

Testing Petri Nets for Mobile Robots Using Gröbner Bases

Angie Chandler¹, Anne Heyworth², Lynne Blair¹, Derek Seward³.

¹ Lancaster University, Department of Computing

² University of Wales, Bangor, Department of Mathematics

³ Lancaster University, Department of Engineering

Abstract

As autonomous mobile robots grow increasingly complex, the need for a method of modeling and testing their control systems becomes greater. This paper discusses the use of Petri nets as a means of modeling and testing the control of a mobile robot, concentrating specifically on the reachability testing of the Petri net model through the use of Gröbner bases.

The designing and testing of the Petri net models for the mobile robot is done initially in component form, providing a model which is then automatically converted into a Gröbner basis to provide a simple means of reachability testing. Once the testing process is complete, the Petri net modules, which represent each of the components of the mobile robot are connected to form a single Petri net. This Petri net is then used for the generation of control code for the robot.

In this paper, the process of testing the modules created to represent the components of the autonomous mobile robot is shown through a case study, Star Track (a tracked autonomous mobile robot). Details of both ordinary and coloured Petri nets representing certain components of Star Track are discussed, with both the Petri net model and the equivalent Gröbner basis described.

1 Introduction

The dynamic and asynchronous structure of the Petri net is ideally suited to the modeling of an autonomous mobile robot, provided the model can be thoroughly tested prior to code generation or execution on board the robot. To this end, the reachability test, as one of the most basic means for checking the accuracy of the model compared to its expected execution, provides a great deal of reassurance to the designer of the software, which in turn allows the designer to create more complex systems reliably.

The need for mathematical analysis of the Petri net models created for the mobile robot, provided an ideal opportunity for collaboration between mathematics and engineering departments. As a result of this co-operation, an approach to Petri net analysis formed, based on the relationship between Petri nets and Gröbner bases. To our knowledge, the application of Gröbner bases has not been previously attempted in the field of engineering, but has successfully been used in fields such as operational research and statistics.

The subject of this paper is the application of the Gröbner basis to the testing of reachability in a Petri net model, specifically to a Petri net model of a mobile robot. This application is implemented as an automatic testing facility within a Petri net toolkit, TRAMP (Toolkit for Rapid Autonomous Mobile robot Prototyping) intended to model mobile robots and other mechatronic systems from the stages of conceptual design to a final executable program. These Petri net models are initially formed as individual modules, each related to a component of the system, in order to allow easier testing and analysis prior to creation of the final, global, Petri net model [Chandler 99a].

In section 2 of this paper, some previous Petri net applications will be discussed, providing a background to the choice of Petri nets as a model for the autonomous mobile robot. Section 3 will detail the Petri net toolkit, TRAMP, before the Gröbner bases used as a testing method are studied in

further detail (section 4). A case study showing the use of the method for the mobile robot, Star Track, will be discussed in section 5, followed by conclusions and future work.

2 Choice of Petri Nets

Petri nets are generic enough to provide the capacity for application to a wide variety of applications, although due to their asynchronous nature they are more commonly used for distributed systems [Buchholz 92] and other similar processes. However, their uses in distributed systems by no means exclude applications to the field of robotics. In fact, robots can themselves form part of a distributed system, as can be seen through the example of an orange-picking robot with point-to-point communications [Cavalieri 97]. Other areas of robotics can also find use for Petri net modelling as a method of eliminating deadlock and other temporal inconsistencies [Simon 98], although these properties require testing through timed Petri nets, an extension which has yet to be made to the analysis system used here. Alternatively, Petri nets can be used to allow co-operation between multiple robots [Suh 96], or between a human and a robot [Mascaro 98]. The range of applications for Petri nets is enormously diverse, and limited only by the range of tools available to implement these possibilities.

Our use of Petri nets as a modeling tool in the field of mobile robotics, was initially inspired by their ability to represent both the data flowing in the system and the state of the system, simultaneously. This initial interest was then furthered by the ease with which the model could be translated into executable code, as required by the TRAMP toolkit discussed in the section 3, below, without the need to alter any of the components modeled. These factors, combined with the mathematical background which supported the testing of any models used, and examples of previous applications to the field led to the eventual use of Petri nets within the TRAMP toolkit.

3 TRAMP

The TRAMP toolkit provides a simple means of modeling, testing and generating code for a mobile robot. This is initially done in the form of modules, or objects based on the separate components of the mobile robot, and divided into five categories in an overall object diagram in order to allow the toolkit user to connect the objects as desired. These five categories, sensors, filters, navigation, low level control, and actuators are also used to provide certain attributes to each object which may only be relevant to that category.

