Contracts as Mathematical Entities in Programming Logic

Ralph-Johan Back [a] Joakim von Wright [a]

[a] *Abo Akademi University, Dept. of Computer Science and Turku Centre for Computer Science (TUCS)*

Abstract

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 coalition among these agents can achieve a given goal by following the contract. We define a generalised weakest precondition semantics for contracts that permits us to compute the initial states from which a coalition has a winning strategy to reach their goal. Notions of correctness and refinement with respect to coalitions are introduced, together with proof rules for correctness and principles for refinement and equivalence transformations. We illustrate the framework with a three agent contract, showing how one can reason about the possibilities that different coalitions of agents have for reaching specific goals.

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 model described here generalises earlier work [5–7] in important ways. As before, we interpret contracts as predicate transformers and use this interpretation to reason about correctness and refinement. However, the agents play a significant role in the semantics, and this makes it possible to reason about and to compare what different groups of agents (coalitions) can accomplish with a contract if they work together.

Our syntax for contracts can be seen as a generalisation of Dijkstra's guarded commands [1], and the predicate transformer semantics gives us access to the concepts and methods of the refinement calculus [2,6,12]. When we consider a specific contract and a specific coalition, the question of correctness reduces to proving the existence of a winning strategy, and we have earlier shown how this can be handled in terms of correctness and refinement [4–6]. The idea of considering a system as a game between two players has also been considered by, e.g., by Abadi, Lamport, and Wolper [1]. However, the way that we can reason about different coalitions formed from an arbitrary number of agents is an important generalisation.

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 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* $val_x : \Sigma \to \Gamma$ and the *update function* $set_x : \Gamma \to \Sigma \to \Sigma$. The function val_x returns the value of the attribute x in a given state, while the function set_x returns a new state where x has a specific value, with the values of all other attributes left unchanged. Given a state σ, $val_x.\,\sigma$ is thus the value of x in this state, while $\sigma' = set_x.\,\gamma.\,\sigma$ 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 $(x + y).\,\sigma = val_x.\,\sigma + val_y.\,\sigma$. 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

$$(x := x + y).\,\sigma \;=\; set_x.\,(val_x.\,\sigma + val_y.\,\sigma).\,\sigma$$

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 set of states, i.e., an element of $\mathcal{P}\Sigma$. It can also be seen as a boolean function $p : \Sigma \to \mathsf{Bool}$ on the state (we will prefer 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, $x \le y$ is a boolean expression that has value $val_x.\,\sigma \le val_y.\,\sigma$ 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

$$\begin{aligned}&(x := x' \mid x' > x + y).\,\sigma.\,\sigma' \;\equiv\\ &\qquad(\exists x' \bullet \sigma' = set_x.\,x'.\,\sigma \;\wedge\; x' > val_x.\,\sigma + val_y.\,\sigma)\end{aligned}$$

This notation generalizes the ordinary assignment; we have that $\sigma' = (x := e).\,\sigma$ iff $(x := x' \mid x' = e).\,\sigma.\,\sigma'$. For more details on this way of modelling states and functions on states, see [6]

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*.

3.1 Basic contract syntax

Assume that there is a fixed collection $\mathcal{A}$ of agents, which are considered to be atomic (we assume that we can test for equality between agents). We let A range over sets of agents and a, b, c over individual agents.

We describe contracts using a notation for *contract statements*. The syntax for these is as follows:

$$S \;::=\; \langle f \rangle \mid \mathsf{if}\ p\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi} \mid S_1\,;S_2 \mid \{R\}_a \mid\; S_1 \sqcup_a S_2$$

Here a stands for an agent while f stands for a state transformer, p for a state predicate, and R for a state relation, both expressed using higher-order logic.

Intuitively, a contract statement is carried out (executed) as follows. The *functional update* $\langle f \rangle$ changes the state according to the state transformer f, i.e., if the initial state is σ_0 then the final state is $f.\sigma_0$. An *assignment statement* is a special kind of update where the state transformer is expressed as 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. We use the name skip for the identity update $\langle \mathsf{id} \rangle$, where $\mathsf{id}.\sigma = \sigma$ for all states σ.

In the *conditional composition* $\mathsf{if}\ p\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi}$, S_1 is carried out if p holds in the initial state, and S_2 otherwise. In the *sequential composition* $S_1\ ;\ S_2$, contract S_1 is first carried out, followed by S_2.

The two last constructs (relational update and choice) introduce nondeterminism into the language of contracts. Both are indexed by an agent which is responsible for deciding how the nondeterminism is resolved.

