Towards a Model-Checker for *Circus*

Artur Oliveira Gomes[1(✉)] and Andrew Butterfield[2]

[1] Universidade Federal de Mato Grosso do Sul, Corumbá, Brazil
artur.gomes@ufms.br
[2] School of Computer Science and Statistics,
Trinity College Dublin, Dublin 2, Ireland
butrfeld@tcd.ie

Abstract. Among several approaches aiming at the correctness of systems, model-checking is one technique to formally assess system models regarding their desired/undesired behavioural properties. We aim at model-checking the *Circus* notation that combines Z, CSP, and Morgan's refinement calculus, based on the Unifying Theories of Programming. In this paper, we experiment with approaches for capturing *Circus* processes in CSP, and for each approach, we evaluate the impact of our decisions on the state-space explored as well as the time spent for such a checking using FDR. We also experimented with the consequences of model-checking CSP models that capture both state invariants and preconditions of *Circus* models.

1 Introduction

The use of formal methods provides a way to rigorously specify, develop, and verify complex systems. Among several approaches aiming at the correctness of systems, model-checking formally assesses given systems regarding their desired/undesired behavioural properties, through exhaustive checking of a finite model of that system.

Woodcock and Cavalcanti defined *Circus* [38], which is a formal language that combines structural aspects of a system using the Z language [40] and the behavioural aspects using CSP [36], along with the refinement calculus [23] and Dijkstra's guarded commands [10]. Its semantics is based on the *Unifying Theories of Programming (UTP)* [18]. In addition, a refinement calculus for *Circus* was developed by Oliveira [27], currently considered the de-facto reference for *Circus*, using tool support with ProofPower-Z [28]. More recently, Foster *et al.* introduced Isabelle/UTP, supporting *Circus* [11]. Moreover, *Circus* has a refinement calculator, CRefine [8], and an animator for *Circus*, Joker [26]. However, for model-checking, *Circus* is usually translated by hand to machine-readable CSP (CSP_M) [35] and then FDR [14] is used. We applied that method in our response to the Haemodialysis case study for ABZ'16 [16]. Model checking through FDR allows the user to perform a wide range of analysis, such as checks for refinement, deadlock, livelock, determinism, and termination.

© Springer Nature Switzerland AG 2019
M. H. ter Beek et al. (Eds.): FM 2019, LNCS 11800, pp. 217–234, 2019.
https://doi.org/10.1007/978-3-030-30942-8_14

Some related work on techniques for model-checking *Circus* was presented by Freitas [12] where a refinement model checker based on automata theory [19] and the operational semantics of *Circus* [39] was formalised in Z/Eves [34]. He also prototyped a model checker in Java. Moreover, Nogueira *et al.* [24] also presented a prototype of a model checker based on the operational semantics of *Circus* within the Microsoft FORMULA [21] framework. However, they could not provide a formal proof of the soundness of their approach, since FORMULA does not have an available formal semantics. Yet another approach for model-checking *Circus* was defined by Ye and Woodcock [41], who defined a link from *Circus* to $CSP\|B$ with model-checking using ProB [31]. Finally Beg [4] prototyped and investigated an automatic translation that supports a subset of *Circus* constructs.

Since CSP_M does not have a notion of variables for state as in Z, *Circus* or even the B-Method, we have to somehow capture them in order to obtain a CSP_M model as similar as possible to the original *Circus* one. Therefore, one could either use a memory model [25,30] in order to manage the values of the state variables, or else, to adopt the idea of *state-variable parametrised processes* [4].

Following the results presented in ABZ'16 [16], which involved manual translation, we decided to develop *Circus2CSP* [1], an automatic translator from *Circus* into CSP_M, aiming at model-checking with FDR. Our tool was then built based on the strategy presented in Sect. 5.3 of Deliverable 24.1 [29], from the COMPASS project [37], that defines a rigorous but manual translation strategy aiming at obtaining CSP_M specifications from *Circus*.

This paper reports design decisions regarding different approaches for model checking and experimental results obtained for *Circus* specifications. Such experiments were enough to identify an effective general form for any CSP_M model derived from *Circus*, where FDR could perform refinement checks with reduced time and memory consumption compared to existing approaches from the literature.

2 *Circus* Background

A *Circus* specification is in some sense an extension of Z [40] in that it takes the paragraphs of Z and adds new paragraph forms that can define *Circus* channels, processes and actions. Channels correspond to CSP events:

$$\textbf{channel}\ c : T$$

Circus actions can be considered as CSP processes extended with the ability to read and write shared variables, usually defined using a Z schema:

$$LocVars \mathrel{\widehat=} [v_1 : T_1, \ldots, v_n : T_n]$$

[1] See https://bitbucket.org/circusmodelcheck/circus2csp.

A *Circus process* is an encapsulation of process-local shared variables and *Circus actions* that access those local variables, along with a 'main' action.

$$
\begin{array}{l}
\textbf{process}\ ProcName\ \widehat{=}\ \textbf{begin} \\
\quad \textbf{state}\ PState == LocVars \\
\quad PBody\ \widehat{=}\ \langle \text{action defn.} \rangle \\
\quad PInit\ \widehat{=}\ \langle \text{action defn.} \rangle \\
\quad PMain\ \widehat{=}\ PInit; PBody \\
\bullet\ PMain \\
\textbf{end}
\end{array}
$$

Circus processes can only communicate with the external environment via channels, while *Circus* actions can also communicate via the local variables of their containing process. Processes can be modified and combined with each other, using the following CSP operators: sequential composition ($;$), non-deterministic choice ($\sqcap$), external choice ($\Box$), alphabetised parallel ($[\![\ldots]\!]$), interleaving ($|||$), iterated versions of the above (e.g., $\sqcap_{e \in E} \bullet \ldots$), and hiding ($\backslash$).

Circus actions can be built with the CSP operators detailed above, as well as the following CSP constructs: termination (Skip), deadlock (Stop), abort(Chaos), event prefix ($\rightarrow$), guarded action ($\&$), and recursion (μ). In addition a *Circus* action can be defined by a Z schema, or Dijkstra-style guarded commands, including variable assignment ($:=$). Note that actions cannot be defined as standalone entities at the top level of a *Circus* specification.

