Logic for social software
Pauly, M.

Link to publication

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.
In this final chapter, we take another look at the relationship between Game Logic and Coalition Logic, showing that they embody two different approaches to reasoning about multi-agent systems. As shall be explained, the difference between these two approaches is in fact well-known in computer science. In section 8.2, we then take stock of the theoretical results obtained in this thesis, on the one hand regarding the relationship between programs and games and on the other hand regarding the differences between reasoning about individuals and reasoning about coalitions. Finally, section 8.3 addresses the question how far Game Logic and Coalition Logic go on the way to a logic of social software.

8.1 Bringing it All Together

As shown in chapter 2, both Coalition Logic and Game Logic make use of essentially the same underlying semantics, interpreted either as an internal model of a single game or as an external model of multiple games. In fact, the difference between the two logics and their uses is analogous to the difference between PDL and temporal logic (TL), as summarized by the following equation:

$$\frac{CL}{GL} = \frac{TL}{PDL}$$

As logics for reasoning about software, temporal logics such as CTL, CTL*, etc. differ in a number of ways from program logics such as PDL and its extensions (see e.g. [70] for a more detailed comparison): In the terminology of [77], PDL is an exogenous logic since programs are an explicit part of the logical language. In contrast, temporal logic is endogenous: The model itself is the fixed program over which expressions are interpreted. The difference between endogenous and exogenous logics thus corresponds to the difference between the internal and the external view of games, explained in section 2.4. This central difference has far
reaching technical consequences. First, program verification takes a different form in the two approaches: In the endogenous approach, program verification takes the form of model checking, in the exogenous approach it takes the form of theorem proving or satisfiability checking. Second, the endogenous approach can be used only to reason about systems with a finite number of states. This restriction comes from the use of model checking for verification: If there are infinitely many states, the denotation of a formula may not be computable. The exogenous approach on the other hand is not limited to such finite-state systems. Third and finally, the class of programs one can reason about in the exogenous approach is limited to compositional systems, where complex programs are constructed using a limited number of program constructions for which inductive proof rules exist. The endogenous approach on the other hand can also deal with non-compositional systems and is thus more general in this respect.

The differences between TL and PDL on the right software side of the equation equally apply on the left social software side between GL and CL. Naturally, there are some more differences on the social software side: In contrast to GL, CL can describe games with more than two players and it can express the effectivity of non-singleton coalitions, differences which have no analogue on the right software side of the equation. Still, it is easy to see how GL could be extended to games with more than 2 players, in fact [97] already contains such a proposal.

In terms of applications to social software, chapter 5 has illustrated some applications of Coalition Logic. The examples have illustrated not only verification (via model checking) but also synthesis (via satisfiability tests) of social software. As for applications of Game Logic in the verification of social software, Parikh in [97] provides an example of how Game Logic can be used to verify the fairness of a cake-cutting algorithm, suggesting that even propositional Game Logic can be useful to verify properties of simple multi-agent algorithms. Not being completely convinced by this application, however, we think that stronger logical frameworks such as the refinement calculus [7] (see also section 8.3) are needed to treat interesting examples. There is, however, another way in which Game Logic could indirectly turn out to be very useful for social software verification, namely as game algebra.

Game Algebra

The operations of Game Logic have also been studied from an algebraic perspective [55, 122]. Recall that a complex game expression $\gamma$ of Game Logic denotes a predicate transformer $E_\gamma : \mathcal{P}(S) \rightarrow \mathcal{P}(S)$. Hence it is natural to call two game expressions $\gamma_1$ and $\gamma_2$ equivalent provided that $E_{\gamma_1} = E_{\gamma_2}$ holds for all game models. Put differently, $\gamma_1$ and $\gamma_2$ are equivalent if $(\gamma_1)p \leftrightarrow (\gamma_2)p$ is valid for a $p$ which occurs neither in $\gamma_1$ nor in $\gamma_2$. When $\gamma_1$ and $\gamma_2$ are equivalent, we say that $\gamma_1 = \gamma_2$ is a valid game identity.