The *relational update* $\{R\}_a$ requires the agent a to choose a final state σ' so that $R.\sigma.\sigma'$ is satisfied, where σ is the initial state. In practice, the relation is expressed as a relational assignment. For example, $\{x := x' \mid x' < x\}_a$ expresses that the agent a is required to decrease the value of the program variable x. If it is impossible for the agent to satisfy this, then the agent has *breached* the contract. In $\{x := x' \mid x' < x\}_a$, agent a must breach the contract if x ranges over the natural numbers and its initial value is 0.

An important special case of relational update occurs when the relation R is of the form $(\lambda\sigma\ \sigma' \bullet \sigma' = \sigma \wedge p.\sigma)$ for some state predicate p. In this case, $\{R\}_a$ is called an *assertion* and we write it simply as $\{p\}_a$. For example, $\{x + y = 0\}_a$ expresses that the sum of (the values of) x and y in the state must be zero. If the assertion holds at the indicated place when the agent a carries out the contract, then the state is unchanged, and the rest of the contract is carried out. If, on the other hand, the assertion does not hold, then the agent has breached the contract. The assertion $\{\mathsf{true}\}_a$ is always satisfied, so adding this assertion anywhere in a contract has no effect. Dually, $\{\mathsf{false}\}_a$ is an impossible assertion; it is never satisfied and always results in the agent breaching the contract.

Finally, a *choice* $S_1 \sqcup_a S_2$ allows agent a to choose which is to be carried out, S_1 or S_2. To simplify notation, we assume that sequential composition binds stronger than choice in contracts.

We also permit *recursive contract statements.* A recursive contract is defined using an equation of the form

$$X = S$$

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 $(\mathsf{rec}\ X \bullet S)$ for the contract X defined by the equation $X = S$.

An important special case of recursion is the *while-loop* which is defined in the usual way:

$$\mathsf{while}\ p\ \mathsf{do}\ S\ \mathsf{od} \ \ \triangleq \ \ (\mathsf{rec}\ X \bullet \mathsf{if}\ p\ \mathsf{then}\ S\,;X\ \mathsf{else\ skip\ fi})$$

This syntax for contracts differs from previous presentations [6,7] in two respects. First, we make a clear distinction between deterministic and nondeterministic constructs, and second, we do not associate recursion with an agents (and as we shall see below, this means that an infinite unfolding as "bad" from the point of view of any agent).

3.2 Operational semantics

We give a formal meaning to contract statements in the form of a structured operational semantics. 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 $\perp_a$ (indicating 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 (where 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):

- *Functional update*

$$\frac{}{(\langle f\rangle,\sigma) \to (\Lambda, f.\sigma)} \qquad \frac{}{(\langle f\rangle,\bot_a) \to (\Lambda, \bot_a)}$$

- *Conditional composition*

$$\frac{p.\sigma}{(\textsf{if } p \textsf{ then } S_1 \textsf{ else } S_2 \textsf{ fi}, \sigma) \to (S_1,\sigma)} \qquad \frac{\neg p.\sigma}{(\textsf{if } p \textsf{ then } S_1 \textsf{ else } S_2 \textsf{ fi}, \sigma) \to (S_2,\sigma)}$$

$$\frac{}{(\textsf{if } p \textsf{ then } S_1 \textsf{ else } S_2 \textsf{ fi}, \bot_a) \to (\Lambda,\bot_a)}$$

- *Sequential composition*

$$\frac{(S_1,\gamma) \to (S_1',\gamma')}{(S_1 \,;\, S_2,\gamma) \to (S_1' \,;\, S_2,\gamma')} \qquad \frac{}{(\Lambda \,;\, S,\gamma) \to (S,\gamma)}$$

- *Relational update*

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

- *Choice*

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

- *Recursion*

$$\frac{X = S}{(X,\gamma) \to (S,\gamma)}$$

For the choice and the relational update, we could label the transition relation with the agent responsible for carrying out the move. However, such a labeling is really redundant since it can always be recovered from the configuration in question. We have also implicitly assumed that equations of the form $X = S$ are available in an *environment*, in the standard way.

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.

3.3 Example: a resource game

As noted in [7], 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. We shall now consider more general interaction involving several participants (agents).

The following example will be used throughout this paper, to illustrate different aspects of contracts. Assume that the agents (a_0, a_1, a_2) are placed in a ring, with a collection of resources situated between them (r_i is the collection of resources placed between agents a_{i-1} and a_{i+1}, where arithmetic modulo 3 is used). The situation is illustrated by Figure 1.

