Handshaking Expansion as Action System Refinement

Rimvydas Rukšėnas

Åbo Akademi University, Department of Computer Science,
Lemminkäisenkatu 14, FIN-20520 Turku, Finland

Kaisa Sere

Åbo Akademi University, Department of Computer Science,
Lemminkäisenkatu 14, FIN-20520 Turku, Finland

TUCS

Turku Centre for Computer Science
TUCS Technical Report No 55
October 1996

ISBN 951-650-864-2
ISSN 1239-1891

Abstract

The *action system* framework is a formalism which can model asynchronous (quasi) delay-insensitive circuits. It is expressive enough to give both high level specification of the circuit and its implementation up to the gate level. A derivation of the gate level description from the specification is based on the *Refinement Calculus*. In the paper we present the *handshaking expansion* as a refinement of action systems and thus prove *correctness* of the transformation.

Keywords: action systems, asynchronous circuits, refinement

TUCS Research Group
Programming Methodology Research Group

1 Introduction

Martin [12, 13] has showed that techniques familiar from parallel programming are suitable for asynchronous circuit design as well. Following his approach, a circuit is constructed using a number of transformation steps. One of the steps is called *handshaking expansion*. Eventhough a formal language is used to carry out the transformations, reasoning about the correctness of them is informal. The purpose of this paper is to prove the correctness of the handshaking expansion. The proof is carried out within the *Refinement Calculus* for *action systems*.

Action systems have shown their strength in the construction of parallel and distributed systems [3, 5], as well as in the initial steps of asynchronous circuit design [4, 16]. In an accompanying paper, Plosila et al. [15] propose a method to synthesize delay-insensitive circuits from high-level action system specifications.

Action systems are similar to UNITY programs [8] that have an associated temporal logic as a specification language. The design and reasoning about action systems is carried out within the Refinement Calculus that is based on the use of predicate transformers. The Refinement Calculus was originally developed for sequential programs only by several researchers [1, 11, 14]. Later it has been extended in various ways, again by many researchers. In this paper we use as our basis the Refinement Calculus for reactive action systems as described by Back [2]. The Refinement Calculus allows us to work within one formalism when reasoning about specifications, programs and implementations. Furthermore, the calculus has been mechanized within the HOL proof assistant [10] giving us a way to carry out the proofs mechanically.

Overview of the paper We proceed as follows. In section 2 we introduce action systems and some notational conventions used in the paper. In section 3 we then define refinement of action system to be a transformation preserving behaviour of the original system. In section 4, we describe how communication between circuit components is represented in the action systems. In section 5, the handshaking expansion is shown to be valid by exhibiting a refinement relation between the original and the transformed action systems. In section 6, we show the handshaking expansion for a circuit component called a sequencer. We end in section 7 with some concluding remarks.

2 Action systems

Action systems The action systems formalism is based on an extended version of Dijkstra's guarded command language [9]. This language includes assignment, sequential composition, conditional choice, and iteration. These program statements are defined using *weakest precondition* predicate transformers.

An action A is a *guarded command*, i.e. a statement of the form

$$g \to S$$

where g is a boolean condition, the *guard*, and S is a program statement, the *body*. The weakest precondition of the action is defined as follows:

$$\mathrm{wp}(A, q) \quad \widehat{=} \quad g \Rightarrow \mathrm{wp}(S, q).$$

An action is said to be *enabled* if the guard is *true*, and *disabled* if the guard is *false*. The guard and the body of an action A are denoted gA and sA, respectively.

A *reactive action system* is a program of the form

$$\mathcal{A} :: \ \ |[\ \mathbf{var}\ x.x := x_0;\ \mathbf{do}\ A_1\ [\!]\ \ldots\ [\!]\ A_n\ \mathbf{od}\ ]| : z$$

The identifiers x are called *local variables* of $\mathcal{A}$ and initialized to x_0. The list z contains variables which are referenced both by the system $\mathcal{A}$ and its environment. They are called the *global variables* of $\mathcal{A}$. The local and global variables are assumed to be distinct. The *state variables* of $\mathcal{A}$ consist now of the local variables and the global variables. Within the non-deterministic loop, $A_1, \ldots, A_n$ are the actions of $\mathcal{A}$. The disjunction of guards of the actions $A_1, \ldots, A_n$ is denoted $g\mathcal{A}$.

The behaviour of the action system $\mathcal{A}$ is the following. First, the local variables x are initialized. Then, repeatedly, an enabled action from $A_1, \ldots, A_n$ is nondeterministically selected and executed. Action system without environment terminates when no action in enabled. If two actions are independent, i.e., they do not have any variables in common, they can be executed in parallel [5]. Their parallel execution is then equivalent to executing the actions one after the other, in either order.

Notation We often want to indicate that actions of the system $\mathcal{A}$ have to be executed sequentially. The notation of *sequential action systems* is introduced for such cases:

$$\mathcal{A} :: \ \ |[\ \mathbf{do}\ A_1;\ \ldots;\ A_n\ \mathbf{od}\ ]| : z$$

The notation hides a local variable pc in the actions of $\mathcal{A}$ and actually stands for the following action system:

$$
\begin{array}{lll}
\mathcal{A} :: & |[& \textbf{var}\ pc.\ pc := 1; \\
 & & \textbf{do}\quad pc = 1 \wedge gA_1 \rightarrow sA_1;\ pc := 2 \\
 & & \quad ; \quad \ldots \\
 & & \quad ; \quad pc = n \wedge gA_n \rightarrow sA_n;\ pc := 1 \\
 & & \textbf{od} \\
 &]|: z &
\end{array}
$$

The hidden variable pc is referred to as the *program counter*. We also use the notation $a\!\uparrow$, $a\!\downarrow$ to represent the assignments $a := true$ and $a := false$, respectively.

