A CSP Approach to Control in Event-B

Steve Schneider[1], Helen Treharne[1], and Heike Wehrheim[2]

[1] Department of Computing, University of Surrey
[2] Institut für Informatik, Universität Paderborn

Abstract. Event-B has emerged as one of the dominant state-based formal techniques used for modelling control-intensive applications. Due to the blocking semantics of events, their ordering is controlled by their guards. In this paper we explore how process algebra descriptions can be defined alongside an Event-B model. We will use CSP to provide explicit control flow for an Event-B model and alternatively to provide a way of separating out requirements which are dependent on control flow information. We propose and verify new conditions on combined specifications which establish deadlock freedom. We discuss how combined specifications can be refined and the challenges arising from this. The paper uses Abrial's Bridge example as the basis of a running example to illustrate the framework.

Keywords: Event-B, CSP, control flow, integration, consistency, deadlock-freedom.

1 Introduction

Event-B [1] is an elegant modelling language which is supported by a notion of *refinement* to enable descriptions of systems to be elaborated during refinement. Event-B has proven to be applicable in a wide range of domains, including distributed algorithms, railway systems and electronic circuits. The basic specification construct is a machine that comprises of a number of events in which control flow is implicit within their guards. Hence, Event-B can be classified as being a *state-based language*: control can only be modelled via state variables and guards on the state. On the other side, there are a number of specification formalisms with language support for *explicitly* specifying control, like statecharts, Petri nets or process algebras, which are however not good at specifying state. This observation has lead to introducing *integrated formal methods*, combining state-based formalisms with control-oriented languages. Examples include combinations of (Object-)Z and CSP [17,7,11,20], or closer to the approach presented here, those integrating B with a process algebra [3,4,19].

Though Event-B can be and is used for modelling control-intensive applications, it has recently been observed that explicit control specifications alongside Event-B machines are nevertheless beneficial [10]. Control specifications can serve two purposes: on the one hand they can make the control flow in the Event-B machine explicit, and thus enhance readability but also facilitate analysis. On

D. Méry and S. Merz (Eds.): IFM 2010, LNCS 6396, pp. 260–274, 2010.

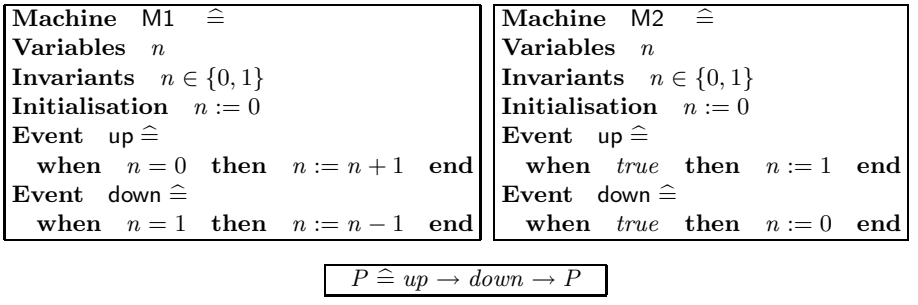

Fig. 1. Event-B machines and control

the other hand, they can be used as a straightforward way of modelling control-oriented requirements, and can thus ease specification. Figure 1 illustrates these two applications. The machine $M1$ on the left hand side regulates its control flow (alternation between `up` and `down` events) via guards on the events. This control flow is made explicit in the Communicating Process Algebra (CSP) [9] process P below. Alternatively, we could have used the machine `M2` on the right hand side to describe the state, and then let the CSP process P *in addition* fix the ordering of events. In this simple case the flow of control in $M1$ is obvious and the variable n might be needed anyway. However, more complex applications might necessitate introducing variables solely acting as program counters. This compromises readability of the specification and ease of modelling.

In this paper we will propose a combination of Event-B with the process algebra CSP which serves these two purposes. The paper begins with a motivating example in order to illustrate how CSP descriptions can be defined alongside an Event-B machine. We will then give a failures divergences semantics (the semantics domain of CSP) to the integration via a weakest precondition semantics for Event-B. This follows previous approaches to integrating B with CSP [19] through relating weakest preconditions with CSP's failures-divergence semantics [13]. The main focus of the paper rests on studying two fundamental issues arising for the combination: how can we determine whether the obtained specifications stay *deadlock free* despite the addition of CSP processes, and how can the central notion of *refinement* used for developing specifications be applied in the combination? The first issue is of particular importance because establishing deadlock-freedom in pure Event-B models is often difficult in practice when the flow of control is not simple, and so it is valuable to investigate approaches which can make that easier. In this paper we introduce proof obligations on Event-B machines which guarantee deadlock-freedom of a combination. For the second part we develop conditions which allow to prove refinement for a combined specification on the Event-B and CSP part in isolation. This gives rise to a compositional refinement framework. Both techniques are illustrated on a running example.

The paper is structured as follows: Section 2 introduces the notation and semantics for CSP and for Event-B; Section 3 introduces the Bridge example

used to illustrate the approach; Section 4 contains the main results for establishing deadlock-freedom; and Section 5 discusses refinement; finally, Section 6 concludes.

