PATRICK GRIM

THE UNDECIDABILITY OF THE SPATIALIZED PRISONER'S DILEMMA

ABSTRACT. In the spatialized Prisoner's Dilemma, players compete against their immediate neighbors and adopt a neighbor's strategy should it prove locally superior. Fields of strategies evolve in the manner of cellular automata (Nowak and May, 1993; Mar and St. Denis, 1993a,b; Grim 1995, 1996). Often a question arises as to what the eventual outcome of an initial spatial configuration of strategies will be: Will a single strategy prove triumphant in the sense of progressively conquering more and more territory without opposition, or will an equilibrium of some small number of strategies emerge? Here it is shown, for finite configurations of Prisoner's Dilemma strategies embedded in a given infinite background, that such questions are formally undecidable: there is no algorithm or effective procedure which, given a specification of a finite configuration, will in all cases tell us whether that configuration will or will not result in progressive conquest by a single strategy when embedded in the given field. The proof introduces undecidability into decision theory in three steps: by (1) outlining a class of abstract machines with familiar undecidability results, by (2) modelling these machines within a particular family of cellular automata, carrying over undecidability results for these, and finally by (3) showing that spatial configurations of Prisoner's Dilemma strategies will take the form of such cellular automata.

KEY WORDS: Undecidability, Prisoner's Dilemma, cellular automata, game theory, decision theory, computability.

1. INTRODUCTION

Each of the two players in the Prisoner's Dilemma has an option to cooperate or defect, with a standard payoff matrix awarding a reward R of 3 points to each player for mutual cooperation, a penalty payoff P of 1 point for mutual defection, a temptation T of 5 points for a player who defects while his opponent cooperates, and the sucker's payoff S of 0 points to a player who cooperates while his opponent defects. Technically any P, R, T and S satisfying $T > R > P > S$ and $R > \frac{1}{2}(S + T)$ characterize a Prisoner's Dilemma, but the particular values mentioned have become traditional and will be used throughout.

Theory and Decision **42:** 53–80, 1997.

In the iterated Prisoner's Dilemma choices of cooperation and defection are repeated against the same opponent. Here general strategies emerge: AllD, for example, which defects regardless of the other's play, AllC, which cooperates regardless, 'grim' strategies which revert to constant defection once crossed, and tit for tat (TFT), which begins by cooperating, cooperates in answer to a cooperation on the other side, and defects following a defection (Axelrod, 1984). In what follows, for the sake of clarity and generalizability, infinitely iterated games will be used (Nowak, 1990; Nowak and Sigmund, 1989, 1992, 1993). In cases in which the pattern of play between players P_1 and P_2 becomes periodic the payoff for P_1 from an infinite game can be calculated as the average of P_1's gains over the series of responses which is periodically repeated. The value calculated for an infinite game is the limit towards which iterated games of increasing length coverage; initial moves before periodicity is established can be ignored. It should be clear that the central results of later sections will also hold for finite games of sufficient length.

The iterated Prisoner's Dilemma has become a standard model for the emergence of cooperation within a community of egoistic agents, frequently cited for implications in both sociology and biology. What will be at issue here is a spatialization of this familiar model, in which a spatial array of players with different strategies is envisaged interacting with their immediate neighbors. This is precisely the kind of model obtained if competing strategies are instantiated as an array of cellular automata (Wolfram, 1984, 1986; Toffoli and Margolus, 1987; Demongeot, Golès, and Tchuente, 1985; Gutowitz, 1990; Nowak and May 1992, 1993; Mar and St. Denis, 1993a,b; Grim 1995, 1996). Each cell of such an array is envisaged as playing against each of eight neighbors and obtaining a local score. Each cell then surveys its neighbors. If none has a higher score, it retains its original strategy. If it has a neighbor or neighbors with higher local scores, on the other hand, it converts to the most successful strategy. The result is a model in which success is in all cases computed against local competitors, with reproduction proceeding locally as well – both features, it can be argued, which constitute a measure of realism with regard to either biological or sociological application (Grim 1995, 1996).

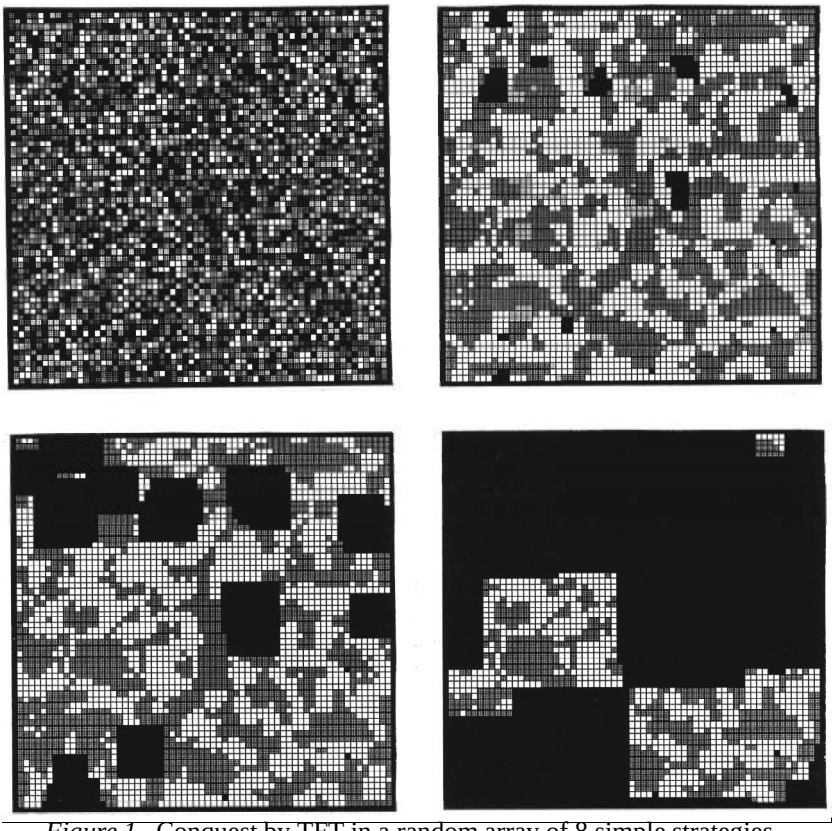

Figure 1. Conquest by TFT in a random array of 8 simple strategies.

Figure 1 shows progressive stages in the evolution of such an array. The field is a 64 $\times$ 64 cell torus or wraparound array, started on a random distribution of just eight simple strategies. The strategies used here are those specifiable in terms of (i) an initial cooperation or defection, (p) a specified response to cooperation by an opponent on the previous move, and (q) a specified response to an opponent's defection on the previous move. Envisaged as ordered triples $\langle i, p, q \rangle$ the eight strategies are the following:

$\langle 0, 0, 0 \rangle$	AllD
$\langle 0, 0, 1 \rangle$	Suspicious Tat for Tit
$\langle 0, 1, 0 \rangle$	Suspicious Tit for Tat
$\langle 0, 1, 1 \rangle$	
$\langle 1, 0, 0 \rangle$	C-then-allD

$\langle 1, 0, 1 \rangle$

$\langle 1, 1, 0 \rangle$ TFT

$\langle 1, 1, 1 \rangle$ AllC

As indicated above, each player in the array is pitted in an infinite game with each of its neighboring strategies and amasses a total score from these local competitions. Each player then compares its score with that of its neighbors; if any has a higher score it adopts the most successful strategy in sight (or is replaced by the most successful strategy, depending on one's point of view). Should two or more neighbors be tied for the highest score, one is chosen randomly.