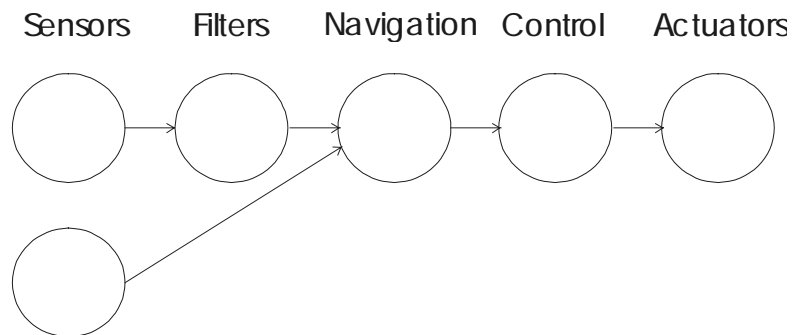


Figure 1 Object Diagram Layout

As the arcs in Figure 1 suggest, the flow of information in the system leads from sensors to actuators via various processing alternatives. Once data has been read in from a sensor, such as a compass, the data may then be filtered to remove any noise from the readings, before the navigation uses the data to make a decision on the next move of the robot. With the commands to be issued to the actuators decided, the navigation module will then pass the information either directly to the actuator (for example a motor) or via a low level controller, which will translate the information into a form readable by the actuator and ensure that it behaves exactly as it should.

Once the modules are defined and linked in the object diagram, as shown in Figure 1, the user may then access the Petri net modules of each of the separate components. These components remain

completely unconnected whilst they are tested, which may include testing on board the robot as a separate module, after which the modules may be linked according to a precise protocol. This is discussed below.

Linking

Once all testing of the Petri net is complete, the user may then link the individual components according to the connections defined in the object diagram, and a specific hierarchy. This hierarchy makes the navigation module the highest level element, and works outwards in the object diagram making the sensors and actuators the lowest. The navigation module (or modules) is designed so that whilst it can be tested in simulation as it stands, several of its transitions actually represent groups of transitions for use when the Petri net is finally connected. As the Petri nets are linked, the navigation module (Figure 2) fully expands its complex transitions.

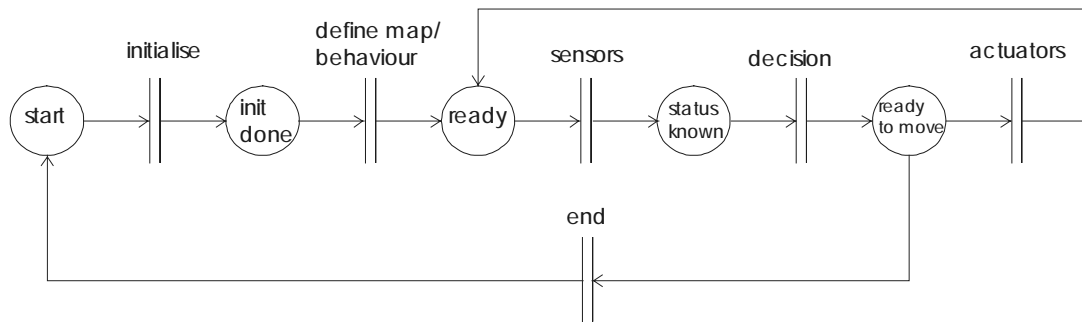


Figure 2 Navigation Module

The transitions “initialise”, “end”, “sensors” and “actuators” each expand into several transitions, providing connections to the other modules. The remaining “define” and “decision” modules represent the actual method of navigation required of the robot, and can easily be exchanged for a number of standard defaults, or left for the user to fully implement.

Initialisation and End Expansion

The “initialise” and “end” transitions connect directly to every other module in the system, ensuring that every initialise routine is called before the main program starts, and that the program shuts down correctly when it finishes. The expansion of these is shown below.

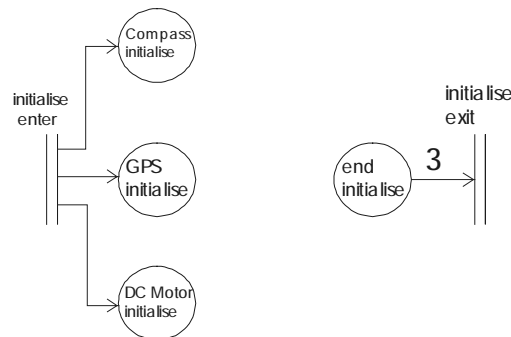


Figure 3 Initialise Transition Expansion

Figure 3 shows the expansion of the initialise transition to connect to two sensors (the compass and the GPS – Global Positioning System) and one actuator (the DC motor). Here, there were no filters or controllers in the system.

Sensor and Actuator Expansion

Similarly, the “sensors” and “actuators” transitions can be expanded. However, here the links created in the object diagram come into play, as the only connections made are those which are directly connected to the navigation module. For the expansion of a “sensors” transition, this includes any filters which are connected to the navigation module, but not any sensors connected only to the filter as they must be connected through a similar process in the filter module.

