Modeling Component Environments and Interactive Programs Using Iterative Choice

Ralph Back, Anna Mikhajlova,
Joakim von Wright

Turku Centre for Computer Science
TUCS Technical Report No 200
September 1998
ISBN 952-12-0267-X
ISSN 1239-1891

Abstract

The unifying ground for component-based systems and interactive programs is the interaction between the user and the system or between a component and its environment. Modeling both kinds of systems in a formal framework appears to be critical for reasoning about the systems' reliability and correctness. A mathematical foundation, based on the idea of contracts, permits this kind of reasoning. In this paper we study the iterative choice statement which models an event loop allowing the user to repeatedly choose from a number of actions an alternative which is enabled and have it executed. We study the properties of this statement and demonstrate its modeling capabilities by specifying an interactive dialog box, and a component environment which describes all actions the environment can take on a component.

Keywords: Modeling, component-based systems, interactive programs, contracts, angelic iteration, iterative choice, correctness, refinement

TUCS Research Group
Programming Methodology Research Group

1 Introduction

Component-oriented approach to software design and development is rapidly gaining popularity and stimulates research on methods for analysis and construction of reliable and correct components and their compositions. Component compositions consist of cooperating or interacting *components*, and for each component all the other components it cooperates with can be collectively considered as the *environment*. Although various standard methods can be used for reasoning about separate components, component environments present, in this respect, a challenge. The ability to model component environments appears to be critical for reasoning about component-based systems. The designer of a component should be aware of the behavior of the environment in which the component is supposed to operate. He can then, e.g. analyze the effect the change to the component will have on the environment, design the appropriate component interface, etc. Also, most programs, to be useful, involve interaction with a user. An interactive program should be able to react in a specified way in response to user actions, and, as any other program, it should be reliable and correct.

Of course, component-based systems can be interactive, and interactive systems can be component-based. Also, component-based systems, even those not involving interaction with the user, can be considered as interactive since components interact with other components. The unifying ground for both kinds of systems is the interaction between the user and the system or between a component and its environment.

The mathematical foundation for reasoning about interactive systems, based on the idea of contracts, has been introduced in [3, 5]. In particular, Back and von Wright proposed using an *iterative choice* statement which describes an event loop, allowing the user to repeatedly choose from a number of actions an alternative which is enabled and have it executed. In this paper we focus on the iterative choice statement, examine its modeling capabilities, and study properties of the *angelic iteration* operator, which is used to define this statement. We illustrate the expressive power and versatility of the iterative choice statement by specifying an interactive dialog box, and a component environment which describes all actions the environment can take on a component.

Notation: 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 Contracts and Refinement

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 section we review a notation for *contract statements*. A more detailed description as well as operational and weakest precondition semantics of these statements can be found in [3, 5].

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. The state has a number of *program variables* $x_1, \ldots, x_n$, each of which can be observed and changed independently of the others. A program variable x of type Γ is really a pair of the *value function* $valx : \Sigma \to \Gamma$ and the *update function* $setx : \Gamma \to \Sigma \to \Sigma$. Given a state σ, $valx.\,\sigma$ is 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 *assignment* like $x := x + y$ denotes a state changing function that updates the value of x to the value of the expression $x + y$, i.e.

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

A *state predicate* $p : \Sigma \to \mathsf{Bool}$ is a boolean function on the state. Since a predicate corresponds to a set of states, we use set notation ($\cup$, $\subseteq$, etc.) for predicates. Using program variables, predicates can be written as boolean expressions, for example, $(x + 1 > y)$. Similarly, a *state relation* $R : \Sigma \to \Sigma \to \mathsf{Bool}$ relates a state σ to a state σ' whenever $R.\,\sigma.\,\sigma'$ holds. 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 variables are unchanged.

2.1 Contract notation

Contracts are built from state changing functions, predicates and relations. The *update* $\langle f \rangle$ changes the state according to $f : \Sigma \to \Sigma$. 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 changing function is an assignment. For example, the assignment statement $\langle x := x + y \rangle$ (or just $x := x + y$ when it is clear from the context that an assignment statement rather than an assignment is intended) requires the agent to set the value of program variable x to the sum of the values of x and y.

The *assertion* $\{p\}$ of a state predicate 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 variables) x and y in the state must be zero. If the assertion does not hold, then the agent has *breached* the contract. The *assumption* $[p]$ is dual to an assertion; if the condition p does not hold, then the agent is released from any obligation to carry out his part of the contract.

In the *sequential action* $S_1; S_2$ the action S_1 is carried out first, followed by S_2. A *choice* $S_1 \sqcup S_2$ allows the agent to choose between carrying out S_1 or S_2.

In general, there can be a number of agents that are acting together to change the world and whose behavior is bound by contracts. We can indicate explicitly which agent is responsible for each choice. For example, in the contract

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

the agents involved are a and b. The effect of the update is independent of which agent carries it out, so this information can be lost when writing contract statements.

The *relational update* $\{R\}_a$ is a contract statement that permits an agent to choose between all final states related by the 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 variables.

A *recursive contract statement* of the form $(\mathsf{rec}_a\ X \bullet S)$ is 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 $(\mathsf{rec}_a\ X \bullet S)$. A more convenient way to define a recursive contract is by an equation of the form $X \;=_a\; S$, where S typically contains some occurrences of X. The indicated agent is responsible for termination; if the recursion unfolds infinitely, then the agent has breached the contract.

2.2 Using Contracts

Carrying out a contract can be seen as a game, where an angel plays against the demon. One collection of agents (the *angel*) are assumed to have a common goal and to coordinate their choices in order to achieve this goal. To prepare for the worst, we assume that the the remaining agents (the *demon*) tries to prevent us from reaching our goals. We refer to choices made by our agents it as *angelic choices*, and to choices made by the other agents as *demonic choices*.

Having taken the side of certain agents, we can simplify the notation for contract statements. We write $\sqcup$ for the angelic choice $\sqcup_{angel}$ and $\sqcap$ for the demonic choice $\sqcup_{demon}$. Furthermore, we note that if our agents have breached the contract, then the other agents are released from it, i.e. $\{p\}_{angel} = [p]_{demon}$, and vice versa. Hence, we agree to let $\{p\}$ stand for $\{p\}_{angel}$ and $[p]$ stand for $\{p\}_{demon}$. This justifies the following syntax, where the explicit indication of which agent is responsible for the choice, assertion, or assumption has been removed:

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

This notation generalizes in the obvious way to generalized choices: we write $\sqcup\{S_i \mid i \in I\}$ for the angelic choice of one of the alternatives in the set $\{S_i \mid i \in I\}$ and we write $\sqcap\{S_i \mid i \in I\}$ for the corresponding demonic choice. For relational update, 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 $(\mathsf{rec}_{angel}\ X \bullet S)$ and $(\nu X \bullet S)$ for $(\mathsf{rec}_{demon}\ X \bullet S)$; this notation agrees with the predicate transformer semantics of contracts.