Basic game algebra studies the game operations of sequential composition,
8.1. Bringing it All Together

Choice (demonic and angelic) and duality. The test-operator is excluded since it would take us out of the purely algebraic framework; iteration on the other hand could be added but has not been investigated so far. The central problem of basic game algebra is to axiomatize the set of valid game identities. The conjectured axiomatization of [12] has been proved complete in [55], and an alternative algebraic proof has been given in [122]. Consider the following game identities (taken from [122]), where we write duality as —, angelic choice as ∨ and demonic choice as ∧:

\[ x ∨ x = x \quad x ∧ x = x \] (G1)

\[ x ∧ y = y ∨ x \] (G2)

\[ x ∨ (y ∨ z) = (x ∨ y) ∨ z \quad x ∧ (y ∧ z) = (x ∧ y) ∧ z \] (G3)

\[ x ∨ (x ∧ y) = x \quad x ∧ (x ∨ y) = x \] (G4)

\[ x ∨ (y ∧ z) = (x ∨ y) ∧ (x ∨ z) \quad x ∧ (y ∨ z) = (x ∧ y) ∨ (x ∧ z) \] (G5)

\[ x ∨ x = x \quad x ∧ y = y ∧ x \] (G6)

\[ x ∨ (y ∨ z) = (x ∨ y) ∨ z \quad x ∧ (y ∧ z) = (x ∧ y) ∧ (x ∧ z) \] (G7)

\[ -(x ∨ y) = -x ∧ -y \quad -(x ∧ y) = -x ∨ -y \] (G8)

\[ x; (y; z) = (x; y); z \quad (x ∨ y); z = x; z ∨ y; z \] (G9)

\[ -(x; y) = -x; -y \quad (x ∧ y); z = x; z ∧ y; z \] (G10)

\[ x; y ∨ x; (y ∨ z) = x; (y ∨ z) \] (G11)

Axioms G1-7 are well known axioms of boolean algebra, where angelic choice corresponds to disjunction or join, demonic choice to conjunction or meet, and dual to negation. Axiom G9 is a principle which is already present in process algebra [49, 8]: If a choice of player i between x and y is followed by game z in any case, then player i might as well choose between x; x and y; z directly. Note that the right-distributive law x; (y ∨ z) = x; y ∨ x; z on the other hand is not valid. In the first game, player 1 can postpone her choice until after game x has been played. She may have a winning strategy which depends on how x is played, and hence such a strategy will not necessarily be winning in the second game, where she has to choose before x is played. Axiom G11 may be easier to read as x; y ≤ x; (y ∨ z) or as the quasi-equation

\[ y ≤ z → x; y ≤ x; z, \]

where \( a ≤ b \) abbreviates \( a ∨ b = b \). The axiom states the right-monotonicity of sequential composition, based on the monotonicity of the predicate transformers \( E_γ \) which interpret the game terms.

Soundness of these axioms can easily be verified. Furthermore, one can show that any valid game identity can be derived from these axioms using equational logic. So far, this result has not been extended to a version of game algebra which includes iteration. As shown in [11], game algebra also provides an interesting perspective on the semantic evaluation games of first-order logic.

Moving from theory to practice, how can Game Algebra be useful for the analysis of social software? It can serve as the basis for an algebraic description
language for multi-agent systems. In the same way in which Process Algebra can be used as a language for describing concurrent systems, the language of Game Algebra can be used to describe 2-player games. The laws of Game Algebra can then be used to simplify or more generally transform the games described preserving semantic equivalence. Finally, Coalition Logic (or Alternating Temporal Logic) can then serve as a specification language, i.e., formulas of Coalition Logic can be used to specify properties which are verified by model checking in the model generated from the Game Algebra expression. Succinctly, we obtain the following equation relating Coalition Logic and Game Algebra (GA) to temporal logic and Process Algebra (PA):

\[ \frac{CL}{GA} = \frac{TL}{PA} \]

Needless to say, Game Algebra in its current form can only be the first step in the direction sketched, comparable to basic Process Algebra (BPA) which serves as a basis for a wealth of different extensions including concurrency and communication. Furthermore, the equation suggests that the semantic notion of equivalence employed in Game Algebra may actually not be the appropriate one. If a Game Algebra expression denotes a complex game whose properties are to be verified using Coalition Logic, equivalence defined in terms of simple overall effectivity will be too crude. Instead, generalized bisimulation as introduced in section 2.5 should be much more suitable. Investigating the axiomatic differences between Game Algebra as defined above and its relative which is based on bisimulation equivalence is an interesting open question for further research.

### 8.2 Summary of Theoretical Results

As we have seen, reasoning about programs can be compared to reasoning about games using endogenous as well as exogenous logics. In each case, the technical results obtained for games can be compared to the results we have for programs. Below we put these theoretical insights into a number of slogans and show how standard meta-theoretic results about axiomatization, complexity, bisimulation and expressiveness can yield insights not only about differences between programs and games but also about differences between reasoning about individuals vs. coalitions. Furthermore, we remark on the role which iteration plays in these results, and we point out the relevance of the results for the practice of social software design and analysis.

