Automated Abstraction of Labeled Petri Nets

Kevin Jones

University of Utah

kjones@vlsigroup.ece.utah.edu

Abstract- *Due to the increasing use and complexity of embedded and cyber-physical systems, proper validation of these systems is an increasingly important topic. Because these systems are used in safety-critical situations, even intermittent failures are unacceptable. Formal verification allows for comprehensive validation, but becomes unrealistically complicated for large models. This paper proposes certain transforms to simplify system models without removing any of the details required for verification. These details must be conservative, in that they may not remove any state from the model's behavior. These transforms have shown this method promising in reducing the complexity of embedded system verification.*

I. Introduction

As embedded and cyber-physical systems become more prevalent, especially in safety-critical situations, formal verification of these systems becomes more important. Any verification makes necessary the development of a model that incorporates analog and digital circuitry, embedded software, and the environment. *Labeled Petri nets* (LPNs) are a proposed model that includes each of these elements. This model has been used previously to model AMS circuits, embedded systems, and environmental processes.

The LPN model was inspired by both *hybrid Petri nets* [3] and *hybrid autonoma* [2] and includes elements of both of these formalisms. While these formalisms would be adequate for modeling embedded and cyber-physical systems, we have found them to be difficult targets for automatic compilation. As the transforms described in this paper were developed, the LPN has also simplified the implementation of automated abstraction. Methods have been developed for generating LPNs from assembly language programs and similar low-level descriptions of the system, as well as from VHDL-AMS models and SPICE simulation data [4], [5], [8], [6], [7].

Low level modeling is essential for proper verification of embedded and cyber-physical systems. A high-level model does not include all of the timing information required for adequate verification. However, lower level models can be large and complex, which leads to verification being prohibitively expensive in terms of both time and memory. This paper provides transforms that can be applied to these models to reduce their complexity and ease verification of the model.

To perform verification, the state space of the model is explored until a transition fires that is marked as a failure transition or until the entire state space has been explored. Because of the hybrid nature of the system, the state has an infinite number of states that must be represented using a finite number of convex equivalence classes known as state sets. These sets are represented using zones, which are represented by *difference bound matrices* (DBMs). Once a failure transition fires, the verification is considered as having failed and the failure trace is returned to the user. If the entire state space is explored

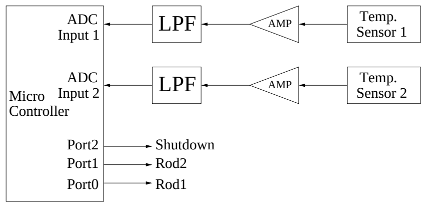

Fig. 1. Fault- tolerant temperature sensor for a nuclear reactor.

without a failure transition having fired, then verification succeeds. No possible failure trace can be removed from the model's behavior by these transforms because all abstractions are conservative.

II. Motivating Example

A useful example for demonstrating the LPN transforms is a temperature controller for a nuclear reactor [1]. This is a classic embedded system that includes hardware, software, and environmental elements. The system consists of two temperature sensors that are fed into two A/D converter inputs of a microcontroller. After the temperature readings are taken, they are compared. If the difference between the two readings is too great, it is assumed that one of the sensors is broken and the reactor is shut down. If both sensors are judged to be working correctly, the microcontroller inserts and removes cooling rods to keep the reactor temperature within set limits. If the reactor's temperature goes outside of those limits, the reactor is also shut down. A block diagram of the system can be found in Fig. 1.

At first glance, this appears to be a simple system, but it has some interesting details that highlight the importance of starting with a detailed, low-level model. First, because the microcontroller only has a single A/D converter, the temperature readings are taken at slightly different times. Second, the instruction that compares the two readings is not an atomic instruction at the assembly level, meaning that the temperature readings may not be from the same cycle. Depending on the system parameters, either one of these faults could cause the reactor to shut down unnecessarily. These details may be missed from a high-level representation of the system, but the timing issues they introduce can be preserved through applying these transforms.

III. The LPN Model

An LPN consists of a set of places, which act as the state holding elements, and transistions, which mark the changes between the states. It also includes Boolean, continuous, discrete, and rate variables. Each variable has an initial condition, and each continuous variable is mapped to its corresponding rate variable. A flow relation connects the places and transitions, a subset of the places are initially marked, and a subset of the transitions are marked as failure transitions. When a failure transition fires, verification has failed. Each transition has a label attached to it, which consists of an enabling condition, which must be satisfied for the transition to fire, a delay, which marks the amount of time that passes between the time that the enabling condition is satisfied and the transition fires, and a set of assignments, which change the value of variables when the transition fires.