The notation for contracts allows us to express all standard programming language constructs, like sequential composition, assignments, empty statements, conditional statements, loops, and blocks with local variables.

2.3 User interaction

Interactive programs can be seen as special cases of contracts, where two agents are involved, the *user* and the *computer system.* The user is the angel, which chooses between alternatives in order to influence the computation in a desired manner, and the computer system is the demon, resolving any internal choices in a manner unknown to the user.

User input during program execution is modeled by an angelic relational assignment. For example, the contract

$$\{x, e := x', e' \mid x' \geq 0 \land e > 0\}; [x := x' \mid -e < x'^2 - x < e]$$

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.

This simple contract *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).

2.4 Semantics, Correctness and Refinement of Contracts

Every contract statement has a weakest precondition predicate transformer semantics. A *predicate transformer* $S : (\Gamma \to \mathsf{Bool}) \to (\Sigma \to \mathsf{Bool})$ is a function from predicates on Γ to predicates on Σ. We write

$$\Sigma \mapsto \Gamma \ \hat{=}\ (\Gamma \to \mathsf{Bool}) \to (\Sigma \to \mathsf{Bool})$$

to denote a set of all predicate transformers from Σ to Γ. A contract statement with initial state in Σ and final state in Γ determines a monotonic predicate transformer $S : \Sigma \mapsto \Gamma$ that maps any postcondition $q : \Gamma \to \mathsf{Bool}$ to the weakest precondition $p : \Sigma \to \mathsf{Bool}$ such that the statement is guaranteed to terminate in a final state satisfying q whenever the initial state satisfies p. Following an established tradition, we identify contract statements with the monotonic predicate transformers that they determine. For details of the predicate transformer semantics, we refer to [3, 5].

The *total correctness assertion* $p \{\!| S |\!\} q$ is said to hold if the user can use the contract S to establish the postcondition q when starting in the set of states p. The pair of state predicates (p, q) is usually referred to as the pre- and post-condition specification of the contract S. The total correctness assertion $p \{\!| S |\!\} q$, which is equal to $p \subseteq S.\, q$, means that the user can (by making the right choices) either achieve the postcondition q or be released from the contract, no matter what the other agents do.

A contract S is *refined by* a contract S', written $S \sqsubseteq S'$, if any condition that we can establish with the first contract can also be established with the second contract. Formally,

$$S \sqsubseteq S' \;\;\widehat{=}\;\; p \{\!| S |\!\} q \Rightarrow p \{\!| S' |\!\} q$$

Refinement is reflexive and transitive. In addition, the contract constructors are monotonic, so a contract can be refined by refining a subcomponent.

Refinement calculus provides rules for transforming more abstract program structures into more concrete ones based on the notion of refinement of contracts presented above. Large collections of refinement rules are given, for instance, in [5, 8].

3 Angelic Iteration

Let S be a monotonic predicate transformer (i.e., the denotation of a contract). We define an iteration construct over S, *angelic iteration*, as the following fixpoint:

$$S^\phi \;\;\widehat{=}\;\; (\mu X \bullet S; X \sqcup \mathsf{skip}) \qquad\qquad (\textit{Angelic iteration})$$

As such, this construct is a dual of the weak iteration S^* defined in [5] by $(\nu X \bullet S; X \sqcap \mathsf{skip})$.

Theorem 1 *Let S be an arbitrary monotonic predicate transformer. Then*

$$S^\phi \;=\; ((S^\circ)^*)^\circ$$

Intuitively, the statement S^ϕ is executed so that S is repeated an angelically chosen (finite) number of times before the iteration is terminated by choosing skip. For example, $(x := x + 1)^\phi$ increments x an angelically chosen finite number of times, and has, therefore, the same effect as the angelic update $\{x := x' \,|\, x \le x'\}$.

A collection of basic properties of angelic iteration follows by duality from the corresponding properties of weak iteration proved in [4].

Theorem 2 *Let S and T be arbitrary monotonic predicate transformers. Then*

$$
\begin{array}{ll}
(a) & S^{\phi} \text{ is monotonic and terminating} \\
(b) & S^{\phi} \text{ preserves termination, strictness, and disjunctivity} \\
(c) & S \sqsubseteq S^{\phi} \\
(d) & (S^{\phi})^{\phi} \;=\; S^{\phi} \\
(e) & S^{\phi} ; S^{\phi} \;=\; S^{\phi} \\
(f) & S \sqsubseteq T \;\Rightarrow\; S^{\phi} \sqsubseteq T^{\phi}
\end{array}
$$

Here, a predicate transformer S is said to be *terminating* if $S.\,\mathsf{true} = \mathsf{true}$, *strict* if $S.\,\mathsf{false} = \mathsf{false}$, and *disjunctive* if $S.\,(\cup i \in I \bullet q_i) \;=\; (\cup i \in I \bullet S.\,q_i)$, for $I \neq \emptyset$.

To account for tail recursion, the angelic iteration can be characterized as follows:

Lemma 1 *Let S and T be arbitrary monotonic predicate transformers. Then*

$$S^{\phi};T \;=\; (\mu X \bullet S;X \sqcup T)$$

This lemma provides us with general unfolding and induction rules. For arbitrary monotonic predicate transformers S and T,

$$
\begin{array}{ll}
S^{\phi};T \;=\; S;S^{\phi};T \sqcup T & \quad (\textit{unfolding}) \\
S;X \sqcup T \sqsubseteq X \;\Rightarrow\; S^{\phi};T \sqsubseteq X & \quad (\textit{induction})
\end{array}
$$

From the unfolding rule with T taken to be skip we get the useful property that doing nothing is refined by angelic iteration:

$$\mathsf{skip} \sqsubseteq S^{\phi}$$

Angelic iteration can also be characterized on the level of predicates:

Lemma 2 *Let $S : \Sigma \mapsto \Sigma$ be an arbitrary monotonic predicate transformer and $q : \mathcal{P}\Sigma$ an arbitrary predicate. Then*

$$S^{\phi}.q \;=\; (\mu x \bullet S.\,x \cup q)$$

When applied to monotonic predicate transformers, the angelic iteration operator has two interesting properties known from the theory of regular languages, namely, the *decomposition property* and the *leapfrog property.*

Lemma 3 *Let S and T be arbitrary monotonic predicate transformers. Then*

$$
\begin{array}{ll}
(S \sqcup T)^{\phi} \;=\; S^{\phi};(T;S^{\phi})^{\phi} & \quad (\textit{decomposition}) \\
(S;T)^{\phi};S \;\sqsubseteq\; S;(T;S)^{\phi} & \quad (\textit{leapfrog})
\end{array}
$$

(if S is disjunctive, then the leapfrog property is an equality).