Each agent a_i has access to the resources in r_{i-1} and r_{i+1} but not to r_i. Resources are non-renewable, and we assume that the agents take turns grabbing one from either side (left or right). An agent can also choose to do nothing.

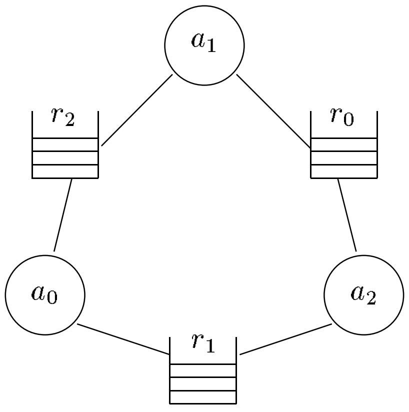

Fig. 1. Resource game

For example, if each r_i contains three resources initially, then the following scenario is possible:

1. a_0 takes one from r_2,
2. a_1 takes one from r_2,
3. a_2 takes one from r_0,
4. a_0 takes one from r_2,

5. a_1 takes one from r_0,
6. a_2 takes one from r_0,
7. a_0 takes one from r_1,
8. a_1 does nothing,
9. a_2 takes one from r_1, and
10. a_0 takes one from r_1.

At this point the nine resources are distributed so that a_0 has four, a_1 has two, and a_2 has three.

Now let us formalise this situation as a contract. Initially, no agent has grabbed any resource. Furthermore, we will assume that the resources are evenly distributed and we are only interested in the number of resources. Thus we can characterise the initialisation as follows:

$$\mathsf{init} \;\triangleq\; n_0, n_1, n_2, r_0, r_1, r_2 := 0, 0, 0, m, m, m$$

where n_i models the number of resources that agent a_i has grabbed and m is the initial number of resources at each point (so m is a free variable).

The alternatives open to agent a_i can be described as follows:

$$S_i \;\triangleq\; \mathsf{grabl}_i \sqcup_{a_i} \mathsf{skip} \sqcup_{a_i} \mathsf{grabr}_i$$

where

$$\begin{aligned}\mathsf{grabl}_i &\triangleq \{r_{i-1} > 0\} \,;\, n_i, r_{i-1} := n_i + 1, r_{i-1} - 1\\ \mathsf{grabr}_i &\triangleq \{r_{i+1} > 0\} \,;\, n_i, r_{i+1} := n_i + 1, r_{i+1} - 1\end{aligned}$$

where the arithmetic in the indices is modulo 3.

Now the whole system can be described as the following contract:

```
init ;
while $r_0 + r_1 + r_2 > 0$ do
   $S_0$ ; $S_1$ ; $S_2$
od
```

Note that the situation is not symmetric with respect to the agents; on every round a_0 gets to choose first. Another possibility would be to have a nondeterministic ordering och choices by introducing a separate *scheduler agent* c and taking $S_0 \sqcup_c S_1 \sqcup_c S_2$ as the loop body. However, this would not guarantee a fair scheduling, since c could decide to ignore one of the alternatives completely..

4 Predicate transformer semantics

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 or a group of agents, it is therefore interesting to know what outcomes are possible regardless of how the other agents resolve their choices.

Consider the situation where the initial state σ is given and a group of agents A agree that their common goal is to use contract S to reach a final state in some set q of desired final states. It is also acceptable that the coalition is released from the contract, because some other agent breaches the contract. This means that the agents should strive to make their 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 $\bot_b$ for some $b \notin A$ (indicating that some other agent has breached the contract).

For the purpose of analysing this specific case we can think of the agents in A as being one single agent and dually, the remaining agents as also being one single agent that tries to prevent the former from reaching its goal. As shown in [7] an execution of the game can then be viewed as a two-person game where one player tries to achieve a certain final situation while the other player tries to prevent this. We also show in [7] how this intuition can be formalised by interpreting contracts with two agents as predicate transformers. We shall now introduce a more general predicate transformer semantics for contracts which allows a more flexible analysis of a single contract from different points of view.

4.1 *Weakest preconditions*

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, and we use $\sqcup$ and $\sqcap$ for these lattice operators.

Assume that S is a contract statement and A a *coalition*, i.e., a set of agents. We want the predicate transformer $\mathsf{wp}.S.A$ to map postcondition q to the set of all initial states σ from which the agents in A jointly have a winning strategy to reach the goal q. Thus, $\mathsf{wp}.S.A.q$ is the *weakest precondition* that guarantees that the agents in A can cooperate to achieve postcondition q.

