Chapter 10
2nd Order Automation Engineering

A scientific theory is intelligible for scientists if they can recognize qualitatively characteristic consequences without performing exact calculations.

H.W. de Regt [61]

In this chapter, semantic closure meets system engineering: we describe how SCL systems can be constructed and controlled in practice, casting a developmental perspective on automation which we call '2nd order automation engineering'. Let us first give context to our objective, starting with a quote from Bundy and McNeil [40], who described in 2006 what they considered to be 'a major goal of artificial intelligence research over the next 50 years':

> For autonomous agents able to solve multiple and evolving goals, the representation must be a *fluent*, i.e., it must evolve under machine control. This proposal goes beyond conventional machine learning or belief revision, because these both deal with content changes within a fixed representation. The representation itself needs to be manipulated automatically.

The *extrinsic* motivation of an SCL system is utilitarian: it is to accept and perform work on command, where work is specified in terms of states to be reached (goals) or avoided (constraints). Thus, the system shall proceed toward constructing a world model such that, ideally, the entire state space is closed under inferencing: starting from as many states as might present themselves, any goal state which concludes the work is predictable (deduction) and conversely, starting from any goal state, as many other states as might occur are reachable (abduction).

By definition, work is a contingent measure for reducing entropy, which is always performed *against* an environment: laws of physics prevail, other agents follow their own agenda and, willfully or not, resist the 'mission' of the system in one way or another. Even the embodiment/agency of the system itself inevitably constrains its prospects of success. In that sense, the *intrinsic* motivation of the system is necessarily structural: it is to broaden the scope and depth ('horizontal' and 'vertical' structure) of

© The Author(s) 2022

J. Swan et al., *The Road to General Intelligence*,
Studies in Computational Intelligence 1049,
https://doi.org/10.1007/978-3-031-08020-3_10

a world model while absorbing the 'dents' caused by its contingencies. The *function* of an SCL system is therefore to accord both motivations, as per the definition of semantic closure (Sect. 7.2). In other words, this means to sustain a structural goal-directed homeostasis 'at the edge of chaos' as Kauffman puts it [167]—beyond that, ignorance unleashes oscillating behaviors, ending up in futility at best, in disaster at worst. Although Kauffman considers a multi-agent epigenetic landscape over evolutionary time scales, we are concerned with the growth of a single mind over the human time scale of work on command. As we will now see, 'adaptation at the edge of chaos' is as relevant for automation engineering as it is for the development of organisms, when re-cast as the hard-coded drive to control the multitude of feedback loops which unfold over the dynamic landscape of fine-grained models.

Kauffman developed his concept of adaptation from a systemic perspective to explain how open systems evolve in complexity to fulfill their intrinsic determination (spontaneous order) *despite* exogenous constraints (evolutionary pressure). But how do *forms*—the recognizable and composable manifestations of complexity—arise in the first place? When in 1917 D'Arcy Thompson published the first edition of 'On growth and form' [337] he probably did not anticipate that, four decades later, his inquiry into the genesis of stable structures would eventually be met by a formal theory that would extend its reach well beyond embryogenesis. In Thom's 'General Theory of Models' [336], forms arise from a substrate, determined by a *kinematics* (the laws which govern the arrangement/interaction of its constituents) whereas the temporal evolution of forms is regulated by a separate *dynamics*. Indeed, forms generally remain invariant under some selected pseudo-group G of interest (stability), yet sometimes break up completely (catastrophe) and new forms arise (morphogenesis). Thom's theory of models is a mathematical framework for eliciting the dynamics necessary to explain (and when possible, to predict) change, *despite* the inevitable under-specification of the underlying kinematics, and *parameterized* by G.

We confront the related inverse problem: to control the morphogenesis of controllers arising from a known kinematics. A general solution to this problem from the perspective of system engineering—'2nd order automation engineering'—ultimately deserves its own book. In the present work, we limit ourselves to presenting a minimal viable design within the scope of the SCL framework.

10.1 Behavioral System Engineering

As we have amply discussed in Part I, autonomous control systems are not systems that routinely switch between compiled behavior routines. Rather, the function of an autonomous control system is to engineer its own behavioral processes in compliance with requirements of scope, correctness, and timeliness. Behaviors result from control feedback loops instantiated over the substrate of a world model. Hence, to sustain the homeostasis introduced above, is to construct and maintain a world model such that the instantiation of desired control loops is predictable within the prescribed operational envelope. To an SCL system, control loops are the essential

observables by which it can assess the adequacy of its world model to the prospects of fulfilling its mission; as such, they constitute the *forms* of interest to be reasoned about.

Relational Control Loops

Functionally, a control loop is the coupling of a subset of rules (the output types of some rules are the inputs types of another) which guarantees the reachability of a goal state, starting from a *terminal* state (for now, the state of an actuator—a definition we will generalize later) constrained by *parameters*; parameters are non-terminal states which are not under the direct control of the loop. The guarantee of reachability is given by the composition of *contracts*, locally defined by rules. A rule is defined by $(A, P_A, f) \rightarrow (B, P_B)$ where A and B are types (generally composite), P_A a predicate on A, P_B a predicate on B, and f a transfer function that maps A to B. The pair (A, P_A) forms a *refinement type* [157], where the predicate P_A defines which subset of the possible representable values actually correspond to valid instances of the type A for the purpose of the rule. A contract is the guarantee that if one predicate holds, so will the other. This perspective of programming-by-contract [245], in which rules are relations (not functions), is close in spirit to the Behavioral System Theory developed by Willems [365] and confers a number of advantages: (1) unlike functions, all relations have a converse, (2) they allow ready modeling of nondeterminism, and (3) they allow identification and composition of invariants.

