Efficient and Mechanised Analysis of Infinite CSP_Z Specifications: strategy and tool support

Adalberto Farias, Alexandre Mota, Augusto Sampaio

Centro de Informática – Universidade Federal de Pernambuco (UFPE)
Caixa Postal 7.851 – 50.740-540 – Recife – PE – Brazil
{acf,acm,acas}@cin.ufpe.br

Abstract. *Model checking is an automatic technique for verifying properties of finite state systems. To deal with infinite state systems, which exhibit the state explosion problem, compression techniques are used before model checking. In this work we present a mechanised way of analysing infinite state CSP_Z specifications based on data abstraction. We have also developed a tool which mechanises the abstraction process.*

Resumo. *Verificação de modelos é uma técnica automática para verificar propriedades de sistemas com finitos estados. Para tratar sistemas com infinitos estados, que exibem o problema da explosão de estados, técnicas de compressão são usadas antes da verificação de modelos. Neste trabalho apresentamos uma maneira de analisar especificações CSP_Z com infinitos estados baseada em abstração de dados. Implementamos também uma ferramenta de suporte que mecaniza o processo de abstração.*

1. Introduction

Model checking [Clarke et al. 1999] is a powerful and automatic technique designed for verifying properties of finite state space systems. For infinite state space systems, compression techniques such as partial reduction [Roscoe 1998], data independence [Lazic 1999], abstract interpretation [Cousot and Cousot 1992], model checking and theorem proving integration [Kesten et al. 1999], etc. have to be first employed.

This work deals with a mechanised approach for model checking infinite state space CSP_Z processes. CSP_Z [Fischer 2000] is a formal combination of the process algebra CSP [Roscoe 1998] and the model-based language Z [Spivey 1992]. The strategy consists of replacing infinite data types with finite ones, while still preserving the original properties. Moreover, the approach combines theorem proving and model checking, and also has tool support.

The *ad hoc* approach for model checking CSP_Z proposed in by Mota and Sampaio [Mota and Sampaio 2001] consists of deriving an equivalent CSP_M (the machine-readable version of CSP) specification in order to reuse the FDR tool [Roscoe 1998] (a CSP model checker). State explosion has naturally emerged, especially due to the abstract data types allowed by Z. To avoid this problem, a mechanical data abstraction technique has been proposed [Mota et al. 2002].

Our work extends previous ones [Mota 2001, Mota et al. 2002] in two directions: theory and practice. Concerning the theoretical contribution, our approach

improves the previous one by considering both parts (CSP and Z) of a CSP_Z process; the previous technique considers only the data (Z) one. The result is a faster algorithm which captures specific situations of the behavioural (CSP) part. More importantly, our algorithm deals with a wider class of specifications, because it analyses only the traces allowed by the CSP part, whereas the original algorithm analyses all possible traces. This can lead to divergence in situations where our algorithm terminates successfully.

Regarding the practical contribution, we have implemented a Java tool which mechanises the proposed approach [Farias et al. 2003, Farias et al. 2004]; the tool has evolved from a previous one [Farias et al. 2001] which only considers the conversion from CSP_Z to CSP_M. The data abstraction feature was introduced to increase the support for analysing infinite CSP_Z specifications as well.

This presentation is organised as follows. Section 2 gives an overview of our approach. Section 3 shows the tool support and Section 4 presents our conclusions.

2. CSP_Z Data Abstraction

A CSP_Z specification has two distinct parts: CSP and Z (see Example 2.1). The first part describes the behaviour and the second one deals with data aspects (data structures, state, initialisation and operations). When converting CSP_Z to CSP_M, the CSP part is rewritten as a controller process (P_{CSP}) and the Z one is translated into a slave process (P_Z^S), which offers all enabled operations and synchronises with P_{CSP} into a parallel composition. These parts are analysed by our abstraction approach simultaneously. However, to simplify the treatment, we consider that the CSP part does not handle data, only events. Therefore, its behaviour does not depend on the data aspects and we say that it is data independent [Lazic 1999]. For the Z part, which contains all data manipulation, we use the abstract interpretation theory [Cousot and Cousot 1992], which allows one to describe concrete structures (data types and operations) by using abstract (simpler) ones. Thus, although we investigate the CSP and the Z parts, the abstraction comes from the Z part only.

To show our data abstraction strategy we use an example of a clock, which performs two events (*tick* and *tack*) forever, while increments an internal counter (see Example 2.1). For each channel *ev*, there exists an operation *com_ev*. This is required to synchronise both parts (CSP and Z); for example, the event *tick* is performed by the CSP part if and only if its corresponding schema *com_tick* is enabled. Figure 1(a) shows an LTS (Labelled Transition System) representation of the *Clock* process.