An LPN model representing the software portion of the motivating example is found in Fig 2.

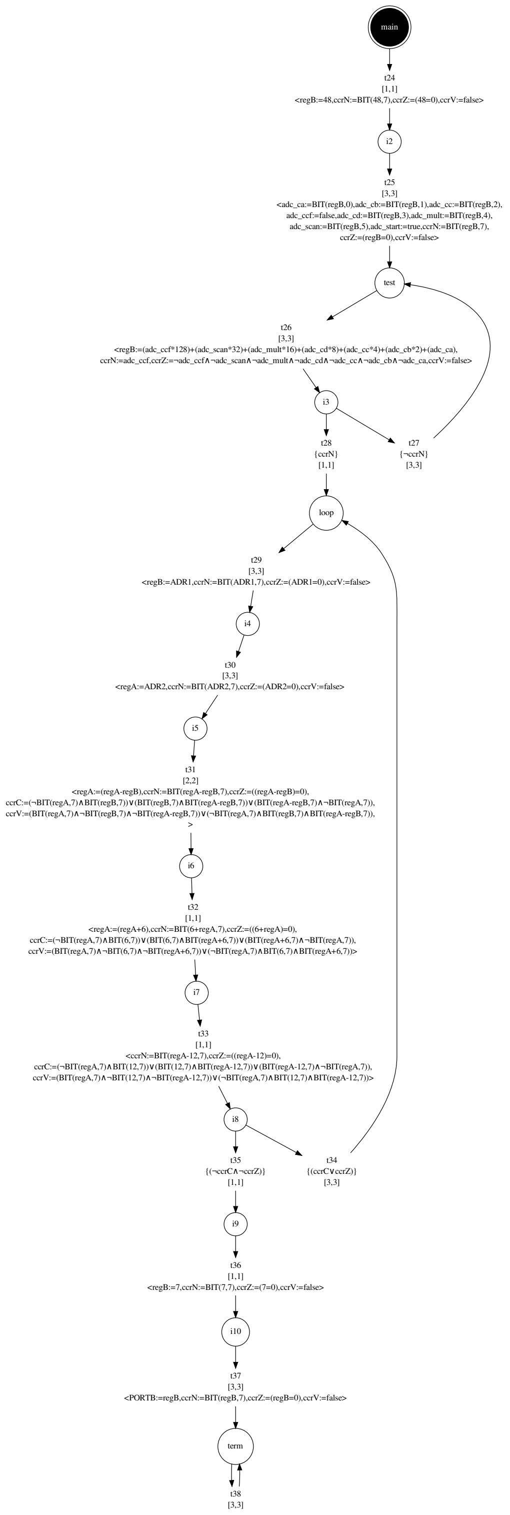
main
t24
[1,1]
<regB:=48,ccrN:=BIT(48,7),ccrZ:=(48=0),ccrV:=false>
i2
t25
[3,3]
<adc_ca:=BIT(regB,0),adc_cb:=BIT(regB,1),adc_cc:=BIT(regB,2),
adc_ccf:=false,adc_cd:=BIT(regB,3),adc_mult:=BIT(regB,4),
adc_scan:=BIT(regB,5),adc_start:=true,ccrN:=BIT(regB,7),
ccrZ:=(regB=0),ccrV:=false>
test
t26
[3,3]
<regB:=(adc_ccf*128)+(adc_scan*32)+(adc_mult*16)+(adc_cd*8)+(adc_cc*4)+(adc_cb*2)+(adc_ca),
ccrN:=adc_ccf,ccrZ:=¬adc_ccf∧¬adc_scan∧¬adc_mult∧¬adc_cd∧¬adc_cc∧¬adc_cb∧¬adc_ca,ccrV:=false>
i3
t28
{ccrN}
[1,1]
t27
{¬ccrN}
[3,3]
loop
t29
[3,3]
<regB:=ADR1,ccrN:=BIT(ADR1,7),ccrZ:=(ADR1=0),ccrV:=false>
i4
t30
[3,3]
<regA:=ADR2,ccrN:=BIT(ADR2,7),ccrZ:=(ADR2=0),ccrV:=false>
i5
t31
[2,2]
<regA:=(regA-regB),ccrN:=BIT(regA-regB,7),ccrZ:=((regA-regB)=0),
ccrC:=(¬BIT(regA,7)∧BIT(regB,7))∨(BIT(regB,7)∧BIT(regA-regB,7))∨(BIT(regA-regB,7)∧¬BIT(regA,7)),
ccrV:=(BIT(regA,7)∧¬BIT(regB,7)∧¬BIT(regA-regB,7))∨(¬BIT(regA,7)∧BIT(regB,7)∧BIT(regA-regB,7)),
>
i6
t32
[1,1]
<regA:=(regA+6),ccrN:=BIT(6+regA,7),ccrZ:=((6+regA)=0),
ccrC:=(¬BIT(regA,7)∧BIT(6,7))∨(BIT(6,7)∧BIT(regA+6,7))∨(BIT(regA+6,7)∧¬BIT(regA,7)),
ccrV:=(BIT(regA,7)∧¬BIT(6,7)∧¬BIT(regA+6,7))∨(¬BIT(regA,7)∧BIT(6,7)∧BIT(regA+6,7))>
i7
t33
[1,1]
<ccrN:=BIT(regA-12,7),ccrZ:=((regA-12)=0),
ccrC:=(¬BIT(regA,7)∧BIT(12,7))∨(BIT(12,7)∧BIT(regA-12,7))∨(BIT(regA-12,7)∧¬BIT(regA,7)),
ccrV:=(BIT(regA,7)∧¬BIT(12,7)∧¬BIT(regA-12,7))∨(¬BIT(regA,7)∧BIT(12,7)∧BIT(regA-12,7))>
i8
t35
{(¬ccrC∧¬ccrZ)}
[1,1]
t34
{(ccrC∨ccrZ)}
[3,3]
i9
t36
[1,1]
<regB:=7,ccrN:=BIT(7,7),ccrZ:=(7=0),ccrV:=false>
i10
t37
[3,3]
<PORTB:=regB,ccrN:=BIT(regB,7),ccrZ:=(regB=0),ccrV:=false>
term
t38
[3,3]