Composing action systems Consider two action systems,

$$
\begin{array}{ll}
\mathcal{A} :: & |[\ \textbf{var}\ x.\ x := x_0;\ \textbf{do}\ A_1\ [\!]\ \ldots\ [\!]\ A_n\ \textbf{od}\ ]| : z \\
\mathcal{B} :: & |[\ \textbf{var}\ y.\ y := y_0;\ \textbf{do}\ B_1\ [\!]\ \ldots\ [\!]\ B_k\ \textbf{od}\ ]| : w
\end{array}
$$

where $x \cap y = \emptyset$. We define the *parallel composition* $\mathcal{A} \parallel \mathcal{B}$ of $\mathcal{A}$ and $\mathcal{B}$ to be the action system $\mathcal{C}$

$$
\begin{array}{lll}
\mathcal{C} :: & |[& \textbf{var}\ x, y.\ x := x_0;\ y := y_0; \\
 & & \textbf{do}\ A_1\ [\!]\ \ldots\ [\!]\ A_n\ [\!]\ B_1\ [\!]\ \ldots\ [\!]\ B_k\ \textbf{od} \\
 &]|: z, w &
\end{array}
$$

The behaviour of a parallel composition of action systems is dependent on how the individual action systems, the *reactive components*, interact with each other via the shared global variables. We have for instance that a reactive component does not terminate by itself: termination is a global property of the composed action system. More on these topics can be found in [2].

3 Refinement of action systems

A formal basis for the stepwise development of action systems is the *Refinement Calculus*. In the Refinement Calculus program statements are identified with their weakest precondition predicate transformers.

Simulation Let S be statement on the program variables a and u whereas S' is statement on the variables c and u. Assume $R(a, c)$ is a relation between

a and c. We say that S is *simulated* by S' using the abstraction relation $R(a, c)$, denoted $S \leq_R S'$, if for any q

$$R \wedge \mathrm{wp}(S, q) \Rightarrow \mathrm{wp}(S', \exists a.R \wedge q).$$

Note that $\exists a.R \wedge q$ is a predicate on the program variables c and u.

From the definition of an action we have that an action A is simulated by an action A' using a relation $R(a, c)$, denoted $A \leq_R A'$, if for any q

(i) Guard: $R \wedge gA' \Rightarrow gA$,

(ii) Body: $R \wedge gA' \wedge \mathrm{wp}(sA, q) \Rightarrow \mathrm{wp}(sA', \exists a.R \wedge q)$.

Refinement of action systems Consider two action systems

$$\mathcal{A} :: \quad |[\ \mathbf{var}\ a.\ a := a_0;\ \ \mathbf{do}\ A_1\ [\!]\ \ \ldots\ [\!]\ \ A_n\ \mathbf{od}\ ]| : u$$

$$\mathcal{C} :: \quad |[\ \mathbf{var}\ c.\ c := c_0;\ \ \mathbf{do}\ C_1\ [\!]\ \ \ldots\ [\!]\ \ C_n\ [\!]\ \ X_1\ [\!]\ \ \ldots\ [\!]\ \ X_m\ \mathbf{od}\ ]| : u$$

where u is a list of global variables. The local variables a in $\mathcal{A}$ are replaced with new local variables c in $\mathcal{C}$. The actions A_i are replaced with C_i, and the auxiliary actions X_j are added into $\mathcal{C}$. We now have the following definition of action system refinement.

DEFINITION 1 *The action system $\mathcal{A}$ is* (trace) refined *by the system $\mathcal{C}$, denoted $\mathcal{A} \sqsubseteq \mathcal{C}$, if there exists relation $R(a, c)$ such that:*

(i) Initialization*:* $R(a_0, c_0)$

(ii) Main actions*:* $A_i \leq_R C_i$*, for* $i = 1, \ldots, n$

(iii) Auxiliary actions*:* $(\mathit{true} \rightarrow \mathit{skip}) \leq_R X_j$*, for* $j = 1, \ldots, m$

(iv) Continuation condition*:* $R \wedge g\mathcal{A} \Rightarrow g\mathcal{C}$

(v) Internal convergence*:* $\mathrm{wp}(\mathbf{do}\ X_1\ [\!]\ \ \ldots\ [\!]\ \ X_m\ \mathbf{od}, \mathit{true})$.

The first condition says that the abstraction relation is established by initializations. The second condition requires that each action A_i is simulated by the corresponding action C_i using $R(a, c)$. The third condition states that the auxiliary actions X_j behave like *skip* with respect to the global variables u while preserving $R(a, c)$. The fourth condition requires that an action in $\mathcal{C}$ is enabled whenever an action in $\mathcal{A}$ is enabled and $R(a, c)$ holds. The last condition stipulates that the execution of auxiliary actions taken separately can not continue forever.