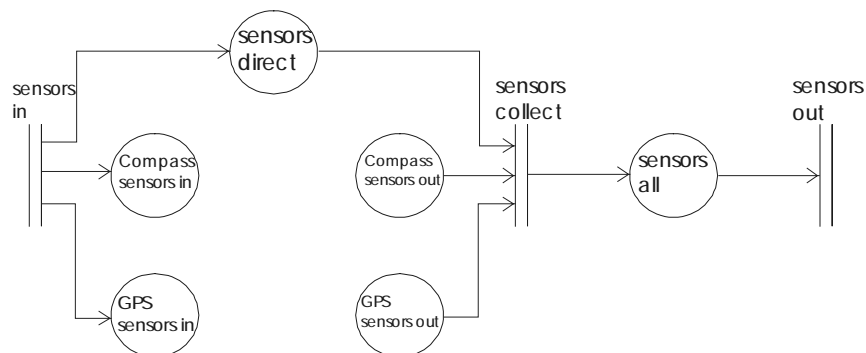


Figure 4 Sensor Transition Expansion

The configuration for sensor transition expansion (Figure 4) is slightly different to allow for the possibility that there may be no sensors or filters connected, but the transition must still operate. The places which link to the lower level modules behave as they do in the “initialise” transition expansion, connecting in place of the test driver initially provided with each module. There are also special standardised tokens which allow this operation to be performed more smoothly. These tokens contain all possible elements of any expected sensor readings or instructions to actuators respectively. These readings or instructions can then be easily converted to or from the tokens created for specific sensor and actuator modules. Here, there were two sensors directly connected to the navigation module and no filters.

Control from Navigation

The method of expansion described in the previous sub-sections is designed specifically to allow the navigation module to maintain control over the system as a whole. The bipartite nature of the Petri net gives the two types of nodes, places and transitions, specific meanings which must be taken into account within the model. Clearly, the transitions perform the actions, whereas the places merely maintain state, but there are further implications which can be put into use here. The nature of the places is such that they have authority over transitions. Transitions cannot fire without, in a sense, instructions from a place as each input place must contain a token in order to enable the transition. It is analogous to the handing out of instructions by a superior, and prior to the receipt of permission the task may not be performed.

Whenever a link is formed between two modules, the module which is further up the hierarchy contains the place which is linked, whilst the lower level module contains the transition. The lower level module knows only that an instruction has been received, whilst the higher level module continues to be aware of its state, despite the departure of the active tokens into a separate module.

This precise method of linking also ensures that the reachability test results performed whilst the modules were still separated will still be accurate once the modules are reconnected, as the individual modules remain essentially separate whilst the pre-tested navigation module and connectors form links to them.

4 Testing through Gröbner Bases

The Gröbner basis forms a basis of polynomials in a manner equivalent to that formed by the vector space for vectors. As every vector in a vector basis must be linearly independent from every other

vector in the basis, so must every polynomial in a Gröbner basis be independent of every other polynomial in the basis.

Definition

Take $K[x_1, \dots, x_n]$ to be a ring of polynomials in n variables over K , a field.

The Gröbner basis is the set of linearly independent polynomials in the polynomial ring K .

A more formal definition can be found in [Chandler 99].

The generation of the Gröbner basis of a set of polynomials, as is used for the Petri net analysis in later sections, is done with the use of Buchberger's algorithm [Buchberger 98]. This algorithm is, again, analogous to methods used to create vector spaces from the simultaneous equations they may represent, requiring the combination of each set of equations (or rows) until there are no more linearly independent combinations available, and will be demonstrated in section 5.

Once created, the Gröbner basis represents all equivalent reachable places, and may be used to find any reachable marking, provided the initial Petri net is reversible, or alternatively determine whether the Petri net is reversible based on results of reachability testing (see section 5).

5 Case Study – Star Track



Figure 5 Star Track

The Petri nets discussed here are based on real life models, created for use on board the mobile robot, Star Track (Figure 5), intended to perform navigation with the use of satellite GPS [Yavuz 99]. This robot's major components consisted of a compass, a GPS receiver, a PC 104 computer, and four DC motors. These four components formed the main objects within the object diagram, two of which are considered in the following examples. It should be noted that the Petri nets shown represent the software interface to the hardware components named, not the hardware components themselves, as the intention of TRAMP is the generation of control software for use with specific hardware.