2 Notation

We start with a short introduction to the two formalisms, CSP and Event-B.

2.1 CSP

CSP [9] is a process algebra and used to specify control oriented applications. In this paper we will use the following subset of the CSP language:

$$P ::= e \rightarrow P \mid P1 \,\Box\, P2 \mid P1 \sqcap P2 \mid P1 \parallel P2 \mid S$$

The event e here is drawn from the set of events in process P, and S is a CSP process variable. Events can either be pure CSP events, or correspond to events in the corresponding Event-B machine. Notationally we will use e for simple atomic CSP events not corresponding to Event-B events, whereas *op* will be used for Event-B event names. In this paper we assume that we have no parameters to channels. Recursive definitions are given as $S \mathrel{\widehat{=}} P$. In a CSP definition, all process variables used are bound by some recursive definition. External choice, $\Box$, is a choice between alternatives which can be influenced by other components running in parallel, whereas $\sqcap$ is an internal choice taken by the process alone. The prefix operator $\rightarrow$ denotes sequencing. The CSP process P in Figure 1 thus specifies a recursive process alternating between *up* and *down* events.

The form of parallel combination we use is *alphabet* parallel, in which processes are associated with an *alphabet* (usually denoted $\alpha(P)$) which is a set of events. The occurrence of an event in the combination requires the participation of all processes whose alphabet contains that event.

The semantics of CSP (see e.g. [16]) is given in terms of its *traces* (the sequences of events it can execute), its *failures* (the events it might refuse after a trace) and its *divergences* (the traces after which it might engage in internal events only):

$$\begin{aligned} &\mathit{traces}(P) \subseteq \alpha(P)^* \\ &\mathit{failures}(P) \subseteq \alpha(P)^* \times 2^{\alpha(P)} \\ &\mathit{divergences}(P) \subseteq \alpha(P)^* \end{aligned}$$

The process $P \mathrel{\widehat{=}} \mathit{up} \rightarrow \mathit{down} \rightarrow P$ for instance has the alphabet $\{\mathit{up}, \mathit{down}\}$ and $\mathit{traces}(P) = \{\langle\rangle, \langle \mathit{up}\rangle, \langle \mathit{up}, \mathit{down}\rangle, \langle \mathit{up}, \mathit{down}, \mathit{up}\rangle, \ldots\}$, $\mathit{failures}(P) = \{(\langle\rangle, \{\mathit{down}\}), (\langle \mathit{up}\rangle, \{\mathit{up}\}), \ldots\}$ and $\mathit{divergences}(P) = \varnothing$. These three semantic domains come with three different notions of process refinement in CSP, two of which we are going to consider.

Definition 1. *Let P_1, P_2 be CSP processes.*
P_2 is a traces refinement *of P_1 ($P_1 \sqsubseteq_T P_2$) if $\mathit{traces}(P_2) \subseteq \mathit{traces}(P_1)$.*
P_2 is a failures refinement *of P_1 ($P_1 \sqsubseteq_F P_2$) if $\mathit{failures}(P_2) \subseteq \mathit{failures}(P_1)$.*

Intuitively, trace refinement is only concerned with *safety*: the refinement may not exhibit more execution traces than the abstract process. Failures on the other hand also treat *liveness*: a pair $(tr, X) \in \mathit{failures}(P)$ specifies events X which may be *refused* to be executed after some trace tr. Failures refinement guarantees that the concrete process may not refuse more events than the abstract one. Further explanation of refinement can be found in [16]

2.2 Event-B

Event-B [1,12] is a state-based specification formalism based on set theory. We cannot describe all of Event-B here, only the basic parts of an Event-B machine, required for this paper. A machine specification usually defines variables (collectively called v), constants c (possibly with axioms $A(c)$, which however do not occur in our examples) and a set of invariants $I(c, v)$ on constants and variables. The core part is the definitions of events, each consisting of a *guard* $G(c, v)$ over constants and variables and a *body* (usually written as an assignment on the variables) which defines a *before-after predicate* $E(c, v, v')$ describing changes of variables upon event execution, in terms of the relationship between the variable values before (v) and after (v'). A machine also has an initialisation T. Proof obligations on events are expressed in terms of *weakest precondition* semantics, where $[S]R$ denotes the weakest precondition for statement S to guarantee to establish postcondition R. A machine will have various proof obligations on it. These includeconsistency obligations, that events preserve the invariant. They can also include (optional) deadlock-freeness obligations, that at least one event guard is always true.

A machine M_1 is refined by another machine M_2 if there is a *linking invariant* (i.e. a predicate) J on the variables of the two machines, which is established by their initialisations, and which is preserved by all events, in the sense that any event of M_2 can be matched by an event of $M1$ to maintain J. This is the standard notion of downwards simulation data refinement (see e.g. [5] for a description). New events can also be introduced as data refinements of *skip* [1]: they need not always be enabled, but their execution should maintain the linking relationship to the same abstract state. Event-B admits a variety of proof obligations depending on what is appropriate for the application. For the purposes of this paper (where we need refinement in Event-B to induce refinement in the CSP semantics), we require the *strong relative deadlock freeness* property *S_DLK_E* of [12], which states that whenever an event E is enabled in machine M_1, then either E or a newly introduced event should be enabled in M_2. We also require the *non-divergence* property *WFD_REF*, which states that newly introduced events cannot always be enabled. When the standard refinement conditions, together with both these conditions, are met we write $M_1 \sqsubseteq_D M_2$.

