := \Phi_1 \ \cup \ [X_2]_{\text{phen}}(L').$$

Note even if X_1 and X_2 affect some of the same components, there is no conflict: those components simply undergo transitions at different ticks, all of which are included in the union. (We do not require disjoint domains for sequential composition.)

3. *If $X = X_1 \oplus X_2$ is parallel with disjoint scopes (so $\operatorname{scope}(X_1) \cap \operatorname{scope}(X_2) = \emptyset$), then*

$$[X_1 \oplus X_2]_{\text{phen}}(L) := [X_1]_{\text{phen}}(L) \ \oplus \ [X_2]_{\text{phen}}(L),$$

using the disjoint-union of phenomena from Definition 1.3.4.

Definition 1.4.12 (State-level action). *[4] For any process X and literal $L \in$ Literal, we define the* state-level action *of X on L by projecting out the final literal from the world-transformer semantics:*

$$\langle X \rangle(L) := \pi_2\big([X]_{\text{world}}(t, L)\big),$$

where $\pi_2(t', L') = L'$, and t is arbitrary because the net effect on Literal *does not depend on the absolute start time. Equivalently, if $[X]_{\text{world}}(t, L) = (t', L')$, then $\langle X \rangle(L) = L'$.*

Definition 1.4.13 (Tagged union of literals). *[4] If $L_1, L_2 \in$ Literal are two literals such that $L_1 \cap L_2 = \emptyset$, then their* tagged union *(or parallel union) is*

$$L_1 \ \uplus \ L_2 \ := \ L_1 \ \cup \ L_2.$$

This operator is used in the parallel world-state semantics of Definition 1.4.10. If L_1 and L_2 share any component then $L_1 \uplus L_2$ is undefined.

Remark 1.4.14 (Consequences of the model). *[31] The above formalism guarantees that any process X composed of atomic actions will itself obey the fundamental constraints. In particular, one can show by induction on the structure of X that:*

1. *$\langle X \rangle$ preserves the set of atomic components (no net creation or destruction), by Axiom 1.4.4 applied to each atomic sub-action.*

2. *X has finite scope and does not affect components outside its scope, by Axiom 1.4.5 and the scope definitions.*

Furthermore, parallel processes by construction never introduce races or conflicts for the same component. Thus, this framework provides a mathematically rigorous foundation for modeling physical systems where composite events are built from independent sub-events and fundamental actions, all while respecting conservation laws and locality.

Demonstration: Center-of-Mass and Global Translation Invariance

Definition 1.4.15 (Center of mass). *[33] Define the center-of-mass map [35]*

$$\text{COM}\colon \text{Component} \times \text{Literal} \ \longrightarrow \ \mathbb{R}^d$$

by

$$\text{COM}(c, L) := \begin{cases} L(c), & c \in C \ \textit{atomic}, \\ \dfrac{1}{\sum_{i=1}^n m(c_i)} \sum_{i=1}^n m(c_i)\, \text{COM}(c_i, L), & c = \mathsf{Comp}(c_1, \ldots, c_n). \end{cases}$$

Objective. Show that for any rigid composite $c \in \text{Component}$ and any global translation by a vector $v \in \mathbb{R}^d$, the center of mass commutes with that translation:

$$\text{COM}\big(c,\, L+v\big) = \text{COM}(c,\, L) \;+\; v,$$

where $(L+v)(c') := L(c') + v$ for all $c' \in C$.

Translation as an atomic action.

Definition 1.4.16 (Translation rune)**.** *[41] Fix $v \in \mathbb{R}^d$. Introduce an atomic action $\tau_v \in \mathcal{A}$ with*

$$\text{Domain}(\tau_v) = C, \quad f_{\tau_v}\big(L|_C\big)(c) = L(c) + v, \quad g_{\tau_v}\big(L|_C\big) = \big\{\, \big(c,\, L(c) \to L(c)+v,\, t, t+1\big) \mid c \in C \big\}.$$

Then

$$[\tau_v]_{\text{world}}(t, L) = \big(t+1,\; L+v\big), \qquad [\tau_v]_{\text{phen}}(L) = g_{\tau_v}(L).$$

Proposition 1.4.17 (Translational invariance of COM)**.** *[41] Let $c \in \text{Component}$ and $L \in \text{Literal}$. Define the translated literal $(L+v)(c') = L(c') + v$. Then*

$$\text{COM}\big(c,\, L+v\big) = \text{COM}(c,\, L) + v.$$

Equivalently, for the translation rune τ_v of Definition 1.4.16,

$$\text{COM}\big(c,\, \langle\tau_v\rangle(L)\big) = \text{COM}(c, L) + v.$$

Proof. We prove the first form by induction on the structure of c using Definition 1.4.15.

Base case. If $c \in C$ is atomic, then by Definition 1.4.15

$$\text{COM}(c, L+v) = (L+v)(c) = L(c) + v = \text{COM}(c, L) + v.$$

Inductive step. Suppose $c = \mathsf{Comp}(c_1, \ldots, c_n)$. By Definition 1.4.15,

$$\text{COM}\big(c,\, L+v\big) = \frac{1}{\sum_i m(c_i)} \sum_{i=1}^{n} m(c_i)\, \text{COM}\big(c_i,\, L+v\big).$$

By the induction hypothesis,

$$\text{COM}\big(c_i,\, L+v\big) = \text{COM}(c_i, L) + v.$$

Hence

$$\begin{aligned}
\text{COM}\big(c,\, L+v\big) &= \frac{1}{M(c)} \sum_{i=1}^{n} m(c_i)\big(\text{COM}(c_i, L) + v\big) \\
&= \frac{1}{M(c)} \Big(\sum_{i=1}^{n} m(c_i)\, \text{COM}(c_i, L) \;+\; v \sum_{i=1}^{n} m(c_i) \Big) \\
&= \frac{1}{M(c)} \sum_{i=1}^{n} m(c_i)\, \text{COM}(c_i, L) \;+\; v = \text{COM}(c, L) + v,
\end{aligned}$$

where $M(c) = \sum_i m(c_i)$ is the total mass. This completes the induction. ☐

Numeric illustration. Let $c = \mathsf{Comp}(c_1, c_2)$ with $m(c_1) = 1$, $m(c_2) = 3$. Choose

$$L(c_1) = (0,0), \quad L(c_2) = (2,0), \quad v = (1,1).$$

Then

$$\text{COM}(c, L) = \frac{1 \cdot (0,0) + 3 \cdot (2,0)}{1+3} = \big(1.5,\, 0\big).$$

Translating,

$$L+v:\; L(c_1) = (1,1),\; L(c_2) = (3,1) \quad\Longrightarrow\quad \text{COM}(c, L+v) = \frac{1 \cdot (1,1) + 3 \cdot (3,1)}{4} = \big(2.5,\, 1\big).$$

Indeed $\text{COM}(c, L+v) = \text{COM}(c, L) + v = (1.5, 0) + (1, 1) = (2.5, 1)$, confirming Proposition 1.4.17 in this case.

☐

Chapter 2

State Variables & Discrete Transitions

Building on the hierarchical component structure established in Chapter 1.1, we now introduce a rigorous treatment of *state* and *time* in our framework. In particular, we formalize how each physical body (component) carries quantifiable state properties and how those states evolve step-by-step in discrete time. We begin by defining atomic vs. composite bodies, then introduce state-valued literals (properties with real-valued data), ticked literals (time-indexed state assertions), and the two basic kinds of time-step transitions (no-change vs. updated). Finally, we define what constitutes a phenomenon—a finite sequence of such transitions—and explain how independent phenomena can be combined. This lays the groundwork for connecting the static component layer to the dynamic layer of runes and spell-based evolution.

2.1 Atomic and Composite Physical Bodies

Every physical body in the system is modeled as a component in a tree-structured hierarchy. We distinguish two kinds of components:

- **Atomic body:** An atomic component is a basic indivisible physical entity with no sub-components. Formally, it corresponds to a leaf node in the component tree (it has no children). Such a body cannot be further decomposed into parts—it is *atomic* in the sense of structure. (For example, an ideal point mass or a rigid particle with no internal parts can be treated as atomic.)
- **Composite body:** A composite component is a physical body that is composed of other components as its parts. In the component hierarchy tree, a composite is an internal node with one or more child components (which themselves can be atomic or composite). Thus, a composite body has an internal structure made up of sub-bodies. In other words, a composite is formed by grouping multiple components into a single whole (e.g., a rigid object made of several pieces, or a solar system composed of a star and planets).

