Information and Computation **156**, 25–45 (2000)
doi:10.1006/inco.1999.2820, available online at http://www.idealibrary.com on IDEAL®

Contracts, Games, and Refinement

Ralph-Johan Back[1] and Joakim von Wright

Department of Computer Science, Abo Akademi University, Lemminkainenkatu 14, Turku SF-20520, Finland
E-mail: backrj@abo.fi

We consider the notion of a contract that governs the behavior of a collection of agents. In particular, we study the question of whether a group among these agents can achieve a given goal by following the contract. We show that this can be reduced to studying the existence of winning strategies in a two-person game. A notion of correctness and refinement is introduced for contracts and contracts are shown to form a lattice and a monoid with respect to the refinement ordering. We define a weakest precondition semantics for contracts that permits us to compute the initial states from which a group of agents has a winning strategy to reach their goal. This semantics generalizes the traditional predicate transformer semantics for program statements to contracts and games. Ordinary programs and interactive programs are special kinds of contracts. © 2000 Academic Press

1. INTRODUCTION

A computation can generally be seen as involving a number of agents (programs, modules, systems, users, etc.) who carry out actions according to a document (specification, program) that has been laid out in advance. When reasoning about a computation, we can view this document as a contract between the agents involved. In this paper we show how contracts can be used as the starting point for a theory of program refinement. We describe a notation for contracts and give them a formal meaning using an operational semantics.

The *refinement calculus* [2, 4, 11] has traditionally based its reasoning on a weakest precondition semantics [7] for program statements. This semantics is based on the notion of total correctness; refinement means preservation of all total correctness properties. We show that the weakest precondition semantics agrees with the intuition of contracts. Furthermore, we show how the operational semantics and the weakest precondition semantics are related through the notion of winning strategies for games.

We use *simply typed higher-order logic* as the logical framework in the paper. The type of functions from a type Σ to a type Γ is denoted by $\Sigma \to \Gamma$ and functions can

[1] Corresponding author.

have arguments and results of function type. Functions can be described using λ-abstraction and we write $f.x$ for the application of function f to argument x.

2. STATES AND STATE CHANGES

We assume that the world that contracts talk about is described as a *state* σ. The *state space* Σ is the set (type) of all possible states. An agent changes the state by applying a function f to the present state, yielding a new state $f.\sigma$. We think of the state as having a number of *attributes* $x_1, \ldots, x_n$, each of which can be observed and changed independently of the others. Such attributes are usually called *program variables*. An attribute x of type Γ is really a pair of two functions, the *value function valx*: $\Sigma \to \Gamma$ and the *update function setx*: $\Gamma \to \Sigma \to \Sigma$. The function $valx$ returns the value of the attribute x in a given state, while the function $setx$ returns a new state where x has a specific value, while the values of all other attributes are unchanged. Given a state σ, $valx.\sigma$ is thus the value of x in this state, while $\sigma' = setx.\gamma.\sigma$ is the new state that we get by setting the value of x to γ.

An *expression* like $x + y$ is a function on states described by $(x + y).\sigma = valx.\sigma + valy.\sigma$. We use expressions to observe properties of the state. They are also used in *assignments* like $x := x + y$. This assignment denotes a state changing function that updates the value of x to the value of the expression $x + y$. Thus,

$$(x := x + y).\sigma = setx.(valx.\sigma + valy.\sigma).\sigma$$

A function $f: \Sigma \to \Sigma$ that maps states to states is called a *state transformer*. We also make use of predicates and relations over states. A *state predicate* is a boolean function $p: \Sigma \to \mathsf{Bool}$ on the state (we use set notation for predicates, writing $\sigma \in p$ for $p.\sigma$). Predicates are ordered by inclusion, which is the pointwise extension of implication on the booleans.

A *boolean expression* is an expression that ranges over truth values. It gives us a convenient way of describing predicates. For instance, $x \leqslant y$ is a boolean expression that has value $valx.\sigma \leqslant valy.\sigma$ in a given state σ.

A *state relation* $R: \Sigma \to \Sigma \to \mathsf{Bool}$ relates a state σ to a state σ' whenever $R.\sigma.\sigma'$ holds. Relations are ordered by pointwise extension from predicates. Thus, $R \subseteq R'$ holds if $R.\sigma \subseteq R'.\sigma$ for all states σ.

We permit a generalized assignment notation for relations. For example, $(x := x' \mid x' > x + y)$ relates state σ to state σ' if the value of x in σ' is greater than the sum of the values of x and y in σ and all other attributes are unchanged. More precisely, we have that

$$(x := x' \mid x' > x + y).\sigma.\sigma' \equiv (\exists x' \cdot \sigma' = setx.x'.\sigma \wedge x' > valx.\sigma + valy.\sigma).$$

This notation generalizes the ordinary assignment; we have that $\sigma' = (x := e).\sigma$ iff $(x := x' \mid x' = e).\sigma.\sigma'$.

3. CONTRACTS

Consider a collection of *agents*, each with the capability to change the state by choosing between different *actions*. The behavior of agents is regulated by *contracts*.

We describe contracts using a notation for *contract statements*. The syntax for these is as follows, where p stands for a state predicate and f stands for a state transformer, both expressed using higher order logic:

$$S ::= \langle f \rangle \mid \{p\} \mid S_1; S_2 \mid S_1 \sqcup S_2.$$

Intuitively, an agent carries out a contract statement as follows: The *update* $\langle f \rangle$ changes the state according to the state transformer f. If the initial state is σ_0 then the agent must produce a final state $f.\sigma_0$. An *assignment statement* is a special kind of update where the state transformer is an assignment. For example, the assignment statement $\langle x := x + y \rangle$ (or just $x := x + y$—from now on we will drop the angle brackets from assignment statements) requires the agent to set the value of attribute x to the sum of the values of attributes x and y.

In the *sequential action* $S_1; S_2$ the action S_1 is first carried out, followed by S_2. A *choice* $S_1 \sqcup S_2$ allows the agent to choose between carrying out S_1 or S_2. To simplify notation, we assume that sequential composition binds stronger than choice in contracts.

The *assertion* $\{p\}$ is a requirement that the agent must satisfy in a given state. For instance, $\{x + y = 0\}$ expresses that the sum of (the values of attributes) x and y in the state must be zero. If the assertion holds at the indicated place when the agent carries out the contract, then the state is unchanged, and the agent carries on with the rest of the contract. If, on the other hand, the assertion does not hold, then the agent has *breached* the contract.

The assertion $\{\mathsf{true}\}$ is always satisfied, so adding this assertion anywhere in a contract has no effect. Dually, $\{\mathsf{false}\}$ is an impossible assertion; it is never satisfied and always results in the agent breaching the contract.