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

$$\begin{array}{rcl}
\mathsf{wp}.\,\langle f\rangle.\,A.\,q & = & (\lambda\sigma \bullet q.\,(f.\,\sigma)) \\
\mathsf{wp}.\,(\mathsf{if}\ p\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi}).\,A.\,q & = & (p \cap \mathsf{wp}.\,S_1.\,A.\,q) \cup (\neg p \cap \mathsf{wp}.\,S_2.\,A.\,q) \\
\mathsf{wp}.\,(S_1\,;\,S_2).\,A.\,q & = & \mathsf{wp}.\,S_1.\,A.\,(\mathsf{wp}.\,S_2.\,A.\,q) \\
\mathsf{wp}.\,\{R\}_a.\,A.\,q & = & \begin{cases} (\lambda\sigma \bullet \exists\sigma' \bullet R.\,\sigma.\,\sigma' \wedge q.\,\sigma') & \text{if } a \in A \\ (\lambda\sigma \bullet \forall\sigma' \bullet R.\,\sigma.\,\sigma' \Rightarrow q.\,\sigma') & \text{if } a \notin A \end{cases} \\
\mathsf{wp}.\,(S_1 \sqcup_a S_2).\,A.\,q & = & \begin{cases} \mathsf{wp}.\,S_1.\,A.\,q \cup \mathsf{wp}.\,S_2.\,A.\,q & \text{if } a \in A \\ \mathsf{wp}.\,S_1.\,A.\,q \cap \mathsf{wp}.\,S_2.\,A.\,q & \text{if } a \notin A \end{cases}
\end{array}$$

(Since we identify sets and characteristic functions, we could also write $\mathsf{wp}.\,\langle f\rangle.\,A.\,q$ as $\{\sigma \mid q.\,(f.\,\sigma)\}$ or simply as $q \circ f$, and similarly for the relational updates).

This means that a contract S is mathematically seen as an element (denoted by $\mathsf{wp}.\,S$) of the domain

$$\mathcal{PA} \to \mathcal{P}\Sigma \to \mathcal{P}\Sigma$$

These definitions can be seen as a generalisation of Dijkstra's original semantics for the language of guarded commands [1] and with later extensions to it, corresponding to nondeterministic assignments, choices, and miracles [2,4,11].

The semantics of a recursive contract is given in a standard way, using least fixpoints. Assume that a recursive contract statement $(\mathsf{rec}\ X \bullet S)$ and a coalition A are given. 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 semantics (for the coalition A). Let us call this function $f_{S,A}$. Then $f_{S,A}$ 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 then define the meaning of the recursion as the least fixpoint:

$$\mathsf{wp}.\,(\mathsf{rec}\ X \bullet S).\,A \;=\; \mu.\,f_{S,A}$$

The details work out in the same way as in the two-agent case [6].

4.2 Healthiness conditions

The fixpoint definition of the semantics of recursion makes use of the fact that for all coalitions A and all contracts S the predicate transformer $\mathsf{wp}.\,S.\,A$ is *monotonic*, i.e.,

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

holds for all predicates p and q. This is in fact the only one of Dijkstra's original four "healthiness" properties [1] that are satisfied by all contracts.

Although no other healthiness conditions hold universally, the property that corresponds to Dijkstra's "Law of Excluded Middle" is of interest; we say that contract statement S is *strict* if

$$\mathsf{wp}.\,S.\,\emptyset.\,\mathsf{false} \;=\; \mathsf{false}$$

A contract statement S is guaranteed to be strict if the relation R in every relational update $\{R\}_a$ in S is total.

4.3 Correctness and winning strategies

We say that *agents A can use contract S in initial state σ to establish postcondition q* (written σ $\{\!|\, S \,|\!\}_A$ q) if there is a *winning strategy* for the agents in A which guarantees that initial configuration (S, σ) will lead to termination in such a way that the final configuration is some (Λ, γ) where γ is either a final state in q or $\bot_b$ for some $b \notin A$. This means that the agents in A can make their choices so that *all* remaining possible scenarios are of the form

$$(S, \sigma) \to \cdots \to (\Lambda, \gamma) \qquad \text{where } \gamma \in q \cup \{\bot_b \mid b \notin A\}$$

Thus σ $\{\!|\, S \,|\!\}_A$ q means that, no matter what the other agents do, the agents in A can (by making suitable choices) either achieve postcondition q or make sure that some agent outside A breaches the contract.