Notation. We will use letters like A, B, C (and so on) to denote components. If B is a sub-component of a composite C, we consider B part of C's structure. The entire system can be viewed as a rooted tree of components, where leaves are atomic bodies and each composite node represents the union of its children. This structural layer provides the stage on which dynamic evolution will occur.

2.2 State-Valued Literals for Atomic Components

Each atomic component carries a set of measurable *state properties* that characterize its physical state. We associate *state-valued literals* to atomic bodies to represent these properties and their numeric values. Common state properties (for classical mechanics) include position q, momentum p, energy E, etc., but in general we assume a fixed set of property names (symbols) relevant to the model.

Definition 2.2.1 (State-valued literal). *[16] [1] Let $A \in C$ be an atomic component and x_i a state-variable name (e.g. q, p, s, E). A* state-valued literal *for A is an assertion of the form*

$$x_i(A) = v,$$

where $v \in \mathbb{R}$ denotes the value of property x_i on component A. For example, $q(A) = 5.2$ might mean A is located at position 5.2 (in some units or frame), or $E(A) = 10$ could mean A has energy 10 (in appropriate units).

These state-valued literals are essentially atomic assertions about a component's condition. In classical physical systems, the *full state* of a body at an instant is given by the values of all such dynamical variables, each with a well-defined real value. We assume all state variables take values in $\mathbb{R}$.

Remark 2.2.2. *We typically assign state-valued literals only to atomic components (the fundamental pieces of matter). Composite bodies inherit their physical state from their constituents—i.e. a composite's overall state is determined by the collection of states of its sub-components. (If needed, one could define aggregate properties for composites, but at this stage the focus is on atomic-level properties.) Thus, one can think of the state of a composite as the union of the state literals of all its atomic parts.*

2.3 Ticked Literals: State at Discrete Time Points

Thus far, a state-valued literal $x_i(A) = v$ has described a static fact (with no explicit time reference). To model dynamics, we introduce *ticked literals* [25] [16]—state literals annotated with a discrete time index. We consider time advancing in uniform discrete steps (ticks) indexed by integers $t \in \mathbb{Z}$ $(\ldots,-2,-1,0,1,2,\ldots)$. A ticked literal is then a binding of a state property to a value at a specific tick. Notationally, we write

$$x_i(A)[t] = v,$$

meaning "at discrete time t, the property x_i of component A has value v." This situates the state-valued literal in time. For example, $p_B(0) = 4.8$ would mean component B's momentum at $t = 0$ is 4.8 (in suitable units), and $q_B(1) = 5.0$ that at the next tick ($t = 1$), B's position is 5.0.

Time indexing. We treat t as a global discrete clock for the entire system. All components "tick" forward in sync, step-by-step. The interval between t and $t + 1$ represents one uniform time step (the fundamental discrete time quantum of our model). By using integer indices, we can talk about consecutive states of the system at $t = 0, 1, 2, \ldots$ (or any starting point). The choice of time origin ($t = 0$) is arbitrary; what matters is the sequence and difference between ticks. We will often compare states at successive ticks to describe how things change.

2.4 Discrete-Time Transitions (ε and Δ-transitions)

With time-indexed state in hand (Definition 2.2.1), we can define the basic transitions that occur from one tick to the next. Consider an atomic component A and one of its state properties x_i. Suppose at time t we have

$$x_i(A)[t] = x, \quad x_i(A)[t+1] = x'.$$

Then exactly one of the following holds:

Definition 2.4.1 (ε-transition (no change)). *[1] [16] If $x' = x$, we say the state of x_i undergoes an ε_{x_i}-*transition *over the interval $[t, t+1]$. In other words, nothing changes for that state variable during the tick, and the ticked literal remains the same. We sometimes call this an "epsilon" transition because it is like an empty update (analogous to an ε-move in automata).*

Definition 2.4.2 (Δ-transition (state update)). *[1] [16] If $x' \neq x$, we say the state of x_i undergoes a Δ_{x_i}-*transition *over $[t, t+1]$. This represents a genuine state update: the value of x_i at the new tick differs from its previous value.*

We often abbreviate these as simply ε- and Δ-transitions when the variable x_i is understood. For example, if $q(A)[5] = 7.0$ and $q(A)[6] = 7.0$, then an ε-transition occurred for the q-property; if instead $q(A)[6] = 7.5$, then a Δ-transition occurred.

Remark 2.4.3. *Thus these two types of transitions exhaust all possibilities for how a single state literal can evolve in one discrete step: it either stays the same (ε) or changes (Δ). At the level of individual components and properties, these atomic transitions will be the building blocks for composite phenomena (Definition 1.4.11).*

2.5 Phenomena as Finite Transition Sequences

With individual time-step transitions defined (Definitions 2.4.1 and 2.4.2), we can describe longer behavior over multiple steps. A *phenomenon* in our framework is defined as a finite sequence of state transitions over a contiguous block of discrete time steps, describing how some component(s) evolve during that interval [29] [25].

Definition 2.5.1 (Phenomenon over an interval). *Let $t_0, t_f \in \mathbb{Z}$ with $t_f \geq t_0$. A* phenomenon Π *over the interval $[t_0, t_f]$ consists of an assignment of values to each relevant state variable at each tick $t_0, t_0+1, \ldots, t_f$, such that for every consecutive pair of times $t, t+1$ in that range, the transition from t to $t+1$ for each state variable is either an ε–transition or a Δ–transition as defined above. In other words, Π can be seen as a sequence of states (a sequence of ticked literals)*

$$\Pi : \{\, \textit{state at } t_0,\ \textit{state at } t_0+1,\ \ldots,\ \textit{state at } t_f \,\},$$

where each adjacent pair of states is related by one of the allowed transitions. We require the time steps to be consecutive with no gaps; a phenomenon doesn't skip any ticks in between t_0 and t_f. The sequence is finite (having length $t_f - t_0 + 1$ states).

Example. If A is an atomic body, a simple phenomenon for A might be:

$$t=0:\ q(A)=7.0, \quad t=1:\ q(A)=7.0, \quad t=2:\ q(A)=7.5, \quad t=3:\ q(A)=7.5$$

Here $t_0 = 0$ and $t_f = 3$. Between $t = 0$ and 1 there is an ε–transition (position stayed at 7.0), between 1 and 2 there is a Δ–transition (position changed from 7.0 to 7.5), and between 2 and 3 another ε–transition. This describes A moving and then staying still over two ticks.

Because phenomena are defined as sequences of transitions, we can naturally *compose* [29] or combine them under certain conditions:

- **Sequential composition (time-concatenation).** [36, 29] If phenomena Π_1 over $[t_0, t_1]$ and Π_2 over $[t_1, t_2]$ satisfy that the end state of Π_1 at t_1 equals the start state of Π_2 at t_1, then their *concatenation* $\Pi_1 \cdot \Pi_2$ is a phenomenon over $[t_0, t_2]$. It simply extends the sequence of transitions. In formal terms, if the sequence of states in Π_1 is $\{s_0, \ldots, s_{t_1}\}$ and in Π_2 is $\{s_{t_1}, \ldots, s_{t_2}\}$, then $\Pi_1 \cdot \Pi_2 = \{s_0, \ldots, s_{t_1}, \ldots, s_{t_2}\}$.
- **Parallel combination (disjoint union of independent phenomena).** [38, 29] If two phenomena Π_1 over $[t_0, t_f]$ on component-set C_1 and Π_2 over the same interval on component-set C_2 satisfy $C_1 \cap C_2 = \emptyset$, then they can be *combined in parallel* by taking the disjoint union of their transitions at each tick. Formally, for each $t \in [t_0, t_f - 1]$, the transition set at $t \to t+1$ is

$$\Pi_1(t \to t+1) \ \oplus \ \Pi_2(t \to t+1)$$

(Definition 1.3.4). The resulting phenomenon faithfully contains all transitions that happened in Π_1 (for components in C_1) and all those in Π_2 (for components in C_2), with no interference.

Essentially, a phenomenon is a self-contained description of how some part of the system changes over a finite time span. We can extend phenomena in time by concatenation, and broaden their scope by parallel union, as long as consistency conditions are met. These properties will be useful for building complex behaviors from simpler pieces.