Figure 10.1 illustrates the relational perspective on control. Further possibilities arising from an alternative categorical presentation via string diagrams due to Baez and Erbele are discussed in Sect. 11.2.4. The generic relational controller (b) is a lens implemented by a single rule

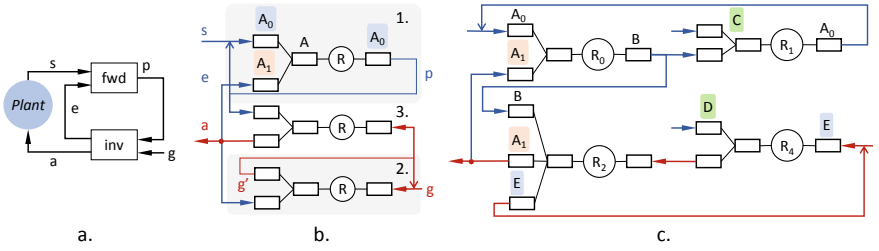

Fig. 10.1 Relational control loops. **a** A generic model-based controller in *functional* form (coupling of inputs/outputs *values*). A forward model *fwd* makes predictions p from sensor readouts s and efference copies e; based on such predictions and set-point g, an inverse model *inv* computes the action a. **b** The same controller in *relational* form (coupling of *constraints* on types) is a lens, implemented by a rule R. The controlled state is A_0, the terminal state A_1, the parameters $\varnothing$. There are two concurrent inference loops: (1) deduction loop and (2) abduction loop. The mixed flow (3) computes possible actions. **c** Complex control loops are composed of several rules/loops. Here, the controlled state is E, the terminal state A_1, the parameters $C \cup D$; see text for details

$$R : (A, P_A, f) \to (A_0, P_{A_0}), A = A_0 \cup A_1$$

through which concurrent inference flows are determined by contract satisfaction. In the first case (b.1), contracts are propagated forward to refine the state of interest (A_0) over an *increasing* time horizon. In the second case (b.3), there exists a solution a such as $f(P_A(a, p))$ satisfies $P_{A_0}(g)$. This is not so in the third case (b.2) where, instead of a, a subgoal g' is abducted such that $P_A(g', e)$ satisfies $P_{A_0}(g)$. The constraint refined by g' is furthermore refined over a *decreasing* time horizon as the b.2 loop iterates until the circumstances (here, p) are eventually such that b.3 holds—then the loop unwinds. This contract-constrained coupling of inferences constitutes the general kinematics of the ruleset and drives the instantiation of control loops at any scale in terms of rules involved.

The SCL framework is parameterized by expression language. For concreteness and ease of explanation, we adopt for the rest of this chapter, the language of *first-order linear arithmetic*[1] (FLA) [33], in which types are subspaces[2] of $\mathbb{R}^n$, rules are linear, and predicates are constructed inductively through Boolean combinations, i.e.,

$$\neg\varphi,\ \varphi \wedge \psi,\ \varphi \vee \psi,\ \varphi \to \psi,\ \varphi \leftrightarrow \psi,\ \forall x_i.\varphi,\ \exists x_i.\varphi$$

where φ and ψ are linear arithmetic formulae. Note that FLA negations express that subregions of the state space are 'to be avoided' to the degree of their likelihood—'forbidden' when the likelihood approaches one—a trait we will leverage in Sect. 10.4. Predicates in FLA are closed under the application of transfer functions, hence the predicate constraining the result of a rule can be automatically derived from the predicate constraining the inputs, namely by applying the transfer function to all of the constants, variables, and linear functions in the input predicate. Contracts can therefore be composed along any arbitrary chain of rules coupled via nonempty intersection between their input/output refinement types. This, in turn, allows representing control loops as (macro) rules and composing them as such; this plays an important part in system identification as we will discuss later.

Hierarchical Behavior Selection

Behaviors result from planning, which results itself from the instantiation of control loops, subjected to the kinematics discussed above. In SCL, abductions are always simulated: this allows exploring, concurrently, several possible courses of action (plans) to achieve a given goal. Of course, at some point in time, the system has to *commit* to one of these plans and start acting.

Each time an efference copy is a premise of a prediction, the corresponding (terminal) goal is added to an auxiliary list associated with the prediction. Such lists are

[1] We discuss other possibilities in Sect. 11.2.1.

