UvA-DARE (Digital Academic Repository)

Logic for social software
Pauly, M.

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
Chapter 1
Let the Games Begin

It is an old idea that thinking to the bottom of our knowledge, all human activity seems merely play. Those who are willing to content themselves with a metaphysical conclusion of this kind should not read this book.

Johan Huizinga

1.1 Logic and Games

Logicians like to play games. In fact, not only logicians like to play games, for homo sapiens seems to be quite a close relative of homo ludens [69]. For the logician, however, playing games can be very useful besides being entertaining. As a consequence, the professional logician comes close to fulfilling one of his childhood dreams: getting paid to play games. In this respect then, he can be compared to professional tennis players, which explains the occasional complaint of a logician about being underpaid. One may even argue that such a complaint gains additional weight by the fact that the logician often manages to come up with winning strategies (i.e., strategies which will be successful against any opponent), something that even the best tennis players can only dream of.

Games have been useful in logic in a variety of ways, some of the most prominent ones being the following (see [12] for a more extensive survey):

Game-Theoretic Semantics: For First-Order Logic, truth in a model can be defined using game-theoretic semantics (see, e.g., [66]). A game-theoretically natural extension of this semantics has led to Independence-Friendly Logic [65, 109]. Likewise, game-theoretic semantics have been proposed for Linear Logic [21, 2].

Dialogue Games: The relation of logical consequence has been viewed as a dialogue game in [83], where the precise rules of the game determine whether
one obtains intuitionistic or classical logical consequence. Extensions of this work have yielded dialogue games for various modal logics and Linear Logic [110, 111].

**Model-Comparison Games:** In recent expositions of model theory (see, e.g., [41, 67]), Ehrenfeucht-Fraïssé games "operationalize" elementary equivalence between models. The number of pebbles used in these games corresponds to the number of variables needed to express a particular property.

Like logic, theoretical computer science also had its share of playfulness, on the most fundamental level through models of computation which are essentially 2-player games [31]. We will have more to say on this issue in section 1.2.

Instead of using games for the purposes of logic, a second line of investigation tries to use logic for the purposes of game theory. And although it seems natural that the study of games and how players should behave in games takes into account players’ reasoning and rationality (the domain of logic, it would seem), formal logical approaches to games are of a rather recent origin. The work in this area can be categorized as follows:

**Epistemic Logic:** Solution concepts such as the Nash equilibrium or the subgame-perfect equilibrium can be investigated concerning their epistemic presuppositions, asking, e.g., which assumptions about rationality and players’ belief/knowledge of rationality are necessary to guarantee a Nash-equilibrium outcome. Modal logics have been used to obtain formal epistemic axiomatizations of various solution concepts, both on the propositional [123, 34, 30] and on the first-order level [74].

**Dynamic Epistemic Logic:** Epistemic logics have also been extended with action modalities in order to express knowledge and belief change. These dynamic epistemic logics can not only express statements like “player 2 knows that player 1 holds the queen of hearts” but also statements like “player 1 knows that after showing his card to player 3, player 2 will know that player 1 holds the queen of hearts” [9, 22]. This approach has been used to formally model the board game *Clue* or *Cluedo* in [40].

**Others:** Besides epistemic logic, temporal logic has been extended with a prediction relation which captures the backward induction solution of a game [23], and similarly an extension of dynamic logic can be used to axiomatize backward induction [62]. Maybe surprisingly, even simple propositional logic can be used as a description language for extensive games of perfect information which also allows one to formulate axioms of rationality [24]. The role of language in economic decision making has been investigated in [112] using formal logical languages. The work in this thesis also falls into the non-epistemic category.
Note that the above classification does take a rather narrow view of logic. On a less restricted interpretation, all of the discussion in game theory which relates to the epistemic aspects of games falls into the scope of logic, and indeed the modal/epistemic logician recognizes quite a number of the technical notions discussed in the game theory literature [10]. On the other hand, we choose to define the use of logic in game theory more narrowly as the use of formal logical languages. Among other things, using formal languages allows one to ask a number of interesting meta-theoretic questions, examples of which shall be given in this thesis.