Lemma 1, Lemma 2, and Lemma 3 follow by duality from the corresponding properties of weak iteration as given in [5].

Let us now study under what conditions the total correctness assertion $p \; \{\!|\, S^{\phi} \,|\!\} \; q$ is valid. In lattice theory, the general least fixpoint introduction rule states that

$$\frac{t_w \;\sqsubseteq\; f.\, t_{<w}}{t \;\sqsubseteq\; \mu\, f}$$

where $\{t_w \mid w \in W\}$ is a ranked collection of elements (so that W is a well-founded set and $v < w \Rightarrow t_v \;\sqsubseteq\; t_w$), $t_{<w}$ is an abbreviation for $(\sqcup v \mid v < w \bullet t_v)$, and $t = (\sqcup w \in W \bullet t_w)$. When used for predicates, with $S^{\phi}.\, q = (\mu x \bullet S.\, x \cup q)$, this rule directly gives a correctness rule for angelic iteration:

$$\frac{p_w \;\subseteq\; S.\, p_{<w} \cup q}{p \; \{\!|\, S^{\phi} \,|\!\} \; q} \qquad \textit{(angelic iteration correctness rule)}$$

Here $\{p_w \mid w \in W\}$ is a ranked collection of predicates and $p = (\cup\, w \in W \cdot p_w)$. If the ranked predicates are written using an invariant I and a termination function t, then we have

$$\frac{I \cap t = w \;\subseteq\; S.\, (I \cap t < w) \cup q}{I \; \{\!|\, S^{\phi} \,|\!\} \; q}$$

where w is a fresh variable. Intuitively, this rule says that at every step, either the invariant I is preserved (with t decreasing) or the desired postcondition q is reached directly and the iteration can terminate. This corresponds to temporal logic assertions "*I until q*" and "*eventually not I*". Since t cannot decrease indefinitely, this guarantees that the program eventually reaches q if started in I.

4 Iterative Choice

Now we consider a derivative of angelic iteration S^{ϕ}, the *iterative choice* statement. This specification construct was defined in [5] as follows:

$$\mathsf{do} \; \lozenge_{i=1}^{n} g_i :: S_i \; \mathsf{od} \;\widehat{=}\; (\mu\, X \bullet \{g_1\}; S_1; X \sqcup \ldots \sqcup \{g_n\}; S_n; X \sqcup \mathsf{skip}) \qquad \textit{(Iterative choice)}$$

As such, iterative choice is equivalent to angelic iteration of the statement $\sqcup_{i=1}^{n} \{g_i\}; S_i$,

$$\mathsf{do} \; \lozenge_{i=1}^{n} g_i :: S_i \; \mathsf{od} \;=\; (\sqcup_{i=1}^{n} \{g_i\}; S_i)^{\phi}$$

and its properties can be derived from the corresponding properties of the angelic iteration.

An angelic iteration is refined if every alternative in the old system is refined by the angelic choice of all the alternatives in the new system.

Theorem 3 *For arbitrary state predicates $g_1, \ldots, g_n$ and $g'_1, \ldots, g'_m$, and arbitrary contract statements $S_1, \ldots, S_n$ and $S'_1, \ldots, S'_m$ we have that*

$$(\forall i \mid 1 \leq i \leq n \bullet \{g_i\}; S_i \sqsubseteq \sqcup_{j=1}^{m} \{g'_j\}; S'_j) \ \Rightarrow$$
$$\mathsf{do}\ \lozenge_{i=1}^{n} g_i :: S_i\ \mathsf{od} \sqsubseteq \mathsf{do}\ \lozenge_{j=1}^{m} g'_j :: S'_j\ \mathsf{od}$$

This can be compared with the rule for Dijkstra's traditional do-loop, where every alternative of the new loop must refine the demonic choice of the alternatives of the old loop (and the exit condition must be unchanged).

Two useful corollaries state that whenever every option is refined, iterative choice of these options is a refinement, and also that adding alternatives in iterative choice is a refinement.

Corollary 1 *For arbitrary state predicates $g_1, \ldots, g_n$ and $g'_1, \ldots, g'_n$, and arbitrary contract statements $S_1, \ldots, S_n$ and $S'_1, \ldots, S'_n$ we have that*

$$g_1 \subseteq g'_1 \wedge \ldots \wedge g_n \subseteq g'_n \ \wedge \ \{g_1\}; S_1 \sqsubseteq S'_1 \wedge \ldots \wedge \{g_n\}; S_n \sqsubseteq S'_n \ \Rightarrow$$
$$\mathsf{do}\ \lozenge_{i=1}^{n} g_i :: S_i\ \mathsf{od} \sqsubseteq \mathsf{do}\ \lozenge_{i=1}^{n} g'_i :: S'_i\ \mathsf{od}$$

Corollary 2 *For arbitrary state predicates $g_1, \ldots, g_{n+1}$ and arbitrary contract statements $S_1, \ldots, S_{n+1}$ we have that*

$$\mathsf{do}\ \lozenge_{i=1}^{n} g_i :: S_i\ \mathsf{od} \sqsubseteq \mathsf{do}\ \lozenge_{i=1}^{n+1} g_i :: S_i\ \mathsf{od}$$

The correctness rule for iterative choice states that for each ranked predicate which is stronger than the precondition there should be a choice decreasing the rank of this predicate or the possibility of establishing the postcondition directly:

$$\frac{p_w \ \subseteq \ \cup_{i=1}^{n}(g_i \cap S_i.\, p_{<w}) \cup q}{p\ \{\!|\ \mathsf{do}\ \lozenge_{i=1}^{n} g_i :: S_i\ \mathsf{od}\ |\!\}\ q} \qquad \textit{(iterative choice correctness rule)}$$

When the ranked predicates are written using an invariant I and a termination function t, this rule becomes

$$\frac{p \ \subseteq \ I \qquad I \cap t = w \ \subseteq \ \cup_{i=1}^{n}(g_i \cap S_i.\, (I \cap t < w)) \cup q}{p\ \{\!|\ \mathsf{do}\ \lozenge_{i=1}^{n} g_i :: S_i\ \mathsf{od}\ |\!\}\ q}$$

From the correctness rule we immediately get the iterative choice introduction rule

$$\frac{p_w \ \subseteq \ \cup_{i=1}^{n}(g_i \cap S_i.\, p_{<w}) \cup q[x' := x]}{\{p\}; [x := x' \mid q] \sqsubseteq \mathsf{do}\ \lozenge_{i=1}^{n} g_i :: S_i\ \mathsf{od}} \qquad \textit{(iterative choice introduction rule)}$$

where x does not occur free in q.