Bridging to dynamics. At this point, we have a formal language to talk about *what* happens to the system's state over time (in terms of ticked literals and transitions). This is the bridge between the static structural layer and the dynamic layer. In the next part of the framework, we will introduce *runes*—the rules or "laws of evolution" that dictate *how* and why those Δ-transitions occur (analogous to the equations of motion or forces in physics)—and *spells*, which are constructed collections of runes that drive the state changes. A phenomenon, as defined above, can be understood as the outcome or execution of some spell; it is the realized sequence of state updates (and holdings) that the spell produces over time. By having rigorous definitions of atomic/composite components, state-valued literals, and transitions in place, we ensure that when we formalize runes and spell-based evolution, they can be precisely described as mechanisms that generate these phenomena in a logically consistent way (without needing any notion of mystical "souls," but simply through component state and dynamic rules). The stage is now set to move from describing *what* evolves to how it evolves under the influence of specified rules.

Chapter 3

Runes

In the proposed framework, an *atomic rune* represents a fundamental physical law or effect by encapsulating two key behaviors: a *world-update* function and a *phenomenon-generation* function. The world-update function specifies how the system's state evolves under the rune's influence; the phenomenon-generation function produces the observable events or transitions from that state.

3.1 Atomic Runes via World-Update and Phenomenon-Generation Functions

Definition 3.1.1 (Atomic rune)**.** *[18] An* atomic rune r *consists of:*

- *A finite* domain $\mathrm{Domain}(r) \subseteq C$*, the set of components (or state-variables) that* r *can affect.*
- *A* next-state function
$$f_r \colon \mathrm{Literal}\big|_{\mathrm{Domain}(r)} \longrightarrow \mathrm{Literal}\big|_{\mathrm{Domain}(r)},$$
which updates the local state of the domain in one tick.
- *A* phenomenon-generation function
$$g_r \colon \mathrm{Literal}\big|_{\mathrm{Domain}(r)} \longrightarrow \mathrm{Phen},$$
which produces the observable transitions (a phenomenon) over the same tick.

These induce semantic maps

$$[r]_{\mathrm{world}}(t, L) \;=\; \big(t+1,\; L'\big), \quad [r]_{\mathrm{phen}}(L) \;=\; \Phi,$$

where

$$L'\big|_{\mathrm{Domain}(r)} \;=\; f_r\big(L\big|_{\mathrm{Domain}(r)}\big), \quad L'\big|_{C\setminus\mathrm{Domain}(r)} \;=\; L\big|_{C\setminus\mathrm{Domain}(r)}, \quad \Phi = g_r\big(L\big|_{\mathrm{Domain}(r)}\big),$$

and $[r]_{\mathrm{world}}, [r]_{\mathrm{phen}}$ *satisfy the atomic-action axioms of Definition 1.4.2 when restricted to* $\mathrm{Domain}(r)$*.*

Remark 3.1.2. *[18] This design cleanly separates each physical law's* local action *(via* f_r*) from its* observable manifestation *(via* g_r*). It mirrors the pattern in state-space models and Mealy machines: one function updates the state, the other produces outputs. By giving each rune its own domain, we ensure modularity—runes in disjoint domains [29] may run in parallel without interference.*

3.2 Parametric Form: Vector Fields and Generating Functions

[22]

Rather than restricting ourselves to two special runes ("F" and "K"), we can *parameterize* atomic runes by arbitrary vector-field and generating functions:

- **Vector fields for world-update.** Many physical laws are naturally expressed as a vector field F on state: for example, Newton's law $F = m\,a$ or Maxwell's equations for (E, B). In our framework, this becomes
$$f_r(s, \alpha) \;=\; s + F(s, \alpha)\,\Delta t,$$
or any discrete-time analogue, where s is the current state restricted to Domain(r) and α are external parameters. Different choices of F yield different runes, all fitting the same template.

- **Generating functions for phenomena.** The phenomenon-generation function g_r can be tailored to whatever observable effect the law produces. In the simplest case g_r is the identity (we observe the updated state itself), but one can also map to sensor-like readings, discrete events (collisions, decays), or even randomized outputs. Plugging in different g_r lets us capture continuous outputs (e.g. field values) or discrete events (e.g. photon emissions).

3.3 Examples of Atomic Runes

[3]

- **Gravity rune.** Domain(r) = {all masses}. f_r implements gravitational acceleration $F(s, p) = G\,m/r^2$, updating velocities and positions. g_r might generate free-fall trajectories or gravitational-wave signals.

- **Electromagnetism rune.** Domain(r) = {charges and currents}. f_r implements Maxwell's equations on (E, B). g_r outputs electromagnetic observables (field measurements, photon events).

- **Thermodynamics rune.** Domain(r) = {particles/heat reservoirs}. f_r enforces heat-flow or energy-distribution laws. g_r produces temperature readings or phase-change events.

- **Custom phenomenon rune.** Domain(r) arbitrary. f_r may be trivial (no continuous update), while g_r generates stochastic events (mutations, random inputs). This captures exogenous or random effects.

Remark 3.3.1. *Each of these is* atomic *in the sense of Definition 3.1.1. In a composite system, all runes act on the shared world-state in parallel; the overall evolution arises by merging their individual world-updates (vector-field contributions) and by taking the disjoint union of their phenomena. This unifies continuous and discrete effects in one coherent model.*

3.4 Power and Flexibility of the Generalized Scheme

[22]

This generalized scheme is extremely powerful: *any* physical effect that changes the state (mathematically via a vector field or update rule f) and produces an observable outcome can be encapsulated as an atomic rune (Domain, f, g). It aligns with control theory's state–output models, drawing on a rich body of formalism to design runic laws. Moreover, whenever new domains or effects emerge, one simply defines a new rune with the appropriate f and g, and it integrates seamlessly with the rest of the framework.

3.5 Potential Improvements and Extensions to the Model

While the two-function template for atomic runes (f_r for world-update and g_r for phenomenon-generation; see Definition 3.1.1) is already quite general, there are several ways one might refine or enhance the framework: [3]

Unification via underlying principles Instead of specifying f_r and g_r separately, one could derive both from a single *energy* or *action* functional (e.g. a Lagrangian or Hamiltonian). In physics, the equations of motion and conservation laws follow from extremizing such a scalar. Formulating each rune via a generator (action potential) would guarantee consistency (e.g. symplectic structure, conserved quantities) automatically. While elegant, this approach is more complex to implement for arbitrary domains and may be overkill when simple update laws suffice.

Composite or hierarchical runes Atomic runes are the building blocks; we can also *compose* them to represent more complex interactions. For example, a "weather" rune could take as input phenomena from a "solar radiation" rune and an "ocean current" rune, producing a coupled climate phenomenon. A formal mechanism for rune-composition (e.g. letting g_{r_1} feed into the domain of r_2) would increase expressiveness, at the cost of added dependency management.

Event triggers and conditions Some phenomena are inherently *event*-based rather than purely continuous. We can enrich g_r with conditional logic that watches for patterns in the local state (e.g. collisions when two bodies coincide, phase transitions when temperature crosses a threshold), and emits discrete events precisely when conditions are met. This hybrid of continuous updates (f_r) and conditional event-triggers (g_r) enables non-linear dynamics common in real-world systems.

Incorporating constraints Certain physical laws act as *constraints* rather than state-updates (e.g. incompressibility in fluid flow or rigidity in mechanics). We could model such laws by having f_r project the post-update state back onto a constraint manifold—or by having g_r emit a phenomenon when the constraint is violated. This extension captures constraint enforcement alongside evolution.

Parameterization and tuning Since runes are parameterized by vector-fields and generating functions, one can expose their parameters (e.g. force constants, material coefficients) for easy adjustment or even *learning* from data. Embedding a tuning layer allows the framework to fit experimental observations, yielding data-driven runes that refine themselves over time.

Each of these extensions trades added expressiveness against greater complexity. Depending on the application domain, one may choose the minimal two-function atomic rune or adopt one of the refinements above to capture richer behaviors.

