2 Automatic Formal Systems

Chess is chess, at any scale

Formal Games

Late in our own century it's hard to take the "paradox" of mechanical reason quite so seriously. Obviously (one insists impatiently) there can be reasoning machines—many of us carry them in our pockets. In philosophy, however, nothing is so treacherous as the "obvious." Even if it's true (as it may be) that the paradox is now resolved, we must see exactly *how* the solution works; we must understand just which new ideas and discoveries finally broke through where Hobbes, Descartes, and Hume (not to mention introspective psychologists, psychoanalysts, and behaviorists) had failed. In other words, we must understand, in fundamental terms, what a *computer* really is. The concepts involved are not particularly difficult; but they're not, in fact, all that obvious either.

A computer is an *interpreted automatic formal system*. It will take us two chapters to spell out what that definition says. More specifically, this chapter will explain what an automatic formal system is; then the next can get on to interpreting them, which is where meaning and rationality come in. In the meantime, before we can talk about automation, there's quite a bit to say about old-fashioned, "manual" formal systems.

A *formal system* is like a game in which tokens are manipulated according to rules, in order to see what configurations can be obtained. In fact, many familiar games—among them chess, checkers, Chinese checkers, go, and tic-tac-toe—simply *are* formal systems. But other games—such as marbles, tiddlywinks, billiards, and baseball—aren't formal at all (in the sense we care about).[1] What's the difference? All formal games have three essential features (not shared by other games): they are "token manipulation" games; they are "digital"; and they are "finitely playable." We must see what these mean.

The *tokens* of a formal game are just the "pieces" ("markers," "counters," or whatever) with which the game is played. For instance, chessmen, checkers, and go stones are the tokens with which chess, checkers, and go are played. These tokens happen to be handy little physical objects, which you can pick up and move with your fingers; but formal tokens don't have to be like that. Thus the tokens for tic-tac-toe are usually O's and X's, written

down with chalk or pencil; and, for many electronic games, the tokens are actually switch settings and colored light patterns.

Manipulating tokens means one or more of the following:

1. Relocating them (e.g., moving them around on some board or playing field);
2. altering them (or replacing them with different ones);
3. adding new ones to the position; and/or
4. taking some away.

In chess and checkers, for example, the pieces are mostly moved, sometimes removed (captured), and occasionally altered (promoted); but new ones are never added. In go, on the other hand, each play consists of adding a new stone to the board and occasionally also removing some; but once a stone is on the board, it is never moved around or altered. Obviously writing down O's and X's, or flipping switches to turn lights on and off, can also be token manipulations, depending on the game. (Often it is convenient to use the term *move* for any sort of token manipulation, not just relocations.)

In order to define completely any particular token manipulation "game"—any formal system—you have to specify three things:

1. what the tokens are;
2. what the starting position is (or what the alternative starting positions are); and
3. what moves (manipulations) would be allowed in any given position—that is, what the rules are.

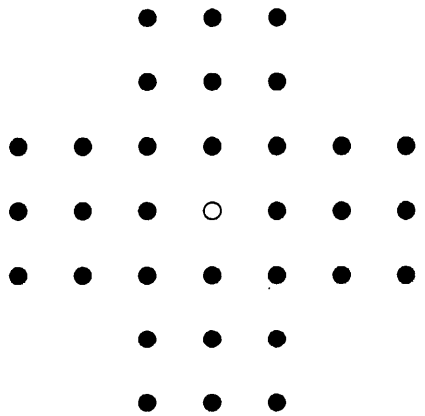
A *position*, clearly, is an arrangement or configuration of tokens (at a given moment). Play begins with some starting position and proceeds by modifying the position, one step at a time. Many formal games (chess and checkers, for instance) have a standard starting position that is always the same; but go has several starting positions, depending on the relative strengths of the players; and other systems may allow any number of possible starting positions.[2]

Positions are modified by (and only by) *legal moves*: that is, by token manipulations permitted by the rules. Just which manipulations the rules permit depends, at each point, on the current position and on nothing else. Thus a move that's legal in one position might not be legal in another; but if it is ever legal in a certain position, then it will always be legal whenever that position occurs. The latter point is emphasized by saying that formal systems are *self-contained;* the "outside world" (anything not included in the current position) is strictly irrelevant. For instance, it makes no difference to a chess game, as such, if the chess set is stolen property or if the building is on fire or if the fate of nations hangs on the outcome—the same moves are legal in the same positions, period. The players or spectators might have wider interests, but all that matters in the game itself are the current and possible positions.[3]

A crucial consequence of formal self-containedness is the irrelevance of meaning. By concentrating so far on games, we have neatly postponed any questions about formal tokens having meanings. But it doesn't take a crystal ball to see that those questions are coming (in the next chapter, to be exact). Suffice it to say here that meaning is not a *formal* property—because (roughly) meanings relate to the "outside world." To take a contrived example, imagine a game in which the following string of letters is part of the position: "The cat is on the mat." (And, for clarity, suppose also that no cats or mats are tokens in the game.) Then the point is: nothing about any cat, mat, or cat/mat relations makes any difference to what formal moves are legal in that game. Of course the string of letters might perfectly well mean something (about some cat and mat, say); but that meaning (if any) cannot be relevant to the formal system as such.

Those formal systems we think of as games typically have some special class of positions designated as the goal or "winning" positions. In such games the player or players strive to achieve one of those positions by carefully planning and selecting their moves; when there is more than one player, they are often opponents, competing to see who can reach a winning position first. But it is essential to realize that not all formal systems are competitive or even have goal positions; these characteristics are common in *games* only because they add to the excitement and fun. But many important formal systems are important for reasons quite apart from amusement (computers being a salient case in point).

A simple example of a formal system not involving competition (but still having a goal position) is the familiar solitaire game played with pegs on a board with a cross-shaped array of holes in it. The starting position appears below. The solid dots represent



holes with pegs in them, and the circle in the center represents an empty hole (the only one in the starting position). This game has only one kind of legal move: if you have two pegs next to each other in a row or column and an empty hole next to one of them in the same row or column, then you can remove the middle peg, while "jumping" the other peg over to the hole that was empty. (Thus the two holes that had pegs in them end up empty, and the one that was empty gets a peg.) The goal of the game is to remove pegs by a sequence of jumps, until only one remains, and it's in the center.

This solitaire game is a simple but genuine formal system. It is not just an analogy or suggestive comparison; it *is* a formal system. All the essential elements are here; and we will refer back to them periodically to make various points explicit. To be sure, miscellaneous subtle and important complications are lacking from such a primitive example—thereby setting the stage for illustrative contrasts as well.

The first "important complication" is really quite basic and should be introduced right away. In the solitaire game there is only one kind of token: they're all pegs, completely alike and interchangeable. But in most interesting formal systems there are numerous distinct token *types*. In chess, for instance, there are six different types of token (for each side): king, queen, rook, bishop, knight, and pawn. Formal tokens are freely interchangeable if and only if they are the same type. Thus it doesn't make any difference which white pawn goes on which white-pawn-square; but switching a pawn with a rook or a white pawn with a black one could make a lot of difference. Of course it can be a legal move to switch tokens of different types; but that results in a new and different position. Interchanging tokens of the same type isn't a move and doesn't affect the position. By the way, "altering" a token (one of the kinds of manipulation mentioned above) really means changing its type (or replacing it with one of a different type).

Ultimately, the rules are what determine the types of the tokens. The rules specify what moves would be legal in what positions. If interchanging two particular tokens wouldn't make any difference to which moves were legal in any position, then it couldn't make any difference to the game at all; and those tokens would be of the same type. To see the significance of this, consider chess again. In some fancy chess sets, *every* piece is unique; each white pawn, for instance, is a little figurine, slightly different from the others. Why then are they all the same type? Because, in any position whatsoever, if you interchanged any two of them, exactly the same moves would still be legal. That is, each of them contributes to an overall position in exactly the same way, namely, in the way that pawns do. And that's what makes them all pawns.