3.1. *Multiple agents*

Above we assumed that there is only one agent bound by a contract. In general, there will be a number of agents that are acting together to change the world and whose behavior is bound by contracts. We then indicate explicitly which agent is responsible for the different choices and assertions. To illustrate this, assume that S is the following contract, binding agent a,

$$S = x := 0; (T \sqcup_a x := x + 1); \{y = x\}_a,$$

where T is the contract

$$T = y := 1 \sqcup_b y := 2$$

which binds b. Combining these two contracts we get

$$S = x := 0; ((y := 1 \sqcup_b y := 2) \sqcup_a x := x + 1); \{y = x\}_a.$$

The effect of the update is independent of which agent carries it out, so we allow this information to be lost when writing contract statements.

3.2. *Operational Semantics*

We can give a formal meaning to contract statements in the form of a *structured operational semantics* [13]. This semantics describes step by step how a contract is carried out, starting from a given initial state.

The rules of the operational semantics are given in terms of a transition relation between configurations. A *configuration* is a pair (S, σ), where

- S is either an ordinary contract statement or the empty statement symbol Λ, and
- σ is either an ordinary state, or the symbol $\perp_a$ (denoting that agent a has breached the contract).

The transition relation $\rightarrow$ (which shows what moves are permitted) is inductively defined by a collection of axioms and inference rules. It is the smallest relation which satisfies the following axioms (in the axioms, we assume that σ stands for a proper state while γ stands for either a state or the symbol $\perp_x$ for some agent x):

- *Update*,

$$\frac{}{(\langle f \rangle, \sigma) \rightarrow (\Lambda, f.\sigma)}, \qquad \frac{}{(\langle f \rangle, \perp_a) \rightarrow (\Lambda, \perp_a)}.$$

- *Assertion*,

$$\frac{\sigma \in p}{(\{p\}_a, \sigma) \rightarrow_a (\Lambda, \sigma)}, \qquad \frac{\sigma \notin p}{(\{p\}_a, \sigma) \rightarrow_a (\Lambda, \perp_a)}, \quad \frac{}{(\{p\}_a, \perp_b) \rightarrow_a (\Lambda, \perp_b)}.$$

- *Sequential composition*,

$$\frac{(S_1, \gamma) \rightarrow (S_1', \gamma'),\ S_1' \neq \Lambda}{(S_1; S_2, \gamma) \rightarrow (S_1'; S_2, \gamma')}, \qquad \frac{(S_1, \gamma) \rightarrow (\Lambda, \gamma')}{(S_1; S_2, \gamma) \rightarrow (S_2, \gamma')}.$$

- *Choice*,

$$\frac{}{(S_1 \sqcup_a S_2, \gamma) \rightarrow_a (S_1, \gamma)}, \qquad \frac{}{(S_1 \sqcup_a S_2, \gamma) \rightarrow_a (S_2, \gamma)}.$$

We have labeled the transition relation with the agent responsible for carrying out the move. This labeling is really redundant since it can always be recovered from the configuration in question.

A *scenario* for the contract S in initial state σ is a sequence of configurations,

$$C_0 \to C_1 \to C_2 \to \cdots,$$

where

1. $C_0 = (S, \sigma)$,
2. each transition $C_i \to C_{i+1}$ is permitted by the axiomatization above, and
3. if the sequence is finite with last configuration C_n, then $C_n = (\Lambda, \gamma)$, for some γ.

Intuitively, a scenario shows us, step by step, what choices the different agents have made and how the state is changed when the contract is being carried out. A finite scenario cannot be extended, since no transitions are possible from an empty configuration. As a matter of fact, it is easy to show that contract statements as we have defined them above permit only finite scenarios.

3.3. *Generalized Choice*

If the language of contracts is to be really useful, we need to be able to describe potentially infinite computation with contracts. We do this by extending the language of contracts with generalized choice, relational updates, and recursion. We describe these constructs briefly below and use them in examples.

The binary choice is generalized to permit *generalized choices.* We write $\bigsqcup_a \{S_i \mid i \in I\}$ for the choice that agent a has to do between the elements of a set of contract statements $\{S_i \mid i \in I\}$. Note that this set could be infinite (or empty). If the index set I is empty, then the agent has no alternative to choose and will have to breach the contract.

This syntax assumes that we are given a way of constructing the contract statement S_i for each i in the set I. For example, if we are given a recursive definition of a contract S_n for each natural number n, then $\bigsqcup_a \{S_n \mid n \in \mathsf{Nat}\}$ means that agent a chooses which of the contracts S_n is to be followed.

The operational semantics for generalized choice is a direct generalization of the semantics for binary choice:

$$\frac{i \in I}{(\bigsqcup_a \{i \in I \bullet S_i\}, \gamma) \to_a (S_i, \gamma)}.$$

We also introduce special notation for the situation where an agent has a choice between different ways of directly changing the state, given in terms of a state relation. The *relational update* $\{R\}_a$ is a contract statement that permits an agent to choose between all final states related by state relation R to the initial state (if no such final state exists, then the agent has breached the contract). For example, the contract statement

$$\{x := x' \mid x < x'\}_a$$

is carried out by agent a by changing the state so that the value of x becomes larger than the current value without changing the values of any other attributes. The operational semantics for the relational update is

$$\frac{R.\sigma.\sigma'}{(\{R\}_a, \sigma) \to_a (\Lambda, \sigma')}, \qquad \frac{R.\sigma = \varnothing}{(\{R\}_a, \sigma) \to_a (\Lambda, \bot_a)}, \qquad \frac{}{(\{R\}_a, \bot_b) \to_a (\Lambda, \bot_b)}.$$

As an example, consider a contract involving four agents a, b, c, and d. Assume that a is a user of a program, whereas b is the main module, and c and d are sub-modules of the program. Agent a chooses some input which must be between 0 and 100. Then b chooses whether to pass on the value to c (which is permitted if the value is below 50) or d (which is always permitted). For instance, we can think that the first alternative gives a more efficient computation, but is only available for small values of x. This is described by the contract statement

$$\{x := x' \mid 0 \leqslant x' \leqslant 100\}_a; (\{x < 50\}_b; S_1 \sqcup_b S_2)$$

where we do not show the details of S_1 (where agent c makes choices) and S_2 (where agent d makes choices).

The relational update $\{R\}_a$ could also be defined in terms of the generalized choice in the following way. Assume that a state relation $R: \Sigma \to \Sigma \to \mathsf{Bool}$ is given. For each state σ in Σ we define a contract S_σ as

$$S_\sigma = \bigsqcup_a \{\langle \lambda\sigma_0 \bullet \sigma' \rangle \mid R.\sigma.\sigma'\};$$

i.e., S_σ changes any initial state σ_0 to some state σ' such that $R.\sigma.\sigma'$ holds. Then we can define

$$\{R\}_a = \bigsqcup \{\{\lambda\sigma_0 \bullet \sigma_0 = \sigma\}; S_\sigma \mid \sigma \in \Sigma\}$$

This corresponds to the intuition that agent a chooses between all state changes to new states σ' such that $R.\sigma_0.\sigma'$ holds, where σ_0 is the current state.