This is easily generalised to a general notion of *correctness*; we define *correctness of contract S for agents A, precondition p and postcondition q* as follows:

$$p\ \{\!|\, S \,|\!\}_A\ q \;\triangleq\; (\forall \sigma \in p \bullet \sigma\ \{\!|\, S \,|\!\}_A\ q)$$

The winning strategy theorem of [6] can now easily be generalised to take into account collections of agents, to give the following:

Theorem 1 *Assume that contract statement S, coalition A, precondition p and postcondition q are given. Then $p \subseteq \mathsf{wp}.S.A.q$ if and only if $p \; \{\!|\, S \,|\!\}_A \; q$.*

From the wp-semantics and Theorem 1 it is straightforward to derive rules for proving correctness assertions, in the style of Hoare logic:

- *Functional update*

$$\frac{(\forall \sigma \in p \bullet q.(f.\sigma))}{p \; \{\!|\, \langle f \rangle \,|\!\}_A \; q}$$

- *Conditional composition*

$$\frac{p \cap b \; \{\!|\, S_1 \,|\!\}_A \; q \qquad p \cap \neg b \; \{\!|\, S_2 \,|\!\}_A \; q}{p \; \{\!|\, \mathsf{if}\ b\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi} \,|\!\}_A \; q}$$

- *Sequential update*

$$\frac{p \; \{\!|\, S_1 \,|\!\}_A \; r \qquad r \; \{\!|\, S_2 \,|\!\}_A \; q}{p \; \{\!|\, S_1 \,;\, S_2 \,|\!\}_A \; q}$$

- *Relational update*

$$\frac{(\forall \sigma \in p \bullet \exists \sigma' \bullet R.\sigma.\sigma' \wedge q.\sigma')}{p \; \{\!|\, \{R\}_a \,|\!\}_A \; q} \; a \in A \qquad \frac{(\forall \sigma \in p \bullet \forall \sigma' \bullet R.\sigma.\sigma' \Rightarrow q.\sigma')}{p \; \{\!|\, \{R\}_a \,|\!\}_A \; q} \; a \notin A$$

- *Choice*

$$\frac{p \; \{\!|\, S_1 \,|\!\}_A \; q}{p \; \{\!|\, S_1 \sqcup_a S_2 \,|\!\}_A \; q} \; a \in A \qquad \frac{p \; \{\!|\, S_2 \,|\!\}_A \; q}{p \; \{\!|\, S_1 \sqcup_a S_2 \,|\!\}_A \; q} \; a \in A$$

$$\frac{p \; \{\!|\, S_1 \,|\!\}_A \; q \qquad p \; \{\!|\, S_1 \,|\!\}_A \; q}{p \; \{\!|\, S_1 \sqcup_a S_2 \,|\!\}_A \; q} \; a \notin A$$

- *Loop*

$$\frac{p \cap b \cap t = w \; \{\!|\, S \,|\!\}_A \; p \cap t < w}{p \; \{\!|\, \mathsf{while}\ b\ \mathsf{do}\ S\ \mathsf{od} \,|\!\}_A \; p \cap \neg b}$$

  Here t (the termination argument for the loop) is assumed to range over some well-founded set W, and w is a fresh variable also ranging over W.

- *Consequence*

$$\frac{p' \subseteq p \qquad p \; \{\!|\, S \,|\!\}_A \; q \qquad q \subseteq q'}{p' \; \{\!|\, S \,|\!\}_A \; q'}$$

These are close to the traditional Hoare Logic rules for total correctness in many ways. We include a rule for the while-loop rather than for recursion, for simplicity. The existential quantifier in the first rule for relational update and the existence of two alternative rules for choice (when $a \in A$) indicate that we can show the existence of a general winning strategy by providing a witness during the correctness proof. In fact, the proof encodes a winning strategy, in the sense that if we provide an existential witness (for a relational update) then we describe how the agent in question should make its choice in order to contribute to establishing the postcondition. Similarly, the selection of the appropriate rule for a choice encodes a description of how an agent should make the choice during an execution.

4.4 Refinement of contracts

The predicate transformer semantics is based on total correctness. Traditionally, a notion of *refinement* is derived from a corresponding notion of correctness, so that a refinement $S \sqsubseteq S'$ holds iff S' satisfies all correctness (as well as all other semantic) properties that S satisfies.

Since we define correctness for a collection of agents (whose ability to guarantee a certain outcome we are investigating), refinement will also be relativised similarly. We say that *contract S is refined by contract S' for coalition A* (written $S \sqsubseteq_A S'$), if S' preserves all correctness properties of S', for A. By Theorem 1, we have

$$S \sqsubseteq_A S' \;\equiv\; (\forall q \bullet \mathsf{wp}.\,S.\,A.\,q \subseteq \mathsf{wp}.\,S'.\,A.\,q)$$

The traditional notion of refinement [2] is here recovered in the case when the coalition A is empty; i.e., if all the nondeterminism involved is demonic. Furthermore, the generalisation of refinement to include both angelic and demonic nondeterminism [4,6] is recovered by identifying the agents in A with as the angel and the agents outside A as the demon.