Formal systems, it was said at the outset, are distinctive in three essential respects: they are token-manipulation games, they are digital, and they are finitely playable. We have seen now something of what "token-manipulation" means, so we may turn to the other two characteristics, beginning with digital systems.

Digital Systems

The word 'digital' generally brings to mind electronic computers or perhaps discrete numeric "readouts" (as in digital clocks, tuners, and thermometers). But digital systems include far more than these—comprising, indeed, a fundamental metaphysical category. Every formal system is digital, but not everything digital is a formal system. The alphabet is digital, as are modern currency (money), playing cards, and the rotary function-switch on your stereo. On the other hand, photographs, gold bullion, pick-up sticks, and stereo volume controls are generally not digital. What's the basic difference?

A *digital system* is a set of positive and reliable techniques (methods, devices) for producing and reidentifying tokens, or configurations of tokens, from some prespecified set of types. Let's distill the main idea out of that: first, we can disregard configurations of tokens ("complex tokens") for the time being and also quietly take reliability and the prespecified types for granted. Second, we can substitute "writing" and "reading" for the more cumbersome "producing" and "reidentifying," but only with two warnings: (1) *writing* isn't just making pen or pencil marks but rather any kind of token manipulation that changes the formal position (thus, putting a peg in the center hole is "writing" a token of the type: peg in center hole); and (2) *reading* implies nothing about understanding (or even recognition) but only differentiation by type and position (an elevator "reads" the button pressings and shows as much by stopping at the proper floors). These simplifications render the definition quite elegant: a digital system is a set of positive write/read techniques.

Clearly everything turns on what 'positive' means. A *positive* technique is one that can succeed absolutely, totally, and without qualification; that is, "positively." The opposite would be methods that can succeed only relatively, partially, or with such qualifications as "almost exactly," "in nearly every respect," or "close enough for government work." Whether a given technique is positive depends heavily on what would count as success. Imagine, for instance, carefully measuring off six feet on a board, drawing a perpendicular line there with a square, and then sawing right along the line. Is that a positive technique? Well, it depends. If the specified goal is to get a board between five feet eleven inches and six feet one inch, then the technique can succeed totally; so it's positive. But if the goal is a board exactly six feet long, then

the method is not positive, since it could succeed at best approximately.

A positive technique isn't guaranteed to succeed; success may not even be very likely. The question isn't whether it will succeed (or how often), but rather how well it can succeed: a positive technique has the possibility of succeeding *perfectly*. When we ask instead about the likelihood of success, that's a question of *reliability*. Thus, the above method for cutting a board within an inch of six feet is not only positive but also reliable because it would be successful almost every time. If the goal were to be within a sixteenth of an inch of six feet, then it would still be positive, but it might or might not be reliable, depending on the skill of the sawyer. But if the specification were plus or minus a millionth of an inch, then the technique would be neither positive nor reliable because it is impossible to cut wood that precisely.

Many techniques are positive and reliable. Shooting a basketball at the basket is a positive method (for getting it through), since it can succeed absolutely and without qualification; moreover, for talented players, it is pretty reliable. Counting is a positive technique for determining the number of pencils in a box, and most people can do it reliably. Closing a switch is a positive method for turning on a furnace, and thermostats can do it reliably. On the other hand, there is no positive method for keeping the temperature at exactly 68 degrees: even the best thermostat could only approximate the goal. No technique can positively determine the exact weight of the pencils in a box or get a ball exactly through the center of a basket, and so on.

Digital techniques are write/read techniques. "Writing" a token means producing one of a given specified type (possibly complex); "reading" a token means determining what type it is. A "write/read cycle" is writing a token and then (at some later time) reading it; a write/read cycle is *successful* if the type determined by the read technique is the same as the type specified to the write technique. A digital system is a set of write/read techniques that are positive and reliable, relative to this standard for write/read success (and some prespecified set of relevant types).

There is considerable latitude in what one counts as producing and then identifying tokens. Thus one could say that getting a ball through a basket and not getting it through are two types of "basket-shooting tokens": shooting the ball is writing a token, and seeing whether it went through or not is reading the token. Then, given a shooter who can reliably hit or miss on command and a judge who can reliably tell hits from misses, you have a simple digital system. But it's kind of silly. Not quite so silly is a multiposition electrical switch. The types are the different possible positions; writing a token is setting the switch to some position; and reading the setting is sending electricity to the switch and having (only) the relevant circuit conduct. This is positive and reliable because the different settings are clearly separated (e.g., by "click stops") and because it is totally unambiguous and determinate which circuit conducts (for each setting).

But the real importance of digital systems emerges when we turn to more complicated cases. Consider, for a moment, the respective fates of Rembrandt's portraits and Shakespeare's sonnets. Even given the finest care, the paintings are slowly deteriorating; by no means are they the same now as when they were new. The poems, by contrast, may well have been preserved perfectly. Of course a few may have been lost and others miscopied, but we probably have most of them *exactly* the way Shakespeare wrote them—absolutely without flaw. The difference, obviously, is that the alphabet is digital (with the standard write/read cycle), whereas paint colors and textures are not.

In the real world there is always a little variation and error. A ball never goes through a basket the same way twice; the angle of a switch is microscopically different each time it's set; no two inscriptions of the letter *A* are quite alike. "Nothing is perfect," we often say. But digital systems (sometimes) achieve perfection, despite the world. How? Essentially, they allow a certain "margin for error" within which all performances are equivalent and success is total. Thus the exact token doesn't matter, as long as it stays "within tolerances."[4] In basketball, a basket is a basket is a basket; and a miss isn't. There are many slightly different ways to inscribe a character which is totally and unqualifiedly an *A*. Hence two copies of a poem can be equally perfect, even though they're in different handwriting (or fonts), as long as each character remains clearly recognizable (and correct).

One reason this is important is that it permits a kind of complexity that is otherwise difficult or impossible. To see how, consider two contrasting conventions for keeping track of money in a poker game. Each uses different colors for different denominations: blue, red, and white for a hundred, ten, and one, respectively. But in one system the unit of each denomination is a colored plastic disk (i.e., a poker chip), whereas in the other system it is a tablespoon of fine colored sand. What are the relative merits of these two arrangements? Well, the sand system allows fractional bets, by using less than a full tablespoon of white sand. You can't do that with chips. But the chip system is digital: there is a positive technique for determining the amount of any wager (namely, counting the chips of each color), and, therefore, every bet is *exact*.

The advantage of the digital arrangement becomes apparent in the case of large, precise wagers. Since a volume of sand can never be measured exactly, there would always be some small error in any sand-system wager—say ± 2 percent, for purposes of discussion. Now imagine trying to be + 914 units. With chips it's easy: you count out nine blue, one red, and four white ones, and you have it exactly. But with the sand, things aren't so nice. Two percent of 900 units is 18 units; so the expected error on the blue sand (the grains that stick to the spoon and so on) is worth more than all the red and white sand combined. In effect, the imprecision in large denominations swamps the entire significance of small denominations and renders them superfluous. But modest "imperfections" don't affect the value of blue chips. Even when they're scratched and worn, they're still worth exactly 100 white chips; therefore additional white chips are never superfluous.

Of course two percent is not a very accurate measurement, and 914 units is kind of an odd wager anyway. But the difference is a matter of principle: at some point the elaboration and versatility afforded by denominational distinctions is only feasible in a system with positive write/read techniques. The same holds for any complicated system where tiny deviations in one part can make all the difference in the world to other parts; there has to be some way to prevent the influential components from accidentally overwhelming the sensitive ones. Digital designs are one of the most

powerful means known for dealing with this problem, which is one of the main reasons they are so important.