In practice we are interested in the refinement of a reactive action system $\mathcal{A}$ in some environment $\mathcal{E}$. The following lemma can be used in such a case:

LEMMA 1 *Let $\mathcal{A}$, $\mathcal{C}$ be as above and let the environment $\mathcal{E}$ be any action system. Then*

$$\mathcal{A} \sqsubseteq \mathcal{C} \Rightarrow \mathcal{A} \parallel \mathcal{E} \sqsubseteq \mathcal{C} \parallel \mathcal{E}.$$

PROOF Follows from the fact that an abstraction relation assumed by the refinement $\mathcal{A} \sqsubseteq \mathcal{C}$ can refer only to the local variables of the action systems $\mathcal{A}$ and $\mathcal{C}$. $\square$

Splitting and merging actions In the proof of the handshaking expansion we will need refinement rules for splitting and merging actions in a sequential action system. The required rules are given and proven below.

LEMMA 2 *Let a sequential action system $\mathcal{A}$ be of the form*

$$\mathcal{A} :: \quad |[\ \textbf{var}\ x.\ \textbf{do}\ A_1;\ \ldots;\ A_n\ \textbf{od}\ ]| : u$$

Then $\mathcal{A} \sqsubseteq \mathcal{A}'$, for any $i \in 1, \ldots, n$, where

$$\mathcal{A}' :: \quad |[\ \textbf{var}\ x.\ \textbf{do}\ A_1;\ \ldots;\ A_{i-1};\ A_i';\ \ldots;\ A_{n+1}'\ \textbf{od}\ ]| : u$$

and for $k = i+1, \ldots, n$

$$\begin{array}{lcl} A_i' & \mathrel{\widehat{=}} & gA_i \rightarrow skip \\ A_{i+1}' & \mathrel{\widehat{=}} & true \rightarrow sA_i \\ A_{k+1}' & \mathrel{\widehat{=}} & A_k. \end{array}$$

The action $gA_i \rightarrow skip$ is a wait action, *whereas $true \rightarrow sA_i$ is a* transition action.

PROOF Recall that the sequential action systems $\mathcal{A}$, $\mathcal{A}'$ contain the local program counters pc, pc', respectively. Define abstraction relation $R(pc, pc')$ as follows

$$\begin{array}{lcl} R(pc, pc') & \mathrel{\widehat{=}} & pc < i \Leftrightarrow pc' = pc \\ & \wedge & pc = i \Leftrightarrow pc' = i \vee pc' = i+1 \\ & \wedge & pc > i \Leftrightarrow pc' = pc + 1 \end{array}$$

It is easy to check that all the five conditions required to establish $\mathcal{A} \sqsubseteq \mathcal{A}'$ hold when we choose $R(pc, pc')$ as an abstraction relation. $\square$

LEMMA 3 *Let a sequential action system*

$$\mathcal{A} :: \quad |[\ \textbf{var}\ x.\ \textbf{do}\ A_1;\ \ldots;\ A_n\ \textbf{od}\ ]| : u$$

be such that $sA_i = skip$*, for some* $i \in 1, \ldots, n-1$*. Then* $\mathcal{A} \sqsubseteq \mathcal{A}'$*, where*

$$\mathcal{A}' :: \quad |[\ \mathbf{var}\ x.\ \mathbf{do}\ A_1;\ \ldots;\ A_{i-1};\ A'_i;\ \ldots;\ A'_{n-1}\ \mathbf{od}\ ]| : u$$

and for $k = i+1, \ldots, n-1$

$$\begin{array}{lcl} A'_i & \widehat{=} & gA_i \wedge gA_{i+1} \rightarrow sA_{i+1} \\ A'_k & \widehat{=} & A_{k+1}. \end{array}$$

Proof The proof of the lemma actually requires more general definition of trace refinement for the reactive action systems (see [6]) which we simplified for this paper. □

4 Communication in the action systems

Within the action systems, a communication between circuit components can be modelled on several levels of abstraction. We show how one communication model, one variable communication, is replaced with a model which is closer to the actual communication in the circuits.

One variable communication Let $\mathcal{A}$ and $\mathcal{B}$ be the following action systems:

$$\begin{array}{rl} \mathcal{A} :: & |[\ \mathbf{do}\ A_1;\ \ldots;\ g_i \rightarrow S_i[c\uparrow];\ \ldots;\ g_k[\neg c] \rightarrow S_k;\ \ldots;\ A_m\ \mathbf{od}\ ]| : u \\ \mathcal{B} :: & |[\ \mathbf{do}\ B_1;\ \ldots;\ h_j[c] \rightarrow T_j;\ \ldots;\ h_l \rightarrow T_l[c\downarrow];\ \ldots;\ B_n\ \mathbf{od}\ ]| : u \end{array}$$

where all occurrences of the variable c are explicitly given in the brackets. Let c be *false* initially. We say that $\mathcal{A}$ and $\mathcal{B}$ communicates through the *communication variable* c in the system $\mathcal{A} \parallel \mathcal{B}$. The communication is called a *one variable* communication. We require that the variable c is not referenced outside the system $\mathcal{A} \parallel \mathcal{B}$ when it is composed with another system $\mathcal{E}$.