Parallel composition of *Circus* actions differs from that in CSP, in that we need to also specify which variables each side is allowed to modify. Parallel action composition, written as $A_1 \, [\![\, ns_1 \mid cs \mid ns_2 \,]\!] \, A_2$ states that action A_i may only modify variables listed in ns_i, where ns_1 and ns_2 are disjoint, and both actions must synchronise on events listed in cs. The semantics is that each side runs on its own copy of the shared variables, and the final state is obtained by merging the (disjoint) changes when both sides have terminated.

Circus also allows the use of local declarations in a variety of both process and action contexts. For actions, we can declare local variables, using $\textbf{var}\ x : T \bullet A$ which introduces variable v of type T which is only in scope within A. Variations of these can be used to define parameterised actions, of which the most relevant here is one that supports read-write parameters.

Finally, there is a refinement calculus for *Circus*, which is a fusion of those for both Z and CSP (failures-divergences)[27].

3 Translating *Circus* to CSP_M using *Circus2CSP*

Our first attempt to model check the *Circus* haemodialysis (HD) specification [16], was to manually translate it into CSP_M, and adjust its state-space until the desired checks could be successfully completed. This manual translation was error-prone, and this motivated the development of a mechanised

translator. Our plan was to provide a high degree of automation to minimise error-prone human interventions, in such a way that we have a basis for arguing for its correctness.

We started the development based on the *Circus*-to-CSP_M translation strategy developed for the EU COMPASS project and described in deliverable D24.1 [29, Section 5]. It specifies the translation in two parts: a function Ω that maps a *Circus* specification to an equivalent *Circus* specification using only the CSP subset of the *Circus* language; and a function Υ that translates CSP-as-*Circus* into machine-readable CSP_M (Fig. 1).

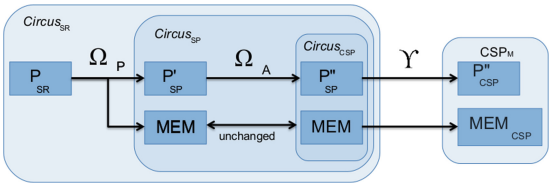

Fig. 1. Mapping *Circus* into CSP_M (derived from [29, Fig. 7, p77])

Function Ω has two phases: Ω_P and Ω_A. Function Ω_P extracts mutable state from the input state-rich ($Circus_{SR}$) process P_{SR} and gathers it in a new $Memory$ action, while replacing direct references to state in P_{SR} with appropriate "get" and "set" messages that communicate with that $Memory$, to obtain a state-poor ($Circus_{SP}$) process P'_{SP}. Function Ω_A then translates P'_{SP} into its CSP equivalent P''_{SP}, by replacing *Circus*-specific actions by CSP-as-*Circus* ($Circus_{CSP}$) equivalents. All of the transformations done by Ω_P and Ω_A are valid *Circus* refinement steps, each of which are in fact equivalences, defined in D24.1 [29, §5.3 and App. A].

3.1 The Memory Model

The need for a memory model arises from the fact that CSP does not naturally capture the notion of *mutable state*. One solution for that is to produce a state-poor process that communicates with a Memory model [25] that stores the values of state components and local variables from the original state-rich processes. Initially, our memory model was very similar to that in D24.1, with some differences in naming conventions. In our approach, we defined a notation for renaming the variables allowing the user to easily identify which are (global) state components, or local variables. Variables are renamed by adding a prefix $sv\_$ or $lv\_$ indicating respectively a state or local variable.

As part of the translation strategy, the CSP_M environment is redefined in terms of the type system. Based on the work of Mota *et al.* [25], D24.1 defined a union type $UNIVERSE$ containing any type defined in the specification. When translated into CSP_M, use is made of the `subtype` facility of that language to

manage the universe construction. Moreover, the names of every state component and local variable are defined as elements of a type $NAME$.

$$NAME ::= sv\_v_1 | sv\_v_2 | \ldots | sv\_v_n | lv\_l_1 | \ldots | lv\_l_k$$

The approach makes use of a set of bindings, $BINDING$, which maps all the names, $NAME$, into the $UNIVERSE$ type. In [29], a function δ is defined as a mapping between each variable in $NAME$ and its type, where each type (T_i is a subtype of $UNIVERSE$), and is used to define $Memory$.

$$BINDING == NAME \rightarrow UNIVERSE$$
$$\delta == \{sv\_v_1 \mapsto T_1, sv\_v_2 \mapsto T_2, \ldots, sv\_v_n \mapsto T_3, \ldots, lv\_l_k \mapsto T_m\}$$

As a result of applying the Ω functions, the state of a *Circus* process is replaced by a $Memory$ action parameterised by a read/write binding (**vres** b), which manages the mutable state, offering $mget$ and $mset$ channels carrying name/value pairs ($n.v$).

$$\begin{aligned} Memory \mathrel{\widehat=} & \textbf{ vres } b : BINDING \bullet \\ & (\Box n : \text{dom } b \bullet \ mget.n!b(n) \rightarrow Memory(b)) \\ & \Box \ (\Box n : \text{dom } b \bullet \ mset.n?nv : (nv \in \delta(n)) \rightarrow Memory(b \oplus \{n \mapsto nv\})) \\ & \Box \ terminate \rightarrow \text{Skip} \end{aligned}$$

Note, that while syntactically a *Circus* action, $Memory$ uses only the CSP subset of *Circus* Such a $Memory$ process runs in parallel with the main action of the translated *Circus* process, communicating through the channels $mget$ and $mset$. Moreover, the process execution ends when the $terminate$ signal is triggered. The above three channels compose the MEM_I channel set: **channelset** $MEM_I == \{|mget, mset, terminate|\}$.

The final specification puts the original process after Ω-translation in parallel with the memory model, synchronising on the MEM_I channels, which are themselves hidden at the top-level, with the binding as a top-level parameter. Note that the semantics of this at the top-level involves a non-deterministic choice[2] of the values in the initial binding b. This results in the following CSP form:

$$\sqcap b : BINDING \bullet \left(\begin{array}{l} (\Omega_A(P); terminate \rightarrow \text{Skip}) \\ \|_{MEM_I} \ Memory(b) \end{array} \right) \backslash MEM_I$$

Deliverable D24.1 contains manual proofs of the correctness of the translation [29, Appendix K].