Given a contract, we can use the predicate transformer formulation of refinement to derive rules that allow us to improve a contract from the point of view of a specific coalition A, 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.

5 Algebra of contracts

We shall now investigate the algebraic properties that are induced by the predicate transformer semantics of contracts. We first consider the semantic operator wp itself, and then the refinement relation.

5.1 *Properties of* wp

We already noted that wp is monotonic in its third argument, i.e., that $\mathsf{wp}.\,A.\,S$ is a monotonic predicate transformer, for arbitrary A and S. Let us now investigate how the semantics changes when we apply various operations to the coalitions under consideration.

Theorem 2 *Let contract statement S and predicate q be arbitrary. Then*

(a) $A \subseteq A' \;\Rightarrow\; \mathsf{wp}.\,S.\,A \sqsubseteq \mathsf{wp}.\,S.\,A'$, *provided that S is strict,*
(b) $\mathsf{wp}.\,S.\,\overline{A}.\,q \;\subseteq\; \neg\mathsf{wp}.\,S.\,A(\neg q)$.

The proof of each of these properties is a tedious but straightforward induction over the structure of contract statements.

Technically speaking, we have investigated to what extent the operator wp is homomorphic in its second argument. Part (a) is a monotonicity property which says that if new agents join a coalition, then the capabilities of the coalition increase (i.e., it becomes easier to reach goals). On the other hand, (b) indicates that if a coalition can establish a goal q then the remaining agents cannot establish $\neg q$ (the reason that we do not have an equality in (b) is that in an infinite scenario neither A nor $\overline{A}$ establishes any postcondition).

An immediate consequence of the monotonicity property is the following:

$$\mathsf{wp}.\,S.\,A \sqcup \mathsf{wp}.\,S.\,A' \;\sqsubseteq\; \mathsf{wp}.\,S.\,(A \cup A')$$

This can be interpreted as saying that the power of two combined coalitions is greater (or at least as great) as the sum of their powers in isolation. The dual property also holds:

$$\mathsf{wp}.\,S.\,(A \cap A') \;\sqsubseteq\; \mathsf{wp}.\,S.\,A \sqcap \mathsf{wp}.\,S.\,A'$$

This property does not have an immediate intuitive interpretation, except as an alternative formulation of monotonicity.

5.2 *Properties of refinement*

It is easy to see that refinement for a given coalition is reflexive and transitive, i.e.,

$$S \sqsubseteq_A S$$
$$S \sqsubseteq_A S' \;\wedge\; S' \sqsubseteq_A S'' \;\Rightarrow\; S \sqsubseteq_A S''$$

holds for arbitrary contract statements S, S', and S''. Thus, contracts are *preordered* by the refinement relation $\sqsubseteq_A$, for a fixed coalition A.

This preordering induces equivalence classes of contract statements and a lattice structure on these equivalence classes. The join operation $\sqcup_A$ corresponds to a choice $\sqcup_a$ for any agent $a \in A$ and dually, the meet operation $\sqcap_A$ corresponds to a choice $\sqcup_a$ for any agent $a \notin A$.

The duality that is thus created by the partitioning of the agents is reflected in the following property:

$$S \sqsubseteq_A S' \;\equiv\; S' \sqsubseteq_{\overline{A}} S$$

i.e., complementing the coalition reverses all refinements.

Next, we investigate homomorphism properties of the refinement relation.

Theorem 3 *Let coalition A be arbitrary. Then*

(a) $R \subseteq R' \;\Rightarrow\; \{R\}_a \sqsubseteq_A \{R'\}_a$ *if $a \in A$, and*
$\quad\;\; R \subseteq R' \;\Rightarrow\; \{R'\}_a \sqsubseteq_A \{R\}_a$ *if $a \notin A$;*
(b) $S_1 \sqsubseteq_A S_1' \wedge S_2 \sqsubseteq_A S_2' \;\Rightarrow\; \textsf{if } p \textsf{ then } S_1 \textsf{ else } S_2 \textsf{ fi} \sqsubseteq_A \textsf{if } p \textsf{ then } S_1' \textsf{ else } S_2' \textsf{ fi}$*;*
(c) $S_1 \sqsubseteq_A S_1' \;\wedge\; S_2 \sqsubseteq_A S_2' \;\Rightarrow\; S_1 \,;\, S_2 \sqsubseteq_A S_1' \,;\, S_2'$*;*
(d) $S_1 \sqsubseteq_A S_1' \;\wedge\; S_2 \sqsubseteq_A S_2' \;\Rightarrow\; S_1 \sqcup_a S_2 \sqsubseteq_A S_1' \sqcup_a S_2'$ *for all a; and*
(e) $(\forall X \bullet S \sqsubseteq_A S') \;\Rightarrow\; (\textsf{rec } X \bullet S) \sqsubseteq_A (\textsf{rec } X \bullet S')$.