The machine $M1$ in Figure 1 for instance defines one variable n, specifies this to only take values 0 and 1 (invariant) and defines the two events `up` and `down`. An initialisation section furthermore fixes the initial value for n.

Morgan's CSP semantics for action systems [13] allows traces, failures, and divergences to be defined for Event-B machines in terms of the sequences of

operations that they can and cannot engage in. This gives a way of considering Event-B machines as CSP processes, and treating them within the CSP framework. Note that the notion of *traces* for machines is dual to that presented in [1], where traces are considered as sequences of *states* rather than our treatment of traces as sequences of *events*.

The alphabet $\alpha(M)$ of a machine M is simply its set of *events*.

The traces of a machine M are those sequences of events $tr = \langle a_1, \ldots, a_n \rangle$ which are possible for M (after initialisation T): those that do not establish *false*:

$$traces(M) = \{tr \mid \neg[T;tr]false\}$$

Here, the weakest precondition on a sequence of events is the weakest precondition of the sequential composition of those events: $[\langle a_1, \ldots, a_n \rangle]P$ is given as $[a_1; \ \ldots \ ; \ a_n]P$.

The failures of a machine M are those pairs (tr, X) for which performing tr followed by refusing X is possible:

$$failures(M) = \{(tr, X) \mid \neg[T;tr]((\bigvee_{op \in X} G_{op}(c, v)))\}$$

In other words, it is not always the case that performance of tr is followed by some event from X being enabled.

A sequence of operations tr is a divergence if the sequence of operations is not guaranteed to terminate, i.e. $\neg[T; \ tr]true$. Thus

$$divergences(M) = \{tr \mid \neg[T;tr]true\}$$

M is divergence-free if $divergences(M) = \varnothing$.

These definitions provide the link between the weakest precondition semantics of the operations, and the CSP semantics of the B machine.

2.3 Combining CSP and Event-B

Figure 1 has defined the process P that alternates between two events. We do not require both the process P and the machine $M1$ in order to capture the requirement of alternating events. Either description independently would have been clear enough. Nonetheless, it is possible to combine the descriptions of P and $M1$ and we could view P as being an annotation of $M1$. This is exactly the way in which control flow expressions are being used in [10]. The author is using *flows* to provide patterns for the events of an Event-B machine, but does not permit the flows to contain events other than those in the Event-B machine. Being able to provide clear annotation of control flow is one benefit of including control flow expressions within Event-B machines. Another is to be able to relieve an Event-B machine of describing control flow explicitly and handing over that responsibility to a CSP process. Then the purpose of the Event-B machine is to define appropriate updates of the state of the machine.

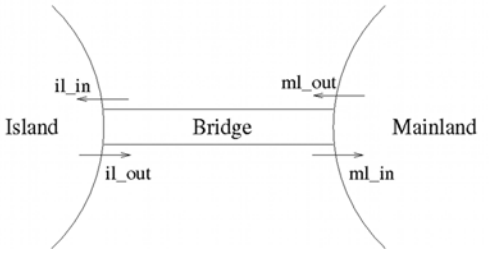

Fig. 2. Single lane bridge between mainland and island

Consider machine $M2$, also defined in Figure 1. The process P combined with $M2$ (i.e. $P \parallel M2$) also provides a definition which specifies alternating events. In this example we have handed over complete control to the CSP. We will illustrate in this paper a mixture of responsibilities in which both the CSP and the Event-B contribute to controlling the ordering of events within a system. It will enable us to see clearly how control flow is also restricted by the state of the system. To this end, we define a joint semantics for the integration in terms of the failures-divergence model of CSP. By giving a CSP semantics to an Event-B machine M, the CSP semantics of $P \parallel M$ follows from the CSP semantics of the parallel operator $\parallel$.

We will also explore in this paper how consistency conditions can be used to ensure deadlock-freedom of an integrated specification and how refinement can be proven in the integration.

3 Motivating and Running Example

We start with an example, inspired by Abrial's car-island example of [1], to exemplify the usefulness of control flow specifications in Event-B machines. The specification is of a single lane bridge going from the mainland (ml) to an island (il). The bridge has a capacity of 10 cars (stored in a **constant** CAP). Unlike [1], our island has no limit on the number of cars on it. The specification needs to ensure that cars only travel in one direction on the bridge; variables a and c are used to track the number of cars on the bridge travelling from mainland to island and vice versa.

Figure 2 shows the bridge and four events which are part of the abstract specification: ml_out and ml_in are events moving cars out of and into the mainland, and il_in and il_out are the corresponding events for the island. The abstract Event-B machine *Bridge*0 given in Figure 3 needs to guarantee that cars on the bridge only travel in one direction and that the bridge does not become overloaded. The guards of the events ensure this, e.g., a car may only move from mainland to bridge (ml_out) if the direction island-to-mainland is currently empty, which we can see from the value of variable c, and if the capacity of the bridge is not already reached, which we can see from a. When it enters the bridge in direction island, variable a is incremented.