Fig. 2. LPN representing the fault-tolerant temperature sensor software.

IV. LHPN Transformations

As mentioned previously, a detailed system model is needed to ensure accurate verification of hybrid systems. However, models with the necessary level of detail are virtually impossible to analyze. Therefore, this paper proposes a series of transforms that can simplify a model while retaining the necessary behavioral details.

Because of the automatic compilation techniques used, in which each individual element of the model compiles directly to a net portion, the model can be simplified in some ways without changing its behavior. These transforms are known as simplifications. Other transforms have been introduced which introduce additional state into the model's behavior. These transforms are known as abstractions. A few transforms are shown as they affect the first portion of the software process model shown in Fig 2. The net fragment as it appears in the original model is shown in Fig 3(a).

A. *Remove Write Before Write*

If a variable is written to twice with no intervening reads, then the first assignment has no effect on the behavior of the system and can be removed. To be sure that the first assignment does not affect net behavior, there should be no concurrent reads or writes to the variable. In the example in Fig 3, the assignments to ccrN, ccrZ, ccrV on transition t24 are not read before assignments are made to the same variables on transition t25. These assignments can, therefore, be removed, as seen in Fig 3(b).

B. *Local Assignment Propagation*

Because there are no concurrent assignments to local variables, timing of assignments to them may be unimportant. If the expression being assigned to the variable is stable, these assignments can be delayed. In the given example, the assignment to the variable regB is moved from transition t24 to transition t25. Everywhere that regB is read in the assignments to other variables on t25, it is replaced with 48. Once this replacement has been made, the expressions in those assignments can be reduced to constant Boolean values. The other assignments on t24 are never read and can be removed. The net fragment after this transform has been applied is shown in Fig 3(c).

C. *Remove Vacuous Transitions*

Once the assignments have been removed from transition t24, it serves no purpose other than to keep track of time. This transition can, therefore, be removed and the time pushed onto the transition following it. This transform is valid as long as the enabling condition of transition t25 cannot be disabled. Fig 3(d) shows the net fragment mentioned above after these transitions have been combined.