4 Upgrading the Memory Model

With the initial version of the tool, we took examples from D24.1 (e.g. the ring-buffer example [29, Appendix D.2, p163]) and automatically translated them and

[2] A non-deterministic choice of values means that the bindings are picked randomly among the possible combinations of bindings.

then succesfully performed FDR checks. However, when we turned our attention to the somewhat larger HD model, we immediately uncovered some limitations of the basic translation, which were overcome by changing the memory model.

4.1 Limitation 1: Z Types vs. CSP_M Types

The use of the $UNIVERSE$ type, the CSP_M `subtype` feature, and a function written in CSP_M to map a name to its specific type, worked fine if all the types in $UNIVERSE$ were a sub-type of one supertype. In the D24.1 examples, all types were sub-types of the natural numbers. However, in the HD model we were developing, we had a mixture of natural sub-types, and enumerations. The type system in CSP_M does not consider enumeration types to be isomorphic to subtypes of any sufficiently large number type. We could have generated those isomorphisms, but these would have complicated the back-annotation problem, whenever a counter-example was found using FDR. Instead, we partitioned $UNIVERSE$ and $BINDING$ into the distinct supertypes present in the *Circus* model.

$$
\begin{aligned}
Memory \mathrel{\widehat=} \mathbf{vres}\ & b_{T_1} : BINDING_{T_1}, \ldots, b_{T_k} : BINDING_{T_k} \\
\bullet\ & \left(\begin{array}{l}
(\Box n_1 : \mathrm{dom}\ b_1 \bullet\ mget.n_1!b_1(n_1) \rightarrow Memory(b_1, \ldots, b_k)) \\
\Box \left(\begin{array}{l} \Box n_1 : \mathrm{dom}\ b_1 \bullet\ mset.n_1?nv : (nv \in \delta(n_1)) \\ \qquad\qquad \rightarrow Memory(b_1 \oplus \{n_1 \mapsto nv\}, \ldots, b_k) \end{array} \right) \\
\Box \ldots \Box (\Box n_k : \mathrm{dom}\ b_k \bullet\ mget.n_k!b_k(n_k) \rightarrow Memory(b_1, \ldots, b_k)) \\
\Box \left(\begin{array}{l} \Box n_k : \mathrm{dom}\ b_k \bullet\ mset.n_k?nv : (nv \in \delta(n_k)) \\ \qquad\qquad \rightarrow Memory(b_1, \ldots, b_k \oplus \{n_k \mapsto nv\}) \end{array} \right) \\
\Box\ terminate \rightarrow \mathrm{Skip}
\end{array} \right)
\end{aligned}
$$

We then changed the top-level view to have a non-deterministic choice over all the distinct bindings.

$$
\begin{aligned}
& \sqcap b_{T_1} : BINDING_{T_1}, \ldots, b_{T_k} : BINDING_{T_k} \\
& \quad \bullet \left(\begin{array}{l} (\Omega_A(P); terminate \rightarrow \mathrm{Skip}) \\ \|_{MEM_I}\ Memory(b_{T_1}, \ldots, b_{T_k}) \end{array} \right) \setminus MEM_I
\end{aligned}
$$

4.2 Limitation 2: FDR Time/Space Explosion

We quickly discovered that using this translation, we could only check *Circus* models with a small number of state variables, usually less than ten, with even the hand-translation of the HD model done for the original case-study being more effective. We proceeded to experiment with transformations to the memory model, justified by the *Circus* refinement laws.

Variables Have Non-deterministic Start Values. We first changed the top-level non-deterministic choice over the various bindings by replacing it with parameters.

$$\mathbf{var}\ b_{T_1} : BINDING_{T_1}, \ldots, b_{T_k} : BINDING_{T_k} \bullet \left(\begin{array}{l} (\Omega_A(P); terminate \rightarrow \text{Skip}) \\ \|_{MEM_I}\ Memory(b_{T_1}, \ldots, b_{T_k}) \end{array} \right) \setminus MEM_I$$

This is an equivalence, as $(\mathbf{var}\ x : T \bullet\ A(x)) = (\sqcap x : T \bullet\ A(x))$. However, FDR treats the latter as being parameterised by x and requires it to be given an initial value. This means that we can only check a very strong proper refinement, rather than the full equivalence. However, we argue that in the safety-critical domain in general, it is always mandatory to initialise all variables. If $Init$ is an action that initialises each variable precisely once with a constant value, with no intervening participation in events, then, regardless of the assignment ordering or any arbitrary initial value of any variable, the outcome is always the same: $s' = S_0$, where S_0 is the assignment of those constants to the coprresponding variables. If we insist on proper initialisation, then equivalence is restored. Given that the main usage of model-checking takes place in safety critical domains, we consider this a reasonable trade-off, particularly because it resulted in FDR performance improvements. However, our experiments revealed that a process translated this way, with more than ten state variables, still could not be checked with FDR in a reasonable time.

Distributed Memory Model. The final step, was to do more partitioning, moving to a situation were every variable gets its own memory process. The supertype bindings were retained at the top-level, but each variable's memory process was parameterised by the relevant binding with its domain restricted to just the name of that variable. So, for example, if variable n_i has a type whose supertype is T, then we first define a binding b_T for that supertype, and use it to parameterise a memory action for all variables of that supertype, which is itself the parallel composition of a memory process for each such variable, all synchronising on $terminate$, but interleaving all the $mget$ and $mset$ events:

$$\begin{array}{l} MemoryT(b_T) \mathrel{\widehat=} \\ \quad [\![\{\!| terminate |\!\}]\!] n : \text{dom}\ b_T \bullet\ MemoryTVar(n, \{n\} \lhd b_T) \end{array}$$

Here $N \lhd \mu$ restricts the domain of map μ to set N. We then define a parameterised process that represents a single variable:

$$\begin{array}{l} MemoryTVar(n, b) \mathrel{\widehat=} \\ \quad mget.n.b(n) \rightarrow MemoryTVar(n, b) \\ \quad \Box\ mset.n?nv : \delta(n) \rightarrow MemoryTVar(n, b \oplus n \mapsto nv) \\ \quad \Box\ terminate \rightarrow \text{Skip} \end{array}$$

The entire memory is constructed by putting the memories for each supertype in parallel, in the same way as for the individual variable processes.

$$\begin{array}{l} Memory(b_{T_1}, \ldots, b_{T_k}) \mathrel{\widehat=} \\ MemoryT_1(b_{T_1}) [\![\{\!| terminate |\!\}]\!] \ldots [\![\{\!| terminate |\!\}]\!] MemoryT_k(b_{T_k}) \end{array}$$

This last transformation produced a marked improvement in the time and memory consumption of FDR when checking models.