Machine Bridge0 $\widehat{=}$
Variables a, c
Constants $CAP = 10$
Invariants $a, c \in \mathbb{N}$
Initialisation $a := 0, c := 0$
Event ml_out $\widehat{=}$ **when** $c = 0 \wedge a < CAP$ **then** $a := a + 1$ **end**
Event ml_in $\widehat{=}$ **when** $c > 0$ **then** $c := c - 1$ **end**
Event il_out $\widehat{=}$ **when** $a = 0 \wedge c < CAP$ **then** $c := c + 1$ **end**
Event il_in $\widehat{=}$ **when** $a > 0$ **then** $a := a - 1$ **end**

Fig. 3. Abstract bridge model

This constitutes our abstract specification. Here, there is no necessity of explicitly modelling control. The ordering of events depends on the data values of a and c only. This could also be modelled in CSP, but state is not CSP's primary domain.

3.1 Bridge with CSP Control

In a development step, the specification is refined so as to introduce traffic lights which regulate the flow of cars onto the bridge. There are two traffic lights here, one between mainland and bridge (`ml_tl`) and the second one between island and bridge (`il_tl`). Each can be either green or red. The single lane use of the bridge should now be achieved by proper switching of traffic light colours. In this setting it becomes obvious that certain orderings among events need to be specified, and CSP provides a natural way of doing so. The first part concerns the behaviour of car drivers: if car drivers would ignore red traffic lights, then the correctness of the system can never be achieved, and so we capture the expectation that drivers will not drive through a red light. The environment assumption about correct driver behaviour is formulated in CSP as $REQ1$ and $REQ2$:

$$REQ1 = \mathsf{ml\_tl\_green} \rightarrow P \qquad REQ2 = \mathsf{il\_tl\_green} \rightarrow Q$$
$$P = \mathsf{ml\_out} \rightarrow P \qquad Q = \mathsf{il\_out} \rightarrow Q$$
$$\Box\ \mathsf{ml\_tl\_red} \rightarrow REQ1 \qquad \Box\ \mathsf{il\_tl\_red} \rightarrow REQ2$$

These two processes specify constraints on cars going past the two traffic lights: ml_out is only possible when the mainland traffic light is green ($REQ1$) and a similar property needs to hold for il_out. $REQ1$ specifies a process which first of all executes event ml_tl_green and then has the choice of allowing ml_out or carrying on with ml_tl_red.

A second constraint contains the switching of traffic lights: at most one of them is allowed to be green, which gives a natural ordering on the events. The process $TL1$ allows the choice of which light to switch at any stage. Thus we model the choice between turning the island or the mainland light to green.

Machine Bridge1 $\widehat{=}$
Variables a, c
Constants $CAP = 10$
Invariants $a, c \in \mathbb{N}$
Initialisation $a := 0, c := 0$
Event ml_out $\widehat{=}$ **when** $a < CAP$ **then** $a := a + 1$ **end**
Event ml_in $\widehat{=}$ **when** $c > 0$ **then** $c := c - 1$ **end**
Event il_out $\widehat{=}$ **when** $c < CAP$ **then** $c := c + 1$ **end**
Event il_in $\widehat{=}$ **when** $a > 0$ **then** $a := a - 1$ **end**
Event ml_tl_green $\widehat{=}$ **when** $c = 0$ **then** *skip* **end**
Event il_tl_green $\widehat{=}$ **when** $a = 0$ **then** *skip* **end**

Fig. 4. The *Bridge*1 machine

$$TL1 = \mathsf{ml\_tl\_green} \rightarrow \mathsf{ml\_tl\_red} \rightarrow TL1$$
$$\Box\ \mathsf{il\_tl\_green} \rightarrow \mathsf{il\_tl\_red} \rightarrow TL1$$

The data dependent part of the system still remains in an Event-B machine, *Bridge*1, given in Figure 4. Observe that the guards $c = 0$ and $a = 0$ have been dropped from events ml_out and il_out respectively. Responsibility for ensuring this element of the condition that these events are enabled is now within the CSP part of the description, arising from the behaviour of the traffic lights, and the assumptions about correct driver behaviour. The combination of CSP and Event-B allows for a natural and clear separation of data-dependent and control-dependent aspects of a model. The resulting model is:

$$TL1 \parallel REQ1 \parallel REQ2 \parallel Bridge1$$

We will wish to show that this model is internally consistent: that the CSP control description is compatible with the Event-B model, and does not introduce new deadlocks. We will also want to be able to relate this model to the original abstract *Bridge*0 model, to demonstrate that it is a refinement. The next sections provide the underlying theory to enable us to consider these issues.

3.2 Event-B Bridge with Control

Figure 5 gives a pure Event-B machine which has the same behaviour as the combined model $TL \parallel REQ1 \parallel REQ2 \parallel Bridge1$, where all the control is managed within the event guards. We introduce a variable for each of the CSP control components: tl, $r1$, and $r2$, for $TL1$, $REQ1$, and $REQ2$ respectively. Their values correspond to the states of the processes: they are used as guards to enable events, and they are updated when events occur in accordance with the control process description. For example, $tl = reds$ is part of the enabling condition for the tl_green events, and tl is updated according to which light turns green.