1.2 Linking Two Fields of Research

Below we discuss more concretely how logic and game theory come together in this thesis. Additionally, we comment on the general virtues of interdisciplinary research.

1.2.1 Logic in Computer Science

Computers do not always work the way they should (some people would use the more radical scoping: computers always do not work the way they should). Since there may be applications, however, where it is extremely important that computers do work as intended (think, e.g., of air traffic control and other safety-critical tasks), it would be useful to be able to prove that a program works as intended by the programmer, or at least that the program satisfies some crucial properties. Formal logics are used in various branches of computer science as tools for reasoning about programs and software systems more generally. We shall briefly discuss two of these branches below.

The theory of program correctness as it has been understood traditionally [82, 73, 91] is concerned with verifying the correctness of software written in imperative programming languages such as PASCAL, C, and so on. As an example, consider the following program \( \gamma_1 \) which is known as Euclid's algorithm:

\[
\begin{align*}
x, y : \text{Nat} \\
\text{while } x \neq y \text{ do} \\
\quad \text{if } x > y \text{ then } x := x - y \\
\quad \quad \text{else } y := y - x \\
\end{align*}
\]

The program is written in a commonly used pseudo-code which is almost self-explanatory: Two variables \( x \) and \( y \) with the value of natural numbers are compared. If they are equal, the program ends. Otherwise, if \( x \) is greater than \( y \), \( y \) is subtracted from \( x \), and if \( y \) is greater than \( x \), \( x \) is subtracted from \( y \). This test and action are repeated until \( x = y \).
It is by no means obvious that this program $\gamma_1$ in fact calculates the greatest common divisor (gcd) of two natural numbers $a$ and $b$. If initially $x = a$ and $y = b$, then the program will terminate with $x = y = \text{gcd}(a, b)$. Proving the correctness of this program thus would entail a proof of the following statement:

If $\gamma_1$ is started in a state where $x = a$ and $y = b$, then $\gamma_1$ will terminate and upon termination $x = y = \text{gcd}(a, b)$.

One natural way to represent programs semantically is by way of a state transformer, i.e., a relation $R_\gamma \subseteq S \times S$ over the set of states $S$. We interpret $sR_\gamma t$ as “starting at state $s$, there is an execution of program $\gamma$ which terminates in state $t$”. Our program $\gamma_1$ has the further property of being deterministic, for every state $s$ there is at most one state $t$ such that $sR_\gamma t$ holds. In general, we allow for programs being nondeterministic. In the case at hand, we can think of $S$ as the set of variable assignments for $x$ and $y$, that is $S = \{s | s : \{x, y\} \rightarrow \mathbb{N}\}$. Proving the above claim then would amount to showing that $sR_\gamma t$ holds if and only if $t(x) = t(y) = \text{gcd}(s(x), s(y))$.

A second way to think about programs is in terms of a predicate transformer $F_\gamma : \mathcal{P}(S) \rightarrow \mathcal{P}(S)$ which maps a set of states $X$ to the set of states $F_\gamma(X)$ at which some execution of program $\gamma$ will terminate in a state in $X$. (Usually one wants all executions to terminate in a state in $X$, but for our present purposes the existential reading is more natural, and for $\gamma_1$ the two coincide anyhow.) This approach to program semantics advocated by Dijkstra in [38, 39] is very well-suited for the kind of backward-reasoning one often does when studying the correctness of a program: If the program should bring about a particular state of affairs after termination, which initial condition has to be satisfied for the program to succeed in doing that? To give an example, assume that we want our program $\gamma_1$ to end in a state where the value of both $x$ and $y$ equals 3, then under what condition will the program succeed in doing that? Let $t$ be the state where $t(x) = t(y) = 3$. Then $F_\gamma(\{t\}) = \{s \in S | \text{gcd}(s(x), s(y)) = 3\}$, i.e., initially the greatest common divisor of $x$ and $y$ must be 3. A general proof of correctness would then amount to showing that if we start in a state where $x = a$ and $y = b$, the program is guaranteed to end in a state where both variables are set to the greatest common divisor of $a$ and $b$. Formally, for all states $t$ with $t(x) = t(y) = \text{gcd}(a, b)$ we have

$$\{s \in S | s(x) = a \text{ and } s(y) = b\} \subseteq F_\gamma(\{t\})$$.

