

CONFRES: Interactive Coding Conflict Resolver based on Core Visualisation

A. Madalinski

University of Newcastle upon Tyne
A.A.Madalinski@ncl.ac.uk

Abstract

The tool supports manual resolution of coding conflicts in asynchronous circuit specification given as Signal Transition Graphs (STGs) and displays them as partial orders (finite and complete prefixes of STG unfoldings). The manual approach although efficient requires a significant effort from the designer. The tool CONFRES assists the designer by visualising the conflict cores, their superposition and the constraints on signal insertion.

1. Introduction

Signal Transition Graphs (STGs) are widely used for specifying the behaviour of asynchronous control circuits. STGs are interpreted Petri nets in which transitions are labelled with the rising and falling edges of circuit signals. There exist a number of methods (reviewed in [1]) for the synthesis of circuits from STG specifications.

Complete State Coding (CSC) is an STG property required for the implementation of next-state functions as circuits. A CSC conflict arises when semantically different states of an STG have the same binary encoding. To resolve it, new signals, helping to distinguish between these states, must be inserted into the STG. The behaviour of the new STG should remain externally equivalent to the original one.

A common approach to detecting and solving CSC conflicts is to construct the reachable state space of the initial STG. While it can be used in completely automatic synthesis, it has several flaws: state graphs are not visual, and this prevents efficient interaction with the user. Moreover, the combinatorial explosion of the state space is a serious issue for highly concurrent STGs. This makes alternative techniques, and in particular those based on Petri net unfoldings, very attractive for this task. In [2] the unfolding technique was applied to check the CSC condition. Since STGs usually exhibit high concurrency, but have few choice points, their unfolding prefixes are often exponentially smaller than the corresponding state graphs; in fact, in most of the experiments conducted in [2] they are just slightly bigger than the original STGs themselves. Therefore, unfolding prefixes are well-suited for both visualising an STG's behaviour and

alleviating the state space explosion problem.

Enforcing the CSC by completely automatic synthesis, which uses heuristics, may produce sub-optimal circuits or fail to solve the problem. Therefore, manual intervention is crucial for finding good synthesis solutions, e.g. when a system's performance is of importance. As a result designers would like to manipulate the model interactively and choose where to insert a new signal. This would help the designer to understand the characteristic patterns of a circuit's behaviour and the cause of each coding conflict, thus facilitating decisions depending on design constraints.

CONFRES facilitates a manual refinement of an STG with CSC. It works on the level of unfolding prefixes where coding conflicts are visualised by cores [3], which are the essential causes of conflicts. All such cores must be eliminated by newly added signals. This eventually results in an STG satisfying the CSC property.

2. Visualisation and resolution of conflicts

In [2] an integer programming technique has been proposed for detecting coding conflicts employing STG unfolding prefixes. A CSC conflict can be represented as an unordered conflict pair of configurations $\langle C_1, C_2 \rangle$ whose final states are in CSC conflict, as shown in Fig. 1(a). [2] builds a system of constraints whose set of solutions comprises such conflict pairs. The *complementary set* CS is the symmetric set difference of C_1 and C_2 , e.g. in Fig. 1(a) $CS_1 = \{ready+, start-, ready-, start+\}$. The binary encoding of the states before and after the firing of the set are the same. This is because the number of a^+ and a^- labelled events in CS is the same for all signals. The states, however, are semantically different, which results in a state coding conflict. A complementary set is a *core* if it cannot be represented as the union of several disjoint complementary sets.

Cores are important for resolving coding conflicts. By introducing an additional internal signal, say $csc+$, one can split a core thus eliminating the corresponding coding conflicts. To preserve the consistency of the STG, the signal's counterpart $csc-$ must also be added to the specification outside the core, in such a way that it is neither concurrent to nor in structural conflict with $csc+$. In addition, inserted signals cannot trigger an input signal.

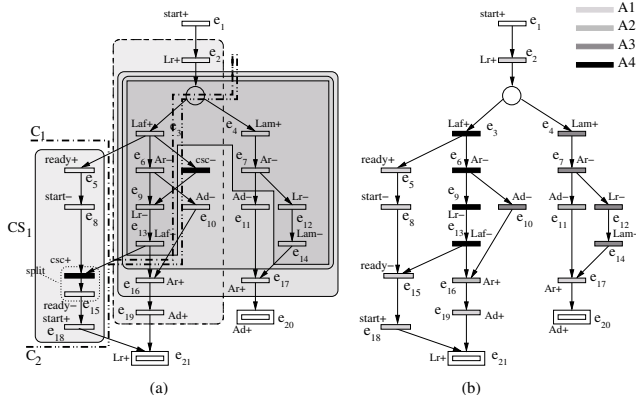


Figure 1. Resolution process: core visualisation (a) with the newly inserted signal highlighted and the height map (b). Inputs: *starts, Lam, Laf, Ad*; outputs: *ready, Lr, Ar*; internal: *csc*.