3.6 Deterministic vs Non-Deterministic Models

A major question is whether our framework should be purely *deterministic*—every rune and process yielding a single predictable outcome—or allow *non-deterministic* behavior (randomness or multiple possible outcomes). The choice depends on the phenomena we wish to capture and the goals of the model. [7]

3.6.1 Why determinism is often sufficient

Most fundamental physical laws are deterministic: given an initial state, their world–update functions f_r (Definition 3.1.1) produce a unique next state. Classical mechanics, electromagnetism, and general relativity all fit this pattern. Even chaotic systems, while extremely sensitive to initial conditions, remain deterministic in that repeated runs with identical inputs yield identical traces of phenomena (see Definition 1.4.11). A deterministic core makes the model conceptually clear, reproducible, and easier to debug.

3.6.2 When non-determinism may be necessary

There are several scenarios in which introducing randomness or choice is beneficial:

- **Quantum phenomena.** At microscopic scales (e.g. radioactive decay, quantum jumps, spin measurements), only probabilistic laws exist. A "radioactive decay rune" must decide at each tick—according to the proper half-life probability—whether to emit a decay event. Here g_r must be stochastic.
- **Unmodeled or external influences.** In simulations of complex environments (Brownian motion, turbulence), one often uses random "kicks" to stand in for countless untracked interactions. A "molecular collision rune" might sample random impulses to approximate thermal fluctuations.
- **Emergent complexity and chaos.** Even deterministic rules can produce effectively unpredictable outcomes when chaotic. To guarantee variation (e.g. in a game or exploratory simulation), one can inject randomness to avoid repetitive trajectories.
- **Simplifying complex processes.** When a fully deterministic model is too detailed to simulate (e.g. each molecule in a fluid), a stochastic approximation can capture high-level behavior more tractably.

3.6.3 Designing for optional non-determinism

A good compromise is to *make runes capable of non-determinism* without requiring it everywhere:

- By default, implement f_r and g_r as deterministic functions.
- In runes that truly require randomness, allow f_r or g_r to draw from probability distributions (e.g. via a seeded RNG).
- Provide a reproducibility option (fixed seed) so that stochastic runs can be replayed deterministically when desired.

This way, nine out of ten runes remain deterministic (aligning with most physical laws), and the special cases use randomness only where it is physically or practically justified.

3.6.4 Recommendations

In summary, choose the level of nondeterminism that matches your project's goals:

- **Theoretical, law-driven modeling:** Stay deterministic to keep the framework clean, reproducible, and explanatory.
- **High-fidelity or exploratory simulations:** Introduce non-determinism in specific runes to capture random or emergent phenomena.
- **Hybrid approach:** Implement all runes with the capacity for randomness but activate it only when needed; otherwise fall back on deterministic behavior.

By making nondeterminism an *option* rather than the default, we preserve both clarity and flexibility in our runic model of physical laws.

Chapter 4

Spell Construction

In this section, we formalize the *construction of spells* from atomic runes, providing an algebra of composition that reflects both sequential (serial) and parallel (concurrent) combinations of effects. We adopt the principle that spells can be composed in sequence or in parallel, but *parallel composition is restricted to disjoint domains* of action. In other words, two subspells may be invoked simultaneously only if they affect separate components of the world state. This restriction avoids interference and ensures that spell composition remains purely functional and deterministic. We thus obtain a minimal yet expressive algebra of spells that is compositional and amenable to future extensions (e.g. adding constraints or multi-resolution updates). We now define the syntax and semantics of spells, and show how to compute the resulting world-state and phenomenon from a spell's structure.

4.1 Spell Syntax and Composition

Definition 4.1.1 (Spell syntax)**.** *[37, 31] Spells are built inductively from atomic runes using two forms of composition. Formally, the class of spells $\mathcal{S}$ is generated by the grammar*

$$S \ ::= \ a \ \mid \ S_1 \ ; \ S_2 \ \mid \ S_1 \oplus S_2 \quad \textit{where } a \in \mathcal{A}.$$

Equivalently:

[31]

- ***Atomic:*** *If $a \in \mathcal{A}$ is an atomic rune, then a is a spell.*
- ***Sequential:*** *If $S_1, S_2 \in \mathcal{S}$, then their* serial composition *$S_1; S_2$ is a spell.*
- ***Parallel:*** *If $S_1, S_2 \in \mathcal{S}$ satisfy* $\text{scope}(S_1) \cap \text{scope}(S_2) = \emptyset$*, then their* parallel composition *$S_1 \oplus S_2$ is a spell.*

The restriction on parallel composition ensures that the two subspells act on disjoint components of the world, so that they do not conflict. We refer to this property as domain disjointness*. (In particular, each atomic rune a has a finite static scope* $\text{scope}(a) \subseteq C$ *by Axiom 1.4.5, and we extend the* scope *map to composite spells below.) We also admit the trivial* unit spell 1*, which performs no action;* 1 *serves as the neutral element for serial composition.*

Definition 4.1.2 (Scope of a spell)**.** *[19] The* scope *of a spell S, denoted* $\text{scope}(S) \subseteq C$*, is defined inductively:*

- *If $S = a$ is an atomic rune, then* $\text{scope}(S) := \text{scope}(a)$ *(the static scope of a, see Axiom 1.4.5).*
- *If $S = S_1; S_2$ is a sequential composition, then* $\text{scope}(S) := \text{scope}(S_1) \cup \text{scope}(S_2)$*.*
- *If $S = S_1 \oplus S_2$ is a parallel composition, then* $\text{scope}(S) := \text{scope}(S_1) \cup \text{scope}(S_2)$*, which is defined only when* $\text{scope}(S_1) \cap \text{scope}(S_2) = \emptyset$*.*

Thus, scope(S) *is the set of all components that spell S might affect.*

By construction, if $S_1 \oplus S_2$ is well-formed, then scope(S_1) and scope(S_2) are disjoint subsets of C [5]. This guarantees that S_1 and S_2 *commute* (their effects do not depend on the order of application) and that we can meaningfully apply them in parallel. The set of all spells so generated will be denoted $\mathcal{S}$. Note that $\mathcal{S}$ is closed under the above compositions and is equipped with an identity element 1; serial composition is associative, and parallel composition is commutative and associative on disjoint subspells [19]. In the special case where an atomic rune a admits an inverse a^{-1}, spells may also include inverse runes and satisfy the usual cancellation rules [8], but we do not focus on inverses here except to give consistency to the algebra.

4.2 Spell Semantics

Every spell $S \in \mathcal{S}$ induces a deterministic transformation of the world state and produces an associated phenomenon (a set of transitions) when invoked. We formalize the *semantics of spells* [36] by defining two functions for each spell: a *world-state update* function (denoted $[S]_{\text{world}}$) and a *phenomenon-generation* function (denoted $[S]_{\text{phen}}$). These extend the atomic rune semantics from Definition 1.4.2 to composite spells. Intuitively, $[S]_{\text{world}}$ computes the new tick and literal state resulting from executing S, while $[S]_{\text{phen}}$ yields the set of state transitions (the phenomenon) that occurred during S's execution. We define these functions inductively on the structure of S [12].

Definition 4.2.1 (World-state transformer)**.** *For a spell S and initial ticked literal $(t, N, P) \in \mathbb{Z} \times \text{Literal}$, the world-update semantics $[S]_{\text{world}}(t, N, P)$ is defined as follows:*

- ***Atomic case:*** *If $S = a$ is an atomic rune, then*

$$[a]_{\text{world}}(t, N, P) := (\, t + 1,\ a(N, P)\,),$$

  where $a(N, P)$ is given by the rune's state-transition function. By Axiom 1.4.4, no components are gained or lost overall—only their classification as N or P may change.

- ***Sequential case:*** *If $S = S_1; S_2$, then*

$$[S_1; S_2]_{\text{world}}(t, N, P) := \textbf{let } (t_1, N_1, P_1) := [S_1]_{\text{world}}(t, N, P) \textbf{ in } [S_2]_{\text{world}}(t_1, N_1, P_1).$$

  Thus the total advance in time is additive: the final tick t_2 equals t plus the number of atomic rune invocations in S_1 plus those in S_2. The final literal (N_2, P_2) is the result of first applying S_1 to (N, P), then S_2 to the intermediate state.