While our exposition of predicate transformers was semantic, reasoning about the correctness of programs will usually be done syntactically using axioms and inference rules. Propositional Dynamic Logic [59, 77] can serve as a very simple formalism which links the syntactic and the semantic perspective on predicate transformers on the propositional level. The formula $\langle \gamma \rangle \varphi$ expresses that there is an execution of program $\gamma$ which terminates in a state satisfying $\varphi$. Thus,
the modality $\langle \gamma \rangle$ is the syntactic analogue of the predicate transformer $F_\gamma$. Logical axioms and rules then syntactically describe the behavior of the predicate transformers, for instance the inference rule

$$
\frac{\varphi \rightarrow \langle \gamma_1 \rangle \psi \quad \neg \varphi \rightarrow \langle \gamma_2 \rangle \psi}{\langle \text{if } \varphi \text{ then } \gamma_1 \text{ else } \gamma_2 \rangle \psi}
$$
says that if the truth of $\varphi$ implies a terminating $\gamma_1$-execution satisfying $\psi$ and the falsity of $\varphi$ implies a terminating $\gamma_2$-execution satisfying $\psi$, then there is a terminating execution of $\text{if } \varphi \text{ then } \gamma_1 \text{ else } \gamma_2$ which satisfies $\psi$.

The two main disadvantages of the traditional approach to program verification are its restricted domain of application and its infeasibility in practice. On the one hand, the kinds of systems which can be analyzed using Dijkstra's approach are programs which can be specified compositionally by means of a fixed number of programming constructs. Furthermore, when started in some initial state, these programs are supposed to terminate in some final state whose properties are then examined. This means that software systems which are intended to run without ever terminating (such as operating systems) as well as systems which have not arisen from a compositional specification cannot be analyzed and verified using a Dijkstra-style approach. On the other hand, even for systems which are amenable to this approach, real-life systems usually turn out to be too complicated to make automatic verification feasible, since such a verification entails theorem proving in a very complex logical formalism.

Because of these drawbacks, another approach to software verification has been investigated more recently which makes use of temporal logics [43, 70]. Here, programs are not part of the logical language but rather the semantic models over which expressions of temporal logic are evaluated. More specifically, process algebras [8, 49] and other tools can be used to describe and generate a process graph which serves as the semantic model of the software system. Temporal logics such as CTL can then express safety and liveness properties of the system which are verified via model checking. This approach allows for the verification of non-terminating systems, and verification is generally more feasible, since it is based on model checking rather than theorem proving. We shall have more to say about the relative merits of the two approaches in section 8.1.

### 1.2.2 Game Theory and Social Choice

Computer programs of the kind we discussed represent processes of a very special kind: They are formally specified and they do not contain any interaction. Social processes on the other hand are usually much harder to analyze because they are interactive by nature and because it is often extremely difficult to model them mathematically. The latter problem can be simplified by abstraction and by studying those social processes which are more regulated to begin with, examples
Chapter 1. Let the Games Begin

being elections, auctions, and the process of obtaining a doctorate degree. Social choice theory [75, 86] studies collective decision making: given the preferences of the individuals of some society, how should the society as a whole choose between the different options? To these considerations game theory [19, 93] adds a strategic component, asking, e.g., how such methods of collective decision making can be manipulated by individuals or groups of individuals.

To model the ability of a group of individuals N, social choice theory has developed the notion of an effectivity function [86, 1]. Given a set of alternatives S between which the individuals must choose, an effectivity function $E : \mathcal{P}(N) \rightarrow \mathcal{P}(\mathcal{P}(S))$ maps groups of individuals to a collection of subsets of alternatives. We interpret $X \in E(C)$ as “coalition C is effective for X”, or as “coalition C is able to achieve an outcome in X”. This interpretation is deliberately noncommittal about how exactly “effectivity” is to be interpreted. In this thesis, we will interpret it as having a strategy which is successful no matter what. Hence, $X \in E(C)$ holds iff coalition C has a joint strategy to achieve an outcome in X no matter what the other players do.