3.4. Recursion

We can make the language of contracts more interesting from a programming point of view by also permitting *recursive contract statements*, which introduce the possibility of infinite scenarios. A recursive contract is defined using an equation of the form

$$X =_a S,$$

where S may contain occurrences of the contract variable X. With this definition, the contract X is intuitively interpreted as the contract statement S, but with each occurrence of statement variable X in S treated as a recursive invocation of the whole contract S. We also permit the syntax $(\mathsf{rec}_a\, X \cdot S)$ for the contract X defined by the equation $X =_a S$.

The mention of a specific agent in the recursive contract requires some explanation. We can think of the unfolding of X (i.e., the replacement of X by S) as carried out by the agent a. Furthermore, we think of this agent as being responsible for termination—in an infinite scenario a is considered to have breached the contract. The operational semantics for recursion is

$$\frac{X =_a S}{(X, \gamma) \to_a (S, \gamma)}.$$

4. USING CONTRACTS

Programs can be seen as special cases of contracts, where two agents are involved, the *user* and the *computer system*. In simple batch-oriented programs, choices are only made by the computer system, which resolves any internal choices (nondeterminism) in a manner that is unknown to the user of the system.

Our notation for contracts already includes assignment statements and sequential composition. The *abort* statement of Dijkstra's guarded commands language can be expressed as $\mathsf{abort} = \{\mathsf{false}\}_{\text{user}}$. If executed, it signifies that there has been a breach of contract by the user, releasing the computing system from any obligations to carry out the rest of the contract. We can also introduce the contract skip which leaves the state unchanged: $\mathsf{skip} = \langle \mathsf{id} \rangle$, where id is the identity function.

We can easily extend the simple language of contracts to include other program constructs, such as conditionals and iteration. A conditional statement like

$$\mathsf{if}\ x \geqslant 0\ \mathsf{then}\ x := x + 1\ \mathsf{else}\ x := x + 2\ \mathsf{fi}$$

is a *conditional contract*. We define it in terms of previous constructs, as equal to

$$\{x \geqslant 0\}_{\text{system}};\, x := x + 1 \sqcup_{\text{system}} \{x < 0\}_{\text{system}};\, x := x + 2.$$

The computer can here choose between two options. As described below, we will in general assume that an agent does not want to breach a contract (but does not mind being released from a contract). The agent will, therefore, always choose the alternative for which the guarding assertion is true; choosing the other alternative would breach the contract.

Iteration (the while loop) is defined in terms of recursion:

$$\mathsf{while}\ g\ \mathsf{do}\ S\ \mathsf{od} = (\mathsf{rec}_{\text{user}}\, X \cdot \mathsf{if}\ g\ \mathsf{then}\ S;\ X\ \mathsf{else}\ \mathsf{skip}\ \mathsf{fi}).$$

This interpretation means that we consider nontermination of the loop as an error which the user should try to avoid.

4.1. User Interaction

The program constructs above do not model user interaction during execution. Once started, execution proceeds to the end if possible, or it fails because the contract is breached (which allows the computer system to do anything, including going into an infinite loop).

The contract statements allows us to model interaction between a user and a computer, by permitting the user to also make choices. The user is an agent who chooses between alternatives in order to influence the computation in the manner she requires. The computer system can also make choices based on some internal decision mechanism, which is unknown to the user, who thus cannot predict the outcome.

As an example, consider the contract

$$x := 0; (x := x + 1 \sqcup_a x := x + 2); (x := x - 1 \sqcup_b x := x - 2).$$

After initialization, the user (a) chooses to increase the value of x by either one or two. After this, the system (b) decides to decrease x by either one or two. The choice of the user depends on what she wants to achieve. If, e.g., she is determined that x should not become negative, she should choose the second alternative. If, again, she is determined that x should not become positive, she should choose the first alternative. We can imagine this user interaction as a *menu choice* that is presented to the user after the initialization, where the user is requested to choose one of the two alternatives.

We could also consider b to be the user and a to be the computing system. In this case, the system starts by either setting x to one or two. The user can then inspect the new value of x and choose to reduce it by either 1 or 2, depending on what she tries to achieve.

4.2. Input Statements and Specifications

A more general way for the user to influence the computation is to give input to the program during its execution. This can be achieved by a relational assignment. The contract

$$\{x, e := x', e' \mid x' \geqslant 0 \wedge e' > 0\}_{\text{user}}; \{x := x' \mid -e < x'^2 - x < e\}_{\text{system}}$$

describes how the user gives as input a value x whose square root is to be computed, as well as the precision e with which the computer is to compute this square root. The system then computes an approximation to the square root with precision e. The system may choose any new value for x that satisfies the required precision.

This simple contract thus *specifies* the interaction between the user and the computing system. The first statement specifies the user's responsibility (to give an input value that satisfies the given conditions) and the second statement specifies the system's responsibility (to compute a new value for x that satisfies the given condition).

The use of contracts allows user and system choices to be intermixed in any way. In particular, the user choices can depend on previous choices by the system and vice versa, and the choices can be made repeatedly within a loop, as exemplified by the Nim game below.

4.3. *Playing Games*

The use of contracts need not be restricted to ordinary user interaction with computers. As an example of another kind of application, consider the game of Nim. Here two players, a and b, take turns removing either one or two sticks from a pile. The player who takes the last stick has lost. Of course, one of the players could here also be a computer system and the other a user who tries to beat the computer in this game, or both players could be computer systems.

Let x stand for the number of sticks in the pile. We can describe the rules of the game as a contract of the form:

$$\begin{aligned} X =_c {} & \{x \neq 0\}_b; \\ & (x := x - 1 \sqcup_a x := x - 2); \\ & \{x \neq 0\}_a; \\ & (x := x - 1 \sqcup_b x := x - 2); \\ & X, \end{aligned}$$

where x ranges over the natural numbers. Player a is going to make the first move. We start by checking whether she already has won. This happens if the number of sticks is zero to start with, forcing player b to breach his contract. Otherwise, player a removes either one or two sticks from the pile (since we are talking about natural numbers, the subtraction operator should be interpreted as "monus," i.e., so that $0 - 1 = 0$). Then we check whether player b has won, which happens if the pile now contains zero sticks (forcing player a to breach the contract). If not, player b removes one or two sticks from the pile. The game is then repeated until either one of the two players breaches the contract (loses the game). As each move by a player removes at least one stick, the game will eventually terminate with one of the players breaking the contract. Agent c acts here as a referee, who takes the blame if the game does not terminate.

5. CONTRACTS AND GAMES

The operational semantics describes all possible ways of carrying out a contract. By looking at the state component of a final configuration we can see what outcomes (final states) are possible, if all agents cooperate. However, in reality the different agents are unlikely to have the same goals, and the way one agent makes its choices need not be suitable for another agent. From the point of view of a specific agent (say a), it is therefore interesting to know what outcomes are possible, regardless of how the other agents resolve their choices.