4.1 Modeling an Interactive Dialog Box

Suppose that we would like to describe a font selection dialog box, where the user is offered the choice of selecting a particular font and its size. The user can select a font by typing the font name in the entry field; the selection is

$$
\begin{array}{rcl}
\mathit{DialogBoxSpec} & = & [\mathit{fentry} \in \mathit{Fonts} \wedge \mathit{sentry} \in \mathit{Sizes}];\\
& & \mathsf{do\ true} :: \{\mathit{fentry} := s \mid s \in \mathit{Fonts}\}\\
& & \ \lozenge\ \mathsf{true} :: \{\mathit{sentry} := n \mid n \in \mathit{Sizes}\}\\
& & \mathsf{od}
\end{array}
$$

Figure 1: Specification of a dialog box

accepted if the entered font name belongs to the set of available fonts. The size of the font can also be chosen by typing the corresponding number in the entry field. The user may change the selections of both the font and the size any number of times before he presses the OK button, which results in closing the dialog box and changing the corresponding text according to the last selection. Using the iterative choice statement, we can model this kind of a dialog box as shown in Fig.1. In this specification $\mathit{fentry} : \mathsf{String}$ and $\mathit{sentry} : \mathsf{Nat}$ are global variables representing current selections of the font name and its size in the corresponding entry fields of the dialog box. The constants $\mathit{Fonts} : \mathsf{set\ of\ String}$ and $\mathit{Sizes} : \mathsf{set\ of\ Nat}$ represent sets of available font names and font sizes. When the user opens the dialog box, he assumes that the default entries for the font name and size are among those available in the system, as expressed by the corresponding assumption in *DialogBoxSpec*. If this assumption is met by the system, the user may enter new font name, or new font size, or leave the current selections intact. The user may select any alternative any number of times until he is satisfied with the choice and decides to stop the iteration. Note that to model dialog closing, we do not need to explicitly maintain a boolean variable *Ok_pressed*, have all the options enabled only when $\neg \mathit{Ok\_pressed}$ holds, and set it explicitly to true to terminate iteration: all this is implicit in the model.

Suppose now that the font "Times" belongs to *Fonts* and the size 12 is in *Sizes*. Can the user, by making the right choices, select this font with this size? The answer to this question can be given by verifying the following total correctness assertion:

$$
\begin{array}{c}\text{“Times”} \in \mathit{Fonts}\ \cap\\ 12 \in \mathit{Sizes}\end{array}
\ \{\!|\ \begin{array}{l}\mathsf{do\ true} :: \{\mathit{fentry} := s \mid s\ \mathsf{in}\ \mathit{Fonts}\}\\ \ \lozenge\ \mathsf{true} :: \{\mathit{sentry} := n \mid n\ \mathsf{in}\ \mathit{Sizes}\}\\ \mathsf{od}\end{array}\ |\!\}\
\begin{array}{c}\mathit{fentry} = \text{“Times”} \cap\\ \mathit{sentry} = 12\end{array}
$$

Using the rule for the correctness of iterative choice with the invariant I and the termination function t such that

$$
\begin{array}{rcl}
I & = & \text{“Times”} \in \mathit{Fonts}\ \cap\ 12 \in \mathit{Sizes}\\
t & = & \#(\{\text{“Times”}, 12\} \setminus \{\mathit{fentry}, \mathit{sentry}\})
\end{array}
$$

we then need to prove two subgoals:

1. $\text{“Times”} \in \mathit{Fonts}\ \cap\ 12 \in \mathit{Sizes}\ \subseteq\ I$
2. $\begin{array}{rcl} I \cap t = w & \subseteq & \mathsf{true} \cap \{\mathit{fentry} := s \mid s\ \mathsf{in}\ \mathit{Fonts}\}.\,(I\ \cap\ t < w)\ \cup\\ & & \mathsf{true} \cap \{\mathit{sentry} := n \mid n\ \mathsf{in}\ \mathit{Sizes}\}.\,(I\ \cap\ t < w)\ \cup\\ & & \mathit{fentry} = \text{“Times”}\ \cap\ \mathit{sentry} = 12 \end{array}$

The first subgoal states that the precondition is stronger than the invariant and is trivially true. The second subgoal states that, when the invariant holds, at least one of the alternatives will decrease the termination function while preserving the invariant. It can be proved by using the definition of angelic relational update and rules of logic.

4.2 Modeling Component Environment

To demonstrate how the iterative choice statement can be used to model a component environment, let us first introduce the notion of a component. We view a component as an abstract data type with internal state and methods that can be invoked on the component to carry out certain functionality and (possibly) change the component's state.

$$
\begin{array}{ll}
c \;=\; \textsf{component} & \\
\quad x : \Sigma := x_0 & \\
\quad m_1\,(\textsf{val}\ x_1 : \Gamma_1, \textsf{res}\ y_1 : \Delta_1) \;=\; M_1, & \\
\quad \ldots & \\
\quad m_n\,(\textsf{val}\ x_n : \Gamma_n, \textsf{res}\ y_n : \Delta_n) \;=\; M_n & \\
\textsf{end} &
\end{array}
$$

Here $x : \Sigma$ are the variables which carry the internal component's state. These variables have some initial values x_0. Methods named $m_1, \ldots, m_n$ are specified by statements $M_1, \ldots, M_n$ respectively. Invocation of a method on a component has a standard procedure call semantics, with the only difference that the value of a component itself is passed as a value-result argument. We will denote invocation of m_i on c with value and result arguments $v : \Gamma_i$ and $r : \Delta_i$ by $c.m_i(v, r)$.