By definition every formal system is a digital system. Formal systems can thus get extraordinarily intricate and complicated without getting out of control—a fact important to Artificial Intelligence in several different ways. But first we should consider the more straightforward cases, beginning with our old solitaire example. Why is that digital? "Writing" a token means producing a new position by making a move (that is, by jumping one peg over another and removing the latter), and "reading" that token means recognizing what the new position is (and thereby determining what moves would be legal next). This cycle is positive and reliable because one can (easily) get the pegs unambiguously and totally in and out of the various holes in the board and then tell exactly and flawlessly just which holes they're in. It doesn't matter if a peg is leaning slightly or is not quite centered in the hole: it's still absolutely and perfectly in that hole and no other, as any child can see.

The same is true for chess, though with the added complication of the different piece types. But positively discriminating the pieces (of any decent set) is as trivial as identifying what squares they're on. Again, the wide margin for error, the fact that the pieces are quite distinct and don't have to be exactly centered on the squares, is what makes the positive techniques possible. Billiards, on the other hand, is not digital: the precise positions of the balls can be quite important. Consider the difference between accidentally messing up a chess game and a billiards game. Chess players with good memories could reconstruct the position perfectly (basically because displacing the pieces by fractions of an inch wouldn't matter). A billiards position, by contrast, can never be reconstructed perfectly, even with photographic records and the finest instruments; a friendly game might be restored well enough, but jostling a tournament table could be a disaster.

The digitalness of formal systems is profoundly relevant to Artificial Intelligence. Not only does it make enormous complexity practical and reliable, but it also underlies another fundamental property of formal systems: their independence of any particular

material medium. Intelligence, too, must have this property, if there are ever to be smart robots.

Medium Independence and Formal Equivalence

Formal systems are independent of the medium in which they are "embodied." In other words, essentially the same formal system can be materialized in any number of different media, with no formally significant difference whatsoever. This is an important feature of formal systems in general; I call it *medium independence*.

To begin with a familiar example, it obviously doesn't matter to pegboard solitaire whether the pegs are made of wood or plastic, are red or green, or are a quarter or a half an inch high. You could change any or all of these and still play the very same game: the same positions, the same moves, the same strategies, the same skills—everything. And, of course, the changes could be bigger: the pegs could be eliminated altogether and replaced with marbles resting in little craters, X's chalked in squares, or even onion rings hanging on nails. On the other hand, changing the starting position or the rules for jumps would matter: that would make it a different game.

Likewise, chess pieces come in many styles and sizes—though their color usually does matter, to distinguish the opposing sides. Or rather, what really matters is that each piece be positively identifiable, both as to type and side (as well as which square it's on). Consistent with that restriction, however, the sky's the limit. Texas millionaires, for instance, could play chess from their opposing penthouses, using thirty-two radio-controlled helicopters and sixty-four local rooftops. Or, if they owned an eight-story hotel with eight rooms per floor, they might use brightly marked window shades (which the hotel staff redistributes on command).

Such outlandish examples are worth stressing just because medium independence is itself so remarkable; most nonformal games are not independent of their media at all. Doubling (or halving) the dimensions of a football, for instance, would make the game significantly different, especially in the passing and kicking strategies. If the ball were magnified by twenty (not to mention replaced by a helicopter), nothing even resembling ordinary football could be played. Billiards is similarly sensitive to the size (and weight

and shape and texture . . .) of billiard balls. Small variations might make only a minor difference; but switching, say, to little wooden pegs or onion rings would be out of the question.

Electronic toys illustrate the point in another way. Each December stores are awash with new and cleverer versions of "pocket hockey," "finger football," and the like, as well as electronic chess and checkers. But there is a dramatic difference between the cases. Pocket hockey has about as much to do with hockey as stick figures do with people: you can see a coarse analogy in a few structural respects, but beyond that they're not even similar. Electronic chess, on the other hand, *is* chess—not grandmaster quality, maybe, but the real thing. Replacing regulation balls and pucks with electronic blips is actually far worse than merely changing their size: a real, live quarterback can't even pick a blip up, let alone pass it or kick it. But luminous displays work just fine as (genuine, full-fledged) chess pieces; Russian champions can attack them (and probably capture them) as well as any other.

The difference isn't simply that some games are intellectual while others are physical. In the first place, professional sports demand a great deal of planning and strategy; and, of course, chess pieces, checkers, and go stones are just as much physical objects as any balls, bats, or cue sticks. Rather it is medium independence itself that explains why, despite these facts, chess still seems so much less "physical" than football or even billiards. The essential point is that nothing in a formal game depends on any *specific* features of its physical medium, as long as the same sequences of legal moves and positions are maintained; hence the game itself seems less directly tied to material embodiment.

Digitalness makes medium independence feasible. Return to the case of Shakespeare versus Rembrandt: since the alphabet is digital, poems are medium independent. They can be chiseled in stone, embossed in Braille, recorded in computer code, or traced romantically in the sand; and, if the order of letters and punctuation remains constant, the poem as such is unaffected. One might think that the same is true for paintings: as long as the "order" of colors, textures, reflectivities, etc. is exactly maintained, a portrait can be indifferently reproduced in oils, acrylics, watercolors, or any medium you like. The trouble is that the relevant properties

of paintings cannot, in fact, be exactly duplicated even in one medium, let alone several. It is the digitalness of alphabets that makes exact reproduction—and hence medium independence—of character sequences feasible.

Note that medium independence does not imply that suitable media are entirely arbitrary. You couldn't really play chess (or record poems) with puffs of smoke, since they can't be maneuvered very well and they dissipate too fast. At the opposite extreme, it would be just as hard to use a solid steel chess set, with all its pieces welded permanently to the board. Clearly the medium (the tokens) of a formal system must be manipulable enough that legal moves can actually be made (written) and also durable enough that positions can still be positively recognized (read) when it comes time for the next move. Though many diverse material systems and procedures can meet these requirements, many others cannot: not everything can be digital.

So far our discussion has been confined to a fairly intuitive and direct notion of "sameness": same game, same move, same poem, etc. But in fact this notion is not simple, and some interesting variations arise when it's pushed. Consider the (unfamiliar) formal game defined as follows:

1. The tokens are thirty-three plastic chips, each with two letters written on it: one from early in the alphabet (A–G), and one from late in the alphabet (T–Z).
2. In the starting position, all of the chips are together in a white basket, except for the one marked *DW*, which is all by itself in a black basket.
3. A move consists of exchanging two chips in the white basket for one in the black; but there's a restriction on which chips can be exchanged for which:
   a. *either* all three chips must have the same early letter and sequential late letters, *or else* they must all have the same late letter and sequential early letters;
   b. *and* the middle letter in the sequence can't be on the one chip going from the black basket to the white.

For example:
AX,BX <=> CX and AX,AW <=> AV are legal moves;
but ET,EV <=> EZ isn't legal (no sequential letters);
AW,BX <=> CY isn't legal (no same letter); and
AX,CX <=> BX isn't legal (middle letter in black basket).

The game is played by one person, whose goal is to finish with all the chips exchanged (i.e., all but *DW* in the black basket).

Obviously this game is digital and formal, and therefore medium independent. The chips could be wooden instead of plastic, marked with digits instead of letters, and/or exchanged between boxes instead of baskets; and, of course, it could all be done with labeled helicopters across the English channel. But could there be more dramatic differences than that? A surprising answer may emerge when we finish defining the game; in particular, we still have to specify *which* letter pairs are marked on those thirty-three chips. The specification is merely a list, but the list can be arranged so as to display the starting position in a revealing pattern.