Machine ControlledBridge $\widehat{=}$
Variables $a, c, tl, r1, r2$
Sets $LIGHTS = \{reds, mlgreen, ilgreen\}$
Constants $CAP = 10$
Invariants $a, c \in \mathbb{N} \wedge r1, r2 \in \{0, 1\} \wedge tl \in LIGHTS$
Initialisation $a := 0, c := 0, r1 := 0, r2 := 0, tl := reds$

Event ml_out $\widehat{=}$ **when** $a < CAP \wedge r1 = 1$ **then** $a := a + 1$ **end**

Event ml_in $\widehat{=}$ **when** $c > 0$ **then** $c := c - 1$ **end**

Event il_out $\widehat{=}$ **when** $c < CAP \wedge r2 = 1$ **then** $c := c + 1$ **end**

Event il_in $\widehat{=}$ **when** $a > 0$ **then** $a := a - 1$ **end**

Event ml_tl_green $\widehat{=}$ **when** $c = 0 \wedge r1 = 0 \wedge tl = reds$
then $r1 := 1 \parallel tl := mlgreen$ **end**

Event il_tl_green $\widehat{=}$ **when** $a = 0 \wedge r2 = 0 \wedge tl = reds$
then $r2 := 1 \parallel tl := ilgreen$ **end**

Event ml_tl_red $\widehat{=}$ **when** $r1 = 1 \wedge tl = mlgreen$
then $r1 := 0 \parallel tl := reds$ **end**

Event il_tl_red $\widehat{=}$ **when** $r2 = 1 \wedge tl = ilgreen$
then $r2 := 0 \parallel tl := reds$ **end**

Fig. 5. The *Bridge* machine with control incorporated within the guards

In contrast to the previous specification, we cannot directly see the flow of control on this machine anymore. There is no way of detecting that variables a and c are used for holding data values, whereas variables $r1$, $r2$ and tl are used for regulating the ordering of events. Furthermore, the switching of traffic lights and the requirement on the car drivers respecting traffic lights is now mixed together. The need for separation of different requirements on the model which we had in the bridge with CSP control is gone.

3.3 Abrial's Event-B Bridge

The standard Event-B approach taken by Abrial in the development of the bridge in [1] is to proceed by a series of refinement steps focusing on the state invariants, each of which introduces new events (such as the traffic lights), and where the control emerges gradually as proof obligations are discharged. For example, the requirement that at least one light must be red emerges from the requirement that all events should preserve the invariant that the bridge should not contain cars travelling in both directions.

The resulting bridge system is quite different to that in Figure 5, reflecting the fact that allowing implicit control to emerge naturally in an Event-B development is a different approach to imposing the flow of control explicitly, as we propose in this paper. However, it is recognised (anecdotally) [8] that establishing deadlock-freedom for such developments is often difficult in practice.

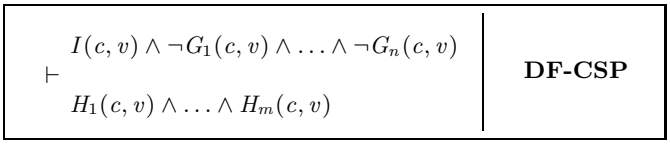

Fig. 6. Deadlock freedom for CSP control

4 Deadlock

Having formally defined the meaning of a combined CSP and Event-B specification now, we will next look at our two main issues: establishing deadlock-freedom in this section, and compositional refinement in the next section.

If a CSP control description P is introduced to a deadlock-free Event-B machine M, then there is a possibility that the additional constraints introduced by P might lead to a deadlock. This is possible since both CSP and Event-B define restrictions on the execution of events, and whenever these restrictions are not consistent for shared events the combined model may deadlock. In terms of the failures-divergence semantics this means that there is a trace after which all events are being refused.

Definition 2. *Let P be a CSP process and M an Event-B machine. The combination $P \parallel M$ is said to* deadlock *if there is a trace $tr \in (\alpha(P) \cup \alpha(M))^*$ such that $(tr, \alpha(P) \cup \alpha(M)) \in failures(P \parallel M)$.*

We will generally introduce a CSP controller over events which will be made available by the Event-B part of the description. Such events might always be enabled, but more generally we would only require them to be enabled at points where none of the other events (i.e. those not in the CSP, and therefore under the control of the Event-B) are enabled. This design principle gives rise to the proof rule **DF-CSP** given in Figure 6. This condition could for instance be established using the Rodin toolset [2]. In this rule, $G_1 \ldots G_n$ are the guards of the operations in $\alpha(M) \setminus \alpha(P)$ and $H_1 \ldots H_m$ are the guards of operations in $\alpha(M) \cap \alpha(P)$. The proof rule requires that whenever all of the events coming from the Event-B machine alone are disabled, then all events jointly controlled by the CSP process and the Event-B machine need to be enabled in the machine. Thus the machine cedes control at that point to the CSP controller.