The evolution in Figure 1 is typical for such an array, shown here for generations 1, 3, 6, and 12. AllD and C-then-AllD seem the early winners (shown using 50% gray and white, respectively). But as these threaten to take over, clusters of TFT, shown in black, thrive in their environment. In the end it is TFT which conquers all other strategies in order to occupy the screen alone; by the twenty-second generation the screen is entirely black. This is, of course, a nice vindication of TFT's robustness in a spatial context, parallel to the classic Axelrod results in non-spatial contexts (Axelrod, 1984; Mar and St. Denis, 1993a,b; Grim, 1995, 1996).

Although this is a standard result for a random configuration of these eight strategies, however, it should be noted that conquest by TFT is *not* inevitable. Figure 2 shows the evolution of an array with the same initial proportions of these eight strategies but in which TFT does not invade to conquest. In the evolution of this second array AllD and C-then-AllD retain their dominance, quickly establishing an equilibrium with each other and with occasional individual islands of TFT and suspicious TFT. Figure 2 shows generations 1, 2, 3 and 5; from this point on equilibrium is established and there is no further change.

There is often a question, with regard to arrays of strategies in the spatialized prisoner's dilemma, whether one strategy or another will grow to conquest (TFT as opposed to AllD, say) or whether some equilibrium between different strategies will remain (Grim, 1995, 1996). With genuinely infinite arrays in mind, rather than merely computer-convenient tori, the question might be posed as follows: Given a particular initial configuration, will a single strategy S eventually dominate any arbitrarily chosen finite area of the array?

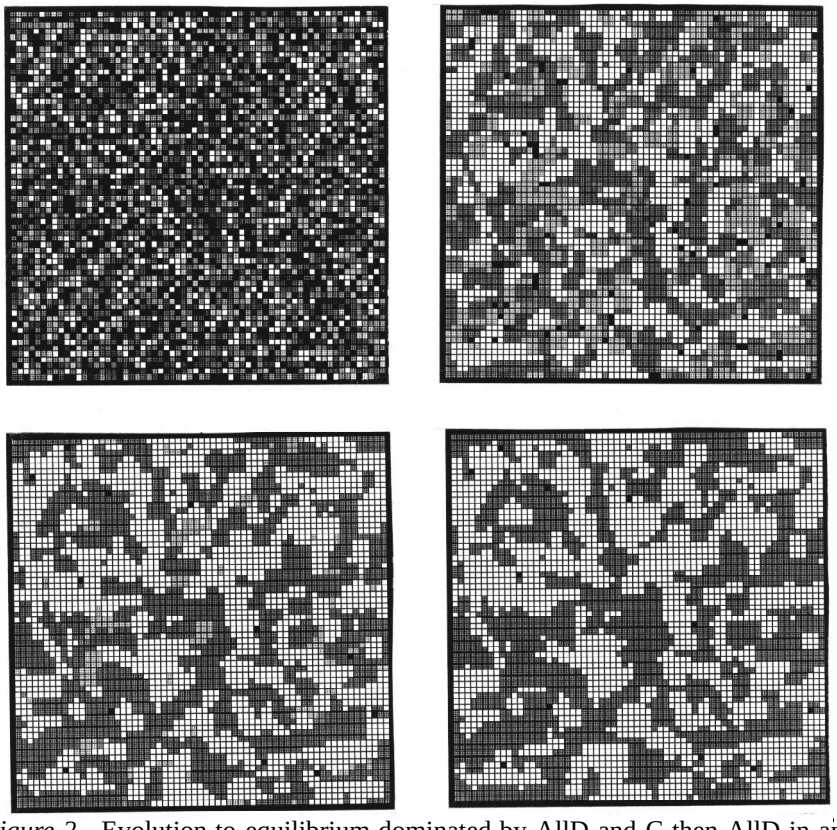

Figure 2. Evolution to equilibrium dominated by AllD and C-then-AllD in an array of the same strategies in same proportions.

It is clear from the simple examples above that the answer to such a question may depend not merely on the strategies represented or on their proportions but will be sensitive to the initial conditions of their spatial arrangement as well.

Let us suppose a standard infinite background B in which arbitrary finite configurations of strategies are embedded. B might be a uniform field consisting of a single strategy, for example, though that background actually used below will be somewhat more complicated. Some finite configurations dropped into our chosen background B may result in progressive conquest by a single strategy, ever growing outward like ripples in an infinite pool. Some may not, pulsing periodically instead or locked in a perpetual equilibrium. The main question at issue here is the following:

Q1. *For any chosen infinite background B within a spatialized Prisoner's Dilemma, is there an algorithm which will tell us in every case whether or not an embedded finite array of strategies will result in progressive conquest by a single strategy?*

The work that follows answers Q1 decisively in the negative: there is in principle no such algorithm. In that sense the spatialized Prisoner's Dilemma is formally undecidable.[1]

The central strategy of the proof is simple: to show that any of a given class of abstract machines can be modelled by finite arrays of strategies in a chosen background B, and that a relative of the halting problem will hold for such machines. The details of the proof require significantly more work. In Section 2 specifications are laid down for the abstract machines at issue – close relatives of both Turing machines and Minsky register machines – together with a discussion of computational universality for such machines and an undecidability result akin to the Halting problem. Section 3 demonstrates that any such machines, wired to an auxiliary 'strategy bomb', can be embedded or instantiated within a relevant species of competitive cellular automata. This family of cellular automata allows instantiation of the details of such machines by way of fairly direct simulations of wires and logic gates. Finally, in Section 4, a set of Prisoner's Dilemma strategies is exhibited the behavior of which within a spatialized Prisoner's Dilemma will model precisely the behavior of the cellular automata of Section 3. My attempt throughout has been to keep the discussion as informally accessible as possible.

In brief, then, the structure of the proof is as follows. If abstract machines can be instantiated as cellular automata arrays of the sort to be outlined in Section 2, then were there a Q1 algorithm which told us whether or not arbitrary arrangements of Prisoner's Dilemma strategies would result in progressive conquest by a single strategy, there would also be an algorithm which told us in all cases whether arrangements consisting of machine-arrays begun on their own encodings as input would or would not result in conquest. This algorithm would itself be computable by an abstract machine instantiable as a machine-array. We consider a slight variation in which we wire a 'strategy bomb' to its output so as to assure that the whole

array will result in conquest by a chosen strategy just in case the encoding fed it is of a machine-array which will *not* result in conquest when fed its own encoding. The supposition of a configuration corresponding to such a machine-array started on its own encoding, however, gives us a contradiction. If such a configuration would result in progressive conquest by a single strategy, it wouldn't. If it wouldn't, it would.

The contradiction tells us that there can be no machine of the sort envisaged, and so there can be no algorithm of the sort envisaged. The last step is to show that arrangements of specifiable Prisoner's Dilemma strategies will operate precisely like the machine-arrays indicated. Since there can be no algorithm which tells us the evolution of the latter in every case, there will be no algorithm which tells us the evolution of former in every case: the spatialized Prisoner's Dilemma is undecidable.