- ***Parallel case:*** *If $S = S_1 \oplus S_2$ with* scope$(S_1) \cap$ scope$(S_2) = \emptyset$*, then*

$$[S_1 \oplus S_2]_{\text{world}}(t, N, P) := \textbf{let } (t_1, N_1, P_1) := [S_1]_{\text{world}}(t, N, P),\ (t_2, N_2, P_2)$$

$$:= [S_2]_{\text{world}}(t, N, P) \textbf{ in } (\,\max(t_1, t_2),\ N_1 \cup N_2,\ P_1 \cup P_2\,).$$

  If one subspell finishes earlier, it idles for the remainder of the tick while the other completes; since their scopes are disjoint, this has no effect on the outcome. The final literal state is assembled by taking each component's new state from whichever subspell acted on it (if any). The union $N_1 \cup N_2$ (resp. $P_1 \cup P_2$) is well-defined by disjointness.

Definition 4.2.2 (Phenomenon generator)**.** *For a spell S and an initial literal state $L = (N, P) \in$* Literal*, the phenomenon semantics $[S]_{\text{phen}}(L)$ is the phenomenon produced by executing S on L. It is defined consistently with the world-update above:*

- ***Atomic case:*** *$[a]_{\text{phen}}(N, P)$ is the phenomenon generated by the atomic rune a on state (N, P). By Definition 1.4.2, $[a]_{\text{phen}}(L) = a(N, P)$ as a set of transitions, and its literal-projection satisfies* proj$([a]_{\text{phen}}(L)) = a(N, P)$.

- ***Sequential case:*** *If $S = S_1; S_2$, let*

$$\Phi_1 := [S_1]_{\text{phen}}(N, P), \quad (N_1, P_1) := \pi_2\big([S_1]_{\text{world}}(t, N, P)\big).$$

  Then

$$[S_1; S_2]_{\text{phen}}(N, P) := \Phi_1 \;\cup\; [S_2]_{\text{phen}}(N_1, P_1).$$

  These transitions occur in successive time intervals; no disjointness is required for serial composition.

- ***Parallel case:*** *If $S = S_1 \oplus S_2$ with disjoint scopes, let*

$$\Phi_1 := [S_1]_{\text{phen}}(N, P), \quad \Phi_2 := [S_2]_{\text{phen}}(N, P).$$

  Then

$$[S_1 \oplus S_2]_{\text{phen}}(N, P) := \Phi_1 \;\oplus\; \Phi_2,$$

  the disjoint-union of the two phenomena (Definition 1.3.4). Because their domains are disjoint, the union introduces no conflict and yields a combined phenomenon covering the union of subsystems.

These definitions enforce that spell semantics is *compositional* [32]: the effect of a composite spell on state and on generated phenomena is determined entirely by the effects of its constituents. Sequential composition corresponds to function composition of state transformations and time-concatenation of transition sets, while parallel composition corresponds to (commuting) independent state transformations and disjoint union of transition sets. By induction on spell structure one can prove homomorphism properties such as *state-update homomorphism* and *phenomenon additivity* (to be proven).

4.3 Examples: Integration Schemes as Spells

The spell construction framework is particularly useful for encoding time-integration schemes in physics simulations [40]. We illustrate this by expressing a few basic integrators as spells composed of simple runes. Consider a single particle with canonical state variables position q and momentum p. Suppose we have defined two atomic runes:

- $a_P(h)$ (Momentum update rune): advances the momentum p by one time-step of size h under the influence of forces (e.g. $p \mapsto p + F(q)\,h$), while leaving q unchanged. This rune's scope is the momentum component.
- $a_Q(h)$ (Position update rune): advances the position q by step h using the current momentum (e.g. $q \mapsto q + \frac{p}{m}h$), while leaving p unchanged. This rune's scope is the position component.

Since these two runes act on disjoint components, they can be composed freely in sequence or in parallel. We now construct spells corresponding to well-known integrators:

Example 4.3.1 (Forward Euler)**.** *The explicit Euler integration step of size h can be realized by the serial spell that first updates momentum and then position:*

$$S_{\text{Euler}}(h) \;=\; a_P(h)\, a_Q(h).$$

Invoking $S_{\text{Euler}}(h)$ on an initial state (q, p) at time t produces a new state (q', p') at time $t + 2$ (since it consists of two atomic updates). In effect, p is incremented by $F(q)\,h$, and then q is incremented by the new momentum p'. The phenomenon $[S_{\text{Euler}}(h)]_{\text{phen}}$ contains two transitions per affected component: one for the momentum change and one for the position change, in sequence [21].

Example 4.3.2 (Symplectic Leapfrog)**.** *A classic second-order symplectic integrator for Hamiltonian systems is the Stormer–Verlet (leapfrog) method. It can be written as three substeps: a half-step momentum update, a full-step position update, and another half-step momentum update. In rune notation:*

$$S_{\text{Leapfrog}}(h) \;=\; a_P\big(\tfrac{h}{2}\big)\; a_Q(h)\; a_P\big(\tfrac{h}{2}\big).$$

This spell first performs the "kick" on p over half the interval, then drifts q over the full interval using the updated momentum, and finally performs another half-kick on p. The resulting state after three atomic updates matches the standard leapfrog scheme. Because the momentum updates straddle the position update, this spell is symplectic*, preserving phase-space volume and energy behavior over long times [14].*

Example 4.3.3 (Parallel independent updates). *As an illustration of parallel composition, suppose our world consists of two non-interacting subsystems (e.g. two distant particles with no forces between them). Let S_1 be a spell that updates the first particle, and S_2 a spell that updates the second (each acting on disjoint components). Then the combined spell*

$$S \;=\; S_1 \oplus S_2$$

advances both subsystems in parallel within one tick. The world-state update $[S]_{\text{world}}(t, L)$ advances each particle's state according to its respective spell (and the global time advances by one tick, or by the maximum of their internal lengths). The phenomenon $[S]_{\text{phen}}(L)$ is the disjoint union $\Phi_1 \oplus \Phi_2$ of the individual phenomena, reflecting simultaneous but independent updates. This compositional property is useful for modeling complex dynamics *where multiple subsystems evolve concurrently [7].*

Example 4.3.4 (Mass–Spring Oscillator via Velocity-Verlet). *Consider a one-dimensional mass–spring system with mass m and spring constant k. We use the two atomic runes*

$$a_P(\tfrac{h}{2})\colon \begin{cases} p \mapsto p - \frac{1}{2}\,k\,q\,h, \\ q \mapsto q, \end{cases} \qquad a_Q(h)\colon \begin{cases} q \mapsto q + \frac{p}{m}\,h, \\ p \mapsto p. \end{cases}$$

The full velocity-Verlet spell of step size h is

$$S_{\text{V}}(h) \;=\; a_P\Big(\frac{h}{2}\Big)\; a_Q(h)\; a_P\Big(\frac{h}{2}\Big).$$

Let the initial tick be t_0 and initial state

$$L_0 = \{\, q(A) = q_0,\; p(A) = p_0 \}.$$

We execute the three atomic sub-runes in sequence:

1. Half-kick $a_P(\frac{h}{2})$:

$$[a_P(\tfrac{h}{2})]_{\text{world}}(t_0, L_0) = (t_1, L_1), \quad L_1 = \big\{q(A) = q_0,\; p(A) = p_1\big\},$$

where $p_1 = p_0 - \frac{1}{2}\,k\,q_0\,h$. The phenomenon $\Phi_1 = [a_P(\frac{h}{2})]_{\text{phen}}(L_0)$ consists of the single transition $\big(p_0 \to p_1,\; t_0 \to t_1\big)$.

2. Drift $a_Q(h)$:

$$[a_Q(h)]_{\text{world}}(t_1, L_1) = (t_2, L_2), \quad L_2 = \big\{q(A) = q_1,\; p(A) = p_1\big\},$$

where $q_1 = q_0 + \frac{p_1}{m}\,h$. The phenomenon $\Phi_2 = [a_Q(h)]_{\text{phen}}(L_1)$ is the transition $\big(q_0 \to q_1,\; t_1 \to t_2\big)$.

3. Half-kick $a_P(\frac{h}{2})$:

$$[a_P(\tfrac{h}{2})]_{\text{world}}(t_2, L_2) = (t_3, L_3), \quad L_3 = \big\{q(A) = q_1,\; p(A) = p_2\big\},$$

where $p_2 = p_1 - \frac{1}{2}\,k\,q_1\,h$. The phenomenon $\Phi_3 = [a_P(\frac{h}{2})]_{\text{phen}}(L_2)$ is $\big(p_1 \to p_2,\; t_2 \to t_3\big)$.