Consider the following example from [52]: Angelina has to decide whether she wants to marry Edwin, the (male) judge, or stay single. Edwin and the judge each can similarly decide whether they want to stay single or marry Angelina. If we assume that the three individuals live in a society where nobody can be forced to marry against his/her will, this situation can be modeled using effectivity functions as follows: The set of players is $N = \{a, e, j\}$ and the set of alternatives is $S = \{s_s, s_e, s_j\}$, where $s_s$ denotes the situation where Angelina remains single, $s_e$ where she marries Edwin, and $s_j$ where she marries the judge. Angelina (a) has the right to remain single, so $\{s_s\} \in E(\{a\})$, whereas Edwin can only guarantee that he does not marry Angelina; whether she marries the judge or remains single is not up to him. Consequently, we have $\{s_s, s_j\} \in E(\{e\})$ and there is no proper subset $X \in E(S)$ that $X \in E(\{e\})$. Analogously for the judge, we have $\{s_s, s_e\} \in E(\{j\})$. Angelina and Edwin together can achieve any situation except the one where Angelina marries the judge (since this alternative would require the judge’s consent), and hence $\{s_s\} \in E(\{a, e\})$. Again, the situation is similar for the judge: $\{s_s\} \cap \{s_j\} \in E(\{a, j\})$.

Another way to think about this marriage constellation is in terms of games. We already suggested that Angelina has three strategies, she can decide to remain single, or go for either the judge or Edwin. (Note that we ignore the temporal/dynamic character of more realistic strategies, where Angelina might propose to the judge and, if he does not accept within 3 days, go for Edwin.) The two men on the other hand only have two options each, either to decide to remain single or to go for marriage. The strategic game below pictures this situation: Angelina chooses one of the three tables, Edwin chooses the upper or lower row, and the judge chooses the left or right column. For each such strategy triple we have given the resulting alternative. As an example, consider the soap opera situation where Angelina wants to marry Edwin (middle table), Edwin wants to remain
single (upper row), but the judge wants to marry Angelina (right column). The result is that everyone remains single (alternative $s_s$).

\[
\begin{array}{ccc}
  s & m \\
  s_s & s_s & s_s & s_s & s_j & s_j \\
  s_e & s_e & s_e & s_e & s_e & s_e \\
  m & m & m & m \\
\end{array}
\]

Depending on the kind of social process under consideration, the effectivity function will satisfy certain properties. Note, e.g., that the effectivity function of the example is monotonic, i.e., if a group of individuals is effective for a set $X$ then it is also effective for any superset of $X$. Another important property which is satisfied is the property of superadditivity. If $X_1 \in E(C_1)$ and $X_2 \in E(C_2)$ then $X_1 \cap X_2 \in E(C_1 \cup C_2)$, provided $C_1 \cap C_2 = \emptyset$. Intuitively, if one group of players is able to achieve an outcome in $X_1$ and another disjoint group is able to achieve an outcome in $X_2$, then they can join their strategies to achieve an outcome in $X_1 \cap X_2$. This and other properties of effectivity functions have been studied in the literature to characterize certain classes of social processes, and we shall see examples of this in chapter 2.

In this thesis, we shall have nothing to say about the preferences which players might have over the set of alternatives. In other words, we only concern ourselves with what players can do, not with what they should or will do. If one does consider players’ preferences, various solution concepts can be investigated. For effectivity functions, a prominent solution concept is the core, the set of all undominated alternatives. An alternative $s$ is dominated by a set of alternatives $X$ if there is some coalition $C$ which is effective for $X$ and every member of $C$ prefers every alternative in $X$ to $s$. It is reasonable to assume that such an alternative $s$ will never be realized, since coalition $C$ will see to it that $X$ is realized instead. An effectivity function is called stable in case its core is nonempty no matter what preferences the players have. Conditions under which an effectivity function is stable have been investigated extensively, and it has been shown in [85] that the problem of checking stability of an effectivity function is NP-complete.