Should this be considered a result in computation theory, or in decision theory? The answer, I believe, is 'both'. Major tools of the proof include standard results from computation theory – it is a family of 'looped' but otherwise fairly traditional abstract machines the embedding of which lies at the core of the proof, and the main 'trick' of the proof is a similarly slight variation on the Halting problem. The phenomena ultimately shown to fall victim to classical undecidability, however, are quite clearly phenomena of decision theory: the behavior of abstract arrays of players following specifiable strategies in the iterated Prisoner's Dilemma. In that sense the argument that follows suffices to show that formal limitative results of the sort familiar in logic and computation theory can be produced quite readily and quite vividly in at least spatialized models within decision theory as well.

Over the past twenty years the iterated Prisoner's Dilemma has been put to work as a primary model within economics, sociology, and theoretical biology. It has been employed for such applications both because of its crystalline simplicity and because it seems intuitively to capture certain tensions between competing rewards of egoistic and altruistic behavior, and it seems safe to say that some of our simplest models of ourselves as social and biological organisms are today written in the decision-theoretic terms of the Prisoner's Dilemma. What the results at issue here suggest, however, is that

the model is not nearly as simple as it might seem – that it is complex enough, at least in infinite spatialized forms, to instantiate both universal computability and classical undecidability. In that sense classical undecidability shows up in close proximity to even some of our simplest attempts to understand ourselves as biological and social organisms.

Here two additional notes are perhaps in order.

The general strategy outlined above follows that of J.H. Conway's proof of universal computability and undecidability for the game of Life (Berlekamp, Conway, and Guy, 1982). Conway quite deliberately selected the rules of Life so as to produce such a result, however, and the appearance of a similar result for any given decision-theoretic model is by no means trivial. In the work below the complexities of (a) instantiating a Turing- or Minsky-like machine within a cellular automata, and (b) defining that cellular automata in terms of Prisoner's Dilemma strategies are in something of a balance. Complications introduced here in (b) allow instantiation of abstract machines at issue in terms of wire and gates in a manner significantly simpler than Conway's.

The question used to exhibit undecidability throughout is Q1, phrased in terms of whether a single strategy will dominate to conquest. It should be noted, however, that this is only an example of a broader class of questions regarding the behavior of arrays which can be addressed in much the same way. Given the basic method of proof other properties of arrays can be shown to be similarly undecidable: whether TFT will ever be completely extinguished in an array, for example, or whether good in the guise of a certain level of generosity will eventually triumph.

2. A SPECIFICATION OF ABSTRACT MACHINES AT ISSUE

As outlined above, the core of the proof is a class of abstract machines, eventually to be instantiated within a spatialized Prisoner's Dilemma, and for which classical undecidability results can be demonstrated.

Each of the abstract machines at issue will consist of two basic components: (1) a finite computational unit and (2) an infinitely expandable tape loop. In Figure 3, block **A** represents the compu-

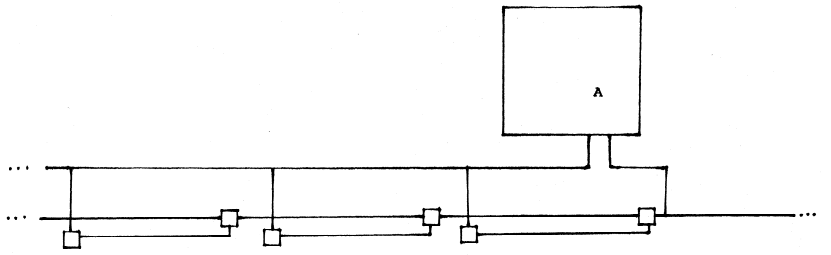

Figure 3. Abstract machine with tape loop.

tational unit. All else in the diagram represents the mechanism of the tape loop, which in the standard manner of Turing machines will serve both as an input device and an infinite external memory. The only difference between particular machines within the abstract family at issue lies in the contents of their computational unit **A**. The basic structure of the tape loop is a constant from machine to machine, and thus machines at issue will eventually be enumerable in terms of encodings for the contents of their computational units.

We can specify that a close-up of the computational unit **A** for any machine at issue will reveal a finite contents composed entirely of wires along which individual electrons pulse at a regularly clocked rate. These wires are able to turn corners and to cross each other either with interaction or without. Within the tangle of wires are recognizable diodes, allowing electron motion in one direction but not another along a wire and perhaps used in the construction of 'or' and 'not' gates. It has long been clear that these elements form a complete base for Boolean functions of any number of variables (Minsky, 1967; Dewdney, 1993).

Each machine requires a tape loop, for which various specifications are possible.

A close-up of one version of the standard tape loop attached to each machine is shown in Figure 4. Here an encoded input is envisaged entering from the right as a finite series of spaced electron pulses along the wire, marked with a particular coding for beginning **b** and end **e**. At the first branch the coded series moves straight ahead, with a diode blocking similar travel northward. The series enters a counter marked **C1**, which is triggered by the beginning code **b** to start a 100-unit 'clock'. If the end of the encoded message **e** arrives before the clock has ticked off its 100 pulses, **C1** returns to

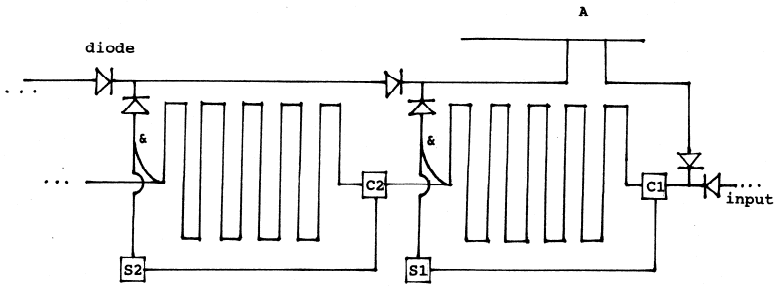

Figure 4. Tape loop details.

its starting position. If not, a single pulse is sent by a southerly route to signal block **S1** and **C1** then returns to its starting position.

The circuitous waffles through which the message is travelling to the left of **C1** are, we might stipulate, 150 clock units long, and are arranged in such a way that a message sent from **C1** can arrive at **S1** before the beginning of the message series arrives at the juncture just above **S1**. On receiving an impulse from **C1**, **S1** starts a constant pulse of, say, 150 electrons at the standard clocked interval and then resets. Marked as **&** in Figure 4 is an 'and' gate. Thus if our series of signals is less than 100 units long, it will travel through the waffles, north to the upper wire and to the right, on and into the computational unit at **A**. It will also continue its travel to the left, but in that direction is lost harmlessly into the infinite ether.

What if the encoded message is more than 100 units long? In that case **S1** will not be triggered, and the encoding will not cycle north at **S1**. It will continue instead to a counter **C2**, which at the beginning signal **b** starts a clock of 200 pulses. If the end signal **e** arrives before 200 pulses, a signal is sent to **S2** which begins a regular series of 250 pulses. If the message series is greater than 100 units but less than 200, then, it will cycle back at the second juncture.