In the next section we describe and discuss our experiments on the HD machine mode comparing some of approaches above. Moreover, we also compare the results obtained using other tools as a way of assessing our results.

5 Experimental Results

In this section we present the tests we performed using our tool, *Circus2CSP*, exploring ways of overcoming any limitations from FDR, as well as comparing our approach with others from the literature. Firstly, we explore the interference of invariants and preconditions in CSP_M. Then, we compare *Circus2CSP* with the model from [16]. We also the effects of using some compression techniques available in FDR. Finally, we compare different approaches for modeling the Ring Buffer case study.

One of the requirements when model-checking a system is to produce a model whose range of values is enough for covering any condition imposed by an operation. However, when including the state invariant, we are also restricting the range of values permitted to be used within the system. From the example of the chronometer [27], we know that both *min* and *sec* was declared as natural numbers. However, while thinking of a chronometer in the real world, we know that neither a second, nor a minute goes beyond 59 units, without flipping the next unit counter. Therefore, it is safe to restrict the range of *min* and *sec* to $0 \,..\, 60$, where 60 is an unexpected value in the system.

We experimented with the impact of explicitly including invariant and precondition checks using the example of the Chronometer [27], with a new process *Chrono*. When using the translation rules presented in [29], we noticed that it is hard for FDR to check the model: it was translated using the conversion from normalised schemas to specification statements and from there, to the appropriate rules that introduce a condition that checks if *pre* is satisfied. If satisfied, it behaves as a non-deterministic choice of values from the state variables that satisfies both invariant and precondition, followed by updating these values in the memory model. Otherwise, if *pre* is not satisfied, it behaves like *Chaos*.

Our example of the chronometer has only two state variables and the results obtained using FDR are enough to show how the invariant checks throughout the specification increase the time spent during the assertion check in FDR. We deliberately modified the original model with the inclusion of the state invariant restricting both *min* and *sec* to values below 60, in order to experiment with the translated model in FDR.

$$
\begin{array}{l}
\textbf{process}\ Chrono \mathrel{\widehat=} \textbf{begin} \\
\quad \textbf{state}\ AState \mathrel{\widehat=} [sec, min : \mathbb{N} \mid min < 60 \land sec < 60] \\
\quad AInit \mathrel{\widehat=} [AState' \mid sec' = 0;\ min' = 0] \\
\quad IncSec \mathrel{\widehat=} [\Delta AState \mid sec' = (sec + 1) \bmod 60] \\
\quad IncMin \mathrel{\widehat=} [\Delta AState \mid min' = (min + 1) \bmod 60] \\
\quad Run \mathrel{\widehat=} \left(\begin{array}{l} tick \rightarrow \big(IncSec\big)\ ;\ \Big(\big(sec = 0\big)\ \&\ \big(IncMin\big) \Box \big(sec \neq 0\big)\ \&\ \mathrm{Skip} \Big) \\ \Box\ time \rightarrow out!(min, sec) \rightarrow \mathrm{Skip} \end{array} \right) \\
\quad \bullet\ AInit\ ;\ (\mu X \bullet Run\ ;\ X) \\
\textbf{end}
\end{array}
$$

We illustrate our experiment in Table 1 while exploring the inclusion of state invariants and precondition verification in the chronometer model, and used the following derived models[3]:

$D241$	Model manually translated using the approach from [29] without invariants and preconditions, using a non-deterministic choice of any set of bindings.
$D241Inv$	Model manually translated using the approach from [29] including the invariants as a restriction to the bindings set.
$D241Pre$	Model manually translated using the approach from [29] which includes precondition checks before the operations, but no invariants in the main action.
$D241InvPre$	Combination of $D241Inv$ and $D241Pre$.
$CTOC$	Model translated using our improved translation rules, the result from our tool *Circus2CSP*, as discussed in Sect. 4 (no invariant checks).
$CTOCPre$	Extension of $CTOC$ model where pre-condition checks, as done for $D241Pre$, are entered manually.

From the models above, our tool is able to automatically generate $CTOC$, $CTOCPre$ was obtained by manually modifying $CTOC$, while the others were generated by hand. We performed checks for deadlock freedom[4] using the translated models in the six variants above, combined with a different range of values for natural numbers, ranging from $\ldots 3$ to $0 \ldots 60$. For example, in a specification where the values for natural numbers are restricted to the range $0 \,..\, 10$, the process state was defined as $[min, sec : 0 \,..\, 10 \mid min < 10 \land sec < 10]$.

We noticed a first difference between models $D241$ and $D241Inv$, on one hand, and $CTOC$ and $CTOCPre$ on the other. The number of states visited for checks with the models $D241$ and $D241Inv$ was over 10-fold larger than for $CTOC$ and $CTOCPre$. However, the influence of a precondition check within an operation makes a significant reduction in the state exploration, but with the price of spending more time computing preconditions, as seen in Table 1, between $CTOC$ and $CTOCPre$. Moreover, we also observed that the checks

[3] The files used in this experiment can be found in the tool repository at https://bit.ly/2ONnk2T.

[4] The tests were performed using *Intel Core i7* 2.8 GHz CPU with 16GB of RAM.

Table 1. Interference of invariants and preconditions in CSP_M—Deadlock freedom checks (in seconds unless indicated otherwise)

	CTOC		CTOCPre		D241		D241Inv		D241InvPre		D241Pre	
Values range	Exec time	States visited	Exec time	States visited	Exec time	States visited	Exec time	States visited	Exec time	States visited	Exec time	States visited
0..3	0.116	68	0.134	21	0.206	1085	0.177	610	0.173	190	0.187	337
0..6	0.242	260	0.373	42	0.416	12734	0.35	9355	0.393	1513	0.428	2059
0..9	0.559	578	1.4	63	1.158	57791	1.138	46810	1.826	5104	1.955	6301
0..12	1.246	1022	4.197	84	2.714	172706	2.57	147157	5.22	12097	5.45	14197
0..15	2.533	1592	9.867	105	5.846	407537	5.452	358186	11.988	23626	12.6	26881
0..60	3m27s	25262	22m29s	1024	2h48	99M	1h40	91M	52m28s	3.7M	1h05	3.8M

for invariants has a weaker effect on states visited, when comparing the results between $D241InvPre$ and $D241Pre$. We also noticed that all variants of $D241$ were executed in a much larger time frame than the approaches using the translation from our tool, *Circus2CSP*. However, the models generated by our tool do not include either invariants or preconditions.