		AV	AW	AX			
		BV	BW	BX			
CT	CU	CV	CW	CX	CY	CZ	
DT	DU	DV		DX	DY	DZ	DW
ET	EU	EV	EW	EX	EY	EZ	
		FV	FW	FX			
		GV	GW	GX			

White basket *Black basket*

Needless to say, each chip corresponds to one hole in the board for our old solitaire example. Transferring a chip from the white basket to the black is like removing a peg from the board, while transferring a chip back is like putting a peg back in an empty hole. And, given that analogy, the rather complicated rule for chip exchanges corresponds exactly to the rule for peg jumps.

So in some sense the two games are the "same." But it's not simply a case of different media, as discussed earlier. The tokens,

the pegs and chips, do *not* correspond; there are thiry-two pegs, all alike, as compared to thirty-three chips, all different. On the other hand, there are thirty-three distinct places where a peg can be, as against only two relevant chip locations. In effect, token distinctions are traded off for place distinctions, leading to the same number of possibilities overall. The correspondence between the two systems is thus complete and exact, but only at a higher or more abstract level than the direct token-for-token, square-for-square correspondence of, say, helicopter chess.

This more abstract correspondence relates complete positions instead of individual tokens and token locations. Moves, then, are conceived as changes in the overall position; so conceived, they also correspond, even though there may be no direct correspondence in the relocations (etc.) of individual tokens. We can express these new intuitions about "higher-level sameness" in an explicit definition. Two formal systems are *formally equivalent* just in case

1. for each distinct position in one system there is exactly one corresponding position in the other system;
2. whenever a move would be legal in one system, the corresponding move (i.e., from the corresponding position to the corresponding position) would be legal in the other system; and
3. (all) the starting positions correspond.

Hence there is also an exact correspondence in the positions legally accessible from the starting positions in the two systems: if you can get there from here in one, you can get there from here in the other.[5] (If the systems are understood as games with goal positions, then these ought to correspond as well.)

Our earlier illustrations of medium independence are, of course, just special cases of formal equivalence: namely, when the positions and moves correspond because the individual tokens and token arrangements correspond. Clearly helicopter chess is formally equivalent to chess played in the more ordinary way. But these are also both formally equivalent to chess played by mail, where the moves are made by writing formulae (like: P–K4) on

postcards rather than by relocating physical pieces. If you let your imagination roam a little, you can see that many otherwise quite disparate formal systems might nevertheless have the same structure of position-accessibility as chess and thus be formally equivalent to it.[6]

The notions of medium independence and formal equivalence are crucially important to Artificial Intelligence and to computational psychology in general. Though some of the significance can only emerge as our discussion proceeds, the basic point can be stated crudely: brain cells and electronic circuits are manifestly different "media"; but, maybe, at some appropriate level of abstraction, they can be media for equivalent formal systems. In that case, a computer mind might be as much a real (genuine) mind as helicopter (or computer) chess is genuine chess, only in a different medium. Of course, we have some work to do before we can turn that into anything more than a tantalizing hint.

Finite Playability and Algorithms

So far we have been fairly casual (not to say "informal") about the definitions of formal systems, avoiding mathematical technicalities wherever possible. One technical matter, however, is central to the very idea of formality: What limits are there (if any) on the size and complexity of formal systems? The crucial issue concerns what it takes "to play the game"—that is, to make and recognize moves allowed by the rules. Roughly, we want to insist that formal games be playable by finite players: no infinite or magical powers are required. On the other hand, our theoretical definition shouldn't be gratuitously parochial: players mustn't be limited to any particular capabilities or resources (such as human or present-day capabilities) as long as they are unquestionably finite. The challenge is to spell this out.

To be able to play a formal game is to be able to follow the rules. But what, exactly, does that involve? In principle, competent players must always (i.e., for any given position) be able to

1. tell, for any proposed move, whether that move would be legal (in that position); and

2. produce at least one legal move (or show that there are none).

Thus, you wouldn't be much of a chess player if you couldn't reliably tell whether your opponent's moves were legal or come up with legal moves of your own. To insist, then, that formal systems be *finitely playable* is to insist that these two necessary abilities be within the compass of finite players. So the question becomes: What can be expected of finite players?

Some operations are manifestly within finite means, essentially because they're trivial. For instance, it doesn't require anything magical or infinite to type a token of the letter *A* on a piece of paper whenever a certain switch is closed. Not only is this clear on the face of it, but we also have an independent demonstration in that (finite, nonmagical) electric typewriters can do it. An ability to identify simple tokens from a standard set is equally mundane in principle, as is shown, for example, by the coin-sorting prowess of common vending machines. Again, it's easy to move, say, a chess piece from one square, indicated by pointing at it, to another; nor is much required to move a pointer to an adjacent square, in a given direction; and so on. It is not important, however, to catalog or even fully delimit these manifestly finite capabilities, as long as it is established that there are some. For they give us a place to start: a finite player is assumed to have some finite repertoire of specific *primitive operations* that it can perform, positively and reliably, any finite number of times.

Primitive operations, of course, are only the beginning. Chess playing requires much more than merely identifying and moving indicated pieces among indicated squares; a player must also determine, for each proposed move, whether it is legal for that type of piece, whether it exposes a king threat, whether it is blocked by intervening pieces, etc. Since such tests typically involve a variety of interrelated factors, they are not altogether simple and trivial; hence, they shouldn't count as primitive. Consequently, finite players must also be capable of operations that are *not* primitive. This is hardly surprising; the task is to define these new, nonprimitive abilities in a way that still keeps them uncontroversially finite.

Players can perform arbitrarily complicated operations by performing ordered combinations of primitive operations. For example, to see whether a king is in check, one might look at each opposing piece in turn and see whether that piece is attacking the king (which might itself involve a combination of still smaller steps). Obviously this sequence of steps is not carelessly or randomly chosen, but explicitly designed for the purpose at hand; moreover, it must be carried out correctly or it won't work. In effect, the particular combination of steps required for a complex operation must be specified by some sort of rule or "recipe," which the player then has to follow to carry it out.

That, unfortunately, is right where we came in: four paragraphs ago we asked what's involved in following rules, and now we find out that it involves following rules. Wonderful! Not all is lost, however, for some rules are easier to follow than others. We need to define a special category of rules that require only primitive rule-following abilities—that is, abilities that are, like the primitive operations themselves, manifestly within finite competence. These special rules can then be used in "building up" both complex operations and complex rule following.

An *algorithm* is an infallible, step-by-step recipe for obtaining a prespecified result. "Infallible" means the procedure is guaranteed to succeed positively in a finite number of steps (assuming each step is carried out correctly). Note that, although the total number of steps must always be finite, there need be no prior limit on how many steps that might be. "Step-by-step" means three things: (1) the recipe prescribes one step at a time, one after another; (2) after each step, the next step is fully determined (there are no options or uncertainties); and (3) after each step, the next step is obvious (no insight or ingenuity is required to figure it out).

Intuitively, algorithms are just mindless "turn the crank" routines that always work (sooner or later). For instance, if you have a ring of keys, one of which fits a certain lock, then trying them in sequence is an algorithm for opening the lock: it's sure to work eventually (no matter how many keys), and "any idiot can do it." Likewise, there are familiar algorithms for sorting arbitrary lists into alphabetical order, for checkmating a lone king with a

king and rook, for transposing songs into different musical keys, for multiplying integers (using Arabic numerals), and so on. These algorithms, notice, are specified in terms of an input/output relation: you start with an input of a certain sort (a lock and keyring, a list of names, a songsheet), and the goal is an output related to that input in some determinate way (that lock opened, that list alphabetized, that song transposed).[7]

The simplest kind of recipe is a *straight schedule* of instructions: do *A* first, then do *B* . . . and, finally, do *Z*. It doesn't take much to "follow" such a recipe: basically, the player has to read the current instruction, obey it, and then move to the next one down the list. But, modest as they are, these schedule-following abilities are importantly distinct from the abilities to perform the listed operations themselves (telling what to do is not the same as doing it). A player needs abilities of both kinds to carry out the recipe.