By sequential composition (Definition 4.2.1):

$$[S_{\text{V}}(h)]_{\text{world}}(t_0, L_0) = (t_3, L_3).$$

And by the time-concatenation rule (Definition 4.2.2),

$$[S_{\text{V}}(h)]_{\text{phen}}(L_0) = \Phi_1 \;\cup\; \Phi_2 \;\cup\; \Phi_3 = \{\, p_0 \to p_1,\; q_0 \to q_1,\; p_1 \to p_2 \,\}$$

with the indicated time stamps. The composite spell $S_{\text{V}}(h)$ faithfully reproduces the standard velocity-Verlet integrator within our formalism [27].

Chapter 5

Conservation and Constraints

In this chapter we incorporate conservation laws (invariants like energy or momentum) and constraints (e.g. fixed geometric links, incompressibility conditions) into our spell-based dynamic model. We address whether such physical laws should be built into the spell semantics (structurally enforced) or treated as optional properties of certain runes/spells, and how to implement constraints either via projection corrections or directly in rune evolution. We favor a formulation that is physically accurate and mathematically elegant, maintaining consistency with the ticked-time, component/phenomenon model established so far.

5.1 Conservation Laws as Invariants

Definition 5.1.1 (Conserved quantity / Invariant)**.** *Let $I\colon \text{Literal} \to X$ be a state-dependent quantity (e.g. total energy, total momentum), where X is typically $\mathbb{R}$ or a vector space. We say I is* conserved *(or an* invariant*) under a spell S if for every literal state L at some global tick t, whenever $S(L) = L'$, the value of I remains unchanged:*

$$I(L') = I(L).$$

When a spell S ensures $I(S(L)) = I(L)$ for all applicable L, we also say S conserves *I, and I is a* conservation law *for S. An invariant of an event Φ (Definition 2.5.1) is defined similarly: I is invariant if I evaluated on the state before the event equals I after the event.*

[34] shows that continuous symmetries of the action correspond to conserved quantities—grounding Definition 5.1.1.

Definition 5.1.2 (Hard vs. soft conservation)**.** *A conservation law I is called* hard *(or* strict*) if it holds exactly at each tick of evolution—i.e. every spell in the model (or every spell in a specified class of "physical" spells) preserves I at all times. In this case I is an inviolable invariant of the system. Conversely, I is* soft *(or approximate) if it is not universally preserved: certain runes or spells may violate the invariant or only maintain it approximately. Soft conservation typically means I might change due to specific spells (e.g. a dissipative or external-force spell) or due to numerical approximation, even if I is mostly conserved in closed subsystems or on average. Hard conservation is appropriate for fundamental physical invariants in closed systems (e.g. total momentum in an isolated interaction), whereas soft conservation allows modeling of open systems or small violations (e.g. energy loss to friction, integration errors).*

In modern numerical schemes, methods like PINN-Proj enforce invariants via projection—ensuring exact conservation [6]. In our formalism, conservation laws can be incorporated in one of two ways:

- **Structural enforcement.** We constrain the form of runes and spells so that certain invariants are preserved *by construction.* This means every applicable spell obeys the conservation law, usually thanks to underlying symmetries or cancellation properties in the spell algebra. For example, if momentum conservation is to be structural, one could require that any rune which increases momentum for some component be paired with a counterpart (in the same spell) that decreases momentum elsewhere by an equal amount, so that the net change cancels out. Such invariant-preserving spell forms ensure the quantity cannot change during evolution.

- **Optional property.** Alternatively, we treat conservation laws as *optional properties* that only certain runes or spells satisfy. In this view, we allow spells that violate an invariant, but we designate which spells $\mathcal{S}_{\text{mom}} \subseteq \mathcal{S}$ are momentum-conserving (each $S \in \mathcal{S}_{\text{mom}}$ satisfies $I_{\text{mom}}(S(L)) = I_{\text{mom}}(L)$ for all L), while other spells outside this class can change total momentum (modeling external impulses or non-physical effects).

Example 5.1.3 (Elastic collision of two masses)**.** *As a concrete instance of conservation and constraints, consider two particles A and B of masses m_A and m_B moving in one dimension. Their state literals are*

$$p_A(A) = p_A, \quad p_B(B) = p_B,$$

and the world-state literal is $L = \{ p_A(A) = p_A, \ p_B(B) = p_B \}$.

Define a single atomic rune a_{coll} (the elastic-collision rune) whose action on the momenta implements the standard one-dimensional elastic-collision update:

$$\begin{cases} p'_A = \dfrac{m_A - m_B}{m_A + m_B}\, p_A + \dfrac{2\, m_B}{m_A + m_B}\, p_B, \\ p'_B = \dfrac{2\, m_A}{m_A + m_B}\, p_A + \dfrac{m_B - m_A}{m_A + m_B}\, p_B. \end{cases}$$

Thus

$$[a_{\text{coll}}]_{\text{world}}\big(t,\ L\big) = \big(t+1,\ L'\big), \quad L' = \{ p_A(A) = p'_A,\ p_B(B) = p'_B \}.$$

The corresponding phenomenon $[a_{\text{coll}}]_{\text{phen}}(L)$ contains two Δ-transitions (one for each particle's momentum).

Now define two invariants:

$$I_{\text{mom}}(L) = p_A + p_B, \qquad I_{\text{E}}(L) = \frac{p_A^2}{2\, m_A} + \frac{p_B^2}{2\, m_B}.$$

One checks algebraically that

$$p'_A + p'_B = p_A + p_B, \quad \frac{{p'_A}^2}{2\, m_A} + \frac{{p'_B}^2}{2\, m_B} = \frac{p_A^2}{2\, m_A} + \frac{p_B^2}{2\, m_B}.$$

Hence a_{coll} conserves *both total momentum and total kinetic energy: it is a hard conservation rune for the invariants I_{mom} and I_{E}.*

In this way, a real-world physical process—an elastic collision—fits cleanly into our model: a single atomic rune whose spell semantics preserves the desired conservation laws by construction.

This elastic-collision update guarantees hard conservation of mass, momentum, and kinetic energy—an example of a symplectic update for collisional dynamics [11].

Lemma 5.1.4 (Closure of invariants under composition)**.** *Let I be a conserved quantity (hard invariant) as in Definition 5.1.1, and let $S_1, S_2 \in \mathcal{S}$ be spells that each conserve I. Then:*

1. *The sequential composition $S_1; S_2$ also conserves I.*
2. *If moreover* $\text{scope}(S_1) \cap \text{scope}(S_2) = \emptyset$*, then the parallel composition $S_1 \oplus S_2$ conserves I.*

Proof. Recall from Definition 1.4.12 that

$$\langle S \rangle(L)$$

denotes the final literal state after applying spell S to L. Since S_1 and S_2 each conserve I, we have

$$I\big(\langle S_i \rangle(L)\big) = I(L) \quad \text{for all literals } L \text{ and } i = 1, 2.$$

(1) Sequential composition. By the semantics of sequential spells,

$$\langle S_1; S_2 \rangle(L) = \langle S_2 \rangle\big(\langle S_1 \rangle(L)\big).$$

Thus

$$I\big(\langle S_1; S_2\rangle(L)\big) = I\big(\langle S_2\rangle(\langle S_1\rangle(L))\big) = I\big(\langle S_1\rangle(L)\big) = I(L),$$

where the middle equality uses that S_2 conserves I, and the last uses that S_1 does. Hence $S_1; S_2$ conserves I.

(2) Parallel composition. Assume $\text{scope}(S_1) \cap \text{scope}(S_2) = \emptyset$. Then by the parallel-semantics of spells (Definition 4.2.1) and the tagged-union of literals (Definition 1.4.13),

$$\langle S_1 \oplus S_2\rangle(L) = \langle S_1\rangle(L) \ \uplus\ \langle S_2\rangle(L),$$

and moreover each component of L lies in exactly one of those two disjoint parts. Since I is additive over disjoint subsets of components, we have

$$I\big(\langle S_1\rangle(L) \uplus \langle S_2\rangle(L)\big) = I\big(\langle S_1\rangle(L)\big) + I\big(\langle S_2\rangle(L)\big).$$

Each term on the right equals $I(L)$ by conservation of S_1 and S_2, and when L itself splits over the same disjoint components one recovers $I(L)$ by additivity. Hence

$$I\big(\langle S_1 \oplus S_2\rangle(L)\big) = I(L),$$

so $S_1 \oplus S_2$ conserves I, completing the proof. $\square$

See [15] for detailed proofs that symplectic (invariant-preserving) maps are closed under both sequential composition and composition on disjoint component sets.