The tape structure continues infinitely to the left with waffle units increasing by any regular interval and the clocks of **C** and **S** modules increasing accordingly. For illustration we have chosen additive units of 100, but increasing powers of 10 would do as well. The purpose of the whole should be clear: any finite message will trigger a recycling loop large enough to accommodate itself. Because **C** and **S** modules are reset every time, the loop is ready for a new message of any finite length emerging from the computational unit and recycling around

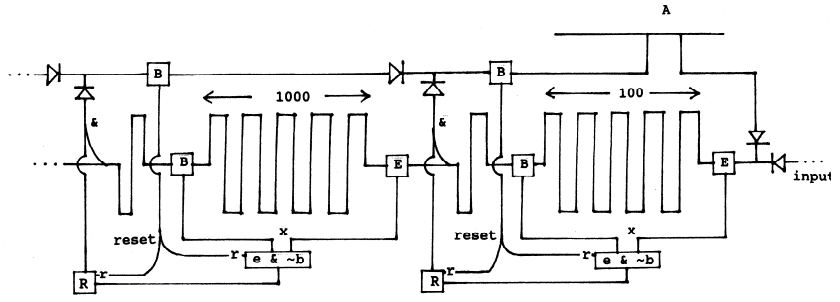

Figure 5. Tape loop with simpler components.

again. Ends of the infinite loop close off to accommodate the current length of the signal.

C and **S** modules can clearly be conceived of as simple finite computational units, themselves composed of wires and standard gates. In each case the crucial element, beyond a beginning- and ending-detector, is a simple loop serving either as a clock or as a pulse-generator of a certain duration.

As indicated, other forms of such a tape loop are possible within the general structure of the proof. Figure 5 shows a significantly simpler form which operates with factory-identical units: a beginning detector **B**, an end detector **E**, a 'repeater' **R** and a signal box. The repeater simply generates a stream of pulses when triggered and until reset. The signal box outputs a single pulse if an input from **E** has been received at **x** and one from **B** has not, resetting on a signal at **r**. Given these elements the progressively longer clock devices above can be replaced simply by increasing wire lengths between **b** and **e** detectors as the tape structure extends infinitely to the left. For present purposes, however, what is important is not the specific design chosen for a standard tape loop but the simple fact that such a loop can be constructed in any of a variety of ways using only wires (including infinite wires) and simple 'or' and 'not' gates.

The messages encoded on our tape loop have been specified above merely as a series of clocked pulses along a wire, marked with a beginning and an end. Here as before there are various alternatives. For the sake of concreteness, however, it may be useful to specify encoding in a particular form:

The beginning signal, let us suppose, consists of two contiguous 1's; the ending signal consists of three. Between beginning

and end are a series of spaces to be treated as registers, separated by dividers which also serve as addresses. The first register space appears immediately following the beginning signal, and requires no further address. The second register space is marked by an initial address of 1111 (using one more 1 than our ending signal), the third by 11111, and so forth. As a whole the message than takes the following form:

11 . . . 1111 . . . 11111 111

Contents of the registers fill the blanks marked with ellipses, and may be envisaged in monadic notation. One way to disambiguate register contents from their addresses is to begin and end them with 0's, inserting 0's between any digits within them as well. A content register of '5' would thus become 01010101010. '0' would become 000. A complete message using only three registers, containing numbers 5, 0, and 1, respectively, would then appear as follows:

11010101010101111000111110101111

Here again specifics are unimportant, of course; all that is required is an unambiguous way of encoding a beginning signal, an end signal, register addresses and their contents.

Given some appropriate convention for message encoding, a tape loop of the form outlined constitutes an infinitely expandable external memory for the computational center at **A**. The tape might be thought of as containing an infinitely expandable number of memory registers, each of which can hold an arbitrarily large integer. Minsky long ago demonstrated that such a memory, together with the arbitrary Boolean power of a finite computation center composed of wires, 'or', and 'not' gates, is sufficient for computational universality (Minsky, 1967).

If we envisage the first two registers of the tape as containing 'position' and 'state' numbers, with all others containing simply 0 or 1, operations on the tape might alternatively be thought of as corresponding to those of a 'semi-infinite' Turing machine, with a tape infinite in only one direction. As is well known, any Turing-computable function is also computable by such a variation (Minsky, 1967; Boolos and Jeffrey, 1989; Dewdney, 1993). The tape contents of a semi-infinite Turing machine tape at any point will correspond

directly to the contents of the message beyond the second register, with a message series lengthened whenever needed by the addition of new register addresses. Changes in position and state in the standard Turing machine will correspond to changes in the position- and state-*numbers* of such a message. Given these conventions the quintuplets of any Turing machine table can be rewritten fairly directly as programmed instructions within the wired computational unit.

Corresponding to any Turing machine, then, will be an abstract machine of the sort outlined. Any Turing-computable function will be computable as well by a wired computation center attached to an infinitely expandable tape loop.[2]

With a parallel to Turing machines, of course, comes a parallel to the Halting Problem. An abbreviated form of the problem is as follows (Berlekamp, Conway, and Guy, 1982):

A Turing machine, and thus an abstract machine of the sort at issue here, can be constructed for the express purpose of investigating any specified, and arbitrarily hard, arithmetical question. We might construct such a machine to search for counter-examples to Goldbach's conjecture, for example, designing it to indicate that it has found a counter-example either by printing a particular message on the tape or by some auxiliary signal – sending a single pulse down a designated signal wire, for example.

Is there an algorithm which will tell us, for any machine of the type at issue, whether it will or will not eventually send a pulse down its signal wire? No. If there were such an algorithm, it would effectively tell us whether arbitrary difficult arithmetical problems have solutions. But as Conway notes, "mathematical logicians have proved that there's no technique which guarantees to tell when arbitrary arithmetical problems have solutions" (p. 847). There can thus be no algorithm which predicts in each case the behavior of our abstract machines.

In somewhat deeper detail, and avoiding the appeal to authority, undecidability for our family of abstract machines can be shown as follows:

Let us conceive of all machines at issue built with a special signal loop at the northeasternmost corner – a loop at the end of a wire down which a pulse may or may not eventually proceed. Some machines, started with particular input messages on their tapes, will eventually

send a pulse down that wire. Some will not. Machines at issue differ only in the finite contents of their computational centers, and can be though of as enumerated or indexed accordingly.

There can be no machine which will send a pulse to its signal loop just in case it is fed the number of a machine which will not send a signal to its loop if started on its own index number. For suppose that there were such a machine, and suppose it were fed its own number. It would then send a pulse to its signal loop just in case it would never send a pulse to its signal loop. By contradiction there can be no such machine.

It follows that there can be no machine which will indicate by such a signal whether or not a numbered machine started on a given message will eventually send a pulse to its signal loop. For given *that* machine, we could construct the machine proved impossible above by arranging signal loops appropriately and attaching an initial message duplicator.

Since any algorithm can be instantiated as a machine of the type at issue, the 'Signalling' Problem – as analogue to the 'Halting' Problem – proves undecidable. There is no effective procedure which will predict in general the signalling behavior of the abstract machines at issue.