Motors

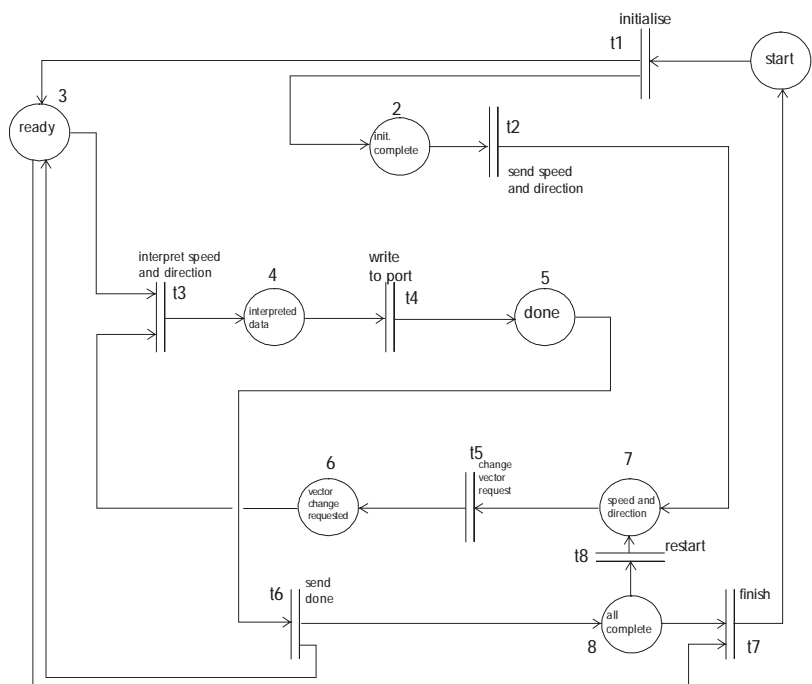


Figure 6 Motors Petri net

As can be seen in the Petri net shown in Figure 6, once the motors have been initialised (t_1) the user may input the required speed and direction (t_2) for each motor. The speed and direction information is then interpreted (t_3) and written to the relevant port (t_4), provided the system is “ready” (place 3) which, combined with the user input token, will enable transition t_3 .

The Gröbner basis for this Petri net is generated from the polynomials of the transitions listed below, where x represents a token in a given place.

For example, a token at place 1 allows the firing of transition t_1 and results in a token in places 2 and 3. This can be represented by the polynomial:

$$\text{pol}(t_1) = x_1 - x_2x_3$$



Polynomials for the other transitions can be similarly generated:

$$\begin{aligned} \text{pol}(t_2) &= x_2 - x_7 \\ \text{pol}(t_3) &= x_3x_6 - x_4 \\ \text{pol}(t_4) &= x_4 - x_5 \\ \text{pol}(t_5) &= x_7 - x_6 \\ \text{pol}(t_6) &= x_5 - x_3x_8 \\ \text{pol}(t_7) &= x_3x_8 - x_1 \\ \text{pol}(t_8) &= x_8 - x_7 \end{aligned}$$

These polynomials are then used to form the Gröbner basis:

$$\{x_4 - x_1, x_5 - x_1, x_6 - x_2, x_7 - x_2, x_8 - x_2, x_2x_3 - x_1\}$$

Which can be calculated automatically, either through TRAMP or through a standard package such as Maple.

This gives a catalogue of markings (reachable places) from an initial marking x_1 (ie, starting with a token in the place “start”) to be:

$$\{x_1, x_4, x_5, x_2x_3, x_3, x_6, x_3x_7, x_3x_8\}$$

As the Grobner basis is only useable when the Petri net is reversible, an undesirable, or unexpected state within this list would indicate either that the Petri net was not reversible, or that there was an error in the Petri net itself, allowing the unexpected state. Once the possibility of either an undesirable reachable place (or alternatively a desirable place which wasn't reached) or a non-reversible Petri net is eliminated, the chance of a serious error occurring on board the mobile robot during execution is greatly reduced.

Should an undesirable state be reachable from the initial marking, it must first be decided whether the error is in the reversibility of the Petri net or in the reachability. This is best done by checking for errors in very simple, and obvious, reachability calculations. If the tester claims that a clearly unreachable state is reachable then it is likely that the Petri net is in fact not reversible, and that is where the error lies. Should the Petri net appear to be reversible, the user can seek further assistance by using the “step through” method, which gives a visual representation of the movement of the tokens through the Petri net, and allows the user to see the error as it occurs.