5.2 Constraints and Their Enforcement

Definition 5.2.1 (Constraint manifold). *A* constraint *is a condition or relation that certain states must satisfy (often reflecting geometric or kinematic restrictions). Formally, let* Literal *denote the set of all literal states. A constraint can be specified as a subset*

$$M \ \subseteq\ \text{Literal},$$

representing the allowed states (the constraint manifold*) in state space. Equivalently, M may be defined by an equation or predicate $g(L) = 0$ (or $g(L) \in \{\texttt{true}\}$) that all states $L \in M$ must fulfill.*

We say a state L *satisfies* the constraint if $L \in M$. The system *enforces* the constraint if the actual state remains in M at all times (for all ticks t).

Definition 5.2.2 (Hard vs. soft constraints). *A constraint $M \subseteq$* Literal *is* hard *if it is strictly satisfied at every step of evolution—i.e. the state is never allowed to leave M. Equivalently, if $L(t) \in M$, then for every subsequent tick $t' \geq t$ the state $L(t')$ remains in M. A* soft *constraint is one that may be violated temporarily or by a small amount; the model does not absolutely prohibit leaving M, but might include forces or corrections that* tend *to keep the state near M.*

Projection-based schemes such as RATTLE/SHAKE enforce holonomic constraints by alternating unconstrained updates with projection steps [2, 23]. When enforcing a hard constraint in a discrete evolution, there are two principal strategies:

1. Constraint-Aware Spell Semantics (Built-In Enforcement). Restrict to spells S whose state-level action

$$\langle S\rangle\colon \text{Literal} \ \longrightarrow\ \text{Literal}$$

maps M into itself. In other words, whenever $\langle S\rangle(L)$ is defined and $L \in M$, one has $\langle S\rangle(L) \in M$. The constraint is then encoded in the rune semantics: each atomic or composite rune is designed (e.g. via Lagrange multipliers or coupled updates) so that it never leaves M. The spell algebra remains valid—sequential and parallel compositions of M-respecting spells still respect M.

2. Projection (Post-Step Correction). Allow spells to compute an intermediate state $L^* = \langle S \rangle(L)$ that may not lie in M, then apply a projection operator

$$\Pi_M \colon \text{Literal} \longrightarrow M,$$

to obtain

$$L' = \Pi_M\big(L^*\big).$$

Here Π_M can be treated as a corrective "rune" or spell invoked instantaneously at the tick boundary. For instance, after moving two linked particles freely to (A', B'), one sets

$$(A'', B'') = \Pi_M(A', B')$$

so that the link length constraint is restored.

Each approach has trade-offs:

- The *constraint-aware* approach preserves algebraic elegance and ensures hard enforcement by construction.
- The *projection* approach is simpler to implement but sits outside the core spell semantics and can break algebraic symmetries.

From a modeling standpoint, one should *encode constraints directly in the rune semantics* whenever possible, making them genuine invariants of the evolution. However, the projection fallback remains a useful, if less elegant, tool for enforcing hard constraints post-hoc.

Example 5.2.3 (Pendulum under Gravity)**.** ***Objective.*** *Model a point mass m on a rigid massless rod of length ℓ in a uniform gravitational field $g\hat{y}$, as a spell in our framework. We will:*

1. *Specify the component set C, literal states* Literal*, and constraint manifold M.*
2. *Define three atomic runes r_g, r_s, and r_d (gravity, spring-force, drift).*
3. *Build the composite spell $S_{\text{pend}}(h)$ via serial and parallel composition.*
4. *Prove that (after a projection) $S_{\text{pend}}(h) \colon M \to M$ (hard-enforces the rod length), and describe its phenomenon.*
5. *Sketch an alternate "constraint-aware" rune sequence preserving M by construction.*

The "kick–drift–kick" scheme is a standard symplectic integrator ("leapfrog") that preserves invariants by construction [39]. When constraints are enforced via built-in updates rather than post-projection, the integrator better preserves both symplectic structure and conservation [28, 9].

1. Component set and literals.

$$C = \{m\}, \quad \text{Literal} = \{\, L : C \to \{q, p \mapsto \mathbb{R}^2\}\},$$

so each L assigns $q(m) \in \mathbb{R}^2$ and $p(m) \in \mathbb{R}^2$. A ticked literal is $L[t]$.

2. Constraint manifold.

Definition 5.2.4 (Pendulum manifold)**.**

$$M \;=\; \big\{\, L \in \text{Literal} \mid \|q(m)\| = \ell \big\}.$$

This defines a holonomic constraint manifold consistent with classical formulations of constrained mechanical systems [30]. By Definition 5.2.1, $M \subseteq$ Literal *is our rigid-rod constraint.*

3. Atomic runes. *Fix time-step $h > 0$. All domains are $\{m\}$.*

Definition 5.2.5 (Gravity rune r_g)**.**

$$f_{r_g}(q, p) = \big(q,\; p + m\, g\, h\, \hat{y}\big), \qquad g_{r_g}(q, p) = \big\{\, (m, p \to p + mgh\,\hat{y},\, t, t+1)\big\}.$$

Definition 5.2.6 (Spring-force rune r_s)**.** *Let $u = q/\|q\|$ and $F_s = -\,k(\|q\| - \ell)\,u$. Then*

$$f_{r_s}(q,p) = \big(q,\ p + F_s\,h\big), \qquad g_{r_s}(q,p) = \big\{\,(m, p \to p + F_s\,h,\ t, t+1)\big\}.$$

Definition 5.2.7 (Drift rune r_d)**.**

$$f_{r_d}(q,p) = \big(q + \tfrac{p}{m}\,h,\ p\big), \qquad g_{r_d}(q,p) = \big\{\,(m, q \to q + \tfrac{p}{m}h,\ t, t+1)\big\}.$$

Each of these satisfies Definition 3.1.1 and Axiom 1.4.2.

4. Composite spell. *We choose the symplectic "kick–drift–kick" step:*

$$S_{\text{pend}}(h) = r_g(h/2)\,;\, r_s(h/2)\,;\, r_d(h)\,;\, r_s(h/2)\,;\, r_g(h/2).$$

(This is purely sequential since $\text{Domain}(r_g) = \text{Domain}(r_s)$*.)*

This step structure matches the leapfrog or Verlet scheme, frequently used with post-step projection to enforce rigid constraints in molecular or mechanical systems [28].

Proposition 5.2.8. *For any $L \in M$, define*

$$L^* \;=\; \langle S_{\text{pend}}(h)\rangle(L) \quad \textit{(state–level action, Def. 1.4.12).}$$

Then the projection

$$L' = \Pi_M(L^*) \quad \textit{satisfies } L' \in M.$$

Moreover the generated phenomenon

$$[S_{\text{pend}}(h)]_{\text{phen}}(L) = \Phi_{g_1} \cup \Phi_{s_1} \cup \Phi_d \cup \Phi_{s_2} \cup \Phi_{g_2}$$

where each Φ records the single-tick transitions in order.

Proof. By Definition 4.2.1, each half-kick and drift advances the tick by 1 and updates only $\{m\}$. Hence

$$L^* = \langle r_g\rangle\Big(\langle r_s\rangle\big(\langle r_d\rangle(\langle r_s\rangle(\langle r_g\rangle(L)))\big)\Big).$$

Explicitly one computes

$$q^* = q + \tfrac{p}{m}\,h, \quad p^* = p + \tfrac{h}{2}\big(F_s(q) + mgh\hat{y}\big) + \tfrac{h}{2}\big(F_s(q^*) + mgh\hat{y}\big).$$

In general $\|q^*\| \neq \ell$, so $L^* \notin M$. Applying the projector Π_M (Def. 5.2.2) along the radial direction restores $\|q\| = \ell$. Since Π_M is defined to map back onto M, we conclude $L' \in M$.