[2] Recall that, as befits the open-ended setting, n varies dynamically, according to the addition of synthetic dimensions.

concatenated as predictions become premises of others, provided that the trajectories they define over the state space are consistent. As a result, a plan is a prediction of a goal state, associated with a list of terminal goals ordered by deadline. We call 'end-plan' a plan whose goal state M is the one imposed by the mission. All end-plans pursue the same goal and, at any point in time, there is at most one end-plan committed to. Anytime the deadline of a goal g (part of an end-plan) is reached, a procedure `commit` is invoked—g is said to be *signaled*. Note that the procedure is invoked concurrently by goals whose deadlines overlap within its WCET. At time t a goal g invokes `commit`:

1. If there is a current plan and its likelihood is below some threshold of acceptance T, then cancel the plan.
2. If there is no current plan, then find the best one (at minima, the one with the highest likelihood above T); if none is found, return.
3. Execute all goals in the current plan, whose deadline falls in $[t - WCET/2, t + WCET/2)$ and which have not been executed yet and are signaled. Due to the composite nature of states, non-terminal goals are the conjunction of sub-goals. If a terminal goal k is in such a conjunction C, then the execution of k is effective if and only if the executions of all other goals in C are effective.
4. If the execution of g is not effective, then cancel g.

For the commitment procedure to be effective requires that end-plans can be produced before the deadline of its earliest goal, which is of course rarely the case in practice. To keep the WCET of computing end-plans within acceptable bounds requires hierarchical planning. Rules are coupled via their types, which in turn are hierarchized by virtue of abstraction; in that sense, type abstraction imposes a hierarchy upon the ruleset. Consider for example, a mobile robot with an arm equipped with a gripper. When the gripper G is actuated, effectively seizing an object O, then when the robot moves, observations that O moves along the same trajectory as G will ensue: a rule R will be synthesized (see next section) to capture these, along with a positive precondition P on R predicting the success of R based on the prior context of A having been close enough to G and G having been actuated. The successful firing of P actually signifies 'G grabbed O', denoted by a new abstract type T (also detailed in the next section). In terms of abduction, T is considered a terminal state in the sense that the modalities of reaching T are irrelevant to the activity of planning the movement of O to some target location.

All type abstractions (super-types, latent states, etc.) actually decouple control loops. Accordingly, we extend the definition of terminal states to 'states transduced to lower levels of abstraction'—sensors and actuators being at level zero. 'Transduced' means that there exists a rule (a *transducer*) whose right-hand state is at a level of abstraction higher than that of its left-hand state. Hence, a terminal state of a control loop at level N may be the controllable state of another loop at $N - 1$, mediated by some transducer, as illustrated in Fig. 10.2. Finally, we can extend our definition of an end-plan to 'a plan whose goal state is a terminal state', considering the top-level mission goal state M a terminal state. The `commit` procedure will be invoked for

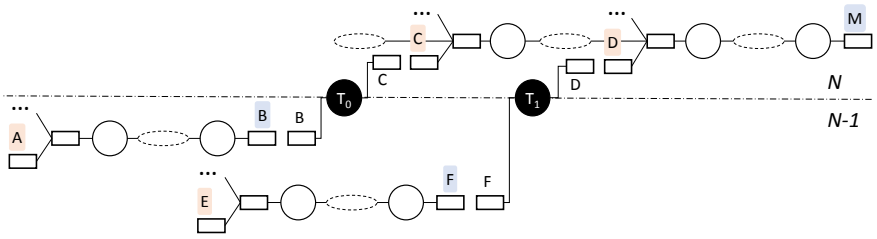

Fig. 10.2 Hierarchical behaviors (an ellipse denotes a path across the ruleset). An end-plan with end-goal M at abstraction level N is composed of terminal goals, some of which (C and D) are transduced (rules T_0 and T_1) into lowers levels: there these global goals become end-goals for local end-plans

all end-plans at some level N, each plan being computed in parallel to assemble the subgoals of plans at level $N + 1$.

System Identification

In automation engineering, system identification refers to the activity (human labor) of modeling an existing system at some appropriate level of description, chosen to confer some expected operationality: this can be, for example, that of predicting, controlling, upgrading, or all of these (e.g. to build a digital twin [280]). Confronted by the variability of both its environment and mission, an SCL system continually adapts its world model to ensure the existence of the necessary control loops, the set of which constitutes a model of its capabilities (self model) while—conversely—delineating the frontiers of its ignorance. Self-identification is the internal process producing and maintaining such a self model for a purpose: deficiencies therein are bound to trigger adaptation of the world model via rules and types *synthesis*. Conversely, adaptation is also triggered externally, any time the actual world is 'surprisingly' at odds with its model. Accordingly, the SCL dynamics is implemented by two complementary and concurrent synthesis processes: one proactive, the other reactive—Fig. 10.3 gives an overview.

In SCL, both the self and world models are written in the same language expressing the same substrate, on which operates the same categorical interpreter—the self model consists of rules which abstract the rules modeling the world. At this operational level of description, a control loop is a mapping between a system trajectory toward a goal state and a subset of the world model at some level of abstraction (its local substrate). To identify a control loop is to find a substrate matching the pattern illustrated in Fig. 10.4. Specifically, the substrate of a loop controlling the state S is a set of rules verifying the following structural properties:

(P_1) There exists an inference loop for predicting and subgoaling S.