Example 2.1 (An Infinite Clock)

spec *Clock*
- chan *tick, tack* — *(CSP part – channel declaration)*
- main = $tick \rightarrow tack \rightarrow$ main — *(CSP part – main process)*
- $State \mathrel{\widehat=} [n : \mathbb{N}]$ — (Z part – state definition)
- $Init \mathrel{\widehat=} [State' \mid n' = 0]$ — *(Z part – initialisation)*
- $com\_tick \mathrel{\widehat=} [\Delta State \mid n \bmod 2 = 0 \land n' = n + 1]$ — *(Z part – operation)*
- $com\_tack \mathrel{\widehat=} [\Delta State \mid n \bmod 2 = 1 \land n' = n + 1]$ — *(Z part – operation)*

end_spec *Clock*

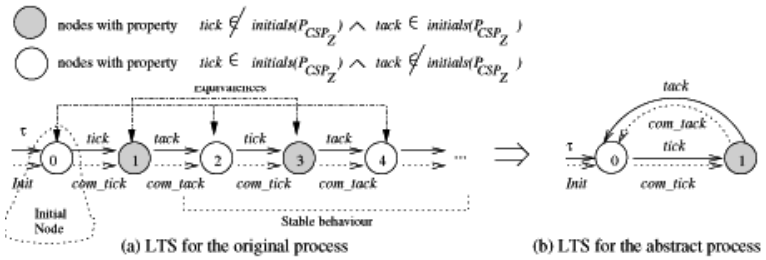

Figure 1: Abstraction of the *Clock* process

After performing ⟨*tick,tack*⟩ the process accepts performing the same trace again. The acceptances set of a CSP_Z process (the set of all events accepted by the CSP part, whose corresponding schema is enabled) is formalised by Definition 2.1, whereas Definition 2.2 establishes the property of a CSP_Z process as a conjunction of its acceptances and refusals (set of events a process refuses for each of its traces). The synchronisation interface is the set of events on which both parts synchronise.

Definition 2.1 (Initials of a CSP_Z Process) *Let P_{CSP_Z} be a CSP_Z process whose CSP part is captured by P_{CSP} and its synchronisation interface is represented by I. Then,*

$$initials(P_{CSP_Z}) = \cup_{ev \in I} \{ev \mid \text{pre } com\_ev \wedge ev \in initials(P_{CSP})\}$$

Definition 2.2 (Property of a CSP_Z Process) *Let P_{CSP_Z} be a CSP_Z process defined with channels ev1, … ev_n. If, in a specific context, the events ev1, … ev_j, with $j \leq n$, are accepted by the P_{CSP_Z}, then its property, at that context, is*

$$\{\wedge_{i=1}^{j} ev_i \in initials(P_{CSP_Z})\} \wedge \{\wedge_{k=j+1}^{n} ev_k \notin initials(P_{CSP_Z})\}$$

The key idea of our approach is the notion of stability–the infinite repetition of a property. When achieving a point whose property has already occurred (node $n = 2$ on Figure 1(a)), one has to investigate the infiniteness of such a repetition. This means checking whether the whole process accepts performing ⟨*tick tack*⟩ forever. For the CSP part, we isolate the trace causing the repetition and build an auxiliary process that reaches such a point and is infinite ($P_{aux} = tick \rightarrow tack \rightarrow P_{aux}$). Then we force *main* to behave like P_{aux} and build a refinement check to determine the stability of *main*:

$$main \underset{\alpha(P_{aux})}{\parallel} P_{aux} \quad \sqsubseteq_{\mathcal{T}} \quad P_{aux}$$

Above, $\alpha(P_{aux})$ denotes the set of all events performed by P_{aux}. For this refinement checking, the *traces* model is sufficient. Nevertheless, the abstracted process is equivalent in the *failures-divergences* model [Roscoe 1998] to the original process (at least when dealing with optimal abstraction [Mota 2001]).

For the Z part we build a schema composition corresponding to ⟨*tick tack*⟩ (for our example, *comp* = *com_tick* ⨾ *com_tack*) and check whether such a composition is allowed to happen infinitely (using a theorem prover like Z-Eves [Saaltink 1997]); this happen when pre *comp* is valid and the execution of *comp* enables it again ((pre *comp*)' is valid). This is formalised by the theorem

$$\forall State; State' \mid (pre\ comp \Rightarrow comp) \bullet (pre\ comp)'.$$

Once both parts are stable, the infinite repetition of ⟨*tick tack*⟩ can be replaced with a cycle (see Figure 1(b)) and an equivalent (abstract) process can be built, such that the original domain (ℕ) is replaced with the finite subset {0,1} where 0 represents the equivalence class of even numbers and 1 that of odd numbers.

spec $Clock^A$
- chan *tick, tack* — *(CSP part – channel declaration)*
- main = *tick* → *tack* → main — *(CSP part – main process)*
- *State* ≙ [*n* : {0,1}] — (Z part – state definition)
- *Init* ≙ [*State'* | *n'* = 0] — *(Z part – initialisation)*
- *com_tick* ≙ [Δ*State* | *n* mod 2 = 0 ∧ *n'* = 1] — *(Z part – operation)*
- *com_tack* ≙ [Δ*State* | *n* mod 2 = 1 ∧ *n'* = 0] — *(Z part – operation)*

end_spec $Clock^A$

Finally, we have showed that the whole CSP_Z process causes less expansions than its Z part (see Theorem 2.1), by proving that the traces performed by the whole process is a subset of those performed by its Z part (see [Farias 2003] for details about the proof).