Finally, Definition 4.2.2 gives

$$[S_{\text{pend}}]_{\text{phen}}(L) = [r_g(h/2)]_{\text{phen}}(L) \cup [r_s(h/2)]_{\text{phen}}(L_1) \cup [r_d(h)]_{\text{phen}}(L_2) \cup [r_s(h/2)]_{\text{phen}}(L_3) \cup [r_g(h/2)]_{\text{phen}}(L_4),$$

which we denote $\Phi_{g_1} \cup \Phi_{s_1} \cup \Phi_d \cup \Phi_{s_2} \cup \Phi_{g_2}$. □

5. Constraint-aware alternative. *Define*

$$r_c : M \to M$$

by

$$f_{r_c}(q,p) = \big(q,\ p - (u \cdot p)\,u\big), \quad g_{r_c}(q,p) = \{\,(m, p \to p - (u \cdot p)\,u,\ t, t+1)\},$$

which projects momentum tangentially (and leaves q unchanged). Then

$$S'_{\text{pend}}(h) = (r_g(h/2); r_s(h/2)); r_d(h); r_c; r_s(h/2); r_g(h/2)$$

maps $M \to M$ by construction, with no separate projection step.

Bibliography

[1] Rajeev Alur and David L. Dill. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid Systems*, pages 209–229. Springer, 1993.

[2] Hans C. Andersen. Rattle: A "velocity" version of the shake algorithm for molecular dynamics calculations. *Journal of Computational Physics*, 52(1):24–34, 1983.

[3] Vladimir I. Arnold. Mathematical methods of classical mechanics. In *Graduate Texts in Mathematics*, volume 60. Springer, 1989. Covers Lagrangian and Hamiltonian formalisms as unifying physical principles.

[4] Jos C.M. Baeten. Process algebra: Equational theories of communicating processes. In *Handbook of Process Algebra*, pages 1–85. Springer, 2010.

[5] Jos C.M. Baeten and W.P. Weijland. *Process Algebra.* Cambridge University Press, 2001. Chapter 4: Parallel composition and algebraic properties.

[6] Anthony Baez, Wang Zhang, Ziwen Ma, Subhro Das, and LamM. Nguyen. Guaranteeing conservation laws with projection in physics-informed neural networks. Proposes PINN-Proj, a projection-based enforcement of physical invariants, 2024.

[7] Christel Baier and Joost-Pieter Katoen. Principles of model checking. *MIT Press*, 2008. Chapters on probabilistic transition systems and non-determinism.

[8] J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *Information and Control*, 60(1-3):109–137, 1984.

[9] Dong Eui Chang, Matthew Perlmutter, and Joris Vankerschaver. Feedback integrators for mechanical systems with holonomic constraints. *Sensors*, 22(17):6487, 2022. Uses Dirac formulas to enforce constraints without projection.

[10] Hartmut Ehrig, Fernando Orejas, Benjamin Braatz, Markus Klein, and Martti Piirainen. A generic component framework for system modeling. In *Fundamental Approaches to Software Engineering (FASE)*, volume 2306 of *LNCS*, pages 33–48. Springer, 2002.

[11] Herbert Goldstein, Charles Poole, and John Safko. *Classical Mechanics.* Addison-Wesley, 3 edition, 2002. See chapter on elastic collisions and conservation laws.

[12] Mike Gordon and Robin Milner. *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic.* Cambridge University Press, 1993. Chapter on inductive definitions and operational semantics.

[13] Gregor Gößler and Joseph Sifakis. Composition for component-based modeling. Technical report, INRIA / VERIMAG, 2004. component composition with tree hierarchy and formal semantics.

[14] Ernst Hairer, Christian Lubich, and Gerhard Wanner. *Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations.* Springer, 2nd edition, 2006. Covers Verlet and structure-preserving schemes.

[15] Ernst Hairer, Christian Lubich, and Gerhard Wanner. *Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations*, volume 31 of *Springer Series in Computational Mathematics*. Springer, 2006. Proves composition properties of symplectic integrators.

[16] David Harel. Statecharts: A visual formalism for complex systems. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 183–196. Springer, 2009.

[17] David Harel and Amir Pnueli. Reactive systems. In *Logic and Software Engineering*, pages 1–44. Springer, 2012.

[18] David Harel and Bernhard Rumpe. Meaningful modeling: What's the semantics of "semantics"? In *Integration of Software Specification Techniques for Applications in Engineering*, volume 3147 of *Lecture Notes in Computer Science*, pages 329–348. Springer, 2005.

[19] Matthew Hennessy. Algebraic theory of processes. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, pages 321–410. Springer, 2003.

[20] Tony Hoare. Communicating sequential processes. In *Theoretical Computer Science*, volume 30, pages 85–138, 1985.

[21] Arieh Iserles. *A First Course in the Numerical Analysis of Differential Equations*. Cambridge University Press, 2nd edition, 2009. Section 1.2 introduces the Forward Euler method as the simplest ODE integrator.

[22] Alberto Isidori. Nonlinear control systems: An introduction. In *Encyclopedia of Systems and Control*, pages 1–27. Springer, 2012.

[23] James A. Izaguirre, Daniel P. Catarello, Michael A. Wozny, and Robert D. Skeel. Langevin stabilization of molecular dynamics. *The Journal of Chemical Physics*, 122(8):084103, 2005.

[24] Leslie Lamport. Proving the correctness of multiprocess programs. *IEEE Transactions on Software Engineering*, SE-3(2):125–143, 1977.

[25] Leslie Lamport. Proving the correctness of multiprocess programs. *IEEE Transactions on Software Engineering*, SE-3(2):125–143, 1977.

[26] Kung-Kiu Lau, Ling Ling, Vladyslav Ukis, and Perla Velasco-Elizondo. Composite connectors for composing software components. In *Proceedings of the 2007 Workshop on Software Composition in Practice*, 2007.

[27] Benedict Leimkuhler and Sebastian Reich. *Simulating Hamiltonian Dynamics*. Cambridge University Press, 2004. Comprehensive treatment of velocity-Verlet and symplectic integrators in classical physics.

[28] Benedict Leimkuhler and Sebastian Reich. *Simulating Hamiltonian Dynamics*. Cambridge University Press, 2004. Covers SHAKE/RATTLE constraint-aware integrators.

[29] Nancy A. Lynch and Mark R. Tuttle. Formal models for component-based systems. *Science of Computer Programming*, 28(1):51–98, 1996.

[30] Jerrold E. Marsden and Tudor S. Ratiu. *Introduction to Mechanics and Symmetry*, volume 17 of *Texts in Applied Mathematics*. Springer, 1994. See sections on holonomic constraints and constrained Hamiltonian systems.

[31] Robin Milner. A calculus of communicating systems. *Lecture Notes in Computer Science*, 92, 1989.

[32] Eugenio Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, 1991. Formalizes compositional semantics via monads; applies to spell composition.

[33] Jacob Moore and Others. Centroids and center of mass via method of composite parts. LibreTexts, Mechanical Engineering, 2019. Describes composite-parts method: $x = \frac{\sum m_i x_i}{\sum m_i}$.

[34] Emmy Noether. Invariante variationsprobleme. *Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse*, pages 235–257, 1918. English translation often cited.

[35] OpenStax College. University physics volume 1: Center of mass. OpenStax, Rice University, 2016. Section 9.6: discrete particle system COM formula.

[36] Gordon D. Plotkin. A structural approach to operational semantics. In *Journal of Logic and Algebraic Programming*, volume 60, pages 17–139, 1981.

[37] Gordon D. Plotkin. A structural approach to operational semantics. In F.L. Bauer and R. Steinbrüggen, editors, *Logic and Algebra of Specification*, pages 3–38. Springer, 2000.

[38] Wolfgang Reisig. Petri nets and other models of concurrency. In *Handbook of Process Algebra*, pages 531–585. Springer, 2010.

[39] J. M. Sanz-Serna and M. P. Calvo. Numerical hamiltonian problems. *Applied Mathematics and Mathematical Computation*, 20:1–16, 1994. Describes symplectic leapfrog/kick–drift–kick integrators.

[40] J. M. Sanz-Serna and M. P. Calvo. Symplectic integrators for hamiltonian problems. *Acta Numerica*, 3:243–286, 1994. Standard reference on symplectic schemes, leapfrog, Euler, etc.

[41] Scott A. Smolka and E. Allen Emerson. Exploiting symmetry in model checking of concurrent systems. *Formal Methods in System Design*, 33(2):171–190, 2008.