None of this should be too surprising: the undecidability of the Halting Problem has simply been carried over to a species of abstract machines which might be envisaged either as wired variants of Turing machines or as loop-memoried forms of Minsky register machines. Details are interesting, perhaps, but the general result uncontroversial. The promised application of such a result to decision theory demands showing in addition that the abstract machines at issue can be embedded within competitive cellular automata which will in turn model the spatialized Prisoner's dilemma.

3. INSTANTIATING ABSTRACT MACHINES BY WAY OF 'WIRES' WITHIN COMPETITIVE CELLULAR AUTOMATA

The cellular automata to be used here will instantiate the abstract machines specified in terms of 'wires' and logical gates constructed from them.

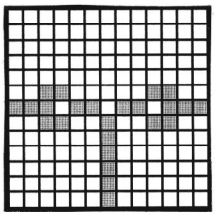

Figure 6. 'Or' gate in Silverman's wireworld, incorporating two diodes.

The background of this approach is two-dimensional cellular automata developed by Brian Silverman to simulate the travel of electrons on a wire (Silverman, 1987; Dewdney, 1990). This 'wireworld' allowed for wire turnings and crossings, the creation of diodes, 'and', 'or', and 'not' gates. With these can be constructed the cellular automaton equivalent of any finite computer.

'Wireworld' operated in terms of just four cellular states: background, wire, electron head and electron tail. Silverman's rules were simple. During each tick of the automata clock electron heads became electron tails and electron tails became wire cells. Wire cells became electron heads just in case they were bordered by one or two electron heads, and background cells never changed. An 'or' gate from Silverman's wireworld, incorporating two diodes, is shown in Figure 6.

Because wireworld can be used to simulate the operation of arbitrary finite computers, and because the infinite tape mechanism of Section 2 can itself be simulated using infinite wires, an easy undecidability result is possible regarding wireworld. Consider any enumeration of finite arrays which might serve as our computational units (or computational units with initial inputs), again thought of as having an upper signalling 'loop'. In wireworld a signal once sent to such a loop continues to circulate there. Is there any algorithm which will tell us whether or not an arbitrary array with Silverman's rules will end up with a signal in its upper loop if embedded in the background of an infinitely expandable tape-loop? By a slight variation on the Signalling Problem of Section 2 the answer is clearly 'no'; the evolution of wireworld displays is formally undecidable.

In what follows it is a significantly more complicated relative of wireworld that will be at issue. The general idea remains the same, however: to model within cellular automata the movement of electrons on wires in such a way as to allow the construction of a complete

Boolean base of operation gates. Beyond that things become more complex. Within Silverman's rules, change in a particular cell could be calculated merely by noting the states of its immediate neighbors and by counting the number of those which are electron heads. In the 'competitive' automata at issue here cells will be thought of as gaining particular scores in competition with their immediate neighbors, thereby amassing a total score in their immediate neighborhood. In a second conceptual step they then compare their score with that of their immediate neighbors and change to a neighboring state should there be one with a higher total score.

More specifically, the competitive automata rules are the following. Four player strategies are instantiated as cells: wire, background, electron head and tail. Each of these players is thought of as competing against other players of the same or different type, with standard scores for such competitions: wires standardly get a score of 3 in competition with electron tails, for example. At each tick of the clock, each player within an array competes against each of its 8 neighbors and adds its total score. Each player then surveys its neighbors and converts to the strategy of a neighboring player should there be one which achieved a higher local score. A wire might thus become an electron head because a neighboring electron head had a higher total score, for example, or an electron tail might become a wire cell. Should a cell be bordered by two with equal higher scores, one is chosen randomly.

Can standard wires, crossings, and Boolean gates be modelled in this more competitive wireworld? The answer is 'yes', although computation is necessarily more complicated throughout and although a somewhat different design is required for standard gates.

For convenience the four strategies will be labelled yellow (background), blue (wire), red (electron head), and pink (electron tail), though for printing purposes our illustrations will use shades of gray (0%, 15%, 88%, and 56%, respectively). The following numerical scores, arrived at by simple but excruciating experimentation, will then give us the basis needed for a wire-like simulation. '**bb**' encodes

the score of blue against itself, '**br**' the score of blue against red, etc:

$$\begin{array}{ll} \mathbf{bb} = 2.412 & \mathbf{rb} = 2.485 \\ \mathbf{br} = 2.534 & \mathbf{rr} = 2.412 \\ \mathbf{by} = 2.472 & \mathbf{ry} = 2.472 \\ \mathbf{bp} = 3.000 & \mathbf{rp} = 2.542 \end{array}$$

$$\begin{array}{ll} \mathbf{pb} = 2.583 & \mathbf{yb} = 0.868 \\ \mathbf{pr} = 2.567 & \mathbf{yr} = 0.868 \\ \mathbf{py} = 2.472 & \mathbf{yy} = 2.667 \\ \mathbf{pp} = 2.412 & \mathbf{yp} = 0.868 \end{array}$$

Though simpler scores are possible here, those listed offer a particularly smooth connection with the work of the following section.

Consider the not-so-simple phenomenon of a simple blue wire maintained on a yellow field. At each tick of the clock, each cell of the display competes against its eight immediate neighbors. In order for a blue cell to remain blue, no immediate neighbor can have a higher score. In order for its yellow neighbors to remain yellow, however, they must be in contact with a yellow cell the total score of which is greater than the score of their blue neighbor. Given the scores above, a blue surrounded by two blues and 6 yellows has a score higher than a yellow in contact with 5 yellows and 3 blues, but less than the score of a yellow surrounded by 8 yellows. At each tick of the clock the blue cells of our wire dominate their neighbors, but are counter-balanced by the high score of yellow against yellow and the proximity of a yellow surrounded by 8 yellow neighbors. A blue wire is thus tenuously balanced between extinction and explosion within a yellow field. A similar trick is used for red and pink cells on a wire.

Travel of an electron along a wire – a red and pink pair on a strip of blue against yellow – is achieved by awarding a red bordering a blue a higher total score in context, a pink bordering a red a higher score than its neighboring red, and a blue bordering a pink a higher score still. The result is the one-cell-per-clock tick simulation of movement illustrated in Figure 7.

Construction of a set of scores adequate for a basic set of gates calls for careful adjustment for turning both solid and nicked corners, and a sensitivity of blue squares to red finely tuned enough to allow

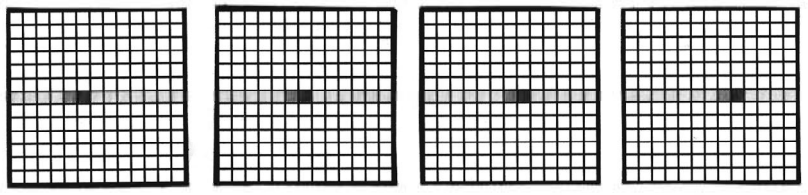

Figure 7. Electron moving along wire in array of competitive automata.