5.1. Winning Strategies

Consider the situation where the initial state σ is given and the goal of agent a is to use contract S to reach a final state in some set q of desired final states. It is also acceptable that the agent is released from the contract, because some other agent breaches its contract. This means that the agent should strive to make its choices in such a way that the scenario starting from (S, σ) ends in a configuration (Λ, γ), where γ is either an element in q, or $\bot_b$, where $b \neq a$ (indicating that some other agent has breached the contract).

We can think of agent a as making its choices according to a *strategy*, i.e., a function that for every configuration of the form $(S_1 \sqcup_a S_2, \gamma)$ returns either (S_1, γ) or (S_2, γ) (and similarly for generalized choices, if these occur in the contract). A strategy tells the agent what to do in every possible choice situation.

We say that agent *a can use contract S in initial state σ to establish postcondition q* (written $\sigma \{\!| S |\!\}_a\, q$) if there is a strategy for a that leads from initial configuration (S, σ) to a final configuration (Λ, γ), where $\gamma \in q \cup \{\bot_b \mid b \neq a\}$. Such a strategy is called a *winning strategy* for a to achieve the goal q. Thus, $\sigma \{\!| S |\!\}_a\, q$ means that a can (by making the right choices) either achieve postcondition q or be released from the contract, no matter what the other agents do.

As an aside we note that $\sigma \{\!| S |\!\}_a\, q$ holds if and only if agent a can avoid breaching the contract $S; \{q\}_a$ when the initial state is σ.

5.2. Taking Sides

Assume that we pick out one or more agents whose side we are taking. These agents are assumed to have a common goal and to coordinate their choices in order to achieve this goal. Hence, we can regard this group of agents as a single agent. The other agents need not share the goals of our agents. To prepare for the worst, we will assume that the other agents are hostile to our goals and try to prevent us from reaching them and that they conspire in order to achieve this (i.e., they coordinate their choices against us). We will make this a little bit more dramatic and call our agents collectively the *angel* and call the other agents collectively the *demon*. We talk about an *angelic choice* when the choice is made by our agents and, about a *demonic choice* when the choice is made by the other agents.

Carrying out a contract can thus be seen as a game, where the angel plays against the demon. The angel tries to achieve its goal, and the demon tries to prevent the angel from reaching its goal. The angel loses if it breaches the contract, and wins if the demon breaches the contract. If no breach of contract occurs, then the angel wins if the last state is in the goal, and loses if it is not.

As an example, consider the game of Nim. This game has two participants, a and b. Let us now investigate under what conditions agent a can win the game. In this game it does not matter whether the referee c works with a or b, because the recursion can only be unfolded a finite number of times. In either case, premature termination is a thing for c to avoid, since it means losing the game. Only in games when c cannot avoid an infinitely unfolding recursion does it matter which side c is on; if c is with a, then an infinite unfolding is a loss for a but if c is with b, then an infinite unfolding is a win for a. If we are interested in finding out whether a can win the game in finite time, then we choose to have the referee c work with a. The angel is thus formed by a and c, while the demon is b.

The winning strategy for the angel in the game of Nim is to always remove so many sticks that $x \mathsf{\ mod\ } 3 = 1$ holds afterwards. This will force the demon to eventually take the last stick. The angel has a winning strategy in Nim, if initially $x \mathsf{\ mod\ } 3 \neq 1$, because then she can establish condition $x \mathsf{\ mod\ } 3 = 1$ with her first move, by either removing one or two sticks. In other words, $\sigma \{\!| \mathrm{Nim} |\!\}_{\mathrm{angel}}\, q$ holds when $\mathit{val} x . \sigma \mathsf{\ mod\ } 3 \neq 1$. In fact, q can here be any postcondition. This is because in this game either no postcondition is achieved or then false is achieved (implying that any postcondition q can be achieved).

Having taken the side of certain agents, we can simplify the notation for contract statements. We write $\bigsqcup$ for the angelic choice $\bigsqcup_{\mathrm{angel}}$ and $\bigsqcap$ for the demonic choice $\bigsqcap_{\mathrm{demon}}$. Furthermore, we let $\{p\}$ stand for $\{p\}_{\mathrm{angel}}$ and $[p]$ (read as "assumption p") stand for $\{p\}_{\mathrm{demon}}$. This justifies the following simpler syntax, where the explicit indications of what agents are responsible for choices and assertions have been removed:

$$S ::= \langle f \rangle \mid \{p\} \mid [p] \mid S_1 ; S_2 \mid S_1 \sqcup S_2 \mid S \sqcap S_2 .$$

This notation generalizes in the obvious way to generalized choices: we write $\bigsqcup \{S_i \mid i \in I\}$ for the angelic choice and $\bigsqcap \{S_i \mid i \in I\}$ for the demonic choice. For relational updates, we write $\{R\}$ if the next state is chosen by the angel and $[R]$ if the next state is chosen by the demon. Furthermore, we write $(\mu X \bullet S$ for $(\mathsf{rec}_{\mathrm{angel}}\, X \bullet S)$ and $(\nu X \bullet S)$ for $(\mathsf{rec}_{\mathrm{demon}}\, X \bullet S)$. Finally, we write $\sigma \{\!| S |\!\}\, q$ for $\sigma \{\!| S |\!\}_{\mathrm{angel}}\, q$. Our reason for choosing this notation will become clear in Section 7, in connection with the predicate transformer semantics.

6. ALGEBRA OF CONTRACTS

We shall now investigate the algebraic properties of contracts with exactly two agents, the angel and the demon.

If we compare two contract statements for our agent, say S and S', then we can say that the latter is at least as good as the former, if any condition that we can establish with the first contract can also be established with the second contract. We will then say that S is *refined by* S', written $S \sqsubseteq S'$. Formally, we define $S \sqsubseteq S'$ to hold if

$$\sigma \{\!| S |\!\}\, q \Rightarrow \sigma \{\!| S' |\!\}\, q \qquad \text{for any } \sigma \text{ and } q.$$

It is easy to see that refinement is reflexive and transitive. We also postulate antisymmetry; i.e., two contracts are equal, if each refines the other. In terms of establishing postconditions, we then have that $S = S'$ if and only if

$$\sigma \{\!| S |\!\}\, q \equiv \sigma \{\!| S' |\!\}\, q \quad \text{for any } \sigma \text{ and } q.$$

For example, from the intuitive description of assertions it is possible to deduce that $\{\mathsf{true}\}$ is equal to skip.

Reflexivity, transitivity, and antisymmetry together imply that contracts are *partially ordered* by the refinement relation.

It is evident that $\{\mathsf{false}\} \sqsubseteq S$ for any contract S, because we cannot use the contract $\{\mathsf{false}\}$ to establish any final condition in any initial state. Hence, any contract is an improvement over this worst of all contracts. Dually, $S \sqsubseteq [\mathsf{false}]$: the assumptions of contract $[\mathsf{false}]$ are never satisfied, so this contract is satisfied in any initial state for any final condition. This means that the partial order of contracts is *bounded*; it has a *least element* $\{\mathsf{false}\}$ and a *greatest element* $[\mathsf{false}]$.