**The External View: Game Logic vs. Program Logic**

*Semantically, disjunctivity and induction distinguish programs from games.*
Semantically, any individual effectivity function \( E \) models ability in a determined 2-player game (corollary 2.13). If \( E \) is disjunctive, it models ability in a 1-player game (theorem 2.16), so disjunctivity makes all the difference between programs and determined 2-player games. As a consequence of disjunctivity, the induction axiom is valid for programs but not for games (theorems 7.9 and 7.10).

\[
\text{Syntactically, program operations + duality = game operations.}
\]

Comparing the program operations of Propositional Dynamic Logic to the game operations of Game Logic, duality is the only difference. Theorem 7.22 suggests that this is no coincidence: The operations of test, sequential composition and choice suffice to obtain all first-order definable programs. Adding duality to these operations yields all first-order definable games.

\[
\text{In the end, games have more expressive power than programs.}
\]

Using duality, Game Logic can express properties of models which cannot be expressed in Propositional Dynamic Logic (theorem 7.2). The expressive difference relies on the presence of iteration which allows one to express the existence of a strategy to achieve something in the long run. Without iteration, duality does not increase expressive power, and hence programs are as expressive as games.

\[
\text{Verifying properties of programs is easier than verifying properties of games.}
\]

At the time of writing, model checking is more complex for games than for programs (theorem 6.22). This claim is only tentative since better polynomial-time algorithms may be found for \( \mu \)-calculus model checking. Hence one should say more accurately that verifying properties of games is equally complex as verifying properties of programs if (and only if [18]) model checking for the \( \mu \)-calculus can be done in polynomial time.

The Internal View: Coalition Logic vs. Temporal Logic

The comparison between Temporal Logic and Coalition Logic is unfortunately more difficult than the comparison between PDL and GL. The reason is that there is a bunch of systems to choose from: On the one hand we have basic Coalition Logic and Extended Coalition Logic, on the other hand we have simple modal/temporal logic, CTL, CTL*, and so on. Furthermore, the models over which these logics are interpreted differ, and interpreting Coalition Logic over Kripke models for purposes of comparison will rob it of all its characteristic features (in contrast to Game Logic).

On the most basic level, we would want to compare basic Coalition Logic with normal modal logic. The complexity of model checking is linear time for
both modal logic and Coalition Logic, although the model size will be different. We saw that the complexity of the satisfiability problem for Coalition Logic is usually PSPACE-complete, as for modal logic. Similarly, there are cases where the complexity of the satisfiability problem turns out to be NP-complete, for strong normal modal logics such as S5 as well as for formulas of the individual fragment of Coalition Logic, when interpreted over extensive games with simultaneous moves. As remarked in section 3.8.2, these complexity results do give rise to an interesting difference between programs and games. If it turns out that NP $\neq$ PSPACE (which we shall assume at least for these slogans):

*Game synthesis is easier than program synthesis, provided we allow for simultaneous moves.*

That is, given an individual specification for a single player, formulated in basic Coalition Logic, finding a satisfying extensive game with simultaneous moves is an NP-complete problem whereas finding a satisfying program/process or a satisfying extensive game without simultaneous moves is PSPACE-complete.

Given the close relationship between Extended Coalition Logic and Alternating Temporal Logic (ATL), one can also compare ATL with its standard temporal counterpart CTL to get a better idea of how closed systems (programs) differ from open systems (games). For the complexity of model checking, such a comparison has been carried out in [3].

Instead of comparing Coalition Logic to temporal or modal logic, below we will compare Coalition Logic to its individual fragment. Putting our results in this light will yield some interesting differences between coalitional and individual reasoning.

*Game synthesis is easier for individual than for coalitional specifications, provided we allow for simultaneous moves.*

For extensive games with simultaneous moves, the satisfiability problem for Coalition Logic is PSPACE-complete (theorems 3.27 and 3.29) while the satisfiability problem of its individual fragment is NP-complete (theorem 3.36). Generating such a game from a specification formulated in the individual fragment of basic Coalition Logic is thus simpler than generating it from a coalitional specification, provided that NP $\subsetneq$ PSPACE. In other words, NP = PSPACE if and only if multi-agent synthesis for individual specifications is equally complex as for coalitional specifications.