Finally, as a way of experimenting with the real world example of the chronometer, we examined the models with numbers ranging from 0 up to 60, as presented in the last row of Table 1. We see a significant difference among the results from the approaches evaluated, where the model using $CTOC$ was evaluated (3 min) by FDR, which is 97% less time than the time spent to check the model using $D241Inv$ (over 1h40) and 94% less than $D241Pre$ (1h05). In general, the CSP_M models ($CTOC$) translated using our tool were evaluated by FDR using a much smaller state space and were checked in less time than all the other models we tried. Such a result shows how different models of the same system can be affected by the checks of invariants and preconditions, as well as how optimising the memory model can result in much smaller state exploration when using FDR. Finally, we observed no correlation between time and state visited, in spite of the use (or not) of compression by default in FDR.

5.1 Haemodialysis (HD) Machine Experiments

The manual translation (herein `byHand`) of the *Circus* [16] HD model resulted in a CSP_M specification with twice as many lines as the *Circus* model. Using the `CTOC` translation results in CSP_M with approximately 75% fewer lines than the corresponding *Circus* file.

Our reference *Circus* model was that of the HD machine running in parallel with a model of one of the case study requirements (**R-1** [2, Section 4.2, p11]). The requirement model is effectively a monitor that observes the machine model, checking that it is satisfied, and deadlocking if it observes a violation. We then check the proposition that the HD model is correct w.r.t **R-1** by showing that the combination is deadlock free. In addition to comparing various translation schemes, we also explored the effect of changing the size of our "natural number" type: $NatValue == 0 \,..\, N$, in order to estimate the number of states visited in FDR.

We explored the `byHand` and `CTOC` translation schemes with four ranges of $NatValue$ size, with N up to a maximum of 90, as shown in Table 2. The only case where we could compare the two approaches was our first case, with $N = 2$: it resulted in 9,409 states visited using `byHand`, in contrast with 811 states visited using `CTOC`, demonstrating a reduction of 91% in terms of states explored. Moreover, the execution time with the model generated using `CTOC` was equally reduced by 91% compared to the model using `byHand`. The "Plys" column indicates how deep the breadth-first search algorithm used by FDR went while checking. This is larger for the `byHand` model, and is independent of the value of N. Interestingly, after waiting more than 2 h, we were unable to obtain results from the model generated with `byHand` when we increased the N to 3. However, the model generated with `CTOC`, when tested using $n = 90$, was executed in 35 s, which is still quicker than `byHand` with $N = 2$. We also note that amount of memory used was constant, at 240 MB approx.

Table 2. Time for asserting deadlock freedom of the HD Machine in FDR4

Approach	NatValue range	Result	States visited	Transitions visited	Plys visited	Exec. time
CTOC	0..1	Passed	811	1,800	39	0.375 s
	0..2	Passed	1,761	3,786	39	0.407 s
	0..10	Passed	21,169	44,586	39	0.937 s
	0..90	Passed	1,369,809	1,369,809	39	35.097 s
byHand	0..1	Passed	9,409	301,617	47	40.826 s
	0..2	Incomplete	?	?	?	>2 h

We could not get results here for the `D241` scheme as its translation of the HD model resulted in type errors being reported by FDR.

In addition to experiments that varied N above, we also explored how the *number* of variables, rather than the size of their datatypes, influenced the checking time. Using a hypothetical example having 12 state variables, checks using `D241` were performed in 35 min, compared to 76 ms using `CTOC`. We observed segmentation faults using `D241` with a more than 12 variables. However, checks using `CTOC` in an example with 42 state variables and $NatValue = 0..30$, were performed in 870 ms. What is clear is that with the `CTOC` translation scheme, namely one memory-process per state-variable, we can now handle *Circus* models of considerable complexity.

5.2 Ring-Buffer Experiments

Another interesting example was to take the *Circus* specification of the bounded reactive ring buffer, RB, from D24.1 [29, Appendix D.2, p. 163], based on the model presented in [7]. We compared the `CTOC` translation of this using *Circus2CSP* (RB_{CTOC}), with the by-hand translation in D24.1 [29, Appendix D.4,

Table 3. *RingBuffer* checks: deadlock and livelock freedom, and determinism.

Test	Model	Result	States visited	Transitions	Plys	Exec. time
Deadlock free	RB_{byH}	Passed	8,297,025	16,805,249	44	26.657 s
	RB_{CTOC}	Passed	1,628	3,109	38	0.145 s
Livelock free	RB_{byH}	Passed	8,297,025	16,805,249	44	25.476 s
	RB_{CTOC}	Passed	1,628	3,109	38	0.151 s
Deterministic	RB_{byH}	Passed	9,869,889	19,852,673	69	54.863 s
	RB_{CTOC}	Passed	2,012	3,853	63	0.159 s

Table 4. Refinement checks between models of the Ring Buffer example

	Refinement check	Result	States visited	Transitions visited	Plys visited	Exec. time
1	$RB_{byH} \sqsubseteq_{FD} RB_{CTOC}$	Passed	1,628	3,109	38	58.019 s
2	$RB_{CTOC} \sqsubseteq_{FD} RB_{byH}$	Passed	8,297,025	16,805,249	44	42.543 s

p166] (RB_{CTOC}). We firstly perform the usual tests like deadlock freedom and termination checks for $the RB_{CTOC}$ and for the RB_{CTOC} specifications, as illustrated in Table 3.

We can see a clear difference between the states visited between the three approaches, notably those between RB_{byH} and RB_{CTOC} where the number of states and transitions visited was reduced considerably, as well as the amount of time spent by FDR4 to check the assertions.