Now consider the contract $S = S_1 \sqcup S_2$. The condition $\sigma\{\!| S |\!\}\ q$ holds if our agent, by choosing either S_1 or S_2, can establish q in initial state σ. Thus, we have that

$$\sigma\{\!| S_1 \sqcup S_2 |\!\}\ q \qquad \text{iff} \quad \sigma\{\!| S_1 |\!\}\ q \quad \text{or} \quad \{\!| S_2 |\!\}\ q.$$

For instance, we have that

$$x = 0\{\!| x := x+1 \sqcup x := x+2 |\!\}\ x = 1$$

holds, because

$$x = 0\{\!| x := x+1 |\!\}\ x = 1$$

holds. A dual argument shows that

$$\sigma\{\!| S_1 \sqcap S_2 |\!\}\ q \qquad \text{iff} \quad \sigma\{\!| S_1 |\!\}\ q \text{ and } \sigma\{\!| S_2 |\!\}\ q.$$

This reflects the fact that the angel cannot influence the choice of the demon and, hence, must be prepared for whichever contract the demon chooses.

The contracts form a *lattice* with the refinement ordering, where $\bigsqcup$ is the *join* operation in the lattice and $\bigsqcap$ is the *meet* operation. The impossible assertion $\{\mathsf{false}\}$ is the bottom of the lattice of contracts, and the impossible assumption $[\mathsf{false}]$ is the top of the lattice.

The sequential composition operation is also important here. Contracts form a *monoid* with respect to the composition operation, with skip as the identity element. Contracts as we have described them above thus have a very simple algebraic structure; i.e., they form a lattice with respect to $\sqsubseteq$ and a monoid with sequential composition.

A further generalization of contracts permits the initial and final state spaces to be different. Thus, the contract may be initiated in a state σ in Σ, but we permit operations in the contract that changes the state space, so that the final state may be in another state space Γ. In this case the simple monoid structure of contracts is not sufficient, and we need to consider the more general notion of a *category* of contracts. The different state spaces form the *objects* of the category, while the *morphisms* of the category are the contracts themselves. The skip action is the identity morphism, and composition of morphism is the ordinary sequential composition of actions.

7. PREDICATE TRANSFORMERS

We shall now link the game-theoretic interpretation with a traditional predicate transformer semantics for contract statements. In fact, the meet, join, and composition operators that we identified for contracts correspond directly to the corresponding operators on predicate transformers.

A *predicate transformer* is a function that maps predicates to predicates. We order predicate transformers by pointwise extension of the ordering on predicates, so $F \sqsubseteq F'$ for predicate transformers holds if and only if $F.q \subseteq F'.q$ for all predicates q. The predicate transformers form a complete lattice with this ordering.

7.1. Predicate Transformer Semantics

Assume that S is a contract statement. We want the predicate transformer $\mathsf{wp}.S$ to map postcondition q to the set of all initial states σ such that $\sigma \{\!| S |\!\} \ q$ holds. In other words, we want $\mathsf{wp}.S.q$ to be the set of initial states from which the angel has a winning strategy to reach the goal q. Thus, $\mathsf{wp}.S.q$ is the *weakest precondition* that guarantees that the angel can achieve postcondition q.

The intuitive description of contract statements can be used to justify the definitions of the weakest precondition semantics:

$$\mathsf{wp}.\langle f \rangle.q = f^{-1}.q$$

$$\mathsf{wp}.\{p\}.q = p \cap q$$

$$\mathsf{wp}.[p].q = \neg p \cup q$$

$$\mathsf{wp}.(S_1; S_2).q = \mathsf{wp}.S_1.(\mathsf{wp}.S_2.q)$$

$$\mathsf{wp}.(S_1 \sqcup S_2).q = \mathsf{wp}.S_1.q \cup \mathsf{wp}.S_2.q$$

$$\mathsf{wp}.(S_1 \sqcap S_2).q = \mathsf{wp}.S_1.q \cap \mathsf{wp}.S_2.q.$$

Here $f^{-1}.q$ denotes the inverse image of q: $f^{-1}.q = \{\sigma \mid f.\sigma \in q\}$. These definitions are consistent with Dijkstra's original semantics for the language of guarded commands [7] and with later extensions to it, corresponding to assertions, assumptions, and choices [2, 5, 10]. The weakest precondition semantics is extended to generalized choices in the obvious way. An important property that $\mathsf{wp}.S$ satisfies for all contracts is that it is *monotonic*:

$$p \subseteq q \Rightarrow \mathsf{wp}.S.q \subseteq \mathsf{wp}.S.q.$$

The definition of sequential composition can be written as $\mathsf{wp}.(S_1; S_2) = \mathsf{wp}.S_1 \circ \mathsf{wp}.S_2$, so the semantic function wp maps composition of contracts to functional composition of predicate transformers. Similarly, it maps the demonic and angelic choice operators for contracts to the meet and join operators on predicate transformers.

7.2. Semantics for Syntax Extensions

The weakest precondition semantics is easily generalized to account for arbitrary choice:

$$\mathsf{wp}.\left(\bigsqcup i \in I \bullet S_i\right).q = \left(\bigcup i \in I \bullet \mathsf{wp}.S_i.q\right)$$

$$\mathsf{wp}.\left(\bigsqcap i \in I \bullet S_i\right).q = \left(\bigcap i \in I \bullet \mathsf{wp}.S_i.q\right).$$

From the description of how relational updates can be defined using generalized choice we also get the semantics for these updates:

$$\mathsf{wp}.\{R\}.q = \{\sigma \mid R.\sigma \cap q \neq \emptyset\}$$

$$\mathsf{wp}.[R].q = \{\sigma \mid R.\sigma \subseteq q\}.$$

This agrees with the traditional definitions of the weakest precondition semantics of (angelic and demonic) nondeterministic state changes. Thus, we have justified considering $\{R\}_a$ as a syntactic abbreviation, according to the definition above.

Finally we consider recursive contracts of the form $(\mu X \bullet S)$ and $(\nu X \bullet S)$. Since S is built using the syntax of contract statements, we can define a function that maps any predicate transformer X to the result of replacing every construct except X in S by its weakest precondition predicate transformer. Let us call this function f. Then f can be shown to be a monotonic function on the complete lattice of predicate transformers, and by the well-known Knaster–Tarski fixpoint theorem it has a complete lattice of fixpoints. We define

$$\mathsf{wp}.(\mu X \bullet S) = \mu f$$

$$\mathsf{wp}.(\nu X \bullet S) = \nu f,$$

where μf is the least and νf the greatest fixpoint of f. The least fixpoint semantics is traditional and associated with the intuition that an infinite unfolding fails to establish any postcondition. Dually, the greatest fixpoint semantics is associated with the intuition that an infinite unfolding establishes any postcondition.