Since there are only finitely many primitive operations in a finite player's repertoire, we may assume that there is for each a unique primitive *instruction*, which the player can reliably obey. Think, for instance, of the separate keyswitches for each of the typewriter's primitive typing operations: closing such a switch is a primitive instruction, which the typewriter can obey. It is equally clear that a finite player can go down a finite list in order; each successive step is fully and obviously determinate. (Consider a typewriter that can be fed a sequence of primitive typing instructions encoded on a paper tape.) Thus we have an initial repertoire of primitive rule-following abilities: (1) obeying primitive instructions and (2) moving to the next instruction in a finite list. Moreover, these abilities are sufficient in principle for following any straight schedule; hence, whenever such a schedule is infallible, it's an algorithm.

The trouble with straight schedules, however, is that they're totally inflexible: exactly the same sequence of steps is prescribed, regardless of the input or any intermediate results. Consequently no recipe with this structure is adequate even for the boring algorithm: try each key in turn, *until* you find the right one. In the first place, the algorithm ought to work for any size keyring (= different inputs); but since a straight schedule has only so many entries, it can go only so far. Second, the algorithm ought to stop

when the right key is found (= intermediate result); but a straight schedule just blindly continues, no matter what.

Both deficiencies are remedied by a single, seemingly obvious device, which turns out to be astonishingly powerful: in general, you let the next step *depend* or be *conditional* on the results of preceding steps. There are various ways to achieve this (all quite interesting to mathematicians); but one approach will suffice for illustration. Consider the recipe:

1. Start with any key, and tie a red ribbon on it.
2. Try that key in the lock.
   DID THE LOCK OPEN?
   IF SO, GO TO LINE 4; IF NOT, GO TO LINE 3.
3. Move to the next key on the ring.
   DOES IT HAVE A RED RIBBON ON IT?
   IF SO, GO TO LINE 5; IF NOT, GO BACK TO LINE 2.
4. Quit, triumphant and happy.
5. Quit, frustrated and disgruntled.

A player following this recipe will repeat steps 2 and 3, trying one key after another (no matter how many), until either the lock opens or every key has been tried; thus the behavior varies appropriately, in accord with the input and intermediate results, which is exactly what we wanted.

Clearly the improvement lies in the special directives, written in capital letters. These don't prescribe particular primitive operations (as do the numbered instructions); rather, they serve as explicit signposts, guiding the player around the recipe by specifying which instruction to obey next, depending on what just happened. To carry out the algorithm, two new recipe-following abilities are needed: answering yes/no questions about the previous step and "branching" to one of two specified instructions, depending on the answer. Such conditional directives (this kind is called a *conditional branch*) provide the flexibility lacking from straight schedules of instructions. Thus the first directive above tests the intermediate result and keeps the search from continuing stupidly after the right key is found. And the "loop back" clause in the second directive (GO BACK TO LINE 2) enables the recipe

to handle an arbitrary number of keys by repeating two steps an arbitrary number of times.

But what is the point of the red ribbon and of the other clause in the second test? Why not just go around the ring, key by key, until one opens the lock? The problem, plainly, is the unlucky case where the lock never opens; but it's more serious than it seems. Since the input keyring can be arbitrarily large, the algorithm must be able to keep looking (repeating its two-step "loop") *without any limit*; this is precisely what gives it its general power. But to be an algorithm, it must also be guaranteed to succeed, eventually; it cannot be allowed to go on forever, looking for a key that isn't there. Normally the routine quits only when it finds a key that works. But if there may be no such key, then there must be some other successful stopping condition, such as proving that none of these keys works this lock. That's what the red ribbon and the "branch to line 5" clause provide.

Obviously it is possible for a finite player to answer some yes/no questions positively and reliably: the vending machine will make change *if* you insert coins larger than needed. It can also easily "go to" (follow) alternative segments of a finite schedule, depending on the answer: the machine will (follow instructions to) give the correct change, once it has identified your coins. So finite players can, in principle, follow any finite *branched schedule* (i.e., schedule with conditional branches), assuming the branching conditions (the yes/no questions) are suitably primitive. Since such a schedule is also a step-by-step recipe, it will be an algorithm if it's infallible—that is, if it's guaranteed to produce its specified result (and stop) in a finite number of steps. Let's call a branched-schedule algorithm with primitive branching conditions and primitive instructions a *primitive algorithm*. Then we have just established the following conclusion: executing primitive algorithms (positively and reliably) is within the compass of finite players.

This conclusion is remarkably important. In the first place, it gets us off the ground with regard to rule following, since no nonprimitive abilities are presupposed. But second, and more spectacular, it effectively extends the range of what can count as "primitive." Nothing concrete has been said about what's primitive

Box 1
Finitude, Magic, and Mechanism

Suppose someone gives you a simple numeric test and asks: What's the smallest positive integer that satisfies it, if any? For some tests, there might be an analytic (e.g., algebraic) solution to the problem. But let's consider a case with no such solution, where the only approach is brute force: try the test on the number 1, if that fails, try 2, and so on.

If there were a guarantee that at least one number would satisfy the test, then brute force would be an algorithm for finding the smallest one; for, no matter how big that number is, counting will get to it eventually. If, on the other hand, no number will satisfy the test, brute force will *never* establish this fact; for, no matter how high you go, there are still higher numbers you haven't tried yet. When a procedure goes on forever like that, it is called *nonterminating*. In a formal game, the procedure(s) for determining whether a proposed move would be legal must always terminate (yes or no); that's the point of finite playability.

But what if we had an oracle: a crystal ball giving (correct) answers, even to cases that are otherwise undecidable? Would that make the game finitely playable after all? No; and this is the point of adding that no magic is needed to play the game. If magic were allowed, then the restriction to finitude (or anything else) would be vacuous.[8]

Magic, by its essence, is unintelligible. It doesn't follow, however, that everything (presently) unintelligible is magic. For instance, the pioneers of mathematical formal systems were uncertain whether mathematical "insight" and "intuition" might be magical. It's not that they thought these were magic; it's just that no one could be sure because nobody really understood how they work. Indeed, the only certain way to abjure magic is to insist on procedures that are transparently intelligible, such as simple physical mechanisms. That's why typewriters and vending machines are so valuable as examples and also why algorithmic procedures are always "mindless and mechanical."

and what's not, as long as positive and reliable performance is clearly within finite competence. But we just saw that executing primitive algorithms is within finite competence; hence an entire primitive algorithm could legitimately serve as a single "primitive" instruction in a more sophisticated algorithm. In other words, if we think of primitive algorithms as ground level, then second-level algorithms can use simple operations and directives that are actually defined by whole ground-level algorithms; and third-level algorithms could use second-level algorithms as their primitives, and so forth.

Our keyring algorithm, for example, could be a single step in a "higher" recipe for trying all the locks around the perimeter of a fortress:

1. Start with any lock, and tie a blue flag on it.
2. Try all the keys in that lock.
   DID THE LOCK OPEN?
   IF SO, GO TO LINE 4; IF NOT, GO TO LINE 3.
3. Move to the next lock around the fort.
   DOES IT HAVE A BLUE FLAG ON IT?
   IF SO, GO TO LINE 5; IF NOT, GO BACK TO LINE 2.
4. Quit, triumphant and happy.
5. Quit, frustrated and disgruntled.

Step 2 involves an arbitrary number of "substeps" (one for each key, no matter how many). But since we have an algorithm for that, we can call it one step and take it for granted. Incidentally, we also see here how important that red ribbon was in the first algorithm: had it been omitted, then this new recipe could never get past the first lock (either that lock would open, or step 2 would continue unsuccessfully forever).