The proof is again a straightforward but tedious induction over the structure of contract statements. Here (b)–(e) essentially say that *contexts are monotonic*, so we can always refine subcontracts. On the other hand, (a) indicates that to refine a relational update $\{R\}_a$ we must know whether the agent a is in the coalition considered or not.

5.3 Semantic equivalence

From the relativised refinement relation we can introduce an absolute one, by requiring refinement for any coalition of agents:

$$S \sqsubseteq S \;\;\triangleq\;\; (\forall A \bullet S \sqsubseteq_A S')$$

In fact (because $S' \sqsubseteq_A S$ is equivalent to $S \sqsubseteq_{\overline{A}} S'$) this non-relativased refinement relation is really the same as *semantic equivalence* $S \equiv S'$ defined by

$$S \equiv S \;\;\triangleq\;\; (\forall A \subseteq \mathcal{A} \bullet \forall q \bullet \textsf{wp}.\, S.\, A.\, q = \textsf{wp}.\, S'.\, A.\, q)$$

Another important consequence of this is that if we want to transform a contract statement S in such a way that all correctness properties are preserved, then we should transform it under this semantic equivalence. The following rules serve as a small example of what kind of transformations are then available:

$$\begin{array}{rcl} S \sqcup_a S & \equiv & S \\ \textsf{while } p \textsf{ do } S \textsf{ od} & \equiv & \textsf{if } p \textsf{ then } S \textsf{ else skip fi} \,;\, \textsf{while } p \textsf{ do } S \textsf{ od} \\ \{R_1\}_a \,;\, \{R_2\}_a & \equiv & \{R_1 \,;\, R_2\}_a \\ \{R_1\}_a \sqcup_a \{R_2\}_a & \equiv & \{R_1 \cup R_2\}_a \\ \{\, |f| \,\}_a & \equiv & \langle f \rangle \end{array}$$

where $|f|$ is the relation corresponding to f, i.e., $|f|.\,\sigma.\,\sigma' \equiv (\sigma' = f.\,\sigma)$.

6 Reasoning about contracts

We shall finally consider how to reason about a contract from different points of view. We focus on correctness reasoning, but we also hint at some refinement possibilities.

We return to the resource game described in Section 3.3:

$$\mathcal{R} \;=\; \textsf{init} \,;\, \textsf{while } r_0 + r_1 + r_2 > 0 \textsf{ do } S_0 \,;\, S_1 \,;\, S_2 \textsf{ od}$$

where we assume that all program variables (and m) range over the natural numbers.

We concentrate on the question of the extent to which it is possible for agents to get their fair share of the resources, or to prevent other agents from getting their fair share. Since the initialisation states that the resources are initially distributed equally ($r_0 = r_1 = r_2 = m$), agent a_i getting a fair share means that execution terminates with $n_i \geq m$ holding.

Intuitive reasoning should indicate that a single agent (say, agent 2) cannot be certain to get its fair share, if the two other agents cooperate to prevent it. This can be shown formally by showing

$$p \;\{\!|\, \mathcal{R} \,|\!\}_{\{a_0, a_1\}}\; n_2 < m$$

where p is some suitable precondition.

We shall outline the proof of this property, showing that it holds under the precondition $m > 0$ (if $m = 0$, then the final situation is trivially fair). This can be done by unfolding the loop once and then showing that

(1) in the first iteration a_0 and a_1 can establishes an unfair situation, and
(2) from that point on the two agents can maintain the unfair situation.

For the first part we can show

$$p_1 \ \{\!| S_0 \,;\, S_1 \,;\, S_2 |\!\}_{\{a_0,a_1\}} \ p_2$$

where

$$\begin{array}{lll} p_1 & \text{is} & n_0 = n_1 = n_2 = 0 \wedge r_0 = r_1 = r_2 = m \wedge m > 0 \\ p_2 & \text{is} & n_0 \geq n_2 \wedge n_1 \geq n_2 \wedge r_0 < m \wedge r_1 < m \wedge r_2 = m \wedge s = 3m \wedge m > 0 \end{array}$$

where s abbreviates $n_0+n_1+n_2+r_0+r_1+r_2$. For this we use the correctness rules for sequential composition and updates. When proving the correctness steps for S_0 and S_1 we have to give a witness for the existential quantifier in the correctness rule; this corresponds to identifying the strategy (which is to leave r_2 untouched as long as possible).