An environment using a component c does so by invoking its methods. Every time the environment has a choice of which method to choose for execution. In general, each option is preceded with an assertion which determines whether the option is enabled in a particular state. While at least one of the assertions holds, the environment may repeatedly choose a particular option which is enabled and have it executed. The environment decides on its own when it is willing to stop choosing options. Such iterative choice of method invocations, followed by arbitrary statements not affecting the component state directly, describes all the actions the environment program might undertake:

$$\textsf{begin var}\ l : \Lambda \bullet p; \textsf{do}\ \ \lozenge_{i=1}^{n} q_i :: c.m_i(g_i, d_i); L_i\ \textsf{od end}$$

Here $l : \Lambda$ are some local variables initialized according to p, predicates $q_1 \ldots q_n$ are the asserted conditions on the state, and statements L_1 through L_n are arbitrary. The initialization p, the assertions $q_1 \ldots q_n$, and the statements $L_1, \ldots, L_n$ do not refer to c, which is justified by the assumption that the component state is encapsulated.

This program statement is a contract between the component c and *any environment* using c. The method enabledness condition q_i corresponds to the assumptions made by the the corresponding method m_i, as stated in its subcontract (the method body definition). For example, in a component *EntryField* the method *SetLength*(val l : Nat) can begin with an assumption that the length l does not exceed some constant value *lmax*. An environment invoking *SetLength* on *EntryField* will then have to assert that the specific *length* does indeed satisfy this requirement:

$$\mathsf{do}\ \mathit{length} \le \mathit{lmax} :: \mathit{EntryField}.\mathit{SetLength}(\mathit{length}); \ldots\ \mathsf{od}$$

The assumption of this condition in the body of *SetLength* will pass through, as $\{p\}; [p] = \{p\}$, for all predicates p.

5 Data Refinement of Iterative Choice

Data refinement is a general technique by which one can change data representation in a refinement. A contract statement S may begin in a state space Σ and end in a state space Γ, written $S : \Sigma \mapsto \Gamma$. Assume that contract statements S and S' operate on state spaces Σ and Σ' respectively, i.e. $S : \Sigma \mapsto \Sigma$ and $S' : \Sigma' \mapsto \Sigma'$. Let $R : \Sigma' \to \Sigma \to \mathsf{Bool}$ be a relation between the state spaces Σ' and Σ. Following [2], the statement S is said to be *data refined* by the statement S' via the relation R, denoted $S \sqsubseteq_{\{R\}} S'$, if $\{R\}; S \sqsubseteq S'; \{R\}$. An alternative and equivalent characterization of data refinement using the inverse relation R^{-1} arises from the fact that $\{R\}$ and $[R^{-1}]$ are each others inverses, in the sense that $\{R\}; [R^{-1}] \sqsubseteq \mathsf{skip}$ and $\mathsf{skip} \sqsubseteq [R^{-1}]; \{R\}$. Abbreviating $\{R\}; S; [R^{-1}]$ by $S{\downarrow}\{R\}$ we have that

$$S \sqsubseteq_{\{R\}} S' \;\equiv\; S{\downarrow}\{R\} \sqsubseteq S'$$

We will call D an *abstraction statement* if D is such that $D = \{R\}$, for some R (in this case, our notion of data refinement is the standard one, often referred to as forward data refinement or downward simulation). For D such that $D = \{R\}$, we will denote its inverse by $\overline{D}$, so that $\overline{D} = [R^{-1}]$.

Data refinement properties of the angelic iteration and the iterative choice statement cannot be proved directly by a duality argument from the corresponding results for traditional iteration operators. However, they can still be proved:

Theorem 4 *Assume that S and D are monotonic predicate transformers and that D is an abstraction statement. Then*

$$S^{\phi}{\downarrow}D \;\sqsubseteq\; (S{\downarrow}D)^{\phi}$$

As a consequence, the angelic iteration operator preserves data refinement:

Corollary 3 *Assume that S, S' and D are monotonic predicate transformers and that D is an abstraction statement. Then*

$$S \sqsubseteq_D S' \;\Rightarrow\; S^{\phi} \sqsubseteq_D S'^{\phi}$$

Proofs of Theorem 4 and Corollary 3 are presented in the Appendix.

Data refinement rules for iterative choice also arise from the corresponding rules for angelic iteration. First, data refinement can be propagated inside iterative choice:

Theorem 5 *Assume that $g_1, \ldots, g_n$ are arbitrary state predicates, $S_1, \ldots, S_n$ are arbitrary contract statements, and D is an abstraction statement. Then*

$$\mathsf{do}\ \between_{i=1}^{n} g_i :: S_i\ \mathsf{od}{\downarrow}D \ \sqsubseteq\ \mathsf{do}\ \between_{i=1}^{n} D.\, g_i :: S_i{\downarrow}D\ \mathsf{od}$$

A more general rule shows how the proof of data refinement between iterative choices can be reduced to proofs of data refinement between the iterated alternatives.

Theorem 6 *Assume that $g_1, \ldots, g_n$ and $g'_1, \ldots, g'_m$ are arbitrary state predicates, $S_1, \ldots, S_n$ and $S'_1, \ldots, S'_m$ are arbitrary contract statements, and D is an abstraction statement. Then*

$$\begin{array}{l}(\forall i \mid 1 \le i \le n \bullet \{g_i\}; S_i \sqsubseteq_D \sqcup_{j=1}^{m} \{g'_j\}; S'_j) \ \Rightarrow \\ \quad \mathsf{do}\ \between_{i=1}^{n} g_i :: S_i\ \mathsf{od} \sqsubseteq_D \mathsf{do}\ \between_{j=1}^{m} g'_j :: S'_j\ \mathsf{od}\end{array}$$

Proofs of Theorems 5 and 6 are presented in the Appendix. A useful special case of these theorems is when the number of choices is the same and they are refined one by one.

Corollary 4 *Assume that $g_1, \ldots, g_n$ and $g'_1, \ldots, g'_n$ are arbitrary state predicates, $S_1, \ldots, S_n$ and $S'_1, \ldots, S'_n$ are arbitrary contract statements, and D is an abstraction statement. Then*

$$\begin{array}{l}D.\, g_1 \subseteq g'_1 \wedge \ldots \wedge D.\, g_n \subseteq g'_n \ \wedge\ \{g_1\}; S_1 \sqsubseteq_D S'_1 \wedge \ldots \wedge \{g_n\}; S_n \sqsubseteq_D S'_n \ \Rightarrow \\ \quad \mathsf{do}\ \between_{i=1}^{n} g_i :: S_i\ \mathsf{od} \sqsubseteq_D \mathsf{do}\ \between_{i=1}^{n} g'_i :: S'_i\ \mathsf{od}\end{array}$$