Theorem 2.1 ($P_Z^S \underset{I}{\|} P_{CSP}$ **refines** P_Z^S) *Let* P_{CSP} *and* P_Z^S *be CSP processes representing the CSP and the Z parts of a* CSP_Z *process, respectively. Then,*

$$P_Z^S \quad \sqsubseteq_{\mathcal{T}} \quad P_Z^S \underset{I}{\|} P_{CSP}$$

Note that the above theorem also holds when the CSP part diverges, stops or terminates. This happens because the parallel composition captures those situations. In [Farias 2003] we have presented several examples to illustrate the importance of Theorem 2.1 in capturing the influence of the behavioural part. Furthermore, we have presented an algorithm, in an imperative style, which implements our strategy.

3. Tool Support

As our approach can be mechanised, we have also implemented a tool (available at *http://www.cin.ufpe.br/~acf*) which applies all steps to derive an abstract CSP_Z specification from a concrete one. Furthermore, as our approach involves theorem proving, the tool is indeed semi-automatic; user assistance can be required. The current version interacts with Z-Eves [Saaltink 1997] by using a file-based strategy. However, the user can provide plugins to interact with other tools.

4. Conclusions

This work is related to CSP_Z data abstraction, a technique which permits to deal with verification of infinite state space specifications. The strategy consists of deriving specifications with a finite state space and with the same properties as the original ones.

We have extended the work reported in [Mota 2001], where a guided approach for CSP_Z data abstraction was proposed. However, Mota´s work considers only the data (Z) part when looking for stable behaviour; it explores only the expansions caused by the Z part. Therefore, the first contribution is an extended strategy which also considers the behavioural (CSP) part. The immediate result concerns the class of specifications

we can deal with. Situations specific of the behavioural part (such as deadlock, termination and divergence) are captured by our approach. Because the whole process is analysed, our algorithm can produce abstractions of specifications with a non-stable Z part (the CSP part can be stable). Furthermore, our algorithm is faster than that proposed in [Mota 2001]. This is a result of Theorem 2.1: exploring the whole CSP_Z process generates less expansion (traces) than considering only its Z part. The limitation of our approach concerns specifications whose both parts never stabilises.

Regarding tool support our contribution consists of providing a flexible and robust tool which implements the technique of data abstraction for CSP_Z. The tool is a pioneer work towards mechanisation of data abstraction.

References

Clarke, E., Grumberg, O. and Peled, D. (1999). Model Checking. The MIT Press.

Cousot, P. and Cousot, R. (1992). "Abstract interpretation frameworks". J. Logic. and Comp., 2(4): 511–547.

Farias, A. (2003). Efficient and Mechanised Analysis of Infinite CSP_Z Processes: strategy and tool support. Master´s dissertation, Federal University of Pernambuco.

Farias, A., Mota, A. and Sampaio, A. (2001). "De CSP_Z para CSP_M: Uma ferramenta transformacional Java". Workshop on Formal Methods (WMF´01), p. 1-10.

Farias, A., Mota, A. and Sampaio, A. (2003). "A Support Tool for CSP_Z Data Abstraction". Tool Exhibition Notes of FME 2003, Pisa, Italy, 2003, p. 11-15.

Farias, A., Mota, A. and Sampaio, A. (2004). "Efficient CSP_Z Data Abstraction". IFM 2004, volume 2999 of LNCS, p. 108-127.

Fischer, C. (2000). Combination and Implementation of Processes and Data: from CSP_{OZ} to Java. PhD thesis, Fachbereich Informatik Universität Oldenburg.

Kesten, Y., Klein, A., Pnueli, A. and Raanan, G. (1999). "A Perfecto Verification: Combining Model Checking with Deductive Analysis to Verify Real-Life Software", volume 1708 of LNCS, p. 173-194.

Lazic, R. (1999). A semantic study of data-independence with applications to the mechanical verification of concurrent systems. PhD thesis, Oxford University.

Mota, A. (2001). Model Checking CSP_Z: Techniques to Overcome State Explosion. PhD thesis, Federal University of Pernambuco.

Mota, A. and Sampaio, A. (2001). "Model-Checking CSP-Z: Strategy, Tool Support and Industrial Application". Science of Computer Programming, 40:59-96.

Mota, A., Borba, P. and Sampaio, A. (2002). "Mechanical Abstraction of CSP_Z Processes". FME 2002, volume 2391 of LNCS, p. 163–183.

Roscoe, A. (1998). The Theory and Practice of Concurrency. Prentice Hall.

Saaltink, M. (1997). "The Z-Eves System". ZUM'97: The Z Formal Specification Notation, volume 1212 of LNCS, Springer.

Spivey, M. (1992). The Z Notation: A Reference Manual. Prentice-Hall International.