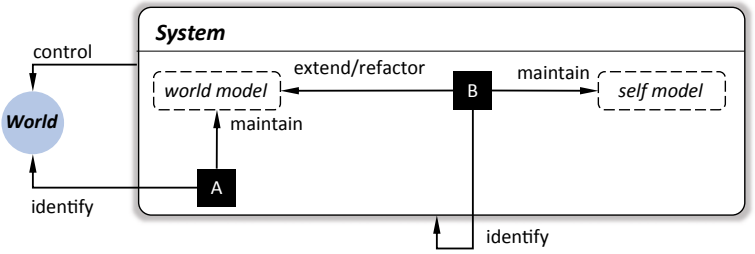

Fig. 10.3 Reflective system adaptation. System identification is performed *on* and *by* the system itself as its world model grows. The dynamics of the system is controlled by two processes by means of rule and type synthesis: *A reacts* to variations of surface phenomena (Sect. 10.2), whereas *B proactively* invokes structural heuristics (intensification/diversification, Sect. 10.3). *B* is domain-independent, decoupled from *A*, and operates deeper in the structure of the world model and at longer time horizons. Constraints on resources and responsiveness are less stringent for *B* than for *A*

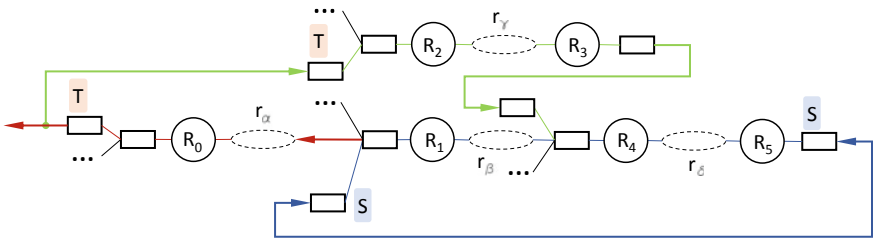

Fig. 10.4 General control loop template. T is the terminal state, S the controlled state. The properties P_1, P_{2a} and P_{2b} are illustrated in blue, red, and green, respectively; see text for details

(P_2) There exists a terminal state T such that (a) T is abducted from S and (b) S is predicted from efferent copies of T.

The self model is both hierarchical and compositional, both properties being inherited from rules and contracts. Technically, two rules

$$X : A \to B \text{ and } Y : C \to D \text{ with } B \cap C \neq \varnothing$$

are composed serially into a third $X + Y = Z : F \to G$, where

$$F = (A \cup C) \setminus (B \cap C) \text{ and } G = (B \cup D) \setminus (B \cap C)$$

We extend the $+$ operator to express joint-serial composition: a set of n rules $\{X_i : A_i \to B_i\}$ is composed with a rule

$$Y : C \to D \text{ with } B_i \cap C \neq \varnothing$$

into a third $X + Y = Z : F \to G$ where

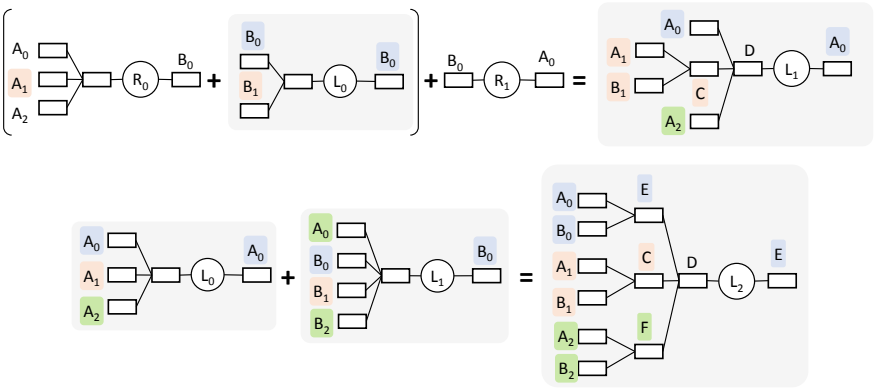

Fig. 10.5 Loop composition. The result of a + operation is a loop describing its arguments at one level of hierarchy higher. Controlled states are marked in blue, terminal states in orange, parameters in green, and control loops in grey; see text for details

$$F = \bigcup_{i=1}^{n} (A_i \cup C) \setminus \bigcup_{i=1}^{n} (B_i \cap C)$$

and

$$G = \bigcup_{i=1}^{n} (B_i \cup D) \setminus \bigcup_{i=1}^{n} (B_i \cap C)$$

Control loops can therefore be expressed by the same formalism used for rules (see Fig. 10.5) and, in particular, the general loop template with terminal state T and controlled state S is

$$L(T, S) = \{R_0 + r_\alpha + R_1 + r_\beta, R_2 + r_\gamma + R_3\} + R_4 + r_\delta + R_5$$

where the r_i are free parameters and the R_i are subject to the following constraints:

$$R_0 : A_0 \to B_0, T \in A_0 \qquad R_1 : A_1 \to B_0, S \in A_1$$
$$R_2 : A_2 \to B_2, T \in A_2 \qquad R_3 : A_3 \to B_3$$
$$R_4 : A_4 \to B_4, B_3 \cap A_4 \neq \varnothing \qquad R_5 : A_5 \to B_5, S \in B_5$$

L is then the search pattern for identifying control loops during system identification at arbitrary levels of abstraction.