7.3. The Winning Strategy Theorem

We can now prove that the weakest precondition predicate transformer has the required property:

THEOREM 1. *Assume that contract statement S, initial state σ and postcondition q are given. Then $\sigma \in \mathsf{wp}.S.q$ if and only if $\sigma \{\!|S|\!\}\, q$.*

This means that the weakest precondition predicate transformer gives us a way of computing the set of initial states for which the angel has a winning strategy for using S to establish postcondition q from initial state σ.

Proof. We prove the theorem for the basic syntax. The extensions (generalized choices, relational updates, and recursion) are treated in Section 7.4. The proof is by induction over the structure of the statement S. We let a stand for the angel and d for the demon. We first need a few additional notions about strategies. Recall that a strategy for the angel is a function that for every *angelic configuration* (a configuration of the form $(S_1 \sqcup S_2, \gamma)$) returns either (S_1, γ) or (S_2, γ). We define the *domain of interest* for a strategy f with respect to configuration (S, σ) (written $\mathsf{IDom}(f, S, \sigma)$) to be the set of all angelic configurations that occur in scenarios for (S, σ) admitted by f. Obviously, the restriction of f to $\mathsf{IDom}(f, S, \sigma)$ determines whether f is a winning strategy or not, since configurations outside $\mathsf{IDom}(f, S, \sigma)$ cannot occur in any scenario.

We now consider the six cases of the inductive proof. Cases (i), (ii), and (iii) are the base cases (update, assertion, and assumption), while cases (iv), (v), and (vi) are step cases (sequential composition, demonic choice, and angelic choice):

(i) For the update $\langle f \rangle$, the domain of interest of any winning strategy is empty, since there is no choice available to the angel (in the derivations, a is the angel and d is the demon):

a winning strategy exists for $\langle f \rangle$ in σ with respect to q
$\equiv$ {definition of winning strategy}
for any transition $(\langle f \rangle, \sigma) \to (\Lambda, \gamma)$ we must have $\gamma \in q \cup \{\bot\}$
$\equiv$ {operational semantics}
$f.\sigma \in q$
$\equiv$ {set theory}
$\sigma \in f^{-1}.q$
$\equiv$ {definition of weakest precondition}
$\sigma \in \mathsf{wp}.\langle f \rangle.q$

(ii) For the assertion $\{p\}$, the reasoning is similar:

a winning strategy exists for $\{p\}$ in σ with respect to q
$\equiv$ {definitions}
for any transition $(\{p\}, \sigma) \to (\Lambda, \gamma)$ we must have $\gamma \in q \cup \{\bot\}$
$\equiv$ {operational semantics}
$\sigma \in p \wedge \sigma \in q$
$\equiv$ {set theory}
$\sigma \in p \cap q$
$\equiv$ {definition of weakest precondition}
$\sigma \in \mathsf{wp}.\{p\}.q$

(iii) For the assumption $[p]$, we have

a winning strategy exists for $[p]$ in a with respect to q
$\equiv$ {definitions}
for any transition $([p], \sigma) \to (\Lambda, \gamma)$ we must have $\gamma \in q \cup \{\bot\}$
$\equiv$ {operational semantics}
$\sigma \notin p \vee \sigma \in q$

$$\begin{aligned}&\equiv \{\text{set theory}\}\\ &\quad \sigma \in \neg p \cup q\\ &\equiv \{\text{definition of update}\}\\ &\quad \sigma \in \mathsf{wp}.[R].q\end{aligned}$$

(iv) Next, we consider demonic choice. For this part of the proof, we need to define a *merge* of strategies. Assume that two strategies f_1 and f_2 are given. A strategy f is a merge of f_1 and f_2 if the following holds for all configurations (S, σ):

$$(S, \sigma) \in \mathsf{Dom}.f_1 \vee (S, \sigma) \in \mathsf{Dom}.f_2 \Rightarrow f.(S, \sigma) = f_1.(S, \sigma) \vee f.(S, \sigma) = f_2(S, \sigma).$$

In other words, we require that if a configuration is in the domain of one or both of the strategies, then f agrees with one of them.

The following observation is now crucial. Assume that f is a winning strategy for game S in initial state σ with respect to goal q and that f is undefined outside $\mathsf{IDom}(f, S, \sigma)$. Similarly, assume that f' is a winning strategy for game S' in initial state σ' with respect to the same goal q and that f' is undefined outside $\mathsf{IDom}(f', S', \sigma')$. Then *any merge of f and f' is also a winning strategy for both S in σ and S' in σ' with respect to q*. This is true, because if f and f' disagree on some configuration that is in the domain of interest of both (f, S, σ) and (f', S', σ'), then both strategies must lead to a win from this configuration.

The induction assumption for the meet is now that there is a winning strategy for game S_1 in initial state σ with respect to postcondition q if and only if $\sigma \in \mathsf{wp}.S_1.q$ holds, and similarly for S_2. Then we have

$$\begin{aligned}&f \text{ is a winning strategy for } S_1 \sqcap S_2 \text{ in } \sigma \text{ with respect to } q\\ &\quad\Rightarrow \{\text{operational semantics, definition of winning strategy}\}\\ &\qquad f \text{ is winning strategy for } S_1 \text{ and } S_2 \text{ in } \sigma \text{ with respect to } q\\ &\quad\equiv \{\text{induction assumption}\}\\ &\qquad \sigma \in \mathsf{wp}.S_1.q \wedge \sigma \in \mathsf{wp}.S_2.q\\ &\quad\equiv \{\text{definition of weakest precondition}\}\\ &\qquad \sigma \in \mathsf{wp}.(S_1 \sqcap S_2).q\end{aligned}$$

so the existence of a winning strategy for the meet implies $\sigma \in \mathsf{wp}.(S_1 \sqcap S_2).q$. For the reverse implication, the induction assumption tells us that there exists a winning strategy f_i for S_i in σ with respect to goal q for $i = 1, 2$. Let f_i' be the result of restricting strategy f_i to the domain of interest of (f_i, S_i, σ). The observation above then guarantees that any merge of the strategies f_1' and f_2' is a winning strategy for $S_1 \sqcap S_2$.