### 1.2.3 Dijkstra meets von Neumann

Programs can be viewed as 1-player games. If the program is deterministic like the $gcd$-algorithm above, the game is of a particularly boring sort since it does not involve any choice points where the player can choose between two or more different possible actions. Nondeterministic programs, however, do involve choices, as the program $\gamma_2$ below illustrates:

\[
\begin{aligned}
x,y &: \text{Nat} \\
\text{case } x \geq y & \rightarrow y := y + 1 \\
y \geq x & \rightarrow x := x + 1 \\
\end{aligned}
\]
In case $x > y$, 1 is added to $y$, and in case $y > x$, 1 is added to $x$, but in case $x = y$, a nondeterministic choice is made between the two increments. In case we have a state $s$ with $s(x) = s(y) = 0$, we thus have $sR_\gamma t_1$ and $sR_\gamma t_2$, where $t_1(x) = t_2(y) = 0$ and $t_1(y) = t_2(x) = 1$.

\[
\begin{align*}
x &= 0, y &= 0 \\
x &= 1, y &= 0 \\
x &= 0, y &= 1
\end{align*}
\]

In words, there is an execution of $\gamma_2$ which starts in a state with $x = y = 0$ and ends in a state with $x = 0$, but not all executions have that property. This program thus presents a more interesting 1-player game, in the sense that the player (which we usually assume to be Nature) can choose which transition to make in case $x = y$.

Predicate transformers can thus be given a game-theoretic interpretation: $s \in F_\gamma X$ holds in case at state $s$, Nature has a strategy in program/game $\gamma_2$ for achieving a state in $X$. Note that this interpretation is in no way dependent on the fact that $\gamma_2$ was a 1-player game, for the notion of “strategy” applies to games generally, independent of the number of players. This move from programs to games has also been carried out in the area of temporal logic [3], and we shall have more to say about this in section 4.4.

The two traditions outlined in the previous section apply a similar mathematical model to capture similar ideas. For an effectivity function $E$, $X \in E(C)$ provided that coalition $C$ has a joint strategy to achieve an outcome in $X$. For a predicate transformer $F$ and a state $s$, $s \in F(X)$ provided that Nature has a strategy to achieve an outcome in $X$. Given this similarity, it may not be surprising that one can discover analogies in the issues investigated in both traditions: The characterization result of social choice theory which isolates the effectivity functions which correspond to strategic games (theorem 2.27) is the multi-agent coalitional analogue of the computer science result which characterizes the predicate transformers which correspond to state transformers (theorem 2.16). Similarly, questions of mechanism design can be related to questions of program synthesis (see chapter 5).

The first difference between the notions of an effectivity function and a predicate transformer is that predicate transformers only capture the abilities of a single player (Nature) whereas effectivity functions model the abilities of many agents and even of groups of agents. The second difference is that effectivity functions are static while predicate transformers are dynamic. An effectivity function describes the social process as a simple one-shot event, the players’ abilities do not depend on the current state. Predicate transformers on the other hand link the player’s powers to the current state, and these powers may change as the state itself changes. We shall try to take the best from both traditions, the multi-agent perspective of effectivity functions and the dynamic approach of predicate
transformers. The result will be a new dynamic multi-agent model of power for which we will develop logical formalisms which can express properties of this type of model.

1.2.4 Praise of FoLLI

The question of why the two traditions mentioned should be brought together can also be viewed as an instance of the more general question regarding the motivation of interdisciplinary research. In the list below, we discuss some general qualities of interdisciplinary study, also referring to some specific results in this thesis which can serve as examples for the case of logic and game theory at hand.

(1.) Answering Old Questions: Probably the most immediate hope one might have is that an open question in one field can be solved using techniques or results from the other field. This hope is also verbalized when the game theorist asks the logician how the logician’s research on logic and games will contribute to solving game-theoretic problems. The logician in turn will be tempted to quote John F. Kennedy’s “ask not what your country can do for you, ask what you can do for your country.”