Theorem 1. *Let P be a deadlock-free CSP process and M a divergence-free Event-B machine which satisfies DF-CSP. Then $P \parallel M$ is deadlock-free.*

This theorem considers the case where, whenever all of the events controlled by M alone are not enabled, then *all* of the events shared with P are enabled. In such a situation, deadlock-freedom of P yields deadlock-freedom of $P \parallel M$.

Observe that if $\alpha(M) \cap \alpha(P) \neq \varnothing$, then the condition on the operation guards of M implies that M is deadlock-free.

Theorem 1 is applicable to the *Bridge*1 example of Section 3. In that example we have that

$$\alpha(Bridge1) \setminus \alpha(TL1 \parallel REQ1 \parallel REQ2) = \{ml\_in, il\_in\}$$
$$\alpha(Bridge1) \cap \alpha(TL1 \parallel REQ1 \parallel REQ2) = \{ml\_out, il\_out, ml\_tl\_green, il\_tl\_green\}$$

If all of the guards in $\alpha(Bridge1) \setminus \alpha(TL1 \parallel REQ1 \parallel REQ2)$ are false, then we have $c = 0 \wedge a = 0$. This implies each of the guards in $\alpha(Bridge1) \cap \alpha(TL1 \parallel REQ1 \parallel REQ2)$, and hence implies their conjunction. This is the condition to conclude deadlock-freedom of $TL1 \parallel REQ1 \parallel REQ2 \parallel Bridge1$.

The condition establishes that *all* of *Bridge*1's events shared with $TL1 \parallel REQ1 \parallel REQ2$ are enabled whenever all of the events that *Bridge*1 controls independently are blocked. In such a state *Bridge*1 does not constrain $TL1 \parallel REQ1 \parallel REQ2$ at all, so $TL1 \parallel REQ1 \parallel REQ2$'s deadlock-freedom extends to $TL1 \parallel REQ1 \parallel REQ2 \parallel Bridge1$.

Theorem 1 is also applicable to the example $P \parallel M2$ of Figure 1. However, it is not applicable to $P \parallel M1$, since the two events of M are shared with P, but their guards are never both true, i.e. $\bigwedge_{op \in \alpha(M1) \cap \alpha(P)} G_{op}(c, v)$ does not hold.

A more general theorem for deadlock-freedom is available, as a specialisation of a result presented in [19] concerned with blocking B operations in classical B. It is applicable to sequential controllers, i.e. those made up of prefix, choice, and recursion, and focuses on the choices provided by the controller after any particular trace.

To state the theorem we need first to define for *sequential* CSP terms P:

- *offers*(P): the offers made by P at stages before a recursive call;
- *pass*(P): the traces corresponding to a single complete pass through a recursively defined process

Definition 3. *For a CSP term P, the set offers(P) is defined inductively as follows:*

$$\begin{aligned}
offers(a \rightarrow P) &= \{(\langle\rangle, \{a\})\} \cup \{(\langle a\rangle \frown tr, Off) \mid (tr, Off) \in offers(P)\} \\
offers(P1 \,\Box\, P2) &= \{(\langle\rangle, Off1 \cup Off2) \mid (\langle\rangle, Off1) \in offers(P1) \\
&\qquad \wedge (\langle\rangle, Off2) \in offers(P2)\} \\
&\quad \cup \{(tr, Off1) \mid (tr, Off1) \in offers(P1) \wedge tr \neq \langle\rangle\} \\
&\quad \cup \{(tr, Off2) \mid (tr, Off2) \in offers(P2) \wedge tr \neq \langle\rangle\} \\
offers(P1 \sqcap P2) &= offers(P1) \cup offers(P2) \\
offers(S) &= \{\}
\end{aligned}$$

Definition 4. *For a CSP term P, the set pass(P) is defined inductively as follows:*

$$\begin{aligned} pass(a \rightarrow P) &= \{\langle a \rangle \frown tr \mid tr \in pass(P)\} \\ pass(P1 \,\Box\, P2) &= pass(P1) \cup pass(P2) \\ pass(P1 \sqcap P2) &= pass(P1) \cup pass(P2) \\ pass(S) &= \{\langle\rangle\} \end{aligned}$$

Theorem 2. *For a recursive definition $N \mathrel{\widehat{=}} P$, if $\alpha(P) = \alpha(M)$ and if there is a (control loop invariant) predicate CLI such that*

- $[T]CLI$
- $\forall\, tr, Off.(tr, Off) \in offers(P) \Rightarrow (CLI \Rightarrow [tr](\bigvee_{op \in Off} G_{op}(c, v)))$
- $tr \in pass(P) \Rightarrow (CLI \Rightarrow [tr]CLI)$

then $P \parallel M$ is deadlock-free.

Consider P and $M1$ from Figure 1. We obtain

$$\begin{aligned} offers(P) &= \{(\langle\rangle, \{up\}), (\langle up\rangle, \{down\})\} \\ pass(P) &= \{\langle up, down\rangle\} \end{aligned}$$

We identify the control loop invariant CLI as $n = 0$, and check the conditions in turn:

- $[n := 0](n = 0)$ is indeed true.
- Checking the condition for the two offers in $offers(P)$: $n = 0 \Rightarrow [\langle\rangle]G_{up})$, and $n = 0 \Rightarrow [\langle up\rangle]G_{down})$ are both true.
- Checking the condition for the single pass: $n = 0 \Rightarrow [\langle up, down\rangle](n = 0)$ is also true.