Communication channels Typically, a communication is related with data passing (and possible computation) from a variable di to a variable do. The transition $c\uparrow$ initiates the data passing from the variable di to the variable do. The transition $c\downarrow$ acknowledges the completion of the data passing. We think of the triple $(c : Bool, di : Data, do : Data)$ as a *communication channel*. The channel (c, di, do) will be simply denoted as c when the data variables are not important. If di is the same variable as do the corresponding channel is a *control*, otherwise it is a *data* channel. In the data channel, data passing from di to do is represented as an assignment statement of the form $do := F(di)$, where the function F defines a computation. Such model of data passing does not make semantic distinction between the *output preserving* and *output non-preserving* data channels. We therefore require the type of data channel to be given explicitly. Note that a control channel (c, d, d) is always considered as output non-preserving.

DEFINITION 2 *We say that the channel* (c_1, d_1, d_2) feeds *the channel* (c_2, d_2, d_3). *If* $c_2 \Rightarrow c_1$ *the channel* (c_1, d_1, d_2) up-feeds *the channel* (c_2, d_2, d_3), *whereas if* $c_2 \Rightarrow \neg c_1$ *it* down-feeds (c_2, d_2, d_3).

EXAMPLE 1 Consider the parallel composition of two action systems:

$$
\begin{array}{lll}
\mathcal{C} :: & |[\ \mathbf{do} & l \rightarrow m\uparrow \\
 & \quad\ ; & \neg m \rightarrow l\downarrow, r\uparrow \\
 & \quad\ \mathbf{od} & \\
 &]|: l, m, r &
\end{array}
$$

$$
\mathcal{D} :: \quad |[\ \mathbf{do}\ m \rightarrow do := F(di); m\downarrow \ \mathbf{od} \ ]| : m, di, do
$$

They communicate through the communication variable m which controls the computation defined by F. Thus, we have a data channel (m, di, do). The action system $\mathcal{C}$ communicates with its environment through the variables l and r as well. Hence, we can think of two other channels: $(l, \_ , di)$ and $(r, do, \_)$. We would need to be more specific about environment of the system $\mathcal{C}$ to identify the variables in those channels denoted by the underscore symbol $\_$.

Handshaking expansion In one variable communication, a communication cycle is the sequence of transitions $c\uparrow;\ c\downarrow$ that starts in the initial state $c = false$. The handshaking expansion replaces each communication variable c with two handshaking variables $req : Bool$ and $ack : Bool$ which are initialized to $false$. In the *four-phase handshaking* the communication cycle is replaced by the following sequence of transitions, called the *handshaking cycle*, on the variables req and ack:

$$
req\uparrow;\ ack\uparrow;\ req\downarrow;\ ack\downarrow \ .
$$

The transition $req\uparrow$ corresponds to the request $c\uparrow$ whereas the acknowledgment $c\downarrow$ in one variable communication is identified with the transition $ack\uparrow$. The two down-going transitions $req\downarrow;\ ack\downarrow$ reset the handshaking variables to the initial values. They are considered as acknowledgment phase transitions. This means that the relation between the communication and the handshaking variables, the *handshaking relation*, is defined as

$$
H(c, req, ack) \quad \hat{=} \quad c = req \wedge \neg ack.
$$

The handshaking relation is an essential part of the abstraction relation which is used in proving the handshaking expansion as a refinement in the next section. In the paper, we do not consider two-phase handshaking, where the down-going transitions are interpreted as a new handshaking cycle.

Interleaving handshaking cycles Usually an action system communicates with its environment through several communication variables (in the Example 1 the system $\mathcal{C}$ refers to variables l, m and r). The down-going parts of the corresponding handshaking cycles can be interleaved in many different ways. The handshaking relation ensure that any interleaving is valid. However, some restrictions, called the *synchronization conditions*, must be imposed on the interleaving when at least one of the channels is output non-preserving. This has to do with the fact that data passing, modelled as an assignment statement, is semantically an output preserving operation.

DEFINITION 3 *For the two channels* (c_1, d_1, d_2) *and* (c_2, d_2, d_3)*, where the channel* (c_1, d_1, d_2) *is output non-preserving, the synchronization condition is:*

(1) (c_1, d_1, d_2) up-feeds (c_2, d_2, d_3): *communication cycle on* (c_2, d_2, d_3) *must be completed before the cycle on* (c_1, d_1, d_2)*, i.e.*

$$\neg req_1 \wedge \neg ack_1 \Rightarrow \neg req_2 \wedge \neg ack_2 \qquad [SC1]$$

(2) (c_1, d_1, d_2) down-feeds (c_2, d_2, d_3): *the transition* $req_1\!\downarrow$ *in the communication cycle on* (c_1, d_1, d_2) *can not be executed before the transition* $ack_2\!\uparrow$ *in the cycle on* (c_2, d_2, d_3)*, i.e*

$$\neg req_1 \vee \neg ack_1 \Rightarrow \neg req_2 \vee ack_2 \qquad [SC2]$$

More on the synchronization conditions can be found in [15].