A concrete example of a result from social choice theory which has been used to solve an open problem in logic will be discussed in section 4.4: Using a characterization result (theorem 2.27) which extends a similar result from the social choice theory literature, we are able to obtain an axiomatization (theorem 3.14) which has been used in [56] to provide a complete axiomatization of Alternating Temporal Logic [3].

(2.) Raising New Questions: More than answering old questions, interdisciplinary research will generate new questions (and answers) in both fields involved. One of the two main topics of this thesis, comparing reasoning about individual ability to reasoning about coalitional ability in various kinds of social processes, seems to exemplify a new type of investigation which could be relevant to social choice theory. Logic has a number of tools and techniques which allow one to carry out such a comparison in terms of expressiveness, complexity, and so on.

An example of a new question for logic will be discussed in section 7.2.2: It turns out that Game Logic can be translated into the propositional modal $\mu$-calculus [76]. This translation has a particular syntactic property, for it uses at most two set variables. This raises the question whether there is a strict finite-variable hierarchy for the $\mu$-calculus, i.e., whether one can show that formulas with $n$ variables can express less than formulas with $n+1$ variables. For the alternation hierarchy, it has been shown that the hierarchy does not collapse [25, 81], whereas the finite-variable hierarchy apparently has not been investigated so far.

(3.) Unification: Obtaining a general framework which describes a wide variety of situations arising in different disciplines is desirable even if no old questions can be answered and no new questions arise. Unification creates links between phenomena and notions which had not been connected before, reducing the num-
ber of basic notions or axioms needed to understand (at least a small part of) the world, increasing order in the mind and creating a pleasant feeling in the stomach.

The coalition models introduced in chapter 2 are general enough to capture ability in extensive games with and without simultaneous moves as well as Kripke models. Interpreting the modality $\Diamond \varphi$ as the existence of a strategy in a game leads to a general coalition logic of which both normal and non-normal modal logics are particular instances (section 3.5).

4. Diversification: As the dual of unification, where, e.g., two notions are unified into one, diversification refers to the opposite process, providing new perspectives on old friends. While physics seems to strive for the theory of everything [57], finding such a theory may only be the beginning of discovering different “equivalent” theories of everything or different interpretations of the theory. In computer science, the various equivalent formalizations of the most central notion, computation, are a paradigm example (see, e.g., [72]).

In the field of logic, modal logic provides an example of a formal system which is open to a wealth of interpretations, and this diversity also testifies to the importance of modal logic. In section 3.5 we shall add yet another view to these multiple perspectives, characterizing normal modal logic as the logic of 1-player games (theorem 3.22) and monotonic non-normal modal logic as the logic of determined 2-player games (theorem 3.24).

1.3 Objectives

Linking the two fields of research discussed above is a natural project to consider, given that the conceptual and mathematical notions involved are very similar. More importantly, however, it will allow us to construct logics which can be used to reason about social software, and to investigate how reasoning about social software differs from reasoning about standard computer software.

1.3.1 Logics for Social Software

In [95], Parikh introduced the term “social software” to refer to the project of analyzing social procedures and processes using the formal methods of computer science. One of his example concerns the well-known problem of dividing a cake fairly among a number of people. For two people, the algorithm “I cut, you choose” is a well-known method to ensure that both people can guarantee themselves at least half of the cake according to their own perception. For more than 2 people, there are extensions of this cut-and-choose algorithm which can yield fair solutions for all the participants [27]. An algorithm which does offer everyone a strategy to guarantee himself a fair share of the cake can be considered
1.3. Objectives

correct, and Parikh calls for the development of logical tools to be able to prove the correctness and efficiency of such an algorithm.

A further example of a social process amenable to formal analysis are voting procedures as studied in social choice theory [84]. The number of alternatives, the order of voting and the different electoral bodies involved can all influence the final outcome. Proving the correctness of a voting procedure would mean to show that it satisfies certain desirable properties (e.g., that there is no dictator, i.e., no individual has the power to determine the outcome by himself), and establishing that it is efficient would entail a proof that there is no simpler procedure with the same properties.