The conditions are all true, so we conclude that $P \parallel M1$ is deadlock-free.

5 Refinement

In addition to introducing control to Event-B models, we are also interested in further *developing* an existing CSP $\parallel$ Event-B model. Both CSP and Event-B come with existing definitions of refinement [5] or development: in CSP this is process refinement and in Event-B data refinement. These guarantee the refinement to only have less traces or less failures than the abstract specification.

In contrast to process refinement, Event-B refinements usually also introduce new events. The corresponding notion of refinement in CSP would need to first *hide* ($\setminus$) these events in the concrete process and then check for trace or failures refinement. Hiding turns visible events into invisible, internal τ events. Thus for instance checking for trace refinement with new events A means checking $P \sqsubseteq_T Q \setminus A$.

The data refinement on machines discussed in Section 2.2 thus induces traces, failures, and divergences refinement. If $M_1 \sqsubseteq_D M_2$ in the Event-B setting, then $M_1 \sqsubseteq M_2 \setminus A$ in each of the CSP semantic models, where $A = \alpha(M_2) \setminus \alpha(M_1)$, the set of new events introduced in M_2.

Our objective is to achieve a compositional framework for refinement (like for integrations of CSP and Object-Z [18,15]). In the case of trace refinement, the refinement relations are compositional. In other words, separately refining the components of a CSP‖Event-B model results in a trace refinement of the model as a whole. Hence safety properties are preserved. This is expressed in Theorem 3.

Theorem 3. *Let P and P' be CSP processes such that $P \sqsubseteq_T P' \setminus A_1$, and M and M' Event-B machines such that $M \sqsubseteq_D M'$ with new events A_2. Then the following holds:*

$$P \parallel M \sqsubseteq_T (P' \parallel M') \setminus (A_1 \cup A_2) \ .$$

Unfortunately a similar result for failures refinement does not hold, and it is not in general possible to deduce particular liveness behaviour of $P' \parallel M'$ from that of $P \parallel M$. This is because parallel composition does not in general preserve liveness properties. However, Theorem 1 is applicable directly to $P' \parallel M'$, thus still allowing deadlock-freedom results to be established directly for the refined models. Furthermore, we are able to obtain a less general result: if there is no intersection between the new events introduced into P and those introduced into M, then failures refinement is preserved.

Theorem 4. *Let P and P' be CSP processes such that $P \sqsubseteq_F P' \setminus A_1$, and M and M' Event-B machines such that $M \sqsubseteq_D M'$ with new events A_2, where $A_1 \cap \alpha(M') = \varnothing$ and $A_2 \cap \alpha(P') = \varnothing$. Then the following holds:*

$$P \parallel M \sqsubseteq_F (P' \parallel M') \setminus (A_1 \cup A_2) \ .$$

Returning to our *Bridge* example, the bridge may occasionally need to be raised (to allow large ships through). This should occur only when there are no cars on the bridge in either direction, and also when the traffic lights are red in both directions. The lights should remain red until the bridge is lowered again.

This new feature is introduced in terms of new events in the Event-B model and the CSP description. The CSP description is augmented to capture the required relationship between the bridge lifting and the lights:

$$\begin{aligned} TL2 = {} & \mathsf{ml\_tl\_green} \rightarrow \mathsf{ml\_tl\_red} \rightarrow TL2 \\ & \Box\ \mathsf{il\_tl\_green} \rightarrow \mathsf{il\_tl\_red} \rightarrow TL2 \\ & \Box\ \mathsf{bridge\_raise} \rightarrow \mathsf{bridge\_lower} \rightarrow TL2 \end{aligned}$$

The requirement that no cars should be on the bridge when it is raised is captured naturally as a new Event-B machine *Bridge*2 consisting of *Bridge*1 augmented with the following event:

$$\mathsf{bridge\_raise} \ \widehat{=}\ \textbf{when}\ \ a = 0 \wedge c = 0\ \ \textbf{then}\ \ \mathit{skip}\ \ \textbf{end}$$

Considering the control and the model separately, we have $TL1 \sqsubseteq_T TL2 \setminus A$, where $A = \{\mathit{bridge\_raise}, \mathit{bridge\_lower}\}$, and also $\mathit{Bridge}1 \sqsubseteq_D \mathit{Bridge}2$ with new events A.

Theorem 3 yields that $TL1 \parallel Bridge1 \sqsubseteq_T (TL2 \parallel Bridge2) \setminus A$, and hence that

$$TL1 \parallel REQ1 \parallel REQ2 \parallel Bridge1 \sqsubseteq_T (TL2 \parallel REQ1 \parallel REQ2 \parallel Bridge2) \setminus A$$

This demonstrates that the new feature is compatible with the existing system.