10.2 Reactive Synthesis

The reactive synthesis of a rule is triggered by one of the following two events: the unpredicted success of a goal, or the failure of a prediction—a missing path in the model structure in the first case, a faulty path in the second. We begin with the first case.

Local Scale: Rules

We are concerned with synthesizing a rule of the form $A \rightarrow B$, where B is the observed goal state which the system could not predict, and A the history of all states observed before B. Of course, a heuristic is needed to limit the scope of A. For example, A can be restricted to the tuple formed by the last N most salient states only—the saliency of a state S being the attention it receives relatively to others, i.e., the relative number of references to S in the jobs of the highest priority (see Sect. 8.2). Of most relevance, we introduce at the end of this section the notion of *postcondition* which summarizes histories of states as hierarchical patterns. Saliency can then be strengthened considering the 'height' of a state in a hierarchy, reminiscent of the 'chunks' theorized to compose human short-term memory and learning—see [114] for a computational model.

As illustrated in Fig. 10.6, the synthesis of a rule $A \rightarrow B$ requires $dim(A) + 1$ successive non-colinear sample pairs (a_i, b_i) of vectors in A and B. Each newly observed vector in A (e.g. a_3) is transformed into its rejection from the hyperplane formed by the previous samples (a_0 and a_1); its associated vector in B (b_2) is transformed accordingly (into b_3). It then suffices to transform the resulting orthogonal basis into an axis-aligned unit basis and to transform the b_i accordingly in order to

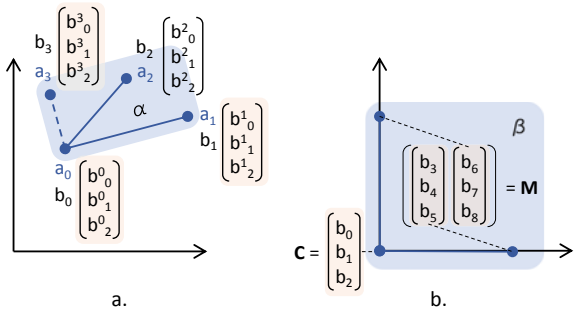

Fig. 10.6 Synthesis of a rule $(A, P_A, f) \rightarrow (B, P_B)$ with $A = \mathbb{R}^2$ and $B = \mathbb{R}^3$. **a** Samples, each a pair (a_i, b_i) of vectors in A and B, are observed in a succession from which an orthogonal basis (α) of A is constructed. **b** α is then transformed into an axis-aligned unit basis (β); f is $b : B = C + M \times a : A$. P_A is the polytope formed by the a_i whereas P_B is that formed by the b_i; see text for details

define the transfer function f as $b : B = C + M \times a : A$. P_A is the polytope enclosing the a_i whereas P_B is that enclosing the b_i. Null columns in M indicate that the inputs located in the tuple A at these column indices are irrelevant to the output; both A and P_A are pruned accordingly.

Polytopes grow as new positive evidences (samples) are observed whereas negative evidences trigger the synthesis of new local restrictions of f. In general, f is not linear globally—over the entire state space—hence applying f outside of P_A may yield an error beyond acceptable tolerance. As a consequence, a new rule R' is to be synthesized as above, whose predicate P'_A is disjoint from P_A. By this construction, P_A is kept convex: convexity warrants interpolation should an input fall within a predicate; outside the predicate, outputs are extrapolated, albeit with a confidence decreasing with the distance of the input to the predicate. Note that it is also possible to subject the likelihood of interpolated outputs to experience. For this, it suffices to maintain the covariance matrix and centroid of all past inputs: the likelihood of interpolated outputs is then inversely proportional to the Mahalanobis distance (times the reliability of the rule and the likelihood of the inputs).

Global non-linear mappings are approximated as the union of local linear ones, each defining its own convex local restriction of a more global domain. The mutual exclusion of the local domains P_A and P'_A is maintained, as they grow, using *preconditions* as follows:

1. If $P'_A \subset P_A$, then a negative precondition on R is synthesized with P'_A as its left-hand predicate.
2. Otherwise, if $P'_A \cap P_A \neq \varnothing$ then one negative precondition is synthesized for each rule, with $P'_A \cap P_A$ as its predicate.
3. Otherwise, no precondition on either rule is synthesized.

Global Scale: Pre- and Postconditions

A second possible trigger for the reactive synthesis of a rule is the failure of a prediction. We proceed as in the case of the unexpected success of a goal, to synthesize a rule N: $(C, P_C, f) \rightarrow (D, P_D)$ where (D, P_D) is the refinement type denoting the failure of some rule R. N is a negative precondition for R: when N fires, the reliability of the inferences produced by R is lowered, proportionally to that of the inference $d : D$ produced by N; during abduction, a goal matching the right side of R triggers the production of a subgoal to prevent N from firing, hence, other subgoals are derived in order to avoid instances of C. Figure 10.7 illustrates the flow of inferences.