Compass

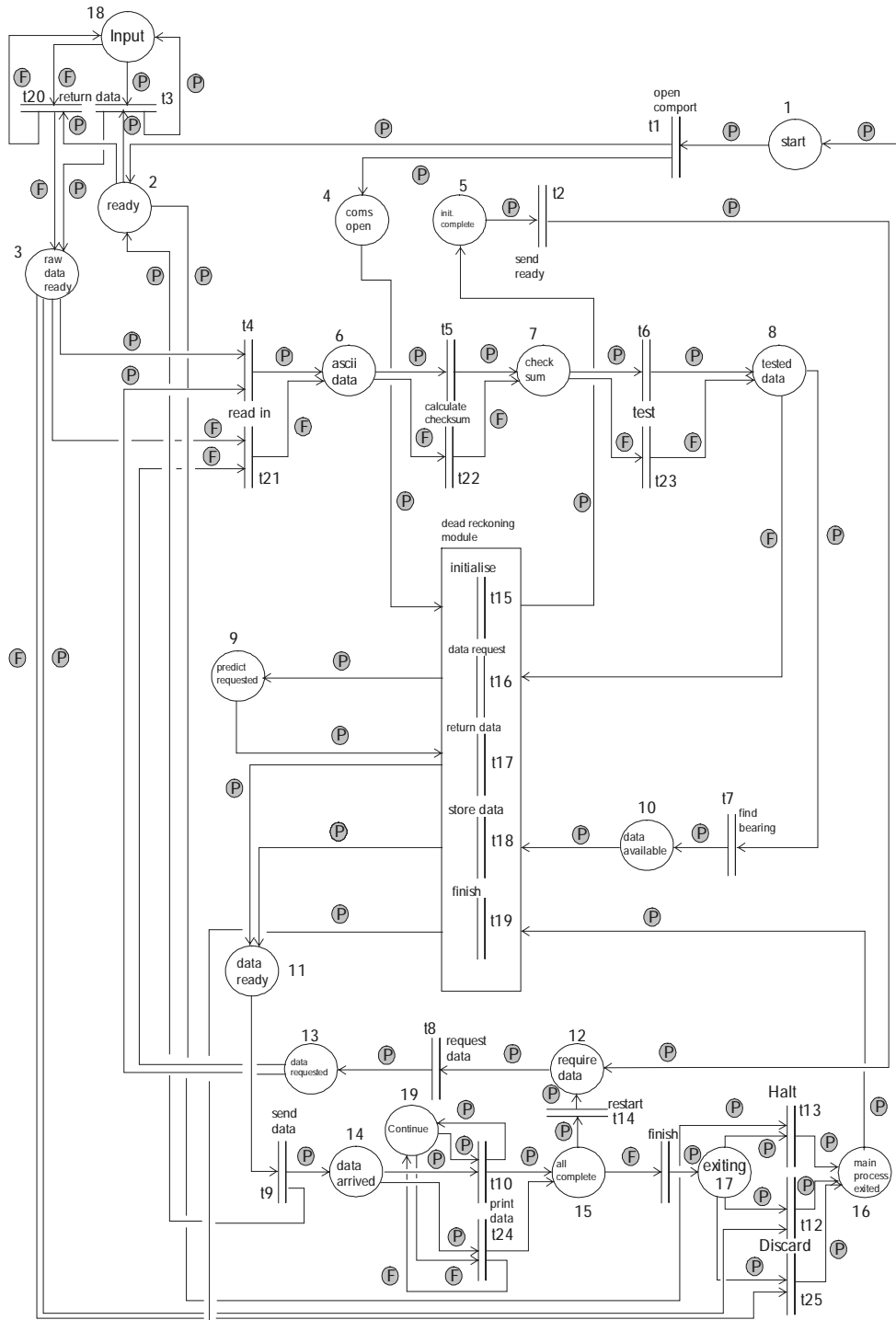


Figure 7 Compass Petri Net

Before analysing the Petri net shown in Figure 7, it is necessary to consider a further extension to the Petri net theory described in section 2. The compass Petri net is a *coloured Petri net*.

The extension of an ordinary Petri net to a coloured Petri net provides the ability to represent different types of token and treat them differently. In a coloured Petri net, the transition is only enabled by an incoming token if the token's colour matches the colours allowed by the transition. A coloured Petri net can always be converted to an ordinary Petri net through additional places and transitions for each different colour.

The compass Petri net contains a number of different token types in order to represent the information types required within the compass program, such as the ASCII characters read in from the compass itself, or the final format of the data when a bearing has been established. However, these additional token types do not affect the choices made in this Petri net, and are therefore irrelevant to its analysis. In the case of the compass Petri net shown in Figure 6, there are essentially only two colours required in the Petri net, "pass" and "fail", as these are the only two colours relevant to the testing of the Petri net in simulation. Any other variations in token type serve no purpose in simulation but to increase the complexity of the analysis.

For the purposes of the calculations to be performed using Gröbner bases, the "pass" tokens have been labelled x , and the "fail" tokens y . The initial marking of the Petri net is described by a single "pass" token in the "start" place (1), and "pass" and "fail" tokens in each of the places "input" (18) and "continue" (19). The additional tokens at places 18 and 19 allow the user to perform the more rigorous testing of the coloured Petri net.