In a similar way, a great algorithmic edifice can be erected on the foundation of chess primitives (identifying pieces and squares, and the like). For out of these we can build algorithms to determine the potential range of any piece, how much of that range is blocked by other pieces, and then which moves would expose the king. Out of these, in turn, we can construct an algorithm to test whether any proposed move would be legal; and then, using that, a

straightforward algorithm can generate (list) *all* the legal moves, for any given position. Needless to say, many other fancy and sophisticated algorithms can be constructed from all sorts of simple primitives. It makes no difference how many levels of complication get piled up, as long as it's algorithms all the way down. Only the ground-level abilities need be genuinely primitive—using these bricks, and algorithmic glue, there's no limit to how high we can build.[9]

Complex Tokens

Before turning to automatic systems, we must consider one last "complication": namely, systems in which individual tokens are composed or built up out of simpler tokens. This is a familiar idea: sentence tokens are composed of word tokens, and (in writing) word tokens are composed of letter tokens; likewise, Arabic numeral tokens are strings of digit tokens (maybe with a decimal point). The idea is so familiar, in fact, that it seems inconsequential; but nothing could be further from the truth.

Actually, there are different cases, some of which are more interesting than others. The difference lies in whether the internal composition of the tokens systematically determines how they can be used. For instance, Morse code is totally boring in this regard: though individual letters are composed of "dots" and "dashes," their specific structure has no significance (it matters only that each character type be unique). The alphabetic compositional system, spelling, is more complicated, since it does provide a rough guide to pronunciation. What's important about words, however, is not how they sound but how they are used to say things; and here their spelling is no help at all. So, aside from phonetics, alphabetic composition is basically just as trivial as Morse code.

The point can be illustrated by contrasting spelling with a system that is emphatically *not* trivial: Arabic numerals. When you know the composition of a complex (i.e., multidigit) Arabic numeral, you know everything you need to know about it. Compare, for example, the following lists of compound tokens, composed respectively in the Arabic numeral and English spelling systems:

783	dol
374	edh
662	mho
519	ret
54,912	kylix
99.44	phiz
2,000.2	ootid
0.0043	yagi

Imagine that you happen never to have seen any of these particular numerals or words before, and suppose you were asked to use each correctly in a typical calculation or sentence. What a difference! It's easy to use complex numerals correctly, even the first time you see them, because you can tell what to do from the way they're built up out of digits. But if those words were new to you, you would have no idea how to use them properly—the spelling wouldn't help a bit.

Box 2
Simple and Complex

Here we use 'simple' and 'complex' in their strictly correct senses. 'Simple' means "having but one part; not put together out of several components." 'Complex' means the opposite: "composed or put together out of several components; having more than one part." The classic example is chemical theory, according to which atoms are simple and molecules are complex (put together out of atoms). The example also brings out an important qualification: whether a thing counts as simple or complex depends on how you look at it. From the point of view of chemical compounds, atoms are simple; but from the point of view of atomic physics, they are complexes, made up of simple nuclei and electrons. (Nuclei themselves are composite from the point of view of nuclear physics; but so far electrons have no known structure.)

Awkwardly, however, both these examples cheat: they trade on the *meanings* of the respective tokens. Thus the composition of a numeral tells you what number it stands for, whereas the spelling of a word doesn't tell you what it means. But since formal systems as such are self-contained, nothing about them can depend on meanings. Hence the examples as presented don't really show anything about formal systems. The way out of the trap is to let the *use* of a token be the contribution it makes to the formal position—that is, the difference it makes to what moves would be legal. If the composition of the tokens determines their "use" in this sense, then that composition is formally significant.

With that strategy in mind, we can even fix up the numeral example, by a modest sleight of hand: the trick is to think of arithmetic itself as *purely a game*. To be more concrete, imagine that you learned the rules and moves of multiplication (in Arabic numerals) before you had any idea that these tokens had anything to do with numbers. (Outrageously, I fear, some children do learn arithmetic this way.) Anyhow, the starting position is a pair of tokens written one above the other, aligned at the right, with a line beneath and a sign to the left:

Starting position:	54912	(first token)
	× 783	(second token)

The player then writes one or more new tokens below these, stairstepping all but the last to the left, and draws another line above the last:

54912	
× 783	
164736	(first move)
439296	(second move)
384384	(third move)
42996096	(last move)

I trust this has a familiar look and that there's no need to spell out the rules in greater detail.

But one point about the rules is crucial: I *could* spell them out, without ever mentioning numbers (or anything else "outside" the system): I just tell you which *token* manipulations to make in which positions, all quite formally. Since the moves are clearly also digital, it follows that Arabic numeral "multiplication" can be defined as a completely formal system, which we might call the *formal multiplication game*.

Except, that is, for one little problem: there are infinitely many distinct types of Arabic numeral tokens. Hence formal multiplication has infinitely many distinct, possible positions and moves. But, by definition, a finite player has only a fixed, finite repertoire of primitive abilities. So how can formal multiplication be finitely playable?

The answer is obvious (once you appreciate the question), but it's quite fundamental. Not only are the individual tokens (the numerals, written one per line) complex (put together out of simple digits); but also, and just as important, the rules for manipulating these compounds are expressed as *recipes*. They are specified sequences of simple steps, going through the component digits one by one. This is how the particular composition of the numerals determines what's legal, and it is also how a small repertoire of primitive operations can handle arbitrarily large numerals (by the method of finite but unlimited repetition). Indeed, there is a complete algorithm for Arabic multiplication (obtaining the partial products, writing them a certain way, adding them up, etc.), including various subalgorithms for performing the required steps (digit by digit), and so on. Therefore multiplication is within finite competence.

This is the example we needed: the tokens are complex; their "use" depends on their composition (so the complexity is not trivial); and yet no meanings are involved—the system is purely formal. It also illustrates the incredible power of combining complex tokens with complex recipes. For here we have a system with literally infinitely many different legal positions and legal moves, and yet there is no question that "playing" it remains fully within finite competence. Formalizing multiplication, of course, is not such a big deal; but the same general point holds for the much

Box 3
Unbounded versus Infinite

There are infinitely many distinct finite numbers; hence there are also infinitely many distinct Arabic numeral types, each with only finitely many digits. How can this be? The answer is that, though each of the numbers (or numerals) in question is finite, there is no upper bound or limit on how big they can be; they are *unbounded*. More precisely, for any positive integer, no matter how large, there is a "next" one that is still larger and therefore distinct from every preceding one; you never come to the end of them.

Why does this mean that there are infinitely many? We can construct a proof by trying to suppose the contrary and then showing that that's untenable. So imagine that there were only finitely many finite integers; in that case there would be some particular number of them, say *N*, and no more. For any positive integer, however, we know that there are exactly that many distinct positive integers up to and including it; thus, there are exactly sixteen distinct integers from 1 through 16, inclusive. So if there were only *N* distinct finite integers, they would be precisely the integers from 1 through *N*; in particular, none of them could be larger than *N*. But for any finite integer, there is always another that is larger (this is where the unboundedness comes in). Hence no finite number can be the number of finite integers—there are always "more." Therefore the number of finite numbers must be infinite.

The conclusion applies as well to Arabic numeral types because every distinct integer has a distinct type of Arabic numeral, with finitely many digits, to stand for it. The notion of "finite but unbounded" is important in other contexts as well. For instance, in algebraic and formal logical systems, the formulae ("sentences") and proofs must all be finite; but there is no upper bound on how long they can be. Similarly, an algorithm must always terminate after finitely many steps; but there's no limit on how many that can be. The same goes in principle for the size of computer memories (such as Turing machine "tapes"; see "Turing Machines and Universality" in chapter 4).

more impressive formalizations of higher mathematics, logic, computer science, linguistics, and so on.