For the second part we want to prove the following:

$$p_2 \ \{\!| \mathsf{while}\ r_0 + r_1 + r_2 > 0\ \mathsf{do}\ S_0 \,;\, S_1 \,;\, S_2\ \mathsf{od} |\!\}_{\{a_0,a_2\}} \ n_2 < m$$

This can be proved using the correctness rule for while-loop with a suitable invariant that describes the unfair situation. The crucial component of this invariant is the following:

$$r_0 \leq r_2 \wedge r_1 \leq r_2 \wedge (r_0 = r_2 \vee r_1 = r_2 \Rightarrow n_0 > n_2 \vee n_1 > n_2)$$

The termination argument for the loop is $r_1+r_2+r_3$ (the two agents can make sure that the loop terminates).

This proof sketch should indicate that we can follow the same proof paradigm as in traditional correctness proofs, i.e., we use the proof rules for correctness to break down the correctness goal step by step, supplying invariants, intermediate predicates, and existential witnesses when they are needed.

Correctness properties for a coalition are preserved by refinements for that same coalition. Thus we could, e.g., change the rules of the game in a way that gives agents a_0 and a_2 more alternatives and a_1 less alternatives, and the correctness property that we proved would still be valid.

The correctness property considered above is just one of many possible examples. Even though one agent working alone cannot be certain to get its fair share, it may be possible for two agents (say, a_0 and a_2) to make sure that they both get their fair share if they work together. To show this, we would have to prove

$$p \,\{\!|\, \mathcal{R} \,|\!\}_{\{a_0,a_2\}}\, n_0 \geq m \wedge n_2 \geq m$$

again for some suitable precondition p.

7 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 describe contracts as syntactic entities, and give them both an operational and a denotational (weakest precondition) semantics. Given a specific goal that a coalition of agents is requested to achieve, we can use traditional correctness reasopning to show that the goal can in fact be achieved by the coalition, regardless of how the remaining agents act.

Earlier formulations of computing systems as contracts [6,7] did not view contracts as mathematical entities in their own right, but assumed a division of agents into two groups (angels and demons) from the outset. The present formulation allows us to analyse a single contract from the point of view of different coalitions and compare the results. For example, it is possible to study whether a given coalition A would gain anything by premitting an outside agent b to join A. We could also model sources of errors as governed by separate agents (in the form $\mathsf{skip} \sqcup_c \langle error \rangle$). Then such an error source can be "turned off" simply by including the agent c into the coalition that is being considered.

The contract approach generalises previous models for nondeterminism, both in relational and in predicate transformer frameworks [3,6] and in process algebra frameworks such as CSP [9] and CCS [35].

References

[1] M. Abadi, L. Lamport, and P. Wolper. Realizable and unrealizable specifications of reactive systems. In G.A. Rozenberg et al. (editors), *Proc. 16th ICALP*, volume 372 of *Lecture Notes in Computer Science*, 1–17, Stresa, Italy, 1989, Springer–Verlag.

[2] R.J. Back. *Correctness Preserving Program Refinements: Proof Theory and Applications*, volume 131 of *Mathematical Centre Tracts.* Mathematical Centre, Amsterdam, 1980.

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

[4] R.J. Back and J. von Wright. Duality in specification languages: a lattice-theoretical approach. *Acta Informatica*, 27:583–625, 1990.

[5] R.J. Back and J. von Wright. Games and winning strategies. *Information Processing Letters* 53(3):165–172, 1995.

[6] R.J. Back and J. von Wright. *Refinement Calculus: A Systematic Introduction.* Springer-Verlag, 1998.

[7] R.J. Back and J. von Wright. Contracts, Games, and Refinement. *Information and Computation*, 156:25–45, 2000.

[8] E.W. Dijkstra. *A Discipline of Programming.* Prentice-Hall International, 1976.

[9] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall International, 1985.

[10] R. Milner. *Communication and Concurrency.* Prentice-Hall International, 1989.

[11] C.C. Morgan. Data refinement by miracles. *Information Processing Letters*, 26:243–246, 1988.

[12] C.C. Morgan. *Programming from Specifications.* Prentice-Hall, 1990.

[13] G.D. Plotkin. A structural approach to operational semantics. Tech. Rpt. DAIMI FN 19, Computer Science Dept., Aarhus University, April 1981.