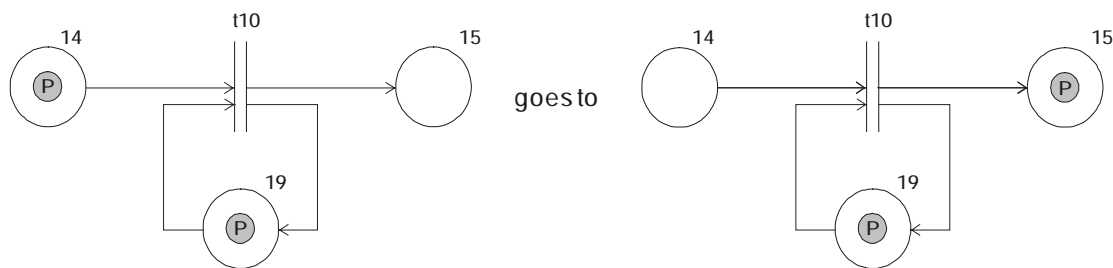
The "pass" and "fail" tokens are primarily used as a distinction between data received with a correct checksum and data received with a failed checksum. This colouring is used to ensure that only uncorrupted data is used to calculate the bearing which will later be output to the main navigation module of the mobile robot. Any failed data is instead sent to the "data request" transition, which can then provide a connection to the "dead reckoning plug-in module" not shown in this diagram. This can be seen through tracing the route of a "pass" and then a "fail" token through the transitions "read in" (t_4 or t_{21}), "calculate checksum" (t_5 or t_{22}) and "test" (t_6 or t_{23}) to the conclusion of the decision at the transitions "find bearing" (t_7) or "data request" (t_{16}).

As shown in the previous example, the first step in the analysis of the Petri net and generation of the Grobner basis is to establish the polynomial for each transition. These polynomials are listed below, with a pass token represented by an x , and a fail token represented by a y .

For example, the polynomial:

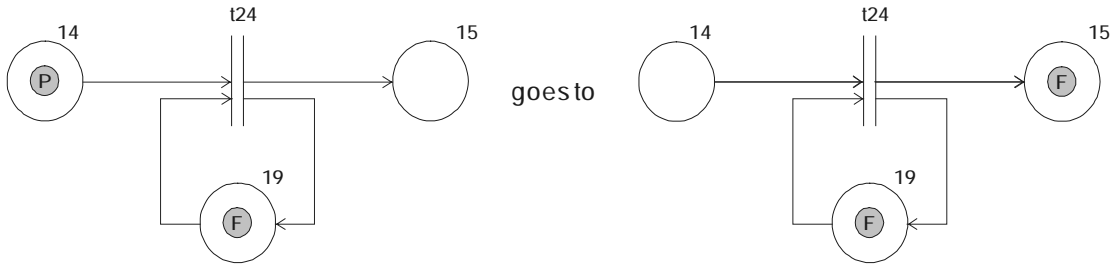
$$\text{pol}(t_{10}) = x_{14}x_{19} - x_{15}x_{19}$$

represents the following possible transition.



Note that $\text{pol}(t_{24})$ is identical to $\text{pol}(t_{10})$ apart from colouring.

$$\text{pol}(t_{24}) = x_{14}y_{19} - y_{15}y_{19}$$



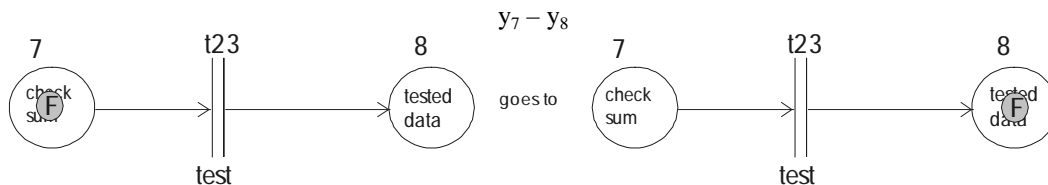
Polynomials can be generated for the other transitions as follows.

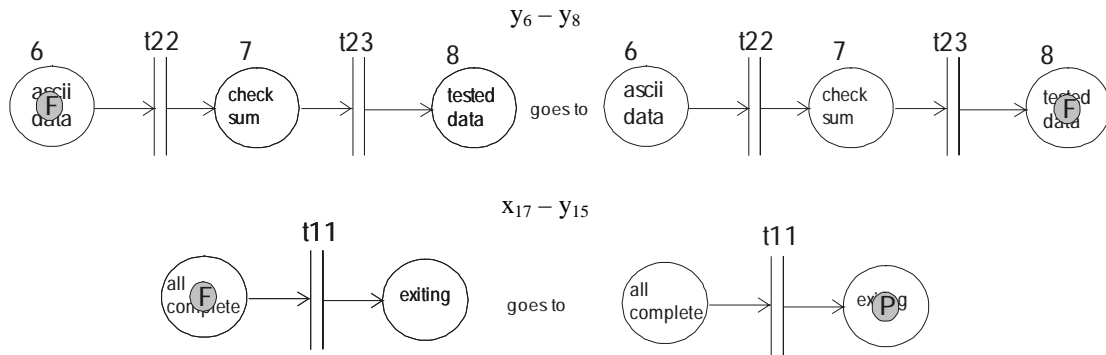
$\text{pol}(t_3) = x_2x_{18} - x_3x_{18},$	$\text{pol}(t_1) = x_1 - x_2x_4$	$\text{pol}(t_{13}) = x_2x_{17} - x_{16}$
$\text{pol}(t_{20}) = y_2y_{18} - y_3y_{18}$	$\text{pol}(t_2) = x_5 - x_{12}$	$\text{pol}(t_{14}) = x_{15} - x_{12}$
$\text{pol}(t_4) = x_3x_{13} - x_6$	$\text{pol}(t_7) = x_8 - x_{10}$	$\text{pol}(t_{15}) = x_4 - x_5$
$\text{pol}(t_{21}) = y_3y_{13} - y_6$	$\text{pol}(t_8) = x_{12} - x_{13}$	$\text{pol}(t_{16}) = y_8 - x_9$
$\text{pol}(t_5) = x_6 - x_7$	$\text{pol}(t_9) = x_{11} - x_2x_{14}$	$\text{pol}(t_{17}) = x_9 - x_{11}$
$\text{pol}(t_{22}) = y_6 - y_7$	$\text{pol}(t_{11}) = y_{15} - x_{17}$	$\text{pol}(t_{18}) = x_{10} - x_{11}$
$\text{pol}(t_6) = x_7 - x_8$	$\text{pol}(t_{12}) = x_3x_{17} - x_{16},$	$\text{pol}(t_{19}) = x_{16} - x_1$
$\text{pol}(t_{23}) = y_7 - y_8$	$\text{pol}(t_{25}) = y_3x_{17} - x_{16}$	