*Game synthesis is easier for extensive games with than for extensive games without simultaneous moves, given individual specifications.*

For individual specifications, the satisfiability problem is NP-complete (theorem 3.36) over extensive games with simultaneous moves but PSPACE-complete (theorem 3.37) over extensive games without simultaneous moves. So again, NP
8.2. Summary of Theoretical Results

= PSPACE if and only if multi-agent synthesis for individual specifications is equally complex in extensive games with and without simultaneous moves. Assuming NP ≠ PSPACE, we can also read this result as demonstrating a difference between environments of perfect and imperfect information: It is simpler to generate multi-agent environments satisfying certain specifications in case we can hide information from the agents. Developing an environment which gives agents certain powers is easier if we have the means to prevent agents from being perfectly informed about the others' actions.

In the end, coalitions have more expressive power than individuals.

For extensive games with simultaneous moves, a gain in expressiveness can already be observed on the local level (theorem 3.35): a language which can express local coalitional ability is more expressive than a language which can only express local individual ability. For extensive games without simultaneous moves, coalitions only add expressiveness in the end, i.e., when the language is enriched to express what can be achieved at some point in the future (theorem 4.12).

Modal logics are game logics.

Normal modal logics describe 1-player games (i.e., Kripke models), in particular the basic normal modal logic K coincides with basic 1-player Coalition Logic over weakly playable coalition models (theorem 3.22). Non-normal monotonic modal logics describe determined 2-player games (i.e., neighborhood models), the basic monotonic modal logic M coincides with the individual fragment of basic 2-player Coalition Logic over weakly playable maximal coalition models (theorem 3.24). While normal modal logic is coalitional (it can express the ability of the empty coalition), non-normal modal logic is not (it can only express the ability of the two players individually), thus providing one explanation of the complexity difference in the satisfiability problem (see above).

The Role of Iteration

Both on the internal and on the external view, increased expressive power can be seen to depend on the presence of some form of iteration. In the case of extensive games without simultaneous moves, coalitions only add expressive power when long-term ability can be expressed. On the other hand, game operations only lead to more expressiveness in case iteration is present.

Iteration is also responsible for the observed complexity differences. But while on the internal view a difference between individuals and coalitions can only be observed with the satisfiability problem, on the external view the difference between programs and games emerges in model checking only: In case the atomic games are actually programs, the satisfiability problems for Game Logic and for Propositional Dynamic Logic are both EXPTIME-complete (theorem 7.14);
in case no restriction is placed on the atomic games, we only know that the satisfiability problem for Game Logic is in EXPTIME (theorem 6.24), but we conjecture that EXPTIME-hardness also holds, for games as well as programs.

**Implications for Social Software**

Besides being interesting on a theoretical level, the technical results obtained are all of practical importance for the development and analysis of social software. This is most easily seen for the axiomatization and complexity results: Based on a complete axiomatization, (semi-)automatic theorem provers can verify properties of social procedures or deduce that a particular specification of a voting procedure is inconsistent. In order to verify properties of social software or synthesize social procedures which meet certain specifications, we also need algorithms for model checking and satisfiability testing. Furthermore, it is important to know how complex these algorithms are. Similarly, studying the expressiveness of the logics involved will allow us to pick the right logical language for the task at hand. If a specification can be formulated in the individual fragment of basic Coalition Logic, we can generate an implementation much more efficiently than if a specification expresses something about the long-term ability of groups of agents, something for which full Extended Coalition Logic is needed. Bisimulation-invariance and -safety results become important once we have generated, e.g., a voting procedure and want to simplify it in various ways. As long as this simplification results in a bisimilar voting procedure, we are guaranteed that its properties will not change, provided these properties are expressible in Coalition Logic.

Note that the kinds of results established in this thesis are certainly not the only ones which could be of relevance for the formal study of social software. To give one example, analogous to the case of temporal logic, it would be useful to have preservation theorems which link a class of model transformations (such as adding/deleting states, etc.) to the class of CL-formulas whose truth values remain unchanged by these transformations.