Furthermore, *Bridge*2 meets **DF-CSP**, and $TL2 \parallel REQ1 \parallel REQ2$ is deadlock-free, so we can also conclude that $TL2 \parallel REQ1 \parallel REQ2 \parallel Bridge2$ is deadlock-free.

6 Conclusion

This paper has illustrated how CSP and Event-B descriptions can be combined and what reasoning can be performed on the combined models. The work resonates closely with [10] but is wider in scope because we want to consider using the process descriptions to specify requirements of a system which may not already be defined in the Event-B model. The benefit of splitting responsibility across both CSP and Event-B is that requirements can be dealt with separately. We must however investigate how global invariants can be expressed. In our example, one might say that if the most recent event is `ml_tl_green` then the number of cars coming the other way should be zero. i.e., $last(tr) = ml\_tl\_green \Rightarrow c = 0$. Since we are now combining descriptions, we lose the benefit of being able to express all invariants as state predicates. Further work is needed before we can conclude what can be expressed using a combination which would have been difficult using only Event-B predicates.

This paper is the basis of our ongoing research; we want to consider developing conditions which ensure that an introduction of a CSP process in a specification constitutes a valid refinement step, possibly using ideas of [6]. Mussat describes in [14] that we should be very clear about the separation between system variables and those that depict the physical environment, and it will be interesting to investigate whether CSP can contribute to the clear delineation of these aspects.

Acknowledgements

We are grateful to the anonymous reviewers for their thoughtful and constructive suggestions.

References

1. Abrial, J.-R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press, Cambridge (2010)
2. Abrial, J.-R., Butler, M.J., Hallerstede, S., Voisin, L.: A Roadmap for the Rodin Toolset. In: Börger, E., Butler, M., Bowen, J.P., Boca, P. (eds.) ABZ 2008. LNCS, vol. 5238, p. 347. Springer, Heidelberg (2008)

3. Butler, M.J.: csp2B: A practical approach to combining CSP and B. In: FACS, pp. 182–196 (2000)
4. Butler, M.J., Leuschel, M.: Combining CSP and B for specification and property verification. In: Fitzgerald, J.S., Hayes, I.J., Tarlecki, A. (eds.) FM 2005. LNCS, vol. 3582, pp. 221–236. Springer, Heidelberg (2005)
5. Derrick, J., Boiten, E.A.: Refinement in Z and Object-Z. Springer, Heidelberg (2001)
6. Derrick, J., Wehrheim, H.: Model transformations incorporating multiple views. In: Johnson, M., Vene, V. (eds.) AMAST 2006. LNCS, vol. 4019, pp. 111–126. Springer, Heidelberg (2006)
7. Fischer, C.: CSP-OZ - a combination of CSP and Object-Z. In: Bowman, H., Derrick, J. (eds.) Second IFIP International Conference on Formal Methods for Open Object-based Distributed Systems, pp. 423–438 (July 1997)
8. Hoang, T.S.: Personal Communication, Email (May 25, 2010)
9. Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall, Englewood Cliffs (1985)
10. Iliasov, A.: On Event-B and Control Flow. Technical report, School of Computing Science, Newcastle University (July 2009)
11. Mahony, B.P., Dong, J.S.: Blending Object-Z and timed CSP: An introduction to TCOZ. In: Futatsugi, K., Kemmerer, R., Torii, K. (eds.) 20th International Conference on Software Engineering (ICSE 1998). IEEE Press, Los Alamitos (1998)
12. Métayer, C., Abrial, J.-R., Voisin, L.: Event-B language. RODIN Project Deliverable 3.2, `http://rodin.cs.ncl.ac.uk/deliverables/D7.pdf` (accessed 25/5/10)
13. Morgan, C.: Of wp and CSP. In: Beauty is Our Business: a Birthday Salute to E. W. Dijkstra, pp. 319–326 (1990)
14. Mussat, L.: Modéles Réactifs. Technical report, ClearSy (July 2008)
15. Olderog, E.-R., Wehrheim, H.: Specification and (property) inheritance in CSP-OZ. Sci. Comput. Program. 55(1-3), 227–257 (2005)
16. Schneider, S.: Concurrent and Real-time Systems: The CSP approach. Wiley, Chichester (1999)
17. Smith, G.: A semantic integration of Object-Z and CSP for the specification of concurrent systems. In: Fitzgerald, J.S., Jones, C.B., Lucas, P. (eds.) FME 1997. LNCS, vol. 1313, pp. 62–81. Springer, Heidelberg (1997)
18. Smith, G., Derrick, J.: Specification, Refinement and Verification of Concurrent Systems-An Integration of Object-Z and CSP. Formal Methods in System Design 18(3), 249–284 (2001)
19. Treharne, H., Schneider, S.: How to drive a B machine. In: Bowen, J.P., Dunne, S., Galloway, A., King, S. (eds.) B 2000, ZUM 2000, and ZB 2000. LNCS, vol. 1878, pp. 188–208. Springer, Heidelberg (2000)
20. Woodcock, J., Cavalcanti, A.: The Semantics of Circus. In: Bert, D., Bowen, J.P., Henson, M.C., Robinson, K. (eds.) B 2002 and ZB 2002. LNCS, vol. 2272, pp. 184–203. Springer, Heidelberg (2002)