D. *Timing Bound Normalization*

The transform that has the biggest impact on actual verification time and memory usage is normalizing the timing bounds. This is the only transform described in this paper that increases the analyzed state space and could introduce false negative verifications. By making the state sets more regular, verification is simplified and can complete much more quickly. To perform this transform, every delay is expanded to be an integer multiple of a user-set normalization factor. Section V describes how effective this transform is in minimalizing verification time.

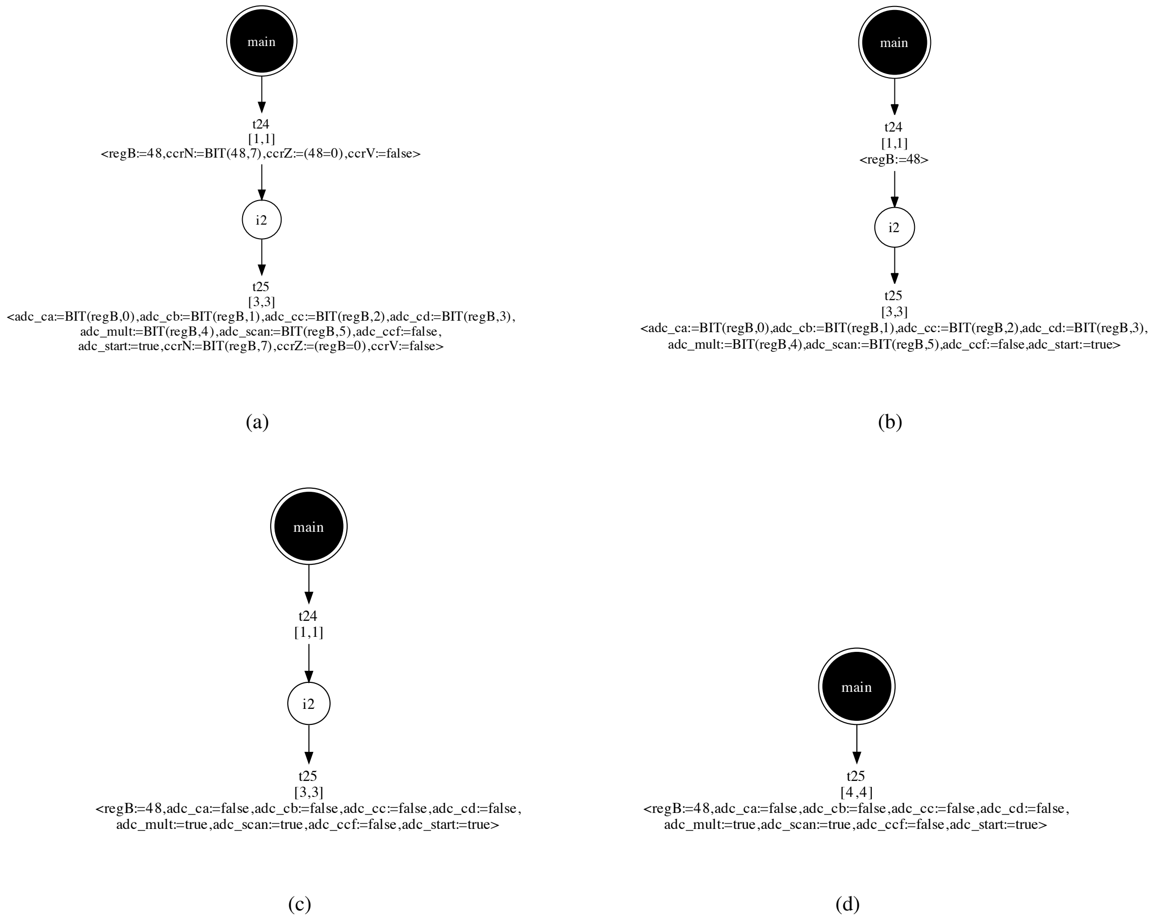
main
t24
[1,1]
<regB:=48,ccrN:=BIT(48,7),ccrZ:=(48=0),ccrV:=false>
i2
t25
[3,3]
<adc_ca:=BIT(regB,0),adc_cb:=BIT(regB,1),adc_cc:=BIT(regB,2),adc_cd:=BIT(regB,3),
adc_mult:=BIT(regB,4),adc_scan:=BIT(regB,5),adc_ccf:=false,
adc_start:=true,ccrN:=BIT(regB,7),ccrZ:=(regB=0),ccrV:=false>
(a)
main
t24
[1,1]
<regB:=48>
i2
t25
[3,3]
<adc_ca:=BIT(regB,0),adc_cb:=BIT(regB,1),adc_cc:=BIT(regB,2),adc_cd:=BIT(regB,3),
adc_mult:=BIT(regB,4),adc_scan:=BIT(regB,5),adc_ccf:=false,adc_start:=true>
(b)
main
t24
[1,1]
i2
t25
[3,3]
<regB:=48,adc_ca:=false,adc_cb:=false,adc_cc:=false,adc_cd:=false,
adc_mult:=true,adc_scan:=true,adc_ccf:=false,adc_start:=true>
(c)
main
t25
[4,4]
<regB:=48,adc_ca:=false,adc_cb:=false,adc_cc:=false,adc_cd:=false,
adc_mult:=true,adc_scan:=true,adc_ccf:=false,adc_start:=true>
(d)