5 Handshaking expansion as a refinement

Consider the following parallel composition of action systems

$$\mathcal{C} :: \ |[\ \mathbf{var}\ c_1, \ldots, c_m : Bool.\ c_1\!\downarrow;\ \ldots; c_j\!\downarrow; \mathcal{A}_1 \parallel \ldots \parallel \mathcal{A}_n\ ]|$$

where each c_j is the communication variable of a channel between some systems $\mathcal{A}_i$ and $\mathcal{A}_k$, $i \neq k$. This means that the systems $\mathcal{A}_i$, $\mathcal{A}_k$ are of the form

$$
\begin{array}{ll}
\mathcal{A}_i :: & |[\ \textbf{do}\ A_{i,1};\ \ldots;\ A_{i,m_i}\ \textbf{od}\ ]| : c_j, u_i \\
\mathcal{A}_k :: & |[\ \textbf{do}\ A_{k,1};\ \ldots;\ A_{k,m_k}\ \textbf{od}\ ]| : c_j, u_k
\end{array}
$$

with the communication actions

$$
\begin{array}{lcl@{\qquad\qquad}lcl}
A_{i,l_1} & \hat{=} & g_{i,l_1} \to S_{i,l_1}[c_j\!\uparrow] & A_{k,l_5} & \hat{=} & h_{k,l_5}[c_j] \to T_{k,l_5} \\
A_{i,l_2} & \hat{=} & g_{i,l_2}[\neg c_j] \to S_{i,l_2} & A_{k,l_6} & \hat{=} & h_{k,l_6} \to T_{k,l_6}[c_j\!\downarrow]
\end{array}
$$

where $1 \leq l_1 < l_2 \leq m_i$, $1 \leq l_5 < l_6 \leq m_k$. Note also that the second index stands for value of the hidden program counter associated with the action. From now on we assume that each communication action refers to only one communication variable. If it is not originally the case such communication action is split into two using Lemma 2.

Auxiliary actions For the channel c_j, two auxiliary actions of the form $\mathit{true} \to \mathit{skip}$ are inserted into the systems $\mathcal{A}_i$, $\mathcal{A}_k$ each. We are allowed to do this because of the Lemma 2. The actions are designated for the down-going transitions of the handshaking cycle on the channel c_j. They are replaced with the corresponding transition and wait actions later. The actions are inserted so that they follow A_{i,l_2} in $\mathcal{A}_i$, and A_{k,l_6} in $\mathcal{A}_k$, respectively. When a communication on the channel c_j is independent of the communications on other channels in $\mathcal{A}_i$ and/or $\mathcal{A}_k$, no further restrictions are needed. Otherwise the order of transitions in the corresponding handshaking cycles must be such that no deadlocks are created in the system.

The same procedure is repeated for all communication channels c_j. The new action systems are of the form

$$
\begin{array}{ll}
\mathcal{A}'_i :: & |[\ \textbf{do}\ A'_{i,1};\ \ldots;\ A'_{i,m'_i}\ \textbf{od}\ ]| : c_j, u_i \\
\mathcal{A}'_k :: & |[\ \textbf{do}\ A'_{k,1};\ \ldots;\ A'_{k,m'_k}\ \textbf{od}\ ]| : c_j, u_k
\end{array}
$$

where, for the channel c_j, the communication actions A'_{i,l'_1}, A'_{i,l'_2}, A'_{k,l'_5}, A'_{k,l'_6} are the same as the old ones only their numbering is changed. The auxiliary actions are denoted A'_{i,l'_3}, A'_{i,l'_4}, A'_{k,l'_7}, A'_{k,l'_8}, respectively. As a result we have that the parallel composition

$$
\mathcal{C}' ::\ |[\ \textbf{var}\ c_1, \ldots, c_m : \mathit{Bool}.\ c_1\!\downarrow;\ \ldots;\ c_j\!\downarrow;\ \mathcal{A}'_1 \parallel \ldots \parallel \mathcal{A}'_n\ ]|
$$

is a refinement of $\mathcal{C}$:

$$
\mathcal{C} \sqsubseteq \mathcal{C}'.
$$

Handshaking Now we replace each communication variable c_j by the two handshaking variables req_j and ack_j. Let new initialization statements be $req_j\!\downarrow$ and $ack_j\!\downarrow$. Define the new communication actions as follows:

$$\begin{array}{lcllcl}
A''_{i,l'_1} & \hat{=} & A'_{i,l'_1}[req_j\!\uparrow / c_j\!\uparrow] & A''_{k,l'_5} & \hat{=} & A'_{k,l'_5}[req_j / c_j] \\
A''_{i,l'_2} & \hat{=} & A'_{i,l'_2}[ack_j / \neg c_j] & A''_{k,l'_6} & \hat{=} & A'_{k,l'_6}[ack_j\!\uparrow / c_j\!\downarrow] \\
A''_{i,l'_3} & \hat{=} & true \to req_j\!\downarrow & A''_{k,l'_7} & \hat{=} & \neg req_j \to skip \\
A''_{i,l'_4} & \hat{=} & \neg ack_j \to skip & A''_{k,l'_8} & \hat{=} & true \to ack_j\!\downarrow
\end{array}$$

Note that the left assignment (predicate) in the brackets above is substituted for the right one in the corresponding action. The transformation gives the new action systems $\mathcal{A}''_1, \ldots \mathcal{A}''_n$. The entire system is as follows:

$$\mathcal{C}'' :: \quad |[\ \mathbf{var}\ req_1, \ldots, ack_m : Bool.\ req_1\!\downarrow ;\ \ldots ;\ ack_m\!\downarrow ;\ \mathcal{A}''_1 \parallel \ldots \parallel \mathcal{A}''_n\ ]|$$