The complex Gröbner basis automatically generated from these polynomials is shown below. Some of the resulting expressions are shown graphically to clarify the meaning of the generated polynomials.

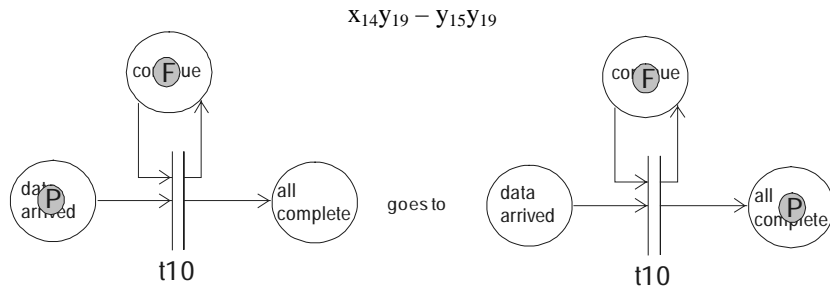
$y_3y_{15}y_{19} - y_8y_{19}$	$x_2y_{18} - y_3y_{18}$	$x_{14}x_3x_{18} - y_8x_{18}$
$y_{18}y_3y_{15} - y_8y_{18}$	$x_2y_{15} - y_3y_{15}$	$x_8 - y_8$
$x_{14}x_{19} - x_{15}x_{19}$	$x_2x_{18} - x_3x_{18}$	$x_7 - y_8$
$x_{16} - y_3y_{15}$	$y_7 - y_8$	$x_6 - y_8$
$y_3x_{15} - y_8$	$y_6 - y_8$	$x_{10} - y_8$
$y_{19}x_{15}x_{19} - x_{19}y_{15}y_{19}$	$x_{17} - y_{15}$	$x_{15}y_8 - y_{15}y_8$
$x_{14}y_8y_{18} - y_{15}y_8y_{18}$	$x_{14}y_{19} - y_{15}y_{19}$	$x_{14}y_{15}y_8 - y_{15}^2y_8$
$x_{14}y_8x_{18} - x_{18}y_{15}y_8$	$x_2x_{15} - y_3y_{15}$	$x_{14}y_8^2 - y_{15}y_8^2$
$x_{14}y_3y_{15} - y_{15}y_8$	$x_2x_{14} - y_8$	$x_{14}y_3y_{18} - y_8y_{18}$
$x_{13} - x_{15}$	$x_1 - y_3y_{15}$	$y_3y_{15}^2 - y_{15}y_8$
$x_{12} - x_{15}$	$x_2y_8 - y_3^2y_{15}$	$y_3y_{15}y_8 - y_8^2$
$x_{11} - y_8$	$x_3y_{15} - y_3y_{15}$	$y_3y_{15}y_{19} - x_{19}y_8$
$x_9 - y_8$	$x_3y_8 - y_3y_8$	$x_{18}y_3y_{15} - y_8x_{18}x_{17}$
$x_5 - x_{15}$	$y_{18}x_3x_{18} - x_{18}y_3y_{18}$	
$x_4 - x_{15}$	$x_3x_{15} - y_8$	

As can be seen through the graphical representation of these first two polynomials, the polynomials which form the basis are not necessary in their simplest form, they show combinations of the polynomials formed by the transitions, with P representing a pass token, and F representing a fail token.





This diagram shows the change between a “fail” token and a “pass” token, possible only at specified transitions.



This diagram shows the token in the “continue” place (19) affecting the output of the transition. The equivalent polynomials for changes made through the “input” place (18) are shown in the first two polynomials below.

The remainder of the polynomials forming the Grobner basis for the compass Petri net (shown in bold) are more difficult to trace through the Petri net diagram, as they require tokens to pass around the Petri net more than once. This is perfectly acceptable, as the Petri net is expected to be reversible, but it does lead to polynomials which are misleading at first glance, and similarly to reachable markings where the path taken is unclear.