Fig. 3. Software initialization transitions. (a) Initial model, (b) after removing write before write, (c) after propogating local assignments, and (d) after removing vacuous transitions.

E. Merge Parallel Transitions

This transform merges transitions that connect the same two places in the LHPN model. For this transition to be applied, the enabling conditions of the transforms must be mutually exclusive. Additionally, the transforms must not become disabled once they are enabled. These conditions verify the existence of certain timing assumptions that are being incorporated into the transform definition. If these conditions are met, then these transforms may be combined into a single transform, for which the enabling condition is a union of the enabling conditions of each of the individual parallel transitions and the delay expression is a mathematical combination of the enabling conditions and delay expressions of the individual transitions.

There are two variations on this transform, a simplification and an abstraction. The simplification generates more complicated expressions but provides a more precise definition of the model's behavior. The abstraction is simpler, but is not as precise in defining the behavior and may introduce additional extraneous system behavior into the model.

F. Remove Uninteresting Variables

This transform is an extension of the previously existing transform that removed unread variables. If a variable is never read, it obviously does not affect the system's behavior and may therefore be safely removed. Some variables, however, are read, but in portions of the net that do not affect the property of interest, i.e. the firing of a failure transition. Such variables may also be removed, for although they affect the system behavior, they do no affect the property of interest. An algorithm has been developed for determining which variables affect properties of interest, and the rest of the variables may effectively be removed from the net. An option is also provided for they user to provide variables that may be interesting for reasons other than their effect on a failure transition.

G. Remove Uninteresting Transitions

Similar to uninteresting variables, uninteresting transitions are transitions in the LHPN that do not affect the behavior of the property of interest. Namely, if a transition does not have any assignments and is not followed by a transition that has any assignments, then the transition does not affect the property of interest. It may be noted that after the removal of uninteresting variables, many transitions that previously contained assignments could be left without any. Any such transition may be removed from the net. Because of the nature of uninteresting transitions, the transitions following an uninteresting transition would also be uninteresting and could be removed. In theory, large portions of the net could be removed if they do not directly affect the property of interest in the verification.

H. Other LHPN Transformations

There are several other LHPN transformations that have been developed but not presented in detail here due to space limitations. These include:

- Combine correlated variables,
- Remove unread variables,
- Remove dead transitions,
- Remove arc after failure transitions,
- Constant enabling condition determination,
- Removing dominated transitions,
- Removing vacuous loops, and
- Removing unnecessary transitions with global variables in their enabling conditions.

For details about these transformations, please see [7].

The final reduced LHPN model of the software process is shown in Fig. 4.

V. Results

The `LEMA` verification tool has been upated to support automatic abstraction of LHPN models. This tool was applied to the fault-tolerant temperature sensor discussed in Section II, with several variations in parameter values. The results are shown in Table I. For each case, the number of state sets found, runtime in seconds, and whether it verifies to be correct are reported. The property being verified is that the reactor never shuts down, assuming that the sensors are working correctly.

TABLE I

VERIFICATION RESULTS FOR THE REACTOR EXAMPLE.

Parameters	States	Time (s)	Verifies
Original model	1,672,714	31,937	Yes
Abstracted model	35,563	133	Yes
With norm. delays	57427	157	Yes
Without init. loop	5	0.006	No
9-bit ADCs	945	0.077	No
Slow ADC	38	0.008	No
$temp$ rates $[-4, 4]$	32	0.009	No
$temp$ rates $[-4, 4]$, 7-bit ADCs	21,787	50	Yes

VI. CONCLUSION

Formal verification, though desirable for embedded systems, is complicated by the size and complexity of the models involved. These models may be reduced through transforms designed to preserve the state space while reducing the complexity of its representation. This project will build on previous work that has been done in this regard. Transforms have been described that merge parallel transitions to reduce model complexity and remove uninteresting portions of the net by intelligently determining which variables and transitions affect the property of verification.