Let the handshaking relation for c_j be

$$H_j(c_j, req_j, ack_j) \quad \hat{=} \quad c_j = req_j \wedge \neg ack_j.$$

The order of transitions in the handshaking cycles ensures that I_j defined as

$$\begin{array}{lcl}
I_j(pc_i, pc_k, req_j, ack_j) & \hat{=} & (pc_i = l'_1 \vee pc_k = l'_5) \Rightarrow \neg ack_j \\
& \wedge & (pc_i = l'_2 \vee pc_k = l'_6) \Rightarrow req_j \\
& \wedge & (pc_i > l'_2 \vee pc_k > l'_6) \Rightarrow ack_j \\
& \wedge & (pc_i > l'_3 \vee pc_k > l'_7) \Rightarrow \neg req_j \\
& \wedge & pc_i = l'_4 \Rightarrow (pc_k > l'_6 \vee 1 < pc_k < l'_5) \\
& \wedge & pc_k = l'_7 \Rightarrow (l'_1 \leq pc_i < l'_4)
\end{array}$$

is an invariant of the parallel composition $\mathcal{A}''_i \parallel \mathcal{A}''_k$.

Let us check for the redefined actions the conditions for action system refinement from the Def. 1. We use $H_j \wedge I_j$ as an abstraction relation. Note that I_j is dropped from the initialization condition and the right-hand side of implications in the other conditions. Such simplification is valid since I_j is an invariant.

(i) *Initialization*: $H_j(false, false, false) \Leftrightarrow (false = false \wedge true) \Leftrightarrow true$

(ii) *Main actions*: We have to check that the redefined actions simulate the old ones. This amounts to checking the simulation condition for the guard and body of the action.

(a) *Guards*: Recall that the simulation condition for the guards of actions A and A' under the relation R is $R \wedge gA' \Rightarrow gA$. We will however show that the stronger property $R \Rightarrow (gA = gA')$ holds for most of the redefined actions. This property implies that the action A' is enabled whenever A is enabled, thus, a part of the continuation condition is satisfied.

The guards of A'_{i,l'_1}, A'_{i,l'_3}, A'_{k,l'_5} and A'_{k,l'_8} remain the same in the redefined actions.

For A''_{i,l'_2} we have: I_j and $pc_i = l'_2$ implies req_j; from H_j and req_j follows $ack_j = \neg c_j$. Thus, the guards of A'_{i,l'_2} and A''_{i,l'_2} are equivalent. Similarly for the action A''_{k,l'_6}.

We can not show the equivalence of gA'_{i,l'_4} and gA''_{i,l'_4}. However, the simulation condition is trivial since $gA'_{i,l'_4} = true$. Similarly for the actions A'_{k,l'_7}, A''_{k,l'_7}.

(b) *Bodies*: The redefined actions affect the global variables in the same way as the old ones. Thus, we only have to show that the auxiliary assignments to the handshaking variables preserve the relation $H_j(c_j, req_j, ack_j)$.

The bodies of A'_{i,l'_2}, A'_{i,l'_4}, A'_{k,l'_5} and A'_{k,l'_7} remain the same in their counterparts of $\mathcal{A}''_i$, $\mathcal{A}''_k$, hence, the relation is preserved.

For A''_{i,l'_1}: the assignments $c_j\uparrow$ and $req_j\uparrow$ establish $H_j(c_j, req_j, ack_j)$, since I_j and $pc_i = l'_1$ implies $\neg ack_j$. Similarly for A''_{k,l'_6}.

For A''_{i,l'_3}: I_j and $pc_i = l'_3$ implies ack_j; from ack_j follows $H_j \Leftrightarrow \neg c_j$. Thus, H_j does not depend on the value of req_j. Similarly for A''_{k,l'_8}.

(iii) *Continuation condition*: We are only left to show that there is an enabled action in $\mathcal{C}''$ whenever either of A'_{i,l'_4} or A'_{k,l'_7} is enabled in $\mathcal{C}'$. An enabledness of A'_{k,l'_7} implies that $pc_k = l'_7$; from I_j and $pc_k = l'_7$ follows $l'_1 < pc_i < l'_4$. Now:

(a) if $pc_i = l'_2$ or $pc_i = l'_3$ then A_{i,l'_2} or A_{i,l'_3} is enabled, respectively.

(b) if $pc_i > l'_3$ then from I_j follows $\neg req_j$. Thus, A_{k,l'_7} is enabled.

(c) otherwise $pc_i = l$, where $l'_1 < l < l'_2$ or $l'_2 < l < l'_3$. If $A_{i,l}$ is a transition action then it is enabled. If it is a wait action, we have a communication on another channel and exactly the same argument applies again. Eventually, this process stops since we have chosen insertion places for the wait and transition actions so that the system is deadlock-free.

Similar considerations hold for A'_{i,l'_4}. This means that the continuation condition is satisfied.

Thus, the abstraction relation

$$H_1 \wedge I_1 \wedge \ldots \wedge H_n \wedge I_n$$

proves the refinement

$$\mathcal{C}' \sqsubseteq \mathcal{C}''.$$

Now we can merge all action sequences of the form $(g_1 \rightarrow \mathit{skip})$; $(g_2 \rightarrow S)$ into one action $g_1 \wedge g_2 \rightarrow S$ in the system $\mathcal{C}''$. By Lemma 3, we have that $\mathcal{C}'' \sqsubseteq \mathcal{C}'''$. Since the refinement relation is transitive, the following refinement holds:

$$\mathcal{C} \sqsubseteq \mathcal{C}'''.$$

The refinement is a *handshaking expansion* of the action system $\mathcal{C}$.