The first thesis of this study is that the logics developed and studied here, Coalition Logic and Game Logic, provide the means to analyze social processes like the ones given. Chapter 5 will provide arguments for this claim by applying Coalition Logic to a number of examples. Since these examples lie well in the domain of game theory (and social choice theory), the question arises how the perspective of social software differs from the perspective of game theory. On the one hand, the social software approach strives for a formal axiomatic theory of social processes which is very explicit. As a consequence, arguments can be formulated in a logical language which lends itself to automated verification, the advantage being a greater degree of confidence in as well as the possible automation of game-theoretic argumentation. On the other hand, the social software perspective provides new theoretical questions and insights about social processes which broaden the scope of game theory. These questions can be grouped into two comparisons which we shall discuss subsequently.

1.3.2 Programs vs. Games

The first comparison we shall engage in relates programs to games. As we have seen, programs can be viewed as 1-player games, so the question arises in what respect reasoning about 1-player games differs from reasoning about 2-player games. Does the addition of a second player make an essential difference, and if so, in what respect? To use a marginally related example from a different domain, it appears that the addition of a second character complicates the automatic generation of humorous film sequences significantly [88].

One way in which we will investigate this question is using Game Logic. This logic uses the formula \langle \gamma \rangle \varphi to express that Player 1 has a strategy to achieve \varphi in game \gamma. The game \gamma here is an expression denoting a complex game which is constructed by means of a number of operations, like the programming operation if...then...else... we have seen earlier. As an example, consider \gamma_1 = (a \cup b); (c \cap d), denoting the game where first player 1 chooses between doing a or b and then player 2 chooses between doing c or d. To compare games to programs, we compare Game Logic with its program fragment, the formulas containing only those modalities \langle \gamma \rangle \varphi where \gamma is a 1-player game, i.e., where
player 1 makes all the choices. The game $\gamma_1$ is not in the program fragment, but, e.g., $\gamma_2 = (a \cup b); (c \cup d)$ is, where the second choice is also made by player 1. The program fragment can then be compared to full Game Logic in terms of expressive power (are there properties which cannot be expressed using only programs?), axiomatization (what are the basic principles of reasoning about games and programs?) and complexity (is reasoning about games more complex than reasoning about programs?).

Besides this syntactic component of the comparison between programs and games, there is also a semantic component having to do with the basic building blocks from which complex games like $\gamma_1$ are constructed. Two cases can be investigated: First, one can assume that all interaction is introduced through the game operations, and that the atomic games are simple 1-player games where the same player makes all the choices (chapter 7). Second, one can allow interaction in the atomic games as well, yielding a more general system which is studied first in chapter 6.

1.3.3 Individuals vs. Coalitions

Once we have moved from programs to games with 2 or more players, we can reason about what the different players can achieve in the game. But there is still another issue here, namely how reasoning about these individual agents may differ from reasoning about groups of agents. It may seem for instance that reasoning about what single agents can bring about in a complex multi-agent process should be simpler than reasoning about what groups of agents can bring about.

In chapters 3 and 4 we will look at this issue using the semantic models developed in chapter 2. The coalition models developed there will allow us to capture various kinds of multi-agent processes or games (including 1-player games), and for each of these games we will investigate the differences between reasoning about individuals and reasoning about coalitions. We will do this first using Coalition Logic (chapter 3), a logical language which contains expressions of the form $[C]\varphi$ where $C$ is a group of agents. The formula expresses that coalition $C$ is able to achieve $\varphi$ in one move, i.e., $C$ has a strategy for $\varphi$. To compare individuals to coalitions, we compare the full language of Coalition Logic with its individual fragment, the restricted language which can only talk about single-agent coalitions, i.e., which only contains formulas $[i]\varphi$ expressing that agent $i$ has a $\varphi$-strategy.