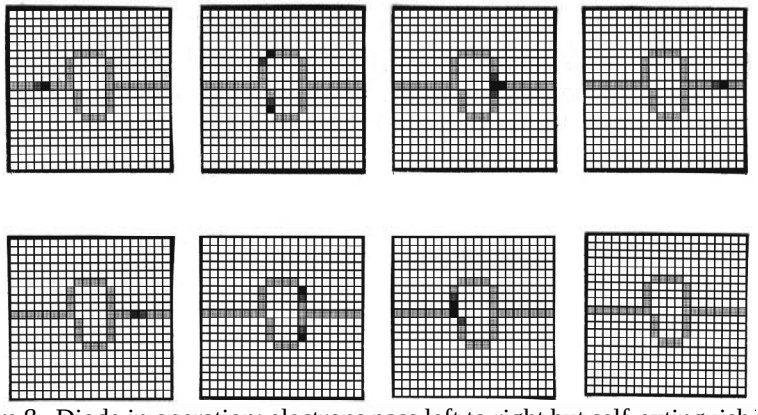

Figure 8. Diode in operation: electrons pass left to right but self-extinguish right to left

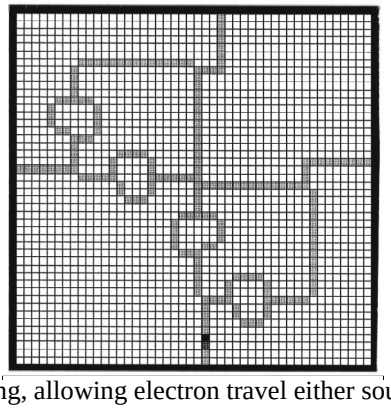

Figure 9. Wire crossing, allowing electron travel either south to north or west to east

both for branching and for the 'kill' function used in diodes and other basic operations. A diode in operation from each direction is shown in Figure 8, allowing electron travel left to right but blocking it by self-extinction right to left.

The complications of wire-crossing are illustrated in Figure 9. Here an electron can travel south to north without propagating east

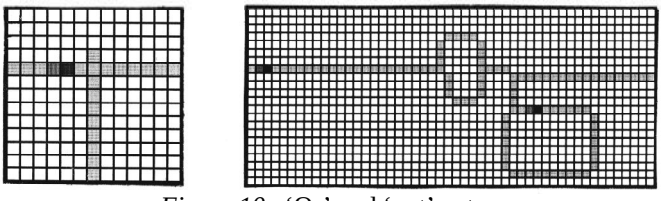

Figure 10. 'Or' and 'not' gates

or west or can travel west to east without propagating north or south. Travelling north, as illustrated, the electron will divide at the first branch with copies proceeding through each of two diodes. At the central cross it will propagate in all three directions. That copy moving west, however, will be killed at the first diode. That moving right will be extinguished by a kill from the doppelgänger which split off earlier. That moving north will continue, with a twin generated to the left at the upper kill site disabled at a later diode.

'Or' and 'not' gates are shown in Figure 10. In the operation of the negation loop a timing convention is assumed for signals sent along a wire; for purposes of illustration 30 'ticks' between consecutive signals have been assumed. The purpose of a negation 'inverter' is to convert a series of spaced signals – 1011001100, say – to their negative image: 0100110011. The 'not' gate achieves this by generating impulses in its lower loop in synch with the signal rate and sending these out to a 'kill' gate. If an impulse representing a '1' arrives a the 'kill' gate from the left, it and the impulse from the lower loop mutually annihilate. No pulse is therefore sent out to the right, and the incoming '1' has effectively been converted to a '0'. If no impulse arrives from the left at the proper time, on the other hand, signalling a '0', the pulse generated from the lower loop travels out to the right undisturbed as a signalled '1'. This achieves the desired result.[3]

Any finite arrangement of wires and standard gates can thus be simulated within the competitive cellular automata outlined. It should also be clear that these components will suffice for the construction in an infinite field of the infinite tape-loop of Section 2. Undecidability results outlined above will therefore carry over as undecidability results regarding our arrays of competitive players.

Here undecidability can also be made somewhat more graphic. Let us first add two additional players, green and fuchsia, to our

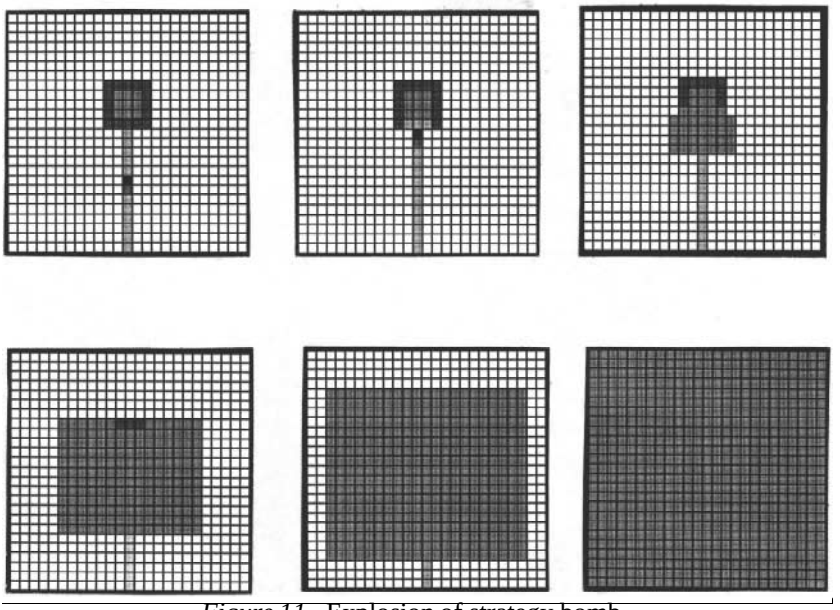

Figure 11. Explosion of strategy bomb.

original set of four. Scores for these, in supplement to our table above, are as follows:

bf = 2.412	**rf** = 2.485
bg = 0.857	**rg** = 0.857

pf = 2.583	**yf** = 0.868
pg = 0.857	**yg** = 0.857

fb = 2.412	**gb** = 4.428
fr = 2.534	**gr** = 4.428
fp = 3.000	**gp** = 4.428
fy = 2.472	**gy** = 4.428
ff = 2.412	**gg** = 2.667
fg = 2.472	**gf** = 0.868

What these two new strategies allow us is the construction of a ‘strategy bomb’: a device which will keep hostage and harmless a small patch of green unless a pulse is sent down a particular wire. Given a pulse down that wire, on the other hand, green will

be released to expand without obstacle ever outward, progressively conquering all strategies in its path. Such a bomb is shown in Figure 11 using a central block of nine green cells surrounded by a protective border of fuchsia. Left alone it remains harmless. A single pulse down its feed wire, however, releases an all-invading cloud of green.

Consider arbitrary finite arrangements of these six players or strategies, embedded in a standard infinite field which contains the cellular equivalent of a tape loop. Is there an algorithm which will tell us in each case whether such a finite arrangement so embedded will result in a progressive conquest by green or not?

No. As noted above, an abstract machine of the type at issue can be constructed to look for a solution to any arbitrarily hard arithmetical problem. Any such machine, starting on any given input, can be instantiated as a finite array of our chosen strategies, embedded in the background at issue. Such an array can be varied slightly by wiring its relevant signal to the strategy bomb above. Thus to arbitrarily difficult arithmetical problems will correspond arrays of our chosen strategies which will or will not result in a progressive conquest by green depending on whether the problem at issue has a solution. Were there an algorithm which sorted such arrangements into those which would result in conquest and those which would not, it would give us as well an algorithm suitable for deciding whether arbitrarily difficult arithmetical problems have solutions. Since there can be no algorithm of the latter sort, there can be no algorithm of the former sort either: there can be no algorithm which will tell us in every case whether a finite cellular arrangement embedded in the standard background will or will not result in a progressive conquest by green.