Output non-preserving channels The above handshaking expansion is a valid refinement when we have only output preserving channels in the system. In the case of an output non-preserving channel (c_j, di_j, do_j) the corresponding synchronization conditions must be satisfied. Namely, for any channel (c_l, di_l, do_l) such that (c_l, di_l, do_l) up-feeds (down-feeds) the channel (c_j, di_j, do_j) the synchronization condition SC1 (SC2) must be an invariant of the system $\mathcal{C}''$.

6 Example: a sequencer

As an example we consider one of the two-channel handshake distributors. We call it a sequencer since it is very similar to the sequencer in the Tangram system [7]. The component is activated via a left channel. Then it sequences the communication cycles on the middle and right channels starting from the middle. We assume that the middle channel is an output non-preserving data channel. On the one variable communication level the component is specified as the following action system:

$$\begin{array}{lll} \mathcal{S}eq :: & |[\ \mathbf{do} & l \rightarrow m\uparrow \\ & \quad\ ; & \neg m \rightarrow r\uparrow \\ & \quad\ ; & \neg r \rightarrow l\downarrow \\ & \quad\ \mathbf{od} & \\ &]| : l, m, r & \end{array}$$

Here l, m and r are the communication variables of the left, middle and right channel, respectively. The environment of the sequencer can be viewed as three action systems, one for the communication through the corresponding channel:

$$\begin{array}{rl} \mathcal{L} :: & |[\ \mathbf{do}\ \neg l \to l\!\uparrow\ \ \mathbf{od}\ ]| : l \\ \mathcal{M} :: & |[\ \mathbf{do}\ m \to do := F(di);\ m\!\downarrow\ \ \mathbf{od}\ ]| : m, di, do \\ \mathcal{R} :: & |[\ \mathbf{do}\ r \to r\!\downarrow\ \ \mathbf{od}\ ]| : r \end{array}$$

The entire system is the following parallel composition:

$$\mathcal{C} ::\ \ |[\ \mathbf{var}\ l, m, r : \mathit{Bool}.\ l\!\downarrow ; m\!\downarrow ; r\!\downarrow ;\ \ \mathcal{L} \parallel \mathcal{S}eq \parallel \mathcal{M} \parallel \mathcal{R}\ ]|$$

In the handshaking expansion of the sequencer $\mathcal{S}eq$, three handshaking cycles are to be interleaved. Since (m, di, do) is an output non-preserving data channel and it down-feeds the control channel (r, do, do), the interleaving of the two handshaking cycles must satisfy the synchronization condition $SC2$:

$$req_r \wedge \neg ack_r \Rightarrow req_m \wedge ack_m.$$

The other down-going transitions can be interleaved in any order. One of the alternatives is given below. It was constructed by splitting the third action of $\mathcal{S}eq$, then adding one auxiliary action (for the transition $req_m\!\downarrow$) in the splitting point and five auxiliary actions at the end of $\mathcal{S}eq$. The new communication actions of the sequencer were then defined as in Section 3. Finally, some actions were merged to get more compact representation of the sequencer:

$$\begin{array}{rlll} \mathcal{S}eq' :: & |[& \mathbf{do} & req_l \wedge \neg ack_r \to req_m\!\uparrow \\ & & ; & ack_m \to req_r\!\uparrow \\ & & ; & ack_r \to req_m\!\downarrow \\ & & ; & \neg req_m \to ack_l\!\uparrow \\ & & ; & ack_l \to req_r\!\downarrow \\ & & ; & \neg req_l \wedge \neg req_r \wedge \neg ack_m \to ack_l\!\downarrow \\ & & \mathbf{od} & \\ &]|\!: & \multicolumn{2}{l}{req_l, ack_l, req_m, ack_m, req_r, ack_r} \end{array}$$

Note that only the request phases of the middle and right channel are sequenced in $\mathcal{S}eq'$ which makes it different from the Tangram sequencer. The new environment action systems

$$
\begin{array}{rll}
\mathcal{L}' :: & |[& \textbf{do} \quad \neg ack_l \to req_l\!\uparrow \\
& & \quad\textbf{;} \quad ack_l \to req_l\!\downarrow \\
& & \textbf{od} \\
&]|: & req_l, ack_l \\
\mathcal{M}' :: & |[& \textbf{do} \quad req_m \to do := F(di);\ ack_m\!\uparrow \\
& & \quad\textbf{;} \quad \neg req_m \to ack_m\!\downarrow \\
& & \textbf{od} \\
&]|: & req_m, ack_m, di, do \\
\mathcal{R}' :: & |[& \textbf{do} \quad req_r \to ack_r\!\uparrow \\
& & \quad\textbf{;} \quad \neg req_r \to ack_r\!\downarrow \\
& & \textbf{od} \\
&]|: & req_r, ack_r
\end{array}
$$

should be interpreted as the requirements for the actual environment of the sequencer $\mathcal{S}eq'$. They basically specify the order of transitions on the handshaking variables which must be preserved by any potential environment. The handshaking expansion is

$$\mathcal{C} \sqsubseteq \mathcal{C}'$$

where $\mathcal{C}'$ is defined as

$$
\begin{array}{rll}
\mathcal{C}' :: & |[& \textbf{var } req_l, ack_l, req_m, ack_m, req_r, ack_r : Bool. \\
& & req_l\!\downarrow;\ ack_l\!\downarrow;\ req_m\!\downarrow;\ ack_m\!\downarrow;\ req_r\!\downarrow;\ ack_r\!\downarrow; \\
& & \mathcal{L}' \parallel \mathcal{S}eq' \parallel \mathcal{M}' \parallel \mathcal{R}' \\
&]| &
\end{array}
$$

Note that the condition $SC2$ for the middle and right channels is an invariant of $\mathcal{C}'$.