(v) Now consider angelic choice. First,

$$\begin{aligned}&f \text{ is a winning strategy for } S_1 \sqcup S_2 \text{ in } \sigma \text{ with respect to } q\\ &\quad\Rightarrow \{\text{operational semantics, definition of winning strategy}\}\\ &\qquad f \text{ is winning strategy for } S_1 \text{ or } S_2 \text{ in } \sigma \text{ with respect to } q\\ &\quad\equiv \{\text{induction assumption}\}\\ &\qquad \sigma \in \mathsf{wp}.S_1.q \vee \sigma \in \mathsf{wp}.S_2.q\end{aligned}$$

$$\begin{aligned}&\equiv \{\text{definition of weakest precondition}\}\\&\quad \sigma \in \mathsf{wp}.(S_1 \sqcup S_2).q\end{aligned}$$

so the existence of a winning strategy for the angelic choice implies $\sigma \in \mathsf{wp}(S_1 \sqcup S_2).q$. For the opposite implication here, we note that if f is a winning strategy for S_i in σ with respect to q (where $i=1$ or $i=2$), then we can adjust f so that $f.(S_1 \sqcup S_2, \sigma) = (S_i, \sigma)$ to get a winning strategy for $S_1 \sqcup S_2$. This works because $(S_1 \sqcup S_2, \sigma)$ is necessarily outside $\mathsf{IDom}(f, S_i, \sigma)$.

(vi) Finally, we consider sequential composition. First, we note from the definition of the operational semantics that a scenario for $S_1; S_2$ is of the form

$$(S_1; S_2, \sigma_0) \to \cdots \to (S_2, \sigma_m) \to \cdots \to (\Lambda, \sigma_n)$$

and furthermore,

(a) $(S_1, \sigma_0) \to \cdots \to (\Lambda, \sigma_m)$ is a scenario for S_1 in initial state σ_0, and

(b) $(S_2, \sigma_m) \to \cdots \to (\Lambda, \sigma_n)$ is a scenario for S_2 in initial state σ_m.

The induction assumption is that the statement of the theorem is true for S_1 and S_2. Assume that f is a winning strategy for $S_1; S_2$ in initial state σ with respect to q and let p be the set containing all second (state) components of the final configurations of scenarios for (S_1, σ) admitted by f. Then define a strategy f' by $f'.(S, \gamma) = f.(S; S_2, \gamma)$ so that f' is a winning strategy for S_1 in σ with respect to p. The induction assumption then tells us that $\sigma \in \mathsf{wp}.S_1.p$ holds. Furthermore, we know that f is a winning strategy for S_2 in any initial state $\gamma \in p$ with respect to q, so the induction assumption tells us that $(\forall \gamma \in p \bullet \gamma \in \mathsf{wp}.S_2.q)$. Then

$$\begin{aligned}&\sigma \in \mathsf{wp}.S_1.p \wedge (\forall \gamma \in p \bullet \gamma \in \mathsf{wp}.S_2.q)\\&\quad\equiv \{\text{definition of bounded quantification}\}\\&\qquad \sigma \in \mathsf{wp}.S_1.p \wedge (\forall \gamma \bullet \gamma \in p \Rightarrow \gamma \in \mathsf{wp}.S_2.q)\\&\quad\equiv \{\text{definition of set inclusion}\}\\&\qquad \sigma \in \mathsf{wp}.S_1.p \wedge p \subseteq \mathsf{wp}.S_2.q\\&\quad\Rightarrow \{\text{monotonicity of } \mathsf{wp}.S_1\}\\&\qquad \sigma \in \mathsf{wp}.S_1.(\mathsf{wp}.S_2.q)\\&\quad\Rightarrow \{\text{definition of sequential composition}\}\\&\qquad \sigma \in \mathsf{wp}.(S_1; S_2).q\end{aligned}$$

For the opposite direction, we assume that $\sigma \in \mathsf{wp}.(S_1; S_2).q$. Again, this means that we have, with $p = \mathsf{wp}.S_2.q$,

$$\sigma \in \mathsf{wp}.S_1.p \wedge (\forall \gamma \in p \bullet \gamma \in \mathsf{wp}.S_2.q).$$

The induction assumption then tells us that there is a winning strategy f for S_1 in initial state σ with respect to p and that for every $\gamma \in p$ there is a winning strategy g_γ for S_2 in initial state γ with respect to q. Now define f' such that $f'.(S; S_2, \delta) = (T; S_2, \gamma)$ whenever $f.(S, \delta) = (T, \gamma)$ for all S and all δ, and f' is undefined everywhere else. Next let g'_γ be the result of restricting g_γ to $\mathsf{IDom}(g_\gamma, S_2, \gamma)$, for all $\gamma \in p$, and let g be a merge of f' and all the strategies in $\{g'_\gamma \mid \gamma \in p\}$ (we use the axiom of choice to deduce that such a merge exists). This makes g a winning strategy for $S_1; S_2$ in initial state σ with respect to q, and the proof is finished. ∎

7.4. Strategies and Syntax Extensions

The arguments for angelic and demonic choice in the proof can be directly generalized to choices with an arbitrary (even infinite) number of alternatives. Thus, the winning strategy theorem holds also if we choose to include the generalized choice in our basic syntax. Because relational updates could be defined as abbreviations for certain generalized choices, the winning strategy theorem holds also for contracts with relational updates.

The argument for recursion is more elaborate. Consider the recursive contract $(\mu X \bullet S)$, where we assume for simplicity that there is no nested recursion. Our argument is based on the construction of an ordinal-indexed sequence of *approximation contracts* for $(\mu X \bullet S)$. We define

$$\begin{aligned} S_0 &= \{\mathsf{false}\} \\ S_{\alpha+1} &= S_\alpha \sqcup S[X := S_\alpha] && \text{for arbitrary ordinals } \alpha, \\ S_\alpha &= \left(\bigsqcup \beta \,\middle|\, \beta < \alpha \bullet S_\beta \right) && \text{for nonzero limit ordinals } \alpha. \end{aligned}$$

Now consider the ordinal-indexed chain of predicate transformers $\mathsf{wp}.S_\alpha$. From traditional theory of program semantics we know that there exists an ordinal γ (in fact there exists an ordinal which depends only on the cardinality of the underlying state space) such that $\mathsf{wp}.S_\delta = \mathsf{wp}.S_\gamma$ whenever $\delta > \gamma$ and that $\mathsf{wp}.S_\gamma = \mu.f$, where f is the function that maps any predicate transformer X to the result of replacing every construct except X in S by its weakest precondition predicate transformer.

We can now argue that the contracts $(\mu X \bullet S)$ and S_γ are equivalent, in the sense that there is a one-to-one correspondence between the winning strategies of the two, with respect to any given initial state σ_0 and any given postcondition q. The initial unfolding step for $(\mu X \bullet S)$ (performed by the angel) corresponds to the choice that the angel makes between the contracts in the set $\{S_\beta \mid \beta < \gamma\}$. Similarly, any recursive unfolding corresponds to resolving a choice of the form $(\bigsqcup \beta \mid \beta < \alpha \bullet S_\beta)$ for some ordinal α. The general fixpoint argument guarantees that the original ordinal γ is large enough, so that this choice never becomes empty if the recursive unfolding terminates.