In more traditional detail the proof is as follows:

As outlined above, any abstract machine at issue can be embedded as a finite arrangement of strategies within a standard background corresponding to a tape loop. In each case the arrangement can be configured so as to feed a relevant output signal to a strategy bomb. Any machine configuration, so wired, can be encoded, either directly or via the abstract machine embedded.

There can, to begin with, be no algorithm which decides, for arbitrary machine configuration encodings, whether the array composed of that machine configuration started on its own encoding as input

will result in conquest by green or not. If there were, that algorithm could be computed by some abstract machine of the type outlined in Section 2. That machine could in turn be instantiated as a cellular array with its signal appropriately wired to a strategy green invasion. That machine configuration would in turn be assigned an encoding.

Consider the array composed of that machine configuration begun on its own encoding as input. Were that array not to result in conquest by green, the core machine would give a negative answer, igniting the strategy bomb which would result in precisely the conquest at issue. Were the array to result in conquest, on the other hand, the instantiated machine would never send the signal required for any such conquest to take place. The array at issue would thus result in conquest by green just in case it would not. The contradiction shows us that there can be no such array, and thus can be no machine and no algorithm of this first type.

It follows that there can be no algorithm which decides in all cases, for configurations specified in terms of machine configuration encodings and inputs, whether the result will be unlimited conquest by green or not. If there were, that algorithm would be computable by an abstract machine from which the machine configuration above could be obtained by a careful wiring of a strategy bomb to the proper output and addition of an input duplicator (for the machine arrays at issue here, achievable simply by a branching wire). Since there can be no machine and no algorithm of the first type, there can be no machine and no algorithm of this more general second type either.

There can, finally, be no algorithm which decides, for an arbitrary finite configuration of the specified strategies, whether it will result in unlimited expansion by green or not when embedded in the standard background specified. Since specification in terms of machine configuration and input will be effectively translatable into any standard encoding for finite configurations, this final algorithm would offer a decision procedure which told us, for any machine configuration and input, whether its configuration would result in unlimited expansion by green or not. By the argument above there can be no such algorithm, and thus there can be no algorithm of this final sort either: there can be no algorithm which will tell us in every

case whether a finite cellular arrangement embedded in the standard background will or will not result in a progressive conquest by green.

4. THE UNDECIDABILITY OF THE SPATIALIZED PRISONER'S DILEMMA

The final step of the proof is to show that the undecidability discussed for a class of formal machines in Section 2 and carried over to a particular form of cellular automata in Section 3 is ultimately an undecidability within decision theory as well: the undecidability of the spatialized Prisoner's Dilemma.

This is in fact an easy step to show, though significantly less easy to construct. What is required is simply a set of Prisoner's Dilemma strategies with payoffs against each other in infinite games corresponding to the competition scores used in constructing the competitive cellular automata in Section 3. Given such payoffs the strategies at issue will behave precisely like the blue, yellow, red, pink, fuchsia and green cells outlined above.

Though other sets are possible, the following Prisoner's Dilemma strategies do the trick. The general idea of this particular construction is to have each strategy begin with a set series of cooperations and defections which in effect constitute a code for that strategy alone. The behavior of each strategy is then geared to the code of its opponent, allowing a great deal of flexibility in fine tuning the scores required. As noted in introduction, it is infinite games that are at issue, and thus the average score over a repeated period of play that is of importance. Scoring during the short initial coding can thus be effectively ignored. Since finite games of increasing length will progressively swamp any initial scoring in precisely the same way, results will hold for finite games of sufficient length as well.

Strategy **b**: Start with 000. Then:

If opponent started 000, play 1111111111100000, repeat.
If opponent started 001, play $\underbrace{010101\ldots01}_{\text{100 plays}}$ followed by 011, repeat.
If opponent started 010, play 111111111100, repeat.
If opponent started 011, play $\underbrace{1000000000}_{\text{10 times}}$ followed by 000000, repeat.
If opponent started 100, play 1111111111100000, repeat.
If opponent started 101, play 1111111, repeat.

Strategy **r**: Start with 001. Then:
If opponent started 000, play $\underbrace{101010\ldots10}_{100\text{ plays}}$ followed by 111, repeat.
If opponent started 001, play 1111111111100000, repeat.
If opponent started 010, play $\underbrace{010101\ldots01}_{180\text{ plays}}$ followed by $\underbrace{111\ldots111}_{20\text{ plays}}$ and 111, repeat.
If opponent started 011, play $\underbrace{1000000000}_{10\text{ times}}$ followed by 000000, repeat.
If opponent started 100, play $\underbrace{101010\ldots10}_{100\text{ plays}}$ followed by 111, repeat.
If opponent started 101, play 1111111, repeat.

Strategy **p**: Start with 010. Then:
If opponent started 000, play 111111111101, repeat.
If opponent started 001, play $\underbrace{101010\ldots10}_{180\text{ plays}}$ followed by $\underbrace{111\ldots111}_{20\text{ plays}}$ and 011, repeat.
If opponent started 010, play 1111111111100000, repeat.
If opponent started 011, play $\underbrace{1000000000}_{10\text{ times}}$ followed by 000000, repeat.
If opponent started 100, play 111111111101, repeat.
If opponent started 101, play 1111111, repeat.

Strategy **y**: Start with 011. Then:
If opponent started 000, 001, 010, or 100, play $\underbrace{1111000000}_{10\text{ times}}$, followed by 111100, repeat.
If opponent started 011, play 111111111100, repeat.
If opponent started 101, play 1111111, repeat.

Strategy **f**: Start with 100. Then:
If opponent started 000, play 1111111111100000, repeat.
If opponent started 001, play $\underbrace{010101\ldots01}_{100\text{ plays}}$, followed by 011, repeat.
If opponent started 010, play 111111111100, repeat.
If opponent started 001, play $\underbrace{010101\ldots01}_{10\text{ times}}$, followed by 000000, repeat.
If opponent started 100, play 1111111111100000, repeat.
If opponent started 101, play $\underbrace{1000000000}_{10\text{ times}}$ followed by 000000, repeat.

Strategy **g**: Start with 101. Then:
If opponent starts 000, 001, 010, or 011, play 0000011, repeat.
If opponent starts 100, play $\underbrace{1111000000}_{10\text{ times}}$ followed by 111100, repeat.
If opponent starts 101, play 111111111100, repeat.

An example of strategies in operation and their relation to the scores used in section 3 is perhaps not out of order. In an infinite

game the strategies specified for **b** and **y** above generate the following patterns of play:

$$\begin{aligned}
\mathbf{b}: \; & 000 \underbrace{1000000000}_{10 \text{ times}} 000000 \underbrace{1000000000}_{10 \text{ times}} 000000\ldots \\
\mathbf{y}: \; & 011 \underbrace{1111000000}_{10 \text{ times}} 111100 \underbrace{1111000000}_{10 \text{ times}} 111100\ldots
\end{aligned}$$

The longer a finite game with these patterns the closer **b**'s score will approach its average over the repeated period: $262/106 = 2.47169811$. The longer a finite game the closer **y**'s score will approach 0.867924528. Scores in an infinite game, on the other hand, are simply taken as these limits. Rounded off, these give us precisely the scores used for **by** and **yb** in Section 3.