We also experimented to check the failures-divergences refinement ($P \sqsubseteq_{FD} Q$) between the three approaches, each pair in both directions. Since we know that the specification RB_{CTOC} is a translation from the same *Circus* model of the handmade translation of RB_{byH}, we expect that RB_{byH} and RB_{CTOC} are equivalent to each other, $RB_{byH} \sqsubseteq_{FD} RB_{CTOC}$ and $RB_{CTOC} \sqsubseteq_{FD} RB_{byH}$, which is true, as seen below in row 1 and 3.

Interestingly, if we compare the states and transitions visited, as well as the execution time from Tables 3 with 4, given a refinement $A \sqsubseteq_{FD} B$, the states and transitions visited are almost the same as when checking B for deadlock freedom.

During our experiments, we also compared our *Circus2CSP* model with the Ring Buffer model RB_{KW}, based on [40, Chapter 22], produced using the approach of Ye and Woodcock [41] for translating *Circus* into `CSP||B`, for model checking using ProB [22]. Such an approach is similar [29, p. 116] but makes use of Z schemas as *Circus* actions that are currently not available in our translation scheme. In our experiments, we observed that the model RB_{KW} is refined by both RB_{CTOC} and RB_{byH}, but the refinement in the reverse direction does not hold, *i.e.*, RB_{KW} is not a refinement of neither RB_{CTOC} nor RB_{byH}, as it is a more abstract model since its data aspects of specification are defined in B.

Unfortunately, the structure defined for our translation strategy is not fully supported by ProB, which was used to test RB_{KW} [42]. ProB is another model-

checker, which like FDR, also allows the user to animate specifications. It was originally developed for the B language, but it has been extended and now it supports other formal languages such as CSP, Z, Event-B [1], as well as combined languages such as `CSP||B`. We observed that the use of `subtype`, in our models, is not fully supported by the ProB tool, causing some commands like "model-check" to result in errors. However, we were able to animate our translated specification using ProB, and to execute the same assertion check, as in FDR: we obtained similar results to those when running FDR.

On the other side, the tests performed with the CSP_M specification of RB_{KW} using FDR failed the checks for deadlock freedom and determinism. The results obtained from ProB can be related to what we obtained in FDR in terms of the behavior of the system: the counterexample given can be used to animate the `CSP||B` model in ProB, causing the same effect: deadlock. However, we have no way to fully compare both approaches since `CSP||B` takes into account the system state in ProB, whereas we only have the CSP_M side of the model, which captures the behavior of the system, but does not captures the system state. The most obvious explanation for the deadlock in RB_{KW} is that the state (modeled in `B`) influences control-flow that results in deadlock situations being avoided.

5.3 Compression Experiments

An important aspect when using FDR is the availability of compression techniques [33] in order to reduce the number of states, reducing the time spent for refinement checking. A compression transforms a labelled-transition system (LTS) into a corresponding one, which is expected to be smaller and more efficient whilst using it for checks in FDR. Currently, FDR applies compressions in parallel compositions by default, which is the main structure we use in our memory model. We explored a few other compression tecniques, such as *sbisim*, which determines the maximal strong bisimulation [5], and *wbisim*, which computes the maximal weak bisimulation. Depending on the compression used, the number of states visited, were indeed reduced, as illustrated in Table 5.

Table 5. Experimenting CSP_M compression techniques with the HD Machine

	sbisim+diamond		No compression		sbisim		wbisim	
Values range	States visited	Exec time (seconds)	States visited	Exec time (seconds)	States visited	Exec time (seconds)	States visited	Exec time (seconds)
0..10	77	0.499	21,169	0.458	302	0.479	87	0.56
0..120	77	25.096	2,416,749	18.805	302	21.793	87	35.839
0..240	77	114.845	9,556,509	84.803	302	100.112	87	175.846
0..360	77	327.815	21,419,469	235.236	302	269.414	Killed	286.079
0..480	77	668.437	38,005,629	467.602	302	523.825	Killed	525.889

Although the states/transitions/plys visited were considerably reduced using the compression techniques mentioned above, there was little impact on overall

execution time, and the number of states visited are independent of the size of $NatValue$, while the number of transitions grows slowly. However, the results obtained here are related to the model of the HD machine, and it is difficult to identify which compression technique will be most effective in a general case, and indeed, further experiments are required.

6 Future Work

Our plans for future work include exploring other industrial-scale case studies [3,15,17], as a way of identifying the kind of *Circus* constructs that would be suitable to have available in our translation tool. We have a particular interest in specifying a translation strategy for Z schemas used as *Circus* actions within a process. The best approach would be to use Z Refinement Calculus [6]. For now, our tool deals only with those schemas that in fact can be translated into assignments. We intend to explore the operators for Z schemas and the refinement laws that can be applied accordingly.

In addition, we also plan to establish a link between *Circus2CSP* and Isabelle/UTP [11], so that we can use their mechanised UTP semantics for *Circus* to verify the correctness of our Haskell implementation. Moreover, our tool also has a *Circus* refinement "calculator" embedded in it, which implements the laws listed in Appendix A of the Deliverable 24.1 [29, p.147], which can easily be extended to the other refinement laws proved by Oliveira [27] in the near future.

We can eliminate the use of CSP_M subtyping in `CTOC` (the process-per-variable model), and simplify "get" and "set" prefixes of the forms $mget.n.v$ and $mset.n.v$ to $get\_n.v$ and $set\_n.v$ respectively, where we now have dedicated channels per variable. However, the relationship of this new form to `CTOC` is no longer a simple equivalence as there are now different events in the two models.

Finally, in terms of improvement of our tool, compared to other approaches [9], it would also be interesting to review the parser of Z and *Circus* from *Circus2CSP* in order to rewrite it to be in conformance with the International Standards Organization (ISO) standards, ISO/IEC 13568:2002 [20], which describes the syntax, type system and semantics of Z formal notation. Moreover, we would like to include the *libcspm* library[5] into *Circus2CSP* in order to be able to parse the relevant code included in our definition of the *assertion* LaTeX environment. Such an attempt would help a *Circus2CSP* user wishing to review any fault in the CSP_M specification translated from *Circus*.