It is often the case that cores overlap. In order to minimise the number of inserted signals, and thus the area and latency of the circuit, it is advantageous to insert a signal in such a way that as many cores as possible are eliminated by it. That is, a signal should be inserted into the intersection of several cores. As an example, consider the cores shown in Fig. 1(a). There are five cores altogether, but by exploiting the fact that four of them overlap, it is possible to eliminate them all by adding just one new signal: it should be inserted into the intersection of the four cores, and its counterpart — into the remaining core.

A key feature in the visualisation process is the *height map*, showing the quantitative distribution of the cores. The events located in conflict cores are highlighted by shades of colours. The shade depends on the "altitude" of an event, i.e., on the number of cores it belongs to. (It is similar to a physical map in geography.) Consider the height map in Fig. 1(b). The events e_3, e_6, e_9 and e_{13} are labelled with the highest altitude A4. "Peaks" with the highest altitude are good candidates for insertion of a new signal, since they correspond to the intersection of maximum number of cores.

From this representation, the designer can select an area for inserting a new signal and obtain a local, more detailed description of the cores overlapping with the selection. When an appropriate core cluster is found, the designer can decide how to insert a new signal transition optimally, taking into account the design constraints and their knowledge of the system being developed.

An overview of this process supported by the tool is shown in Fig. 2. Given an STG, a finite complete prefix of its unfolding is constructed, and the cores are computed. If there are none, the process stops. Otherwise, locations for the insertion of a transition and its counterpart are determined in phases 1 and 2, respectively. The inserted signal is then transferred to the STG, and the process is repeated. Depending on the number of CSC conflicts, the resolving process can consist of several cycles.

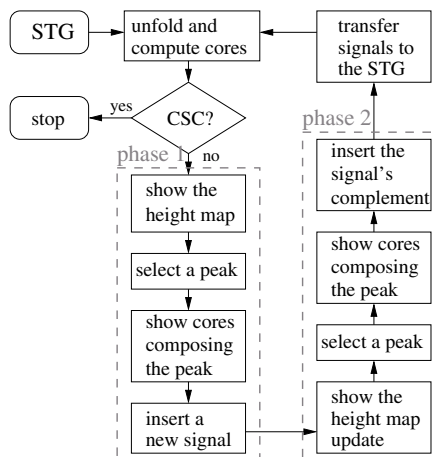


Figure 2. The process of resolving conflicts

3. Tool description

CONFRES is an interactive state coding conflict resolver, which is based on core visualisation and employs finite and complete prefixes of STG unfoldings. It takes an STG in the '.g' format supported by PETRIFY [1], an STG-based synthesis tool. It uses PUNF [4], a Petri net unfold, to produce a finite and complete prefix of the STG, and CLP [4], a linear programming model checker, to detect coding conflicts in the STG. After the detection of conflicts, cores are computed and the resolution process described in Fig. 2 is applied. The tool guides the designer through the stages in phase 1 and 2. During this process the cores and the corresponding height map are visualised using DOT [5], a graph drawing software by AT&T, and the designer can interactively insert new signals to obtain a customised solution. After the resolution process is completed a synthesis tool, e.g. PETRIFY, can be used to synthesise the circuit.

References

- [1] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, *Synthesis of Asynchronous Controllers and Interfaces*, Springer Verlag, 2002.
- [2] V. Khomenko, M. Koutny, and A. Yakovlev, "Detecting State Coding Conflicts in STGs Using Integer Programming", *Proc. of DATE'02*, IEEE Comp. Soc. Press, 2002, 338-345.
- [3] A. Madalinski, V. Khomenko, A. Bystrov, and A. Yakovlev, "Visualisation and Resolution of Coding Conflicts in Asynchronous Circuit Design", *Proc. of DATE'03*, IEEE Comp. Soc. Press, 2003, 926-931.
- [4] V. Khomenko, "Model Checking Based on Petri Net Unfolding Prefixes", PhD Thesis, Department of Computing Science, University of Newcastle, 2002.
- [5] E. Koutsofios, and S. North, "Dot User's Manual", AT&T Labs-Research, 2002.