While basic Coalition Logic only contains modalities to talk about what groups of agents can achieve in one move of the game, chapter 4 introduces Extended Coalition Logic which can also express long-term ability. The formula $[C^*]\varphi$ expresses that coalition $C$ has a strategy to bring about $\varphi$ at some point in the future. As with basic Coalition Logic, one can compare individuals to coalitions in this richer language.
1.4 Overview

In section 1.2.1, we briefly discussed 2 approaches to program verification, an internal approach based on temporal logic and an external approach based on programming logics like PDL. Both of these approaches shall be lifted from programs to games, and hence this thesis naturally falls into two parts, chapters 3 to 5 which focus on the internal approach using the logical framework of Coalition Logic, and chapters 6 and 7 which deal with the external approach using the framework of Game Logic. Chapter 2 will introduce a semantic framework which unifies both the internal and the external approach. We shall now proceed to describe the content of the different chapters in some more detail.

Chapter 2 develops a very general dynamic ability model based on the notion of an effectivity function. This model serves to capture group ability in programs and extensive games with and without simultaneous moves among 2 or more players. Quite some time will be spent on characterizing the precise class of ability models which corresponds to each of these processes. We illustrate how these ability models are open to two different interpretations, either as internal descriptions of a single game or as external descriptions of a collection of games. We also consider models of individual ability, asking again which conditions on individual ability characterize ability in various classes of games, and in which situations individual ability completely determines group ability. Finally, we extend the notion of bisimulation to serve as an equivalence notion for the ability models defined.

Chapter 3 introduces Coalition Logic, a modal logic which contains formulas $[C]\varphi$ expressing that the group of agents $C$ is able to achieve in one move an outcome where $\varphi$ is true. The formulas are evaluated over the ability models defined in chapter 2 and interpreted as an internal description of a social process. To compare reasoning about individuals to reasoning about coalitions, we isolate the individual fragment of Coalition Logic and compare some of its properties (complexity and expressiveness) to those of full Coalition Logic. Finally, we show how normal and non-normal modal logics can be viewed as instances of this more general Coalition Logic.

Chapter 4 extends basic coalition logic with an additional modality $[C^*]\varphi$ asserting that coalition $C$ can achieve $\varphi$ at some point in the future. This additional modality opens the door to a variety of different applications of Coalition Logic in the analysis of social processes. We also consider two meta-theoretic questions concerning the complexity of Extended Coalition Logic and its expressive power. Related formalisms such as Alternating Temporal Logic and the multi-agent logics developed in distributed artificial intelligence are compared to (extended) Coalition Logic.

Chapter 5 applies Coalition Logic and Extended Coalition Logic to problems mainly from the area of social choice theory. Put succinctly, we show how Coalition Logic can be used to guarantee personal liberties, avoid excessive or obscure
legislative procedures and reduce telephone costs.

Chapter 6 changes from the internal to the external view of games. Ability models are viewed as a collection of interlinked games and the formal language of Game Logic is used to describe what players can bring about in complex games. Here, the modal expression $\langle \gamma \rangle \varphi$ states the existence of a strategy for player 1 in game $\gamma$ which achieves $\varphi$. We define a generalized version of the $\mu$-calculus of which Game Logic is a fragment. This perspective allows us to locate an interesting difference between programs and games which will lead to different complexity results for model checking. Finally, we also consider the complexity of the satisfiability problem.

Chapter 7 continues the study of Game Logic, now focusing on Game Logic when interpreted over the restricted class of Kripke models. Over these models, Game Logic can be compared to Propositional Dynamic Logic (PDL) on the one hand and the standard modal $\mu$-calculus on the other hand in terms of expressiveness and complexity. In particular the comparison with PDL will yield some interesting differences between programs and games, one of them being that games allow for two different kinds of iteration. Using bisimulation, we also obtain a characterization of the set of game operations needed to construct all first-order definable games. This result can then be compared to an analogous result obtained for programs.

Chapter 8 summarizes the main results of this thesis, addressing again the relationship between Game Logic and Coalition Logic, and also discussing the algebraic counterpart of Game Logic and its application in the development of social software.

Since chapters 4 and 6 make use of fixpoint constructions to define long-term ability and iteration, we have summarized some (mostly well-known) results concerning fixpoints in appendix A which will be appealed to in these chapters.