The strategy specifications above are constructed so as to give the same kind of match to the scores used in Section 3 for all strategies involved. Given those scores, as the work of Section 3 shows, the evolution of arrays of competitive cellular automata proves formally undecidable. Because the strategies above generate precisely these scores, spatial arrays of Prisoner's Dilemma strategies will prove undecidable as well.

5. CONCLUSION

The proof outlined carries classical undecidability results into decision theory through the medium of cellular automata. Undecidability for the abstract machines of Section 1 – close relatives of both Turing machines and Minsky register machines – is firmly based in familiar results from computation theory. The strategy of embedding or instantiating abstract machines within cellular automata arrays, following the general lines of Conway's undecidability proof for the game of Life, is employed here in terms of simpler wires and gates and for a quite different family of competitive automata. The latter is crucial for the present application. It is because the spatialized Prisoner's Dilemma will be instantiated as precisely that kind of cellular automata that classical undecidability will carry over into decision theory: the spatialized Prisoner's Dilemma proves undecidable.

None of this, of course, indicates that modelling in terms of the Prisoner's Dilemma is in any way doomed or hopeless, any more than standard undecidability results regarding formalized arithmetic indicate that ordinary arithmetic is doomed or hopeless. What the present results do show is that there will be classes of questions regarding the behavior of our models – at least ideal models both spatial and infinite – for which no algorithm or effective procedure will be capable of supplying an answer in all cases.

The Prisoner's Dilemma is of interest precisely because it has seemed so promising as a model – promising both because of its abstract simplicity and because of its intuitive application to questions of egoism and altruism within economics, sociology, and theoretical biology. One thing undecidability means from the modeler's perspective is that idealized forms of that model are significantly more complex than may at first appear. The standard matrix for the Prisoner's Dilemma may give one the impression that all that is at issue is a small handful of numerical comparisons. In the idealized infinite case and once strategy interaction is taken into account, however, the model becomes precisely as complex as standard paradigms from computation theory, exhibiting the same forms of undecidability for precisely the same reasons.

ACKNOWLEDGEMENTS

I owe a great debt to Gary Mar and Paul St. Denis for sharing their work on the Prisoner's Dilemma in cellular automata and for the core of programming employed here. I am grateful to Paul St. Denis and Tobias Müller for fruitful discussion, and to an anonymous reviewer for very helpful comments.

NOTES

[1] A related NP-Completeness result for finite arrays is sketched in Grim 1994b.

[2] A form of the proof is also possible using a two-register Minsky machine which dispenses with the infinite structure required here for the tape loop. This makes the result more graphic in some ways, in that only a uniform infinite background **B** of a single strategy is required. Major complications are required in the outline of relevant Prisoner's Dilemma strategies, however. See Grim 1994a.

[3] A diskette with programs in Trubasic for these components, unprotected so as to facilitate further research, is available from the author on request.

REFERENCES

Axelrod, R.: 1984, *The Evolution of Cooperation*, Basic Books, New York.

Berlekamp, E., Conway, J., and Guy, R.: 1982, *Winning Ways for your Mathematical Plays*, Vol. II, Academic Press, London.

Boolos, G., and Jeffrey, R.: 1989, *Computability and Logic*, Third Edition, Cambridge Univ. Press, New York.

Demongeot, J., Golès, E., and Tchuente, M. (Eds.): 1985, *Dynamical Systems and Cellular Automata*, Academic Press, New York. Dewdney, A.K.: 1990, 'The cellular automata programs that create wireworld, rugworld and other diversions', Computer Recreations, *Scientific American* **262**(1), 146–149.

Dewdney, A.K.: 1993, *The (New) Turing Omnibus*, Computer Science Press, New York.

Grim, P.: 1994a, 'Computation and Undecidability in the Spatialized Prisoner's Dilemma,' Research Report No. 94-02, Group for Logic and Formal Semantics, Dept. of Philosophy, Suny at Stony Brook.

Grim, P.: 1994b 'An NP-Complete Question Regarding the Spatialized Prisoner's Dilemma,' Research Report No. 94-03, Group for Logic and Formal Semantics, Dept. of Philosophy, SUNY at Stony Brook.

Grim, P.: 1995, 'The Greater Generosity of the Spatialized Prisoner's Dilemma', *Journal of Theoretical Biology* **173**, 353–359.

Grim, P.: 1996, 'Spatialization and greater generosity in the stochastic Prisoner's Dilemma', *BioSystems* **37**, 3–17.

Gutowitz, H. (Ed.): 1990, *Cellular Automata: Theory and Experiment*, North-Holland, New York.

Mar, G., and St. Denis, P.: 1993a, 'The Evolution of Dynamical Meta-Strategies in the Prisoner's Dilemma', International Conference on Game Theory, SUNY at Stony Brook, July 1993, and research report No. 93-01, Group for Logic and Formal Semantics, Dept. of Philosophy, SUNY at Stony Brook.

Mar, G., and St. Denis, P.: 1993b, 'Chaos in Cooperation: Two-Dimensional Prisoner's Dilemmas in Infinite-Valued Logic,' research report No. 93-02, Group for Logic and Formal Semantics, Dept. of Philosophy, SUNY at Stony Brook, and *International Journal of Bifurcation and Chaos* **4** (1994), 943–958.

Minsky, M.: 1967, *Computation: Finite and Infinite Machines*, Prentice-Hall, Englewood Cliffs, N.J.

Nowak, M.: 1990, 'Stochastic Strategies in the Prisoner's Dilemma,' *Theoretical Population Biology* **38**, 93–112.

Nowak, M., and May, R.: 1992, 'Evolutionary games and spatial chaos', *Nature* **359**, 826–829.

Nowak, M., and May, R.: 1993, 'The Spatial Dimensions of Evolution', *International Journal of Bifurcation and Chaos* **3**(1), 35–78.

Nowak, M., and Sigmund, K.: 1989, 'Game-Dynamical Aspects of the Prisoner's Dilemma', *Applied Mathematics and Computation* **30**, 191–213.

Nowak, M., and Sigmund, K.: 1992, 'Tit for tat in heterogeneous populations', *Nature* **355**, 250–252.

Nowak, M., and Sigmund, K.: 1993, 'A strategy of win-stay, lose-shift that outperforms tit-for-tat in the Prisoner's Dilemma game', *Nature* **364**, 56–58.
Silverman, Brian.: 1987, *The Phantom Fish Tank: An Ecology of Mind*, Logo Computer Systems, Montreal.
Toffoli, T., and Margolus, N.: 1987, *Cellular Automata Machines: A New Environment for Modelling*, MIT Press, Cambridge, Mass.
Wolfram, S.: 1984, 'Cellular automata as models of complexity', *Nature* **311**, 419–424.
Wolfram, S.: 1986, *Theory and Applications of Cellular Automata*, World Scientific, Philadelphia.

Group for Logic & Formal Semantics,
Department of Philosophy,
SUNY at Stony Brook,
Stony Brook, NY 11794, U.S.A.
pgrim@ccmail.sunysb.edu