Finally, we can envisage work in the future that might extend the benefits gained here to the wider model-checking community. One possibility is extending the translator to target model-checkers other than FDR. This would require us to have either a rigourosly defined embedding, of the subset of CSP that we produce, into the modelling language of the proposed checker, or have a way of linking the semantics of the target modelling language to *Circus* and/or CSP to verify the correctness of direct output in that language. The second aspect

[5] https://github.com/tomgr/libcspm.

concerns the possibility that our approach can be adapted to work within another model-checking eco-system entirely. ne key advantage in having a state-rich form is the ability to easily describe state changes that only modify small parts of the state (compare $P = w := y - x; Q$ with $P(u, v, w, x, y, z) = Q(u, v, y - x, x, y, z)$). We note that the CADP system, which is based on LOTOS (state-poor), has already moved in this direction, with tools now working with LTN (LOTOS New Technology, state-rich), using a LTN to LOTOS translator [13]. Do other modelling notations have state-rich forms that are hard to check, but have good checkers for state-poor forms?

7 Conclusions

In this paper we evaluated possible approaches for translating *Circus* into CSP_M, for model checking using FDR. Our main concern was how the state of a *Circus* process could be captured in CSP_M in such a way that FDR could handle a large amount of state variables and an even larger range of values. We then produced several models of CSP_M specifications translated from *Circus* and also explored the consequences of including both state invariants and preconditions of *Circus* actions in the CSP_M models. Such a research resulted in the development of *Circus2CSP*, a tool for model checking *Circus*, through the automatic translation from *Circus* to CSP_M, and therefore, being able to use FDR for refinement checks. *Circus2CSP* development was developed in 24 months, and has a total of over 26 thousand lines of Haskell code.

We observed that a distributed memory model, rather than a centralised one, as proposed by Mota *et al.* [24] is beneficial for larger states. Moreover, the time spent as well as the state exploration from FDR's refinement checks is larger when capturing preconditions and state invariants. Another observation from our experiments is that we were able to reduce the state exploration even more by refining our model to one where the bindings were explicitly defined by *Circus2CSP*, rather than considering a non-deterministic choice over such bindings, as per the original manual translation. This is justified by assuming that every state variable should be initialised prior to its use in the process. The outcome is that we now have a mechanised translator from *Circus* to CSP_M that produces tractable models, and allows the use of FDR on larger case studies than has been possible up to now.

We should clarify that our approach to produce parametrised processes is not an attempt to use the bindings *data-independently* [32, p. 453]. That is solving a different problem, namely finding a finite size of a type that is suitable to demonstrate the correctness for any finite or even infinite size of such type. Moreover, to date, our approach is unable to generate counterexamples or any kind of back annotation to the *Circus* models, and thus is in our plans for future work.

We used the HD machine and the ring buffer case studies as examples in order to test the capabilities of our tool whilst model checking the automatically translated models in FDR. Our aim was to contribute to reducing FDR's

workload in order to model check larger systems. We learned that a practical implementation/mechanisation of a theory may reveal difficulties that could not otherwise be discovered without extensive use of a tool prototype, especially when applying it to larger case studies.

Acknowledgments. This work was funded by CNPq (Brazilian National Council for Scientific and Technological Development) within the Science without Borders programme, Grant No. 201857/2014-6, and partially funded by Science Foundation Ireland grant 13/RC/2094.

References

1. Abrial, J.R.: Modeling in Event-B: System and Software Engineering, 1st edn. Cambridge University Press, New York (2010)
2. Mashkoor, A.: The hemodialysis machine case study. In: Butler, M., Schewe, K.-D., Mashkoor, A., Biro, M. (eds.) ABZ 2016. LNCS, vol. 9675, pp. 329–343. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-33600-8_29
3. Beg, A., Butterfield, A.: Linking a state-rich process algebra to a state-free algebra to verify software/hardware implementation. In: Proceedings of the 8th International Conference on Frontiers of Information Technology - FIT 2010, pp. 1–5 (2010). http://portal.acm.org/citation.cfm?doid=1943628.1943675
4. Beg, A., Butterfield, A.: Development of a prototype translator from Circus to CSPm. In: Proceedings of ICOSST 2015–2015 International Conference on Open Source Systems and Technologies, pp. 16–23, December 2016
5. Boulgakov, A., Gibson-Robinson, T., Roscoe, A.W.: Computing maximal weak and other bisimulations. Form. Asp. Comput. **28**(3), 381–407 (2016). https://doi.org/10.1007/s00165-016-0366-2
6. Cavalcanti, A., Woodcock, J.C.P.: ZRC - a refinement calculus for Z. Form. Asp. Comput. **10**(3), 267–289 (1998). http://link.springer.com/10.1007/s001650050016, https://doi.org/10.1007/s001650050016
7. Cavalcanti, A.L.C., Sampaio, A.C.A., Woodcock, J.C.P.: A refinement strategy for Circus. Form. Asp. Comput. **15**, 146–181 (2003). https://doi.org/10.1007/s00165-003-0006-5
8. Conserva Filho, M., Oliveira, M.V.M.: Implementing tactics of refinement in CRefine. In: Eleftherakis, G., Hinchey, M., Holcombe, M. (eds.) SEFM 2012. LNCS, vol. 7504, pp. 342–351. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33826-7_24
9. CZT Partners: Community Z tools, October 2006. czt.sourceforge.net/
10. Dijkstra, E.W.: Guarded commands, nondeterminacy and formal derivation of programs. Commun. ACM **18**(8), 453–457 (1975). http://portal.acm.org/citation.cfm?doid=360933.360975%5Cn
11. Foster, S., Zeyda, F., Woodcock, J.: Isabelle/UTP: a mechanised theory engineering framework. In: Naumann, D. (ed.) UTP 2014. LNCS, vol. 8963, pp. 21–41. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-14806-9_2
12. Freitas, L.: Model checking Circus. Ph.D. thesis, Department of Computer Science, The University of York, UK (2005)
13. Garavel, H., Lang, F., Serwe, W.: From LOTOS to LNT. In: Katoen, J.-P., Langerak, R., Rensink, A. (eds.) ModelEd, TestEd, TrustEd. LNCS, vol. 10500, pp. 3–26. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68270-9_1