Data Refinement of the Interactive Dialog Box

Let us now demonstrate how our original specification of a dialog box can be data refined to a more concrete one. Suppose that we would like to describe a dialog box, where the user can select a font by choosing it from the list of available fonts or by typing the font name in the entry field. The size of the font can also be chosen either from the list of sizes or by typing the corresponding number in the entry field. Using the iterative choice statement, we can model this kind of a dialog box as shown in Fig.2. In this specification $\mathit{fonts} : \mathsf{array}\ 1..\mathit{fmax}\ \mathsf{of}\ \mathsf{String}$ and $\mathit{sizes} : \mathsf{array}\ 1..\mathit{smax}\ \mathsf{of}\ \mathsf{Nat}$ are used to represent lists of the corresponding items. When the user opens the dialog box, the system initializes *fonts* and *sizes* to contain elements from the constant sets *Fonts* and *Sizes*. The function *array_to_set*, used for this purpose, is given as follows:

$$\mathit{array\_to\_set} \ = \ (\lambda\,(a, n)\,.\,\{e \mid \exists\, i \bullet 1 \le i \le n \wedge a[i] = e\})$$

$$
\begin{array}{lll}
DialogBox & = & [fentry, sentry, fonts, sizes, fpos, spos := \\
& & \quad fentry', sentry', fonts', sizes', fpos', spos' \mid \\
& & \quad\quad array\_to\_set(fonts', fmax) = Fonts \ \wedge \ \#Fonts = fmax \wedge \\
& & \quad\quad array\_to\_set(sizes', smax) = Sizes \ \wedge \ \#Sizes = smax \wedge \\
& & \quad\quad fentry' = fonts'[fpos'] \ \wedge \ sentry' = sizes'[spos'] \wedge \\
& & \quad\quad 1 \le fpos' \le fmax \ \wedge \ 1 \le spos' \le smax]; \\
& & \mathsf{do}\ \mathsf{true} :: \{fentry := fentry' \mid \exists\, i \bullet 1 \le i \le fmax \ \wedge \ fonts[i] = fentry'\}; \\
& & \qquad\qquad [fpos := fpos' \mid fonts[fpos'] = fentry] \\
& & \ \ \Diamond\ \mathsf{true} :: \{sentry := sentry' \mid \exists\, i \bullet 1 \le i \le smax \ \wedge \ sizes[i] = sentry'\}; \\
& & \qquad\qquad [spos := spos' \mid sizes[spos'] = sentry] \\
& & \ \ \Diamond\ \mathsf{true} :: \{fpos := fpos' \mid 1 \le fpos' \le fmax\}; fentry := fonts[fpos] \\
& & \ \ \Diamond\ \mathsf{true} :: \{spos := spos' \mid 1 \le spos' \le smax\}; sentry := sizes[spos] \\
& & \mathsf{od}
\end{array}
$$

Figure 2: Specification of a dialog box refinement

The initialization conditions $\#Fonts = fmax$ and $\#Sizes = smax$ state, in addition, that the arrays contain exactly as many elements as the corresponding constant sets. Indices $fpos : \mathsf{Nat}$ and $spos : \mathsf{Nat}$ represent the currently chosen selections in the corresponding arrays and are initialized to index some items in $fonts$ and $sizes$; the variables $fentry$ and $sentry$ are initialized with values of these items. The implicit invariant maintained by $DialogBox$ states that $fonts[fpos] = fentry$ and $sizes[spos] = sentry$, i.e. the currently selected font in the list of available fonts is the same as the one currently typed in the font entry field, and similarly for font sizes.

The iterative choice statement is the contract stipulating the interaction between the user making choices and the system reacting to these choices. Consider, for example, the case when the user wants to select a font by directly choosing it from the list of available fonts, modeled by the third alternative. First, the user is offered to pick an index $fpos'$, identifying a certain font in the list of fonts, and then the system updates the variable $fentry$ to maintain the invariant $fonts[fpos] = fentry$.

The abstraction relation coercing the state of $DialogBox$ to the state of $DialogBoxSpec$ is essentially an invariant on the concrete variables:

$$
\begin{array}{l}
array\_to\_set(fonts, fmax) = Fonts \ \wedge \ \#Fonts = fmax \ \wedge \ 1 \le fpos \le fmax \wedge \\
array\_to\_set(sizes, smax) = Sizes \ \wedge \ \#Sizes = smax \ \wedge \ 1 \le spos \le smax \wedge \\
fentry = fonts[fpos] \ \wedge \ sentry = sizes[spos]
\end{array}
$$

Strictly speaking, we should distinguish between $fentry, sentry$ of $DialogBoxSpec$ and $fentry, sentry$ of $DialogBox$; the abstraction relation also includes the conditions $fentry = fentry_0$ and $sentry = sentry_0$, where $fentry_0$ and $sentry_0$ denote $fentry$ and $sentry$ of $DialogBoxSpec$. It can be shown that $DialogBoxSpec \sqsubseteq_{\{R\}} DialogBox$, where

$$
\begin{aligned}
R\ concrete\ abstract \ = \ & array\_to\_set(fonts, fmax) = Fonts \ \wedge \ \#Fonts = fmax \ \wedge \\
& array\_to\_set(sizes, smax) = Sizes \ \wedge \ \#Sizes = smax \ \wedge \\
& 1 \le fpos \le fmax \ \wedge \ 1 \le spos \le smax \ \wedge \\
& fentry = fonts[fpos] \ \wedge \ sentry = sizes[spos] \ \wedge \\
& fentry = fentry_0 \ \wedge \ sentry = sentry_0
\end{aligned}
$$

with $concrete = fentry, sentry, fonts, sizes, fpos, spos$ and $abstract = fentry_0$, $sentry_0$.

6 Conclusions and Related Work

We have described an interactive computing system in terms of contracts binding participating agents and stipulating their obligations and assumptions. In particular, we have focused on the iterative choice contract and studied its algebraic properties and modeling capabilities. This work extends [3] where Back and von Wright introduced the notions of correctness and refinement for contracts and defined their weakest precondition semantics.

The notion of contracts is based on the fundamental duality between demonic and angelic nondeterminism (choices of different agents), abortion (breaching a contract), and miracles (being released from a contract). The notion of angelic nondeterminism goes back to the theory of nondeterministic automata and the nondeterministic programs of Floyd [7]. Broy in [6] discusses the use of demonic and angelic nondeterminism with respect to concurrency. Some applications of angelic nondeterminism are shown by Ward and Hayes in [10]. Adabi, Lamport, and Wolper in [1] study realizability of specifications, considering them as "determined" games, where the system plays against the environment and wins if it produces a correct behavior. Specifications are identified with the properties that they specify, and no assumptions are made about how they are written. Moschovakis in [9] studies non-deterministic interaction in concurrent communication also considering it from the game-theoretic perspective.