The following classical example illustrates this argument. Consider the recursion $(\mathsf{rec}_a\, X \bullet S)$, where

$$\begin{aligned} X =_a\ & \{x = 0\}_a;\ \{x := x' \mid x' \geqslant 1\}_d;\ X \sqcup_a \\ & \{x = 1\}_a \sqcup_a \\ & \{x \geqslant 1\}_a;\ x := x + 1;\ X. \end{aligned}$$

When a stands for the angel and d for the demon, this can be written as the recursion

$$(\mu X \bullet \{x=0\}; [x:=x' \mid x' \geqslant 1]; X \sqcup \{x=1\} \sqcup \{x \geqslant 1\}; x:=x-1; X)$$

(this recursion establishes the postcondition $x=1$, regardless the value of x in the initial state).

It can be shown that the ordinal γ in this case can be chosen to be $\omega+1$ (where ω is the first infinite ordinal; for details see [4]). Assume that the value of x is 0. At the first unfolding, the demon chooses a value for x, and after this, the value of x is decremented by 1 for each recursive call. Thus, the demon also chooses the number of recursive calls, which can thus be unboundedly large.

The approximation $S_{\omega+1}$ is

$$S_\omega \sqcup (\{x=0\}; [x:=x' \mid x' \geqslant 1]; S_\omega \sqcup \{x=1\} \sqcup \{x \geqslant 1\}; x:=x-1; S_\omega).$$

Here the angel can choose the right-hand side of the top-level choice and is then forced to choose the contract $\{x=0\}; [x:=x' \mid x' \geqslant 1]; S_\omega$, if x has the value 0. After that, the demon chooses a value for x and only after that does the angel choose S_n for some $n<\omega$, i.e., the number of remaining unfoldings. The use of ordinals guarantees that the angel does not have to "play all its cards too early."

Since there is a one-to-one correspondence between the winning strategies for $(\mu X \bullet S)$ and for S_γ, we know that the winning strategy theorem is also valid for μ-recursion. The argument for the greatest fixpoint semantics of $(\nu X \bullet S)$ is dual.

7.5. Correctness and Refinement

The predicate transformer semantics is based on total correctness. Intuitively, a contract S is correct with respect to precondition p and postcondition q, written $p\{\!|S|\!\}\, q$, if $\sigma\{\!|S|\!\}\, q$ holds for every σ in p. Thus, $p\{\!|S|\!\}\, q$ expresses that for any initial state in p, the angel can choose an execution of S that establishes q (or leads to some assumption being violated). Theorem 1 immediately gives us the following corollary.

COROLLARY 2. *Assume that contract statement S, precondition p, and postcondition q are given. Then $p \subseteq \mathsf{wp}.S.q$ if and only if $p\{\!|S|\!\}\, q$.*

We can prove $p \subseteq \mathsf{wp}.S.q$ using standard program verification techniques, such as using loop invariants to establish that a loop is guaranteed to achieve a specific postcondition. Most of the techniques in Dijkstra's weakest precondition method for proving total correctness can be generalized to contracts in this more general setting (for example, we can use loop invariants to prove correctness of the game of Nim, as shown in [6]).

We are also interested in refinement of contracts. By Theorem 1, we have the following corollary.

COROLLARY 3. *Assume that contract statements S and S' are given. Then $S \sqsubseteq S'$ if and only if $\mathsf{wp}.S \sqsubseteq \mathsf{wp}.S'$.*

Given a contract, we can use the predicate transformer formulation to derive rules that allow us to improve a contract, in the sense that any goals achievable with the original contract are still achievable with the new contract. These refinement rules can be used for *stepwise* refinement of contracts, where we start from an initial high level specification with the aim of deriving a more efficient (and usually lower level) implementation of the specification.

8. CONCLUSION

We have described a computing system in terms of a (global) state that is changed by a collection of agents. These agents are bound by contracts that stipulate their obligations and assumptions. We can study what a specific group of agents can achieve in such a system by taking sides, partitioning the agents into friendly and hostile agents. This reduces the computing system to a two-person game between an angel representing the friendly agents and a demon representing the hostile agents. Given a specific goal that the angel is requested to achieve, we can compute the set of initial states from which this can be done with certainty, in the sense that the angel has a winning strategy to achieve the goal. The computation rules are given by the weakest precondition semantics. The weakest precondition semantics can also be used to reason about correctness and refinement of contracts.

Contracts give an intuition for both angelic and demonic nondeterminism (choices of different agents), abortion (breaching a contract), and miracles (being released from a contract) in programs. Miracles and angelic nondeterminism have been introduced into the refinement calculus for algebraic reasons [3, 10, 12], but it has not been clear how they could be interpreted intuitively in a consistent way. The contract approach also means that we handle choices in a way similar to that of process algebras such as CSP [8] and CCS [9]. A particularly interesting situation arises when our agent is the environment or user of some system and the other agents are parts of the system. This view of a system as a game has been considered by Abadi, Lamport, and Wolper [1]. A more thorough investigation of the notion of contracts, games, and refinement is presented in [4].

Received January 1998; final manuscript received June 14, 1999

REFERENCES

[1] Abadi, M., Lamport, L., and Wolper, P. (1989), Realizable and unrealizable specifications of reactive systems, *in* "Proc. 16th ICALP, Stresa, Italy" (G. A. Rozenberg *et al.*, Eds.), Lecture Notes in Computer Science, Vol. 372, pp. 1–17, Springer-Verlag, New York/Berlin.

[2] Back, R. J. (1980), "Correctness Preserving Program Refinements: Proof Theory and Applications," Mathematical Centre Tracts, Vol. 131, Mathematical Centre, Amsterdam.

[3] Back, R. J. (1989), Changing data representation in the refinement calculus, *in* "21st Hawaii International Conference on System Sciences."

[4] Back, R. J., and von Wright, J. (1998), "Refinement Calculus: A Systematic Introduction," Springer-Verlag, New York/Berlin.

[5] Back, R. J., and von Wright, J. (1990), Duality in specification languages: A lattice-theoretical approach, *Acta Inform.* **27**, 583–625.

[6] Back, R. J., and von Wright, J., (1995), Games and winning strategies, *Inform. Process. Lett.* **53**(3), 165–172.

[7] Dijkstra, E. W. (1976), "A Discipline of Programming," Prentice–Hall, Englewood Cliffs, NJ.

[8] Hoare, C. A. R. (1985), "Communicating Sequential Processes," Prentice–Hall, Englewood Cliffs, NJ.

[9] Milner, R. (1989), "Communication and Concurrency," Prentice–Hall, Englewood Cliffs, NJ.

[10] Morgan, C. C. (1988), Data refinement by miracles, *Inform. Process. Lett.* **26**, 243–246.

[11] Morgan, C. C. (1990), "Programming from Specifications," Prentice–Hall, Englewood Cliffs, NJ.

[12] Morris, J. M. (1987), A theoretical basis for stepwise refinement and the programming calculus, *Sci. Comput. Programming* **9**, 287–306.

[13] Plotkin, G. D. (1981), "A Structural Approach to Operational Semantics," Tech. Rep. DAIMI FN 19, Computer Science Dept., Aarhus University.