Above all, however, it holds for Artificial Intelligence. For if Hobbes is right that thinking is "computation," it is certainly highly complex computation—far more so, indeed, than Hobbes could ever have imagined. As we shall see (for instance in "Commonsense Stereotypes" in chapter 5), the formal structures that an AI system would have to manipulate, even to have an ordinary conversation in natural English, are enormous and very elaborate. Their detailed internal composition, moreover, is absolutely critical to how they may be used.

Automatic Systems

An *automatic* formal system is a formal system that "works" (or "plays") by itself. More precisely, it is a physical device (such as a machine), with the following characteristics:

1. some of its parts or states are identified as the tokens (in position) of some formal system; and
2. in its normal operation, it automatically manipulates these tokens according to the rules of that system.

So an automatic formal system is like a set of chess pieces that hop around the board, abiding by the rules, all by themselves, or like a magical pencil that writes out formally correct mathematical derivations without the guidance of any mathematician. These bizarre, fanciful images are worth a moment's reflection, lest we forget the marvel that such systems (or equivalent ones) can now be constructed.

The actual playing of a game involves more than just the positions and moves: there must also be one or more players and a referee. The players make the moves, whenever it's their turn. The referee doesn't make any moves but rather determines whose turn it is—that is, which player should move next, and, perhaps, which tokens it should work on. The referee also provides the starting position, decides when the game is over, announces the official results, and so on. For friendly games, we tend to overlook the referee function, since it's often so straightforward that the

players perform it for themselves. But strictly speaking, refereeing is always separate from playing (making the legal moves), even when the difference is inconspicuous; and when the cases get complicated, it is essential to keep the distinction clear, on pain of hopeless confusion.

Since an automatic game actually plays by itself, it must include all these elements; in fact, we can think of it as essentially a combination of the player(s), the referee, and the ordinary "manual" game (the tokens). Sometimes, of course, a formal game is only partially automated. For instance, electronic chess sets typically automate only one of the players (plus the referee); the other player is external to the machine—such as the person who bought it. But this makes no important difference to the theory of automatic systems, and we might as well consider all the players as included.

So far we have said what an automatic formal system is: a manual game, combined with automated player(s) and referee. We have not said how they work, what they're made of, or what principles they're based on—because none of that is relevant to explaining what they are. To borrow a wonderful term from engineering, the player(s) and referee are treated as *black boxes*: things you don't look inside of. Thus it is often useful to consider only the "opaque surface" of a component (*what* it does), while taking its inner workings (*how* it does it) for granted. For example, an audio engineer assembling a concert sound system with dozens of microphones, mixers, amplifiers, speakers, etc., can't be worried about how each part does its individual job, as long as it does. The separate components must be treated as black boxes, and taken for granted, in the design of the overall system.

What counts as a black box depends on your point of view; one engineer's component is another engineer's whole system. For instance, an amplifier is itself a complicated system, which some electronics engineer had to design out of still smaller black boxes: transistors, resistors, integrated circuits, etc. The amplifier designer takes these for granted; but, of course, somebody else had to design them too. If you need to know only what something does, then you can look just at its surface (or consult the manufacturer's specification sheet); but if you want to understand how

it does whatever it does, then you must look inside (or consult the manufacturer's design staff).

Coming to understand how something does what it does, by considering what it's made of and how it's put together, is called *analysis*. Explaining how a complex system works, in terms of the interactions of its functional components, is one kind of analysis.[10] Suppose, for instance, you are perplexed and amazed by your automatic chess system. You know that it comprises (besides the tokens) two chess-player components and a referee component; and you know that they play legal chess games. Now you would like to understand *how* they do that. In other words, you don't want to treat the components as black boxes anymore; you want to analyze.

One possibility is analysis into smaller automatic formal systems; that is, a single component in the overall system might itself be an entire automatic formal system, with its own private little tokens, manipulated by a group of inner players, guided by an inner referee—all quite distinct from the outer tokens, players, and referee. Take, for instance, one of the "outer" players. It makes a legal chess move whenever the referee indicates a position and says, "It's your move now." What might the "innards" of such a player look like, supposing it were itself a complete automatic formal system?

The answer, of course, depends on the specific design; but here's one design that would work (see figure 1). The tokens of the inner game are a description (or copy) of the current outer position (provided by the outer referee), plus some standard notation for specifying chess moves. There are two inner players: a "move lister" and a "move chooser." The inner referee first indicates the position (inner copy) and signals the move lister, who proceeds to list all the legal moves in that position. Then the referee, indicating the position and list together, signals the move chooser, who proceeds to choose some move from the list. Finally, the referee announces the chosen move as the subsystem's output (that is, the outer player makes this move in the chess game).

That's *one* level of analysis. But there can be many levels: boxes, within boxes, within . . . until the components are so basic that they can't be analyzed further (or perhaps can be analyzed only

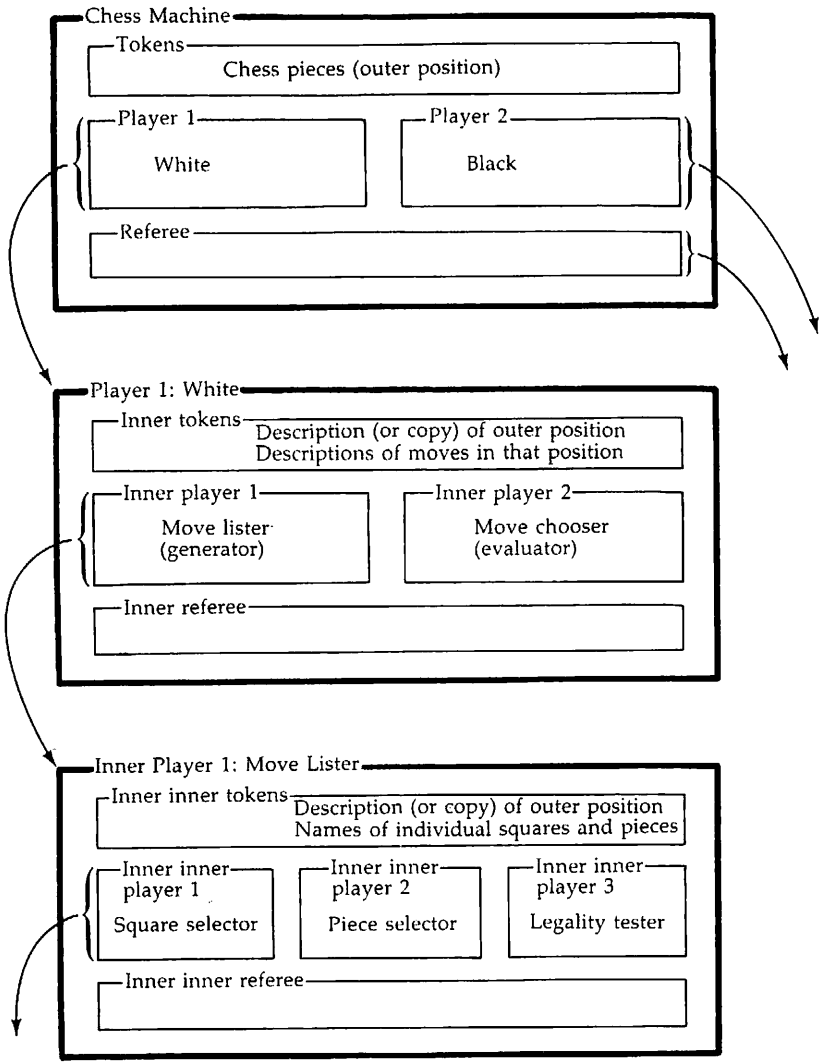


Figure 1 Partial Analysis of a Chess Machine

in some other way). For instance, we might be able to analyze the move-lister and/or the move-chooser components (from the above system) into still smaller systems; the two cases would be quite different, with each illustrating an important moral. (The inner referee might be analyzable too; but this particular one happens to be fairly boring.)