Whereas predicates in left-hand refinement types impose *local* constraints on forward input bindings, negative preconditions impose *global* constraints on the target rule, i.e., they define the context in which the rule is likely to fail, regardless of the validity of the bindings of its left-hand refinement type. Just as there exist contexts for failure, there exist contexts for success: these are captured in the left-hand refinement types of positive preconditions—see Fig. 10.8. The synthesis of the positive precondition P is triggered by the success of R if not already predicted

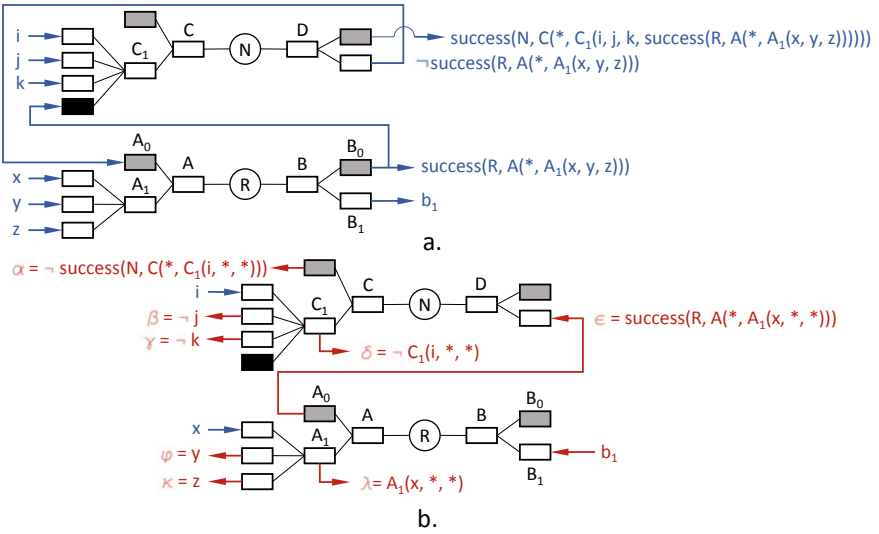

Fig. 10.7 A negative precondition N for a rule R is represented. The first type (in grey) embedded in the type of either the left- or right-hand term of R always denotes the success of R; inputs matching A_0 or B_0 do not trigger inferencing. **a** Forward inference (in blue): any time an instance $a : A$ triggers a forward inference, a prediction $success(R, a)$ is produced by R, which in turn may trigger the production by N of its negation: this lowers the likelihood of b_1. **b** Backward inference (in red): $b_1 \rightarrow \lambda \wedge \epsilon, \lambda \rightarrow \varphi \wedge \kappa, \epsilon \rightarrow \delta \vee \alpha, \delta \rightarrow \beta \vee \gamma$

by another positive precondition. Note that preconditions are rules and as such, can themselves be subjected to other preconditions.

Any time a precondition $P : A \rightarrow S$ (where S is the success/failure of some rule R) is synthesized, an abstract type X and a transducer T are also surmised: X is defined as a new dimension of the state space, and T as $A \rightarrow X$. The transfer function of T is learned as any other function, with the minor difference that since X is abstract (therefore not observable), evidences for the outcome of R are taken as proxies for evidences of X.

Mirroring preconditions, whereas predicates in right-hand refinement types define local guarantees on forward output bindings, *postconditions* define global consequences of rule firings. Recall that forward rule firing statements (e.g., $success(R, ...)$ or $\neg success(R, ...)$) are first-class states: hence they can be surmised as input types embedded in the composite left-hand type of a rule. Postconditions are rules taking the (forward) firing of other rules as input types. Such input types summarize (parts of) the state history: if a rule R admits A_i as input types composing its left-hand type, then the single input type $success(R, ...)$ in a postcondition S is equivalent to S admitting all the A_i (and the conjunction of the negations of all negative preconditions for R) as its own input types. Conversely, an input type $\neg success(R, ...)$ in S is equivalent to the negation of at least one of the A_i or the assertion of at least

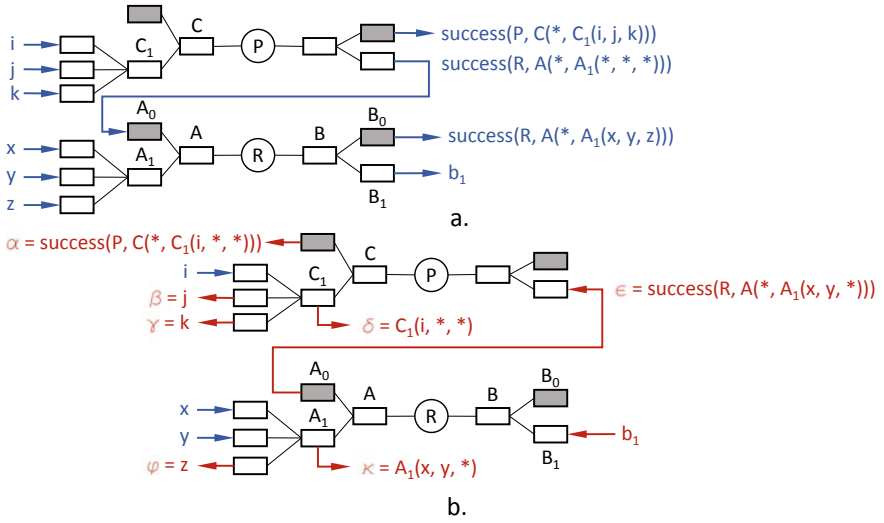

Fig. 10.8 A positive precondition P for a rule R is represented. **a** Forward inference: any time an instance $c : C$ triggers a forward inference, a prediction $success(R, *)$ is produced by P: this increases the likelihood of b_1. **b** Backward inference: $b_1 \rightarrow \kappa \wedge \epsilon, \kappa \rightarrow \varphi, \epsilon \rightarrow \delta \wedge \alpha, \delta \rightarrow \beta \wedge \gamma$

one of the negative preconditions for R. Note that some of the A_i may themselves be the firing of postconditions: this enables postconditions to capture state histories as temporal hierarchical patterns.