Another direction of related work concentrates on studying the role of interaction in computing systems. Wegner in [11] proposes to use *interaction machines* as "a formal framework for interactive models". Interaction machines are described as extensions of Turing machines with unbounded input streams, which "precisely capture fuzzy concepts like open systems and empirical computer science". The main thesis of work presented in [11] and further developed in [12] is that "Logic is too weak to model interactive computation" and, instead, empirical models should be used for this purpose. Apparently, first-order logic is meant by the author, which is indeed too weak for modeling interaction. However, our formalization is based on an extension of higher-order logic and, as such, is perfectly suitable for this purpose. Also, it is claimed in [11] that "Interaction machines are incomplete in the sense of Gödel: their nonenumerable number of true statements cannot be enumerated by a set of theorems. [...] The incompleteness of interactive systems implies that proving correctness is not merely hard but impossible." We believe that our work presents a proof to the contrary.

References

[1] M. Abadi, L. Lamport, and P. Wolper. Realizable and unrealizable specifications of reactive systems. In *Proceedings of 16th ICALP*, volume 372 of *LNCS*, pages 1–17, Stresa, Italy, 11–15 July 1989. Springer-Verlag.

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

[3] R. J. R. Back and J. von Wright. Contracts, games and refinement. Technical Report 138, Turku Centre for Computer Science, October 1997.

[4] R. J. R. Back and J. von Wright. Reasoning algebraically about loops. Technical Report 144, Turku Centre for Computer Science, November 1997.

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

[6] M. Broy. A theory for nondeterminism, parallelism, communication, and concurrency. *Theoretical Computer Science*, 45:1–61, 1986.

[7] R. Floyd. Assigning meanings to programs. In Schwartz, editor, *Proc. ACM Symp. on Applied Mathematics*, number 19, pages 19–32, 1967.

[8] C. C. Morgan. *Programming from Specifications*. Prentice–Hall, 1990.

[9] Y. N. Moschovakis. A model of concurrency with fair merge and full recursion. *Information and Computation*, 93(1):114–171, July 1991.

[10] N. Ward and I. Hayes. Applications of angelic nondeterminism. In P.A.C.Bailes, editor, *6th Australian Software Engineering Conference*, pages 391–404, Sydney, Australia, 1991.

[11] P. Wegner. Interactive software technology. In J. Allen B. Tucker, editor, *The Computer Science and Engineering Handbook*. CRC Press, in cooperation with ACM, 1997.

[12] P. Wegner. Interactive foundations of computing. *Theoretical Computer Science*, 192(2):315–351, Feb. 1998.

Appendix

Theorem 1 *Let S be an arbitrary monotonic predicate transformer. Then*

$$S^{\phi} \;=\; ((S^{\circ})^{*})^{\circ}$$

Proof

$$\begin{aligned}
& ((S^{\circ})^{*})^{\circ} \\
= & \{\text{definition of weak iteration}\} \\
& (\nu X \bullet S^{\circ}; X \sqcap \mathsf{skip})^{\circ} \\
= & \{\text{least fixpoint is the dual of the greatest fixpoint}\} \\
& (\mu X \bullet (S^{\circ}; X^{\circ} \sqcap \mathsf{skip})^{\circ})) \\
= & \{\text{properties of the dual operator}\} \\
& (\mu X \bullet S; X \sqcup \mathsf{skip}) \\
= & \{\text{definition of angelic iteration}\} \\
& S^{\phi}
\end{aligned}$$

□

Theorem 3 *For arbitrary state predicates $g_1, \ldots, g_n$ and $g'_1, \ldots, g'_m$, and arbitrary contract statements $S_1, \ldots, S_n$ and $S'_1, \ldots, S'_m$ we have that*

$$\begin{aligned}
&(\forall i \,|\, 1 \leq i \leq n \bullet \{g_i\}; S_i \;\sqsubseteq\; \sqcup_{j=1}^{m}\{g'_j\}; S'_j) \;\Rightarrow \\
&\quad \mathsf{do}\ \lozenge_{i=1}^{n} g_i :: S_i\ \mathsf{od} \;\sqsubseteq\; \mathsf{do}\ \lozenge_{j=1}^{m} g'_j :: S'_j\ \mathsf{od}
\end{aligned}$$

Proof

$$\begin{aligned}
& \mathsf{do}\ \lozenge_{i=1}^{n} g_i :: S_i\ \mathsf{od} \;\sqsubseteq\; \mathsf{do}\ \lozenge_{j=1}^{m} g'_j :: S'_j\ \mathsf{od} \\
= & \{\text{definition of iterative choice}\} \\
& (\sqcup_{i=1}^{n}\{g_i\}; S_i)^{\phi} \;\sqsubseteq\; (\sqcup_{j=1}^{m}\{g'_j\}; S'_j)^{\phi} \\
\Leftarrow & \{\text{Theorem 2(f)}\} \\
& \sqcup_{i=1}^{n}\{g_i\}; S_i \;\sqsubseteq\; \sqcup_{j=1}^{m}\{g'_j\}; S'_j \\
\Leftarrow & \{\text{use assumption to change every } \{g_i\}; S_i \text{ to its refinement}\} \\
& \sqcup_{i=1}^{n}(\sqcup_{j=1}^{m}\{g'_j\}; S'_j) \;\sqsubseteq\; \sqcup_{j=1}^{m}\{g'_j\}; S'_j \\
= & \{\text{logic}\} \\
& \mathsf{T}
\end{aligned}$$

□

In the following proofs we will use the rules

$(\sqcup_{i=1}^{n} S_i)\!\downarrow\! D \;\sqsubseteq\; \sqcup_{i=1}^{n} S_i\!\downarrow\! D$ (*data refinement for angelic choice*)

$(S;T)\!\downarrow\! D \;\sqsubseteq\; S\!\downarrow\! D; T\!\downarrow\! D$ (*data refinement for sequential composition*)

$\{p\}\!\downarrow\! D \;\sqsubseteq\; \{D.\,p\}$ (*data refinement for assertion*)