14. Gibson-Robinson, T., Armstrong, P., Boulgakov, A., Roscoe, A.W.: FDR3 – a modern model checker for CSP. Tools Algorithms Constr. Anal. Syst. **8413**, 187–201 (2014). http://www.cs.ox.ac.uk/projects/fdr/manual/
15. Gomes, A.O.: Formal Specification of the ARINC 653 Architecture Using Circus (2012). http://etheses.whiterose.ac.uk/id/eprint/2683
16. Gomes, A.O., Butterfield, A.: Modelling the haemodialysis machine with *Circus*. In: Butler, M., Schewe, K.-D., Mashkoor, A., Biro, M. (eds.) ABZ 2016. LNCS, vol. 9675, pp. 409–424. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-33600-8_34
17. Gomes, A.O., Oliveira, M.V.M.: Formal specification of a cardiac pacing system. In: Cavalcanti, A., Dams, D.R. (eds.) FM 2009. LNCS, vol. 5850, pp. 692–707. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-05089-3_44
18. Hoare, C., He, J.: Unifying Theories of Programming. Prentice-Hall, Upper Saddle River (1998)
19. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation - International Edition, 2nd edn. Addison-Wesley, Boston (2003)
20. ISO/IEC: ISO/IEC 13568:2002 Information Technology - Z formal specification notation - Syntax, type system and semantics. Technical report (2002). http://standards.iso.org/ittf/PubliclyAvailableStandards/c021573_ISO_IEC_13568_2002(E).zip
21. Jackson, E.K., Levendovszky, T., Balasubramanian, D.: Reasoning about metamodeling with formal specifications and automatic proofs. In: Whittle, J., Clark, T., Kühne, T. (eds.) MODELS 2011. LNCS, vol. 6981, pp. 653–667. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24485-8_48
22. Leuschel, M., Butler, M.: ProB: a model checker for B. In: Araki, K., Gnesi, S., Mandrioli, D. (eds.) FME 2003. LNCS, vol. 2805, pp. 855–874. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45236-2_46
23. Morgan, C.: Programming from Specifications. Prentice Hall International Series in Computer Science, vol. 16, 2nd edn. Prentice Hall, Upper Saddle River (1994). https://dl.acm.org/citation.cfm?id=184737
24. Mota, A., Farias, A., Didier, A., Woodcock, J.: Rapid prototyping of a semantically well founded *Circus* model checker. In: Giannakopoulou, D., Salaün, G. (eds.) SEFM 2014. LNCS, vol. 8702, pp. 235–249. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10431-7_17
25. Nogueira, S., Sampaio, A., Mota, A.: Test generation from state based use case models. Form. Asp. Comput. **26**(3), 441–490 (2014)
26. Oliveira, D., Oliveira, M.V.M.: Joker: an animation framework for formal specications. In: SBMF 2011 - Short Papers, pp. 43–48. ICMC/USP, September 2011
27. Oliveira, M.V.M.: formal derivation of state-rich reactive programs using Circus. Ph.D. thesis, University of York, UK (2005). http://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.428459
28. Oliveira, M.V.M., Cavalcanti, A., Woodcock, J.C.P.: Unifying theories in ProofPower-Z. Form. Asp. Comput. **25**, 133–158 (2013). https://doi.org/10.1007/s00165-007-0044-5
29. Oliveira, M.V.M., Sampaio, A., Antonino, P., Ramos, R., Cavalcanti, A., Woodcock, J.C.P.: Compositional analysis and design of CML models. Technical report D24.1, COMPASS Deliverable (2013). http://www.compass-research.eu/Project/Deliverables/D241.pdf

30. Oliveira, M.V.M., Sampaio, A.C.A., Conserva Filho, M.S.: Model-checking *Circus* state-rich specifications. In: Albert, E., Sekerinski, E. (eds.) IFM 2014. LNCS, vol. 8739, pp. 39–54. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10181-1_3
31. Plagge, D., Leuschel, M.: Validating Z specifications using the ProB animator and model checker. In: Davies, J., Gibbons, J. (eds.) IFM 2007. LNCS, vol. 4591, pp. 480–500. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73210-5_25
32. Roscoe, A.W.: The Theory and Practice of Concurrency. Prentice Hall PTR, Upper Saddle River (1973)
33. Roscoe, A.W., Gardiner, P.H.B., Goldsmith, M.H., Hulance, J.R., Jackson, D.M., Scattergood, J.B.: Hierarchical compression for model-checking CSP or how to check 10^{20} dining philosophers for deadlock. In: Brinksma, E., Cleaveland, W.R., Larsen, K.G., Margaria, T., Steffen, B. (eds.) TACAS 1995. LNCS, vol. 1019, pp. 133–152. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-60630-0_7
34. Saaltink, M., Meisels, I., Saaltink, M.: The Z/EVES reference manual (for version 1.5). Reference manual, ORA Canada, pp. 72–85 (1997). http://dl.acm.org/citation.cfm?id=647282.722913
35. Scattergood, B.: The semantics and implementation of machine-readable CSP, pp. 1–179 (1998). http://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.299037
36. Schneider, S.: Concurrent and Real-Time Systems. Wiley, Chichester (2000)
37. Woodcock, J.C.P., Bryans, J., Canham, S., Foster, S.: The COMPASS modelling language: timed semantics in UTP, pp. 1–32 (2014)
38. Woodcock, J.C.P., Cavalcanti, A.: The semantics of Circus. In: ZB 2002: formal specification and development in Z and B. In: 2nd International Conference of B and Z Users Grenoble (2002)
39. Woodcock, J., Cavalcanti, A., Freitas, L.: Operational semantics for model checking Circus. In: Fitzgerald, J., Hayes, I.J., Tarlecki, A. (eds.) FM 2005. LNCS, vol. 3582, pp. 237–252. Springer, Heidelberg (2005). https://doi.org/10.1007/11526841_17
40. Woodcock, J.C.P., Davies, J.: Using Z, Specification, Refinement, and Proof. Prentice Hall International Series in Computer Science. Prentice-Hall Inc., Upper Saddle River (1996)
41. Ye, K.: Model checking of state-rich formalisms. Ph.D. thesis, University of York (2016)
42. Ye, K., Woodcock, J.C.P.: Model checking of state-rich formalism Circus by linking to CSP——B. Int. J. Softw. Tools Technol. Transf. **19**(1), 73–96 (2017). https://doi.org/10.1007/s10009-015-0402-1