10.3 Proactive Synthesis

The purpose of proactive synthesis is to mitigate deficiencies observed in the set of macro-rules (control loops) describing the system, resulting from the process of system identification, as discussed in Sect. 10.1. We now proceed to describe an elementary strategy for proactive synthesis which frames intrinsic motivation in terms of control—possible alternative approaches are clearly an open-ended research issue. Within the discipline of metaheuristic search, it is often useful to characterize the state trajectory of a system as *diversifying* or *intensifying*.[3] According to Glover and Laguna [113]:

> The main difference between intensification and diversification is that during an intensification stage the search focuses on examining neighbours of elite solutions. The diversification stage on the other hand encourages the search process to examine unvisited regions and to generate solutions that differ in various significant ways from those seen before.

[3] The related notions of *exploration* and *exploitation* also appear in the literature on evolutionary computation and intrinsic motivation.

As observed by Blum and Roli [26], the notions of intensification and diversification are not mutually exclusive. For example, while random search is strongly diversifying and gradient descent is strongly intensifying, simulated annealing [172] lies somewhere in-between, progressing from diversification at high temperatures to intensification at low temperatures.

The control mechanism we describe here is derived from that of the *Reactive Tabu Search* (RTS) [18, 19] an extension of Glover's original tabu search [113]. Tabu search is a local search metaheuristic [208] in the same family of 'single-point search' techniques as stochastic gradient descent and simulated annealing. The basic mechanism of tabu search (termed *recency-based memory*) maintains a restricted local neighbourhood by prohibiting the choice of a neighbouring state if (some attribute of) that neighbour has been encountered recently. The simplest (or *fixed-tabu*) implementation implements the recency structure as a sequence of the last k states encountered, where k is the *tabu-tenure*. In addition to the recency-based memory structure (which could be said to model 'short-term' memory), many implementations also maintain a 'long-term' or *frequency-based* memory, which is essentially a frequency-histogram of attributes with a tenure much larger than k.

The essential idea of RTS is to inform control via dynamical system metrics. We therefore briefly recap dynamical systems terminology. Informally, an attractor of a dynamical system is a set of points in state space such that all 'nearby' points in the state space eventually move close to it. The simplest dynamical system is one in which there is a single fixed point which acts as an attractor for all states. The next simplest attractor is a *limit cycle*, in which trajectories converge to a closed loop. The *cycle length* of a dynamical system in state s is the number of iterations since s was last encountered (or ∞ if no encounter with s is recorded). It is also possible for the trajectory to be confined within some region of phase space but exhibit no obvious periodicity, due to the presence of a so-called *strange attractor*. RTS thus instruments the search space with recency and frequency information, in order to drive control mechanisms that:

1. are self-adaptive (tabu-tenure is a function of the moving average of the detected cycle-length);
2. maintain a good balance between intensification (i.e. exploration of promising regions) and diversification;
3. recognize when the search lies in the attractor of an unpromising region.

The presence of previously-encountered attributes in the recency list triggers the 'fast-reaction' mechanism which leads on successive iterations to a geometric increase in recency tabu-tenure. This tends to force the search to choose neighbours with unexplored attributes and will eventually break any limit cycle. Conversely, a 'slow-reaction' mechanism acts to counter redundant prohibition by reducing the tabu-tenure when the fast-reaction mechanism has been idle for some time. In order to detect the presence of strange attractors, the number of repeated attributes in the frequency structure is examined. When the number of such attributes exceeds a threshold level an 'escape mechanism' is activated, which (as per Battiti's original

implementation) consists of a random walk of length proportional to the moving average of the current cycle length.

Measures which characterize intensification and diversification are obtained from the recency and frequency structures and used to select of an inference method. Describing any particular selection mechanism in detail would be overly specific: capturing entire families of indexing strategies via representations such as fuzzy logic [372] is clearly possible. Hence, we intentionally give here a qualitative description:

1. When intensifying, the system tends to:
   - apply or synthesize rules which tend to move towards goal states;
   - compress known regions of the state space via union of more primitive regions;
   - invent synthetic types via the application of abstraction.
2. When diversifying, the system tends to:
   - invent some target type which does not overlap with any explored region;
   - place low priority on goal states (e.g. potentially bypassing a theoretically reachable goal state in favor of traversing new areas of the state space);
   - synthesize new rules and types via analogy (i.e. pick two rules with a common domain).

The heuristics above are merely intended to give a flavor of the relationship between metrics and strategies. Cross-domain heuristics for proactive synthesis are clearly of great potential value, and are rightfully the subject of an extended empirical investigation.