The shown completed Gröbner basis, once generated, can be used to find every reachable marking of the compass Petri net is represents. However, due to the complexity of this example, these results are not listed here.

Testing the reachability of this Petri net, given a certain type of token, will confirm that the choices made by colouring of a given token will behave as the user would expect, beyond the simple testing of a Petri net with no colourings. This will enable the user to determine errors of this nature prior to the final generation of executable code for the mobile robot and decrease the necessary debugging time.

6 Conclusions and Future Work

The method for Petri net analysis described here has proved highly reliable and accurate. It is particularly successful in the detection of Petri nets which have falsely been assumed to be reversible, and in finding badly designated initial markings of the Petri net which may also stop it from being reversible.

However, the time taken for performance of these calculations remained, as with many previous methods, unacceptably long. The motors example shown in section 5 was completed successfully within a few minutes, but once a colouring was added alongside a number of places and transitions for the compass example (section 5) the time taken increased beyond that which could be considered reasonable for the user to wait, Gröbner basis generation taking approximately an hour.

The primary concern of any further work on this technique must be the reduction of the time taken for results of reachability testing to become available to the user. There are three possible avenues of research available, which may lead to an appropriate reduction in complexity. The first may consider the reduction of the Petri net itself. Whilst the TRAMP method of separating objects into individual components has already greatly decreased the Petri net complexity, it is clear that further effort must be put into this in order to provide a usable analysis service. This may be implemented through the use of standard Petri net reduction techniques [Murata 89].

The processing time may then be further reduced through the improved implementation of the Grobner basis techniques [Fröberg 97], which themselves have a number of efficiency algorithms which has yet to be utilised in these initial testing procedures.

A further alternative to the Petri net reduction and Gröbner basis efficiency techniques is the introduction of on-the-fly matrix generation method commonly applied to automata in order to improve the efficiency of the equivalent of reachability testing [Larsen 97]. If this method were to be applied to the Gröbner basis reachability test for the Petri net, then the overheads from large Petri nets would decrease dramatically.

A further desirable development may arise from the introduction of timings to the Petri net, which are already a widely used tool, and provide a valuable additional level of analysis to the Petri net.

References

- [Buchberger 98] An Algorithmic Criterion for the Solvability of a System of Algebraic Equations, Buchberger B (translation Abramson M and Lumbert R). Grobner Bases and Applications. Proc. London Math Soc. Vol 251. 1998.
- [Buchholz 92] A hierarchical View on GCSPNs and its Impact on Qualitative and Quantitative Analysis. Buchholz P. Journal of Distributed Computing, 1992, Vol 15, pp 207 – 224
- [Cavalieri 97] Impact of Fieldbus on Communication in Robotic Systems. Cavalieri S, DiStefano A, Mirabella O. IEEE Transactions on Robotics and Automation, 1997, Vol 13, No. 1, pp 30-48
- [Chandler 99] Grobner Basis Procedures for Testing Petri Nets. Chandler A, Heyworth A. UWB Math preprint 99.11. 1999
- [Chandler 99a] An Object-Oriented Petri Net Toolkit for Mechatronic System Design. Chandler A. PhD Thesis, Lancaster University, Engineering Department, 1999.
- [Fröberg 97] An Introduction to Gröbner Bases. Fröberg R. John Wiley and Sons, 1997.
- [Larsen 97] Efficient Verification of Real-Time Systems: compact data structure and state-space reduction. Larsen KG, Larsson F, Pettersson P, Yi W. Proceedings – Real-Time Systems Symposium, 1997.
- [Mascaro 98] Hand-in-Glove Human-Machine Interface and Interactive Control: Task Process Modelling Using Dual Petri Nets. Mascaro S, Asada HH. Proceedings - IEEE International Conference on Robotics and Automation, 1998, Vol 2, pp 1289-1295
- [Murata 89] Petri Nets: Properties, Analysis and Applications. Murata T. Proceedings of the IEEE, 1989, Vol 77, No. 4, pp 541 – 580
- [Simon 98] Design and Analysis of Synchronisation for Real-Time Closed-Loops Control in Robotics. Simon D, Castaneda EC, Freedman P. IEEE Transactions on Control Systems Technology, 1998, Vol 6, No. 4, pp 445-461

- [Suh 96] Design of a Supervisory Control System for Multiple Robotic Systems. Suh IH, Yeo HJ, Kim JH, Ryoo JS, Oh SR, Lee CW, Lee BH. IEEE International Conference on Intelligent Robots and Systems, 1996, Vol 1, pp 332-339
- [Yavuz 99] Conceptual Design and Development of a Navigation System for a Mobile Robot. Yavuz H, Chandler AK, Bradshaw A, Seward DW. Proceedings of CACD 99 (International Workshop on Engineering Design), 1999, pp 65 - 80