**8.3 The Future of Social Software**

As discussed in the previous sections, Game Logic and Coalition Logic represent two very different approaches to reasoning about social software, one being exogenous and the other endogenous. In giving examples of applications, we have focused on Coalition Logic because we think that in contrast to Game Logic, Coalition Logic can be applied to the analysis of social software as it is, analogous to how Temporal Logic is applied to the analysis of, e.g., concurrent systems. The insufficiency of Game Logic does not mean, however, that it cannot be extended to a logical framework for reasoning about multi-agent algorithms. In fact, the refinement calculus can be viewed as one such extension.
The aim of the refinement calculus [7] is to analyze “real” programs like the \textit{gcd}-program discussed in the introduction. For this aim, a simple propositional logic like Game Logic is not sufficient for formalization; instead, higher-order logic is used in [7]. What makes the refinement calculus relevant to the study of programs and games is that it also provides a general framework which can be used to reason about programs as well as games. This is not so surprising after all since the basic semantic notion of the refinement calculus is the predicate transformer, and we have seen that predicate transformers can model programs as well as determined 2-player games. The extended programming language of the refinement calculus is very similar to the language of Game Logic. While the operation of duality is also discussed, its role is less central in the refinement calculus. Preference is given to using demonic and angelic versions of the basic programming operations such as nondeterministic choice. In this enriched programming language, programs are viewed as \textit{contracts} which define the rights and obligations of the parties involved. Contracts are essentially what we have been calling games all along. The advantage of the contract metaphor is that it allows for a natural interpretation of empty games: under the contract interpretation, this is simply the case where one of the agents has breached the contract. 2-player zero-sum games such as Nim can be programmed using the language of the refinement calculus. The result is a game expression $\gamma$ such that (in Game Logic terms) $\langle \gamma \rangle \bot$ holds precisely when Angel has a winning strategy in the game. Besides formally proving the existence of such a winning strategy, the refinement calculus can also be used to extract a concrete winning strategy: Starting with the original game $\gamma$, the choices of angel are restricted step by step until all the choices left are demonic in the final program $\gamma'$. If we ensure that each step in this refinement is semantically sound, we can guarantee that $\gamma'$ is indeed a winning strategy for $\gamma$.

Returning to the topic of social software, how far do Coalition Logic, Game Algebra and (extensions of) Game Logic go in providing formal tools for the analysis and synthesis of social software? First, which kinds of examples of social software can we handle? As argued in chapter 5, the main requirement is that the social process to be analyzed is itself well-defined, allowing us e.g. to identify the set of agents involved in the process and the relevant properties of the states of the process.

Many of the social software examples presented were essentially voting problems since these problems meet the requirement of well-definedness. More generally, there is a practical argument for focusing on social software from the domain of social choice theory (broadly conceived), namely that a lot of research has been done in social choice theory which can be usefully applied, so that social choice theory and logic can meet half way. An example is the rich temporal logic of rights developed in [63, 64] which distinguishes alethic from deontic possibility. The language of this Deontic Logic of Action contains formulas of the form CanDo$_i(t_k, \varphi)$ (agent $i$ can act at time $t_k$ so as to bring about $\varphi$), MayDo$_i(t_k, \varphi)$ (agent $i$ has
the right at time $t_k$ to act in a way which brings about $\varphi$) and others. Preferences are also added to the model so that various paradoxes involving liberalism and constitutional decision making can be discussed and formalized.

Having suggested problems of social choice theory as the kind of social software to which the logics presented can usefully be applied, what kinds of questions do these logics allow us to address? Chapter 5 has given examples of verification and synthesis of social procedures. Furthermore, the example of telephone democracy even suggests that the efficiency of such procedures can be analyzed using basic Coalition Logic. Note, however, that the logics discussed cannot express anything about the preferences of the agents involved, nor about how their actions will be influenced by their preferences. For this reason, we cannot capture any strategic considerations, e.g., in the telephone democracy example. In spite of this limitation, the examples have shown that an interesting analysis of social software can be done even without considering the agents’ preferences. Before asking what people want to do in a social process, we should make sure that the process gives them the rights and duties they should have.

Comparing social software to computer software in terms of its complexity, one might be tempted to think that social processes must be far more complex than computational processes. On the other hand, we conjecture that even the most complex voting system used in any human society will be much less complex than the operating system used on most computers. Consequently, problems which may be intractable for computer software may well turn out to be tractable for social software. As shown in chapter 5, in the case of voting procedures the synthesis of social software reduces to satisfiability testing in basic Coalition Logic. The results of chapter 3 show that this problem is certainly feasible, since it is no harder than theorem proving in standard modal logic.

To conclude, we hope to have convinced the reader that one can treat social processes as social software by developing logics as analytical tools as is done in computer science. Exogenous program logics and in particular endogenous temporal logics can both be extended to yield logics for reasoning about social processes. We do not think that we will arrive at one general framework which is adequate to analyze all or even most social processes. Rather, we expect to see a variety of logics developed for different purposes, varying in complexity and expressive power like the different logics used in computer science. And if the theoretical results obtained and the examples provided still do not manage to convince the reader, we will have to close with a quote by Vince Lombardi, an American football coach:

*We didn’t lose the game; we just ran out of time.*