In order to make a choice of rules and target types in the above, the selection mechanism must ultimately be grounded in a specific state space. There are actually two choices here: (1) the 'first-order' state space described by the current ruleset and (2) the infinite 'second-order' state space described by the prospective addition of rules. By means of defunctionalization [287], second-order rules of the form $A \rightarrow (B \rightarrow C)$ can be 'uncurried' into first-order rules $(A, B) \rightarrow C$, thereby allowing all selection decisions to be grounded in a first order state space as follows:

- The state of a type is given by its predicate.
- The state of a rule is the swept-surface defined by interpolating the application of transfer function between the domain and codomain values associated with predicates.
- The measure of *static-intensification* of some newly proposed type or rule with respect to a ruleset is a function of the degree of overlap with (i.e. polytope intersection) that ruleset.
- The measure of *static-diversification* of some newly proposed type or rule with respect to a ruleset is a function of the distance from the closest point in that ruleset.
- The *dynamic* analog of the above measures is considered with respect to the recent past/near future trajectory of the state of the ruleset, optionally with some time-discounted weighting factor.

In summary, the above allows higher-order cognition to be framed in terms of reactive control, as applied to the representation of the reasoning process itself. One key issue is scalability (i.e. how to avoid 'long-tailed' fragmentation of the representation). The proposed means of addressing this is to favour the frequent application of abstraction as a means of imposing equivalence classes on the search space.

10.4 Safety

Safety is rightfully a dominant concern for autonomous systems. Indeed, taken to the extreme, 'AI safety' has spawned an academic subdiscipline devoted to the most dire of futurist predictions [30]. However, we are concerned here with the mismatch between requirements and the capabilities of contemporary approaches. We can broadly consider safety to be 'confidence that the system will not exceed its operational envelope'. There is clearly a continuum of formality with respect to the degree of confidence and bounds on/communicability of the envelope. Communicability is a bidirectional notion: how accurately has desired behaviour been specified to the system, and how interpretable is the system's representation of it?

Regarding contemporary approaches: at one extreme, we have formal methods, which permit the explicit delineation of behaviours, together with guarantees that forbidden regions of the state space will not be entered. In the manner of traditional symbolic systems and (idealized) theorem provers such as the Gödel machine [306], it is possible to enshrine 'ground truths' within the system by equipping the seed with axioms and sound rules of inference. At the other end are approaches layered on deep (reinforcement) learning, in which behaviour is indirectly mediated via reward, and obtaining confidence in the behavioural envelope is an active research area.

Considered in isolation (i.e. independent of some more autonomous invoking architecture), these approaches are anyway only applicable in 'closed world' environments, the dimensions of which must be prescribed a priori by human ingenuity and which are assumed to be unchanging thereafter.

Experiential Safety

In the open-world setting of genuinely autonomous systems, there may always be possibilities not foreseeable as a function of current inference. This could be because e.g. the sensors are not equipped to detect the associated latent interactions a priori; the butterfly effect; emergent properties of interactions, etc. As befits the scientific method, knowledge obtained via the system's own hypothesizing is therefore provisional, and 'facts' are simply hypotheses that have proved useful in the context of recent tasks. However, overriding constraints remain in place: *to the best of its knowledge* the system will never enter a forbidden region of the state space. Hence a robot tasked with remaining upright could fail to do so, given ignorance of high

winds, for example. In such an 'experiential safety' setting, safety guarantees are at both global and local levels: the local level is concerned with guarantees at the scale of single inference steps, the global scale with the longer-term behaviour of the system. Regarding local guarantees, the framework provided by F-algebras is of particular importance for general intelligence, in that it allows safety to be reconciled with open-endedness. Specifically:

- **Open-endedness**: the *algorithm template* for the F-algebra of a type can be programmatically derived [334], even if the type has been synthesized online by the system. The template orchestrates the invocation of learned rules, as described in Sect. 10.3.
- **Safety**: The interpreter defined by the algorithm template can nonetheless be constrained to well-defined behavior, i.e., mapping only between prescribed input and output types with required constraints.

Global safety properties of general interest are *reachability*, i.e. find the set of states reachable from a given initial state X_0 and *controllability*, i.e. find the set of states controllable to a given final state X_t. Modulo efficiency considerations, the bidirectional nature of rules means that reachability and control can be considered equivalent—they are anyway the same in linear systems, for example (further details of a categorical treatment in this setting can be found in Section 11.2.4). Depending on the properties of the expression language used, variants (e.g. point-to-point reachability) may be more computationally efficient. More generally, such guarantees can be divided into two categories:

Formal Reachability

In this approach, we temporarily assume the soundness of current hypotheses—based, for example, on the reliability of rules and likelihood of states. In linear continuous-time dynamical systems, reachability is decidable [131]. In this setting, reachability can be updated periodically, e.g., whenever a rule/type is added/deleted, or perhaps less frequently as an empirically-determined tradeoff between task urgency and available resources.

Time-Bounded Reachability

For expression languages in which reachability is not statically decidable, an alternative is to assume the world remains unchanged while a simulation procedure runs in the background: effectively iterating (perhaps some approximation of) the current transition relation of the system to determine which states are reached. Hence, this approach is likely to be of greatest value for determining the behavioural envelope in the near-term.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.