7 Conclusions

We have shown the correctness of the handshaking expansion by modelling circuits as action systems and formulating the correctness criteria for this within the Refinement Calculus.

Our approach to handshaking expansion is derived from that of Martin. However, we consider the reshuffling of handshaking cycles to be part of the handshaking expansion allowing several of the cycles to be interleaved. In the original work of Martin these two things were separated. Also the method of Smith and Zwarizo [17] is based on the ideas of Martin. They give a formal proof of correctness for the compilation method, but in their case the reshuffling is also a separate step. van Berkel [7] gives formal proofs for his handshake circuits. The initial designs are carried out in the Tangram language. The proofs are based on trace theory and there is no formal notion of program refinement. In the work of Yakovlev et al. [18], Petri nets are used to specify asynchronous circuits. Their signalling expansion, which is essentially the same as handshaking expansion, is proven in a separate formalism without giving formal refinement arguments as we do here.

Acknowledgments

The basic ideas behind the use of action systems to model the low-level issues in circuit design are joint work with Juha Plosila. He formulated the synchronization conditions and suggested the sequencer example as well. The research has been supported by the Academy of Finland.

References

[1] R. J. R. Back. *On the Correctness of Refinement Steps in Program Development*. PhD thesis, Department of Computer Science, University of Helsinki, Helsinki, Finland, 1978. Report A–1978–4.

[2] R. J. R. Back. Refinement calculus, part II: Parallel and reactive programs. In J. W. de Bakker, W.–P. de Roever, and G. Rozenberg, editors, *Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness. Proceedings. 1989*, volume 430 of *Lecture Notes in Computer Science*. Springer–Verlag, 1990.

[3] R. J. R. Back and R. Kurki-Suonio. Decentralization of process nets with centralized control. In *Proc. of the 2nd ACM SIGACT–SIGOPS Symp. on Principles of Distributed Computing*, pages 131–142, 1983.

[4] R. J. R. Back, A. J. Martin, and K. Sere. Specifying the Caltech asynchronous microprocessor. *Science of Computer Programming*, North-Holland. Accepted for publication.

[5] R. J. R. Back and K. Sere. Stepwise refinement of action systems. *Structured Programming*, 12:17-30, 1991.

[6] R. J. R. Back and J. von Wright. Trace refinement of action systems. In B. Jonsson and J. Parrow, editors, *CONCUR '94: Concurrency Theory, 5th International Conference, Uppsala, Sweden, August 1994*, volume 836 of *Lecture Notes in Computer Science*. Springer–Verlag, 1994.

[7] K. van Berkel. *Handshake circuits: an asynchronous architecture for VLSI programming*. International Series on Parallel Computation. Cambridge University Press, Cambridge, England, 1993.

[8] K. Chandy and J. Misra. *Parallel Program Design: A Foundation.* Addison–Wesley, 1988.

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

[10] T. Långbacka, R. Rukšėnas, and J. von Wright. TkWinHOL: A Tool for Window Inference in HOL. In E. T. Schubert, P. J. Windley, and J. Alves-Foss, editors, *Higher Order Logic Theorem Proving and Its Applications – 8th International Workshop, Aspen Grove, Utah, USA, September 1995*, volume 971 of *Lecture Notes in Computer Science*, pages 245–260, Springer–Verlag, 1995.

[11] C. C. Morgan. The specification statement. *ACM Transactions on Programming Languages and Systems*, 10(3):403–419, July 1988.

[12] A. J. Martin. Compiling communicating processes into delay-insensitive VLSI circuits. *Distributed Computing*, 1:226-234, 1986.

[13] A. J. Martin. Synthesis of Asynchronous VLSI Circuits. *CalTech*, Technical Report, 1993.

[14] J. M. Morris. A theoretical basis for stepwise refinement and the programming calculus. *Science of Computer Programming*, 9:287–306, 1987.

[15] J. Plosila, R. Rukšėnas, K. Sere, and T. Kuusela. Synthesis of DI Circuits within the Action Systems Framework. Manuscript.

[16] J. Plosila and K. Sere. Action Systems in Pipelined Processor Design. Technical Report 54, Turku Centre for Computer Science, 1996. Available from `http://www.tucs.abo.fi/publications/techreports/TR54.ps.gz`.

[17] S. F. Smith and A. E. Zwarico. Correct compilation of specifications to deterministic asynchronous circuits. *Formal Methods in System Design*, 7, 155-226 (1995). Kluver Academic Publishers.

[18] A. V. Yakolev, A. M. Koelmans, and L. Lavagno. High level modelling and design of asynchronous interface logic. Computing Science, University of Newcastle upon Tyne, Technical Report Series No. 460, November 1993.

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**