REFERENCES

[1] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P. H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Comp. Sci.*, 138(1):3–34, 1995.

[2] R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In Robert L. Grossman, Anil Nerode, Anders P. Ravn, and Hans Rischel, editors, *Hybrid Systems*, volume 736 of *LNCS*, pages 209–229. Springer, 1992.

[3] R. David and H. Alla. On hybrid Petri nets. *Discrete Event Dynamic Systems: Theory and Applications*, 11(1–2):9–40, January 2001.

[4] S. Little, N. Seegmiller, D. Walter, C. Myers, and T. Yoneda. Verification of analog/mixed-signal circuits using labeled hybrid Petri nets. In *Proc. International Conference on Computer Aided Design (ICCAD)*, pages 275–282. IEEE Computer Society Press, 2006.

[5] S. Little, D. Walter, and C. Myers. Analog/mixed-signal circuit verification using models generated from simulation traces. In *Automated Technology for Verification and Analysis (ATVA)*, volume 4762 of *LNCS*, pages 114–128. Springer, 2007.

[6] R. Thacker, C. Myers, K. Jones, and S. Little. A new verification method for embedded systems. In *Proc. International Conference on Computer Design (ICCD)*. IEEE Computer Society Press, 2009.

[7] R. A. Thacker. *A New Verification Method for Embedded Systems*. PhD thesis, U. of Utah, January 2010.

[8] D. Walter, S. Little, C. Myers, N. Seegmiller, and T. Yoneda. Verification of analog/mixed-signal circuits using symbolic methods. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, 27(12):2223–2235, 2008.

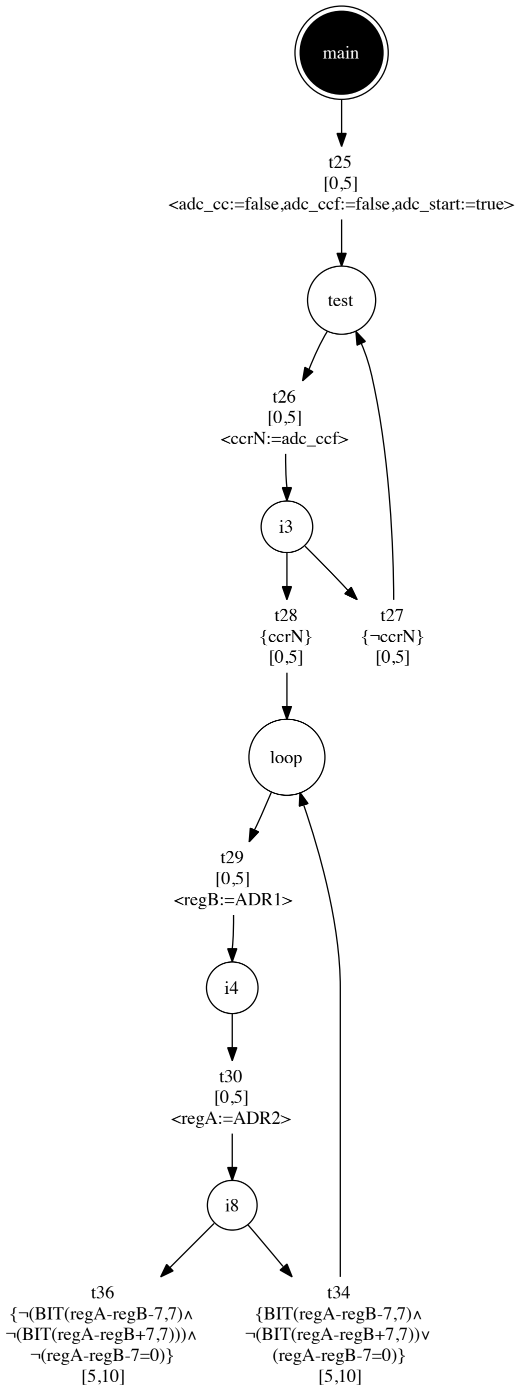

Fig. 4. Final reduced LHPN model for the software process.