Theorem 4 *Assume that S and D are monotonic predicate transformers and that D is an abstraction statement. Then*

$$S^{\phi}\!\downarrow\! D \;\sqsubseteq\; (S\!\downarrow\! D)^{\phi}$$

Proof

$$\begin{aligned}
& S^{\phi}\!\downarrow\! D \\
\sqsubseteq\; & \{\text{general rule } \mathsf{skip} \sqsubseteq \overline{D}; D\} \\
& (S;\overline{D};D)^{\phi}\!\downarrow\! D \\
=\; & \{\text{definition of } \downarrow\} \\
& D;(S;\overline{D};D)^{\phi};\overline{D} \\
=\; & \{\text{leapfrog property, } D \text{ is disjunctive}\} \\
& (D;S;\overline{D})^{\phi};D;\overline{D} \\
\sqsubseteq\; & \{\text{general rule } D;\overline{D} \sqsubseteq \mathsf{skip}\} \\
& (S\!\downarrow\! D)^{\phi}
\end{aligned}$$

□

Corollary 3 *Assume that S, S' and D are monotonic predicate transformers and that D is an abstraction statement. Then*

$$S \sqsubseteq_D S' \;\Rightarrow\; S^{\phi} \sqsubseteq_D S'^{\phi}$$

Proof

$$\begin{aligned}
& S^{\phi} \sqsubseteq_D S'^{\phi} \\
=\; & \{\text{definition of } \downarrow\} \\
& S^{\phi}\!\downarrow\! D \sqsubseteq S'^{\phi} \\
\Leftarrow\; & \{\text{Theorem 4}\} \\
& (S\!\downarrow\! D)^{\phi} \sqsubseteq S'^{\phi} \\
\Leftarrow\; & \{\text{Theorem 2(f)}\} \\
& S\!\downarrow\! D \sqsubseteq S' \\
=\; & \{\text{definition of } \downarrow\} \\
& S \sqsubseteq_D S'
\end{aligned}$$

□

Theorem 5 *Assume that $g_1, \ldots, g_n$ are arbitrary state predicates, $S_1, \ldots, S_n$ are arbitrary contract statements, and D is an abstraction statement. Then*

$$\mathsf{do}\ \between_{i=1}^{n} g_i :: S_i\ \mathsf{od}{\downarrow}D \ \sqsubseteq\ \mathsf{do}\ \between_{i=1}^{n} D.\, g_i :: S_i{\downarrow}D\ \mathsf{od}$$

Proof

$$\begin{aligned}
& \quad \mathsf{do}\ \between_{i=1}^{n} g_i :: S_i\ \mathsf{od}{\downarrow}D \\
= & \ \{\text{definition of iterative choice}\} \\
& \quad (\sqcup_{i=1}^{n}\{g_i\}; S_i)^{\phi}{\downarrow}D \\
\sqsubseteq & \ \{\text{Theorem 4}\} \\
& \quad ((\sqcup_{i=1}^{n}\{g_i\}; S_i){\downarrow}D)^{\phi} \\
\sqsubseteq & \ \{\text{data refinement for angelic choice rule}\} \\
& \quad (\sqcup_{i=1}^{n}(\{g_i\}; S_i){\downarrow}D)^{\phi} \\
\sqsubseteq & \ \{\text{data refinement for sequential composition rule}\} \\
& \quad (\sqcup_{i=1}^{n}\{g_i\}{\downarrow}D; S_i{\downarrow}D)^{\phi} \\
\sqsubseteq & \ \{\text{data refinement for assertion rule}\} \\
& \quad (\sqcup_{i=1}^{n}\{D.\, g_i\}; S_i{\downarrow}D)^{\phi} \\
= & \ \{\text{definition of iterative choice}\} \\
& \quad \mathsf{do}\ \between_{i=1}^{n} D.\, g_i :: S_i{\downarrow}D\ \mathsf{od}
\end{aligned}$$

□

Theorem 6 *Assume that $g_1, \ldots, g_n$ and $g'_1, \ldots, g'_m$ are arbitrary state predicates, $S_1, \ldots, S_n$ and $S'_1, \ldots, S'_m$ are arbitrary contract statements, and D is an abstraction statement. Then*

$$\begin{aligned}
&(\forall i \,|\, 1 \le i \le n \bullet \{g_i\}; S_i \sqsubseteq_D \sqcup_{j=1}^{m}\{g'_j\}; S'_j) \ \Rightarrow \\
&\qquad \mathsf{do}\ \between_{i=1}^{n} g_i :: S_i\ \mathsf{od} \sqsubseteq_D \mathsf{do}\ \between_{j=1}^{m} g'_j :: S'_j\ \mathsf{od}
\end{aligned}$$

Proof

Since $S \sqsubseteq_D T \ \equiv\ S{\downarrow}D \sqsubseteq T$, we prove an equivalent goal

$$\begin{aligned}
&(\forall i \,|\, 1 \le i \le n \bullet (\{g_i\}; S_i){\downarrow}D \sqsubseteq \sqcup_{j=1}^{m}\{g'_j\}; S'_j) \ \Rightarrow \\
&\qquad \mathsf{do}\ \between_{i=1}^{n} g_i :: S_i\ \mathsf{od}{\downarrow}D \sqsubseteq \mathsf{do}\ \between_{j=1}^{m} g'_j :: S'_j\ \mathsf{od}
\end{aligned}$$

as follows:

$$
\begin{aligned}
& \quad \mathsf{do}\ \between_{i=1}^{n} g_i :: S_i\ \mathsf{od}{\downarrow}D \\
= & \ \{\text{definition of iterative choice}\} \\
& \quad (\sqcup_{i=1}^{n}\{g_i\};S_i)^{\phi}{\downarrow}D \\
\sqsubseteq & \ \{\text{Theorem 4}\} \\
& \quad ((\sqcup_{i=1}^{n}\{g_i\};S_i){\downarrow}D)^{\phi} \\
\sqsubseteq & \ \{\text{data refinement for angelic choice rule}\} \\
& \quad (\sqcup_{i=1}^{n}(\{g_i\};S_i){\downarrow}D)^{\phi} \\
\sqsubseteq & \ \{\text{use assumption to change every } (\{g_i\};S_i){\downarrow}D \text{ to its refinement}\} \\
& \quad (\sqcup_{i=1}^{n}(\sqcup_{j=1}^{m}\{g'_j\};S'_j))^{\phi} \\
= & \ \{\text{logic and definition of iterative choice}\} \\
& \quad \mathsf{do}\ \between_{j=1}^{m} g'_j :: S'_j\ \mathsf{od}
\end{aligned}
$$

□

Turku Centre for Computer Science
Lemminkäisenkatu 14
FIN-20520 Turku
Finland

http://www.tucs.abo.fi

University of Turku
- **Department of Mathematical Sciences**

Åbo Akademi University
- **Department of Computer Science**
- **Institute for Advanced Management Systems Research**

Turku School of Economics and Business Administration
- **Institute of Information Systems Science**