In a deep sense, the move lister is trivial because there is an algorithm for listing all the legal moves in any chess position. What do algorithms have to do with automatic formal systems? In a way, everything. Recall that executing an algorithm requires a fixed, finite repertoire of primitive abilities—both primitive operations and primitive recipe following. So suppose each primitive operation gets assigned a single player, whose sole competence is that operation; and suppose the referee has all the primitive recipe-following abilities. Then, roughly, the referee can follow the recipe, signaling the players in the right order (telling them whose turn it is and what tokens to work on), and the players can then carry out the specified operations. The algorithm can be executed automatically by a system composed of these players and this referee.

But what about those primitive abilities? Well, whenever an ability can itself be specified by an algorithm (in terms of simpler abilities), then that whole ability can count as a primitive in a higher-level algorithm. The only restriction is that all the required abilities eventually get analyzed into ground-level abilities that are *really* primitive—abilities so "mindless and mechanical" that even a machine could have them. This analysis of higher-level primitives into lower-level algorithms precisely parallels the analysis of higher-level components (black boxes) into lower-level systems. Moreover, the ultimate, ground-level primitives are exactly the same—except that, in an automatic system, they not only could be but *are* carried out by primitive machines.

These ultimate primitives (both player operations and referee recipe followings) are the fundamental point at which formal systems meet up with automation. Given this foundation, sophisticated automatic systems can be built back up, black box by black box, in just the way that sophisticated algorithms can be

Box 4
Analysis and Reduction

Many philosophers and scientists believe that nature is unified not only in the fundamental stuff of which it's all made (matter), but also in the fundamental laws in terms of which it can all be explained (physics). Advocacy of this further thesis, sometimes summarized as "the unity of science," is called *physicalism* or *reductionism*—the latter because physicalism (in contrast to mere materialism) amounts to the claim that all other sciences can (in principle) be "reduced" to physics.

The standard example is the reduction of thermodynamics, according to which heat is just an alternative form of energy stored in the chaotic bouncing around of individual molecules. Given this, plus miscellaneous technical breakthroughs, it can be shown that thermodynamic laws are derivable from physical laws—implying that thermodynamics is basically a special case of physics. Such reductions are called *nomological* (from the Greek *nomos* = law).

Many explanations, however, don't appeal to laws; so they can't be reduced nomologically. For instance, when Mr. Goodwrench explains how a car engine works, he doesn't use physics; he appeals instead to the capabilities and functions of the various parts, showing how they fit together and interact as an integrated, organized system. In other words, he analyzes the engine and explains it functionally or systematically.

When an explanation appeals to laws, those laws are not themselves explained but merely used: they serve as unexplained explainers. Reduction is explaining the unexplained explainers of one theory in terms of those of another; hence physicalism is really the view that there are no unexplained explainers save those of physics. Accordingly, functional/systematic explanations can be reduced too, if their unexplained explainers (essentially the interactive capabilities of the functional components) can themselves be explained in more basic terms.

Analyzing automatic formal systems into ever smaller sys-

tems of interacting black boxes is therefore reductionistic—not nomologically, but rather in the functional/systematic sense. Connection to physics is made at (or via) the level of operations so primitive that they can be carried out by mindless mechanisms.

built up out of ever cruder algorithms, resting eventually on the ground level. So the essential point may be summed up this way:

AUTOMATION PRINCIPLE: whenever the legal moves of a formal system are fully determined by algorithms, then that system can be automated.

This is why automating the move-lister component of the chess-player component of the chess system is (theoretically) trivial.

The move-chooser component, however, is another story. If the move lister listed only one legal move, then no choice would have to be made, and there would be no problem. But chess, like most interesting formal systems, is *nondeterministic*; that is, in most positions, any of several different moves would be legal. Sometimes a particular move is forced because no other move is legal; but if the whole game were like that, it would just be oppressive and unbearable—no fun. Algorithms, by contrast, are always deterministic: by definition they are step-by-step procedures that never permit any options or uncertainties about the next step. How then can a system allowing choices be automated?

There are a couple of cheap and dirty answers that we might as well dispose of right way. First, the move chooser might just choose whatever move appeared at the top of the lister's list (equally, it could use any other unmotivated principle, such as selecting the move whose description comes first in alphabetical order). Second, the chooser might pick from the list randomly; that is, it could consult some random input (a roll of the electronic dice, say) and let that determine the choice. These designs would yield a system that played *legal* chess; thus, in a superficial sense, they seem to answer the question. But such systems would also

play atrocious chess—so hopelessly crazy and chaotic, in fact, that we should hesitate to call it chess, even though the moves are all strictly legal.

The real problem, then, is to design a chooser that makes good (reasonable, intelligent, wise. . . ?) choices. Unfortunately, there are no known feasible algorithms for telling in general which is the better of two chess moves (let alone for wisdom and the like). Not surprisingly, approaches to this issue begin to involve techniques characteristic of Artificial Intelligence research which we will consider in more detail later. In the meantime, however, a sketch of a chess-move chooser can give a glimpse of what is coming.

Obviously the chooser needn't always find the best move; even the greatest champions don't play perfect chess. The goal, rather, is a system that chooses relatively well most of the time. In other words, an infallible test for the better move (i.e., an algorithm) is not really required; it would be enough to have a fairly reliable test, by which the machine could usually eliminate the worst choices and settle on a pretty good one. Such fallible but "fairly reliable" procedures are called *heuristics* (in the AI literature). Thus heuristics (so defined) are not algorithms because algorithms must be infallible. Intuitively, heuristics are like rules of thumb: rough, generic principles that are not perfectly accurate, but are still accurate enough to be convenient and useful.

There are many rules of thumb for better chess. For instance, other things being equal, it is better to capture opposing pieces while losing none (or only weaker ones); likewise, it is generally preferable to control the center, activate pieces early, keep your king out of the way, and so on. But everybody knows that other things are not always equal: even the best heuristics sometimes fail—as in the apparent blunder ("free" capture) that turns out to be the bait in a brilliant trap. There are various ways to improve the odds, like using lots of heuristics and balancing them off or considering sequences of moves together (looking ahead); and the net results can get pretty good, though still not perfect.

How do heuristics aid automation? Suppose we devise a strict, precise formula for applying and combining various well-defined rules of thumb; the output of this formula is a fallible but relatively

reliable estimate of the best move in a given situation. The formula itself is perfectly explicit and unambiguous, so we can well imagine a routine that infallibly calculates its value (i.e., comes up with the exact estimate) for any given position. Is that routine an algorithm? It depends on how you look at it. If the specified goal is to crank through the formula (calculate the estimate), then it *is* an (infallible) algorithm. But if the goal is specified as finding an optimal chess move, then that very same routine is only a fallible estimator, and hence only a heuristic. The punch line is easy: seen as an algorithm, the routine can be automated like any other; but seen as an estimator, the same routine (now automated) can function as a decent move-chooser. In other words, a good but fallible move-chooser can be automated by an algorithm after all.

This strategy of regarding or *redescribing* the very same thing from more than one point of view is essential throughout science. Physicians must learn to see suffering patients as conglomerations of living tissues and living tissues as seething chemical stews. To understand why sonnets are so easily preserved, one must reconceive them as mere strings of digital characters. We shall find that the possibility of adopting multiple viewpoints is particularly important in Artificial Intelligence—analyzing black boxes and automating rules of thumb being but two examples. The most important redescription by far, however, arises in connection with *meaning*—an issue we can postpone no longer.