Handling Guards and Vacuous Transitions

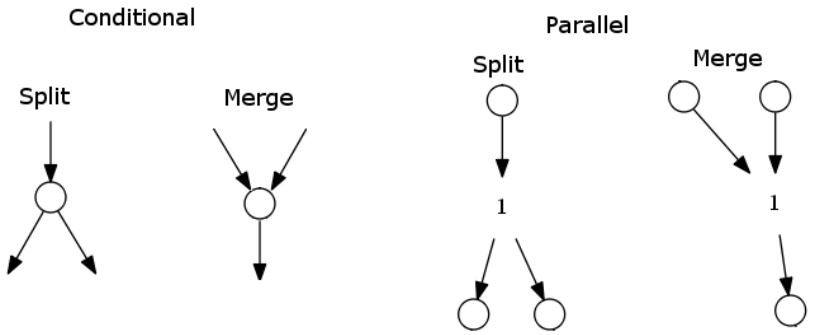
June 16, 2014

I. Background

A. Petri Nets

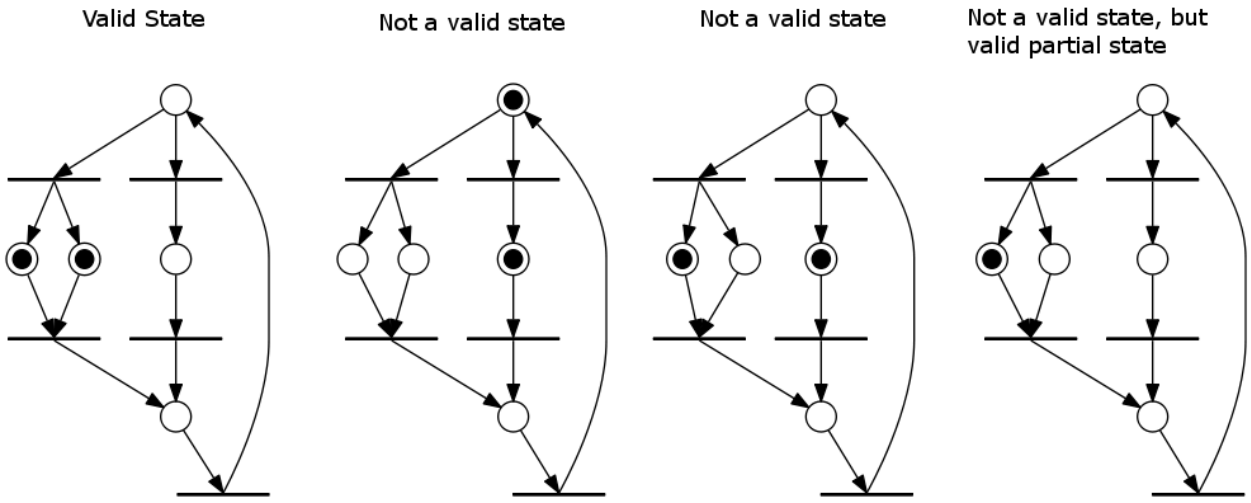
A petri net is a bipartide graph consisting of places, transitions, and arcs where an arc connects a place to a transition or a transition to a place. Generally places are represented by circles and transitions are represented by horizontal lines. In this paper, we will also represent transitions with plaintext giving the action the transition takes. Places from which arcs run to a transition are called input places and places to which arcs run from a transition are called output places. Places in a petri net may contain a discrete number of marks called tokens. A collection of tokens is called a marking. A transition is enabled if there is at least one token at every input place, and that transition may only fire once it is enabled. Upon firing, that transition will consume one token from every input place and generate one token at every output place. A firing is atomic.

A transition that has multiple outgoing arcs is a parallel split, and a transition that has multiple incoming arcs is a parallel merge. A place that has multiple arcs is a conditional split, and a place that has multiple incoming arcs is a conditional merge.



Definition 1: *A **state** is a marking such that no token follows another sequentially, no two tokens are on separate banches of the same conditional, and that marking forms a complete graph cut across all parallel branches.*

Definition 2: *A **partial state** is a marking such that no token follows another sequentially and no two tokens are on separate banches of the same conditional.*



B. Hand Shaking Expansions

Hand shaking expansions are represented by petri nets modified in the following way.

DEFINITION 3: *A **state encoding** is a value assignment of either true or false for every node in the circuit represented by the hand shaking expansion.*

DEFINITION 4: *A **predicate** is a boolean expression which represents the set of possible state encodings for a given marking. A marking and its predicate will be represented by the same symbol.*

DEFINITION 5: *A **transition expression** is a boolean expression representing the action that a transition takes. A transition and its expression will be represented by the same symbol.*

Boolean operators and set operators are interchangable when dealing with predicates and transition expressions. Given two boolean expressions E_0 and E_1 and the two sets of state encodings they represent S_0 and S_1:

- $S_0 \cap S_1 \equiv E_0 \wedge E_1$
- $S_0 \cup S_1 \equiv E_0 \vee E_1$
- $S_0^C \equiv \neg E_0$
- $S_0 \subset S_1 \equiv E_0 \wedge E_1 = E_0$

For the purpose of this paper in all cases, I will use the boolean operators except when I want the subset in which case I will use the set operator.

DEFINITION 6: ***Active transitions** represent assignments. The transition expression for an active transition represents a value assignment for a single node. When an active transition fires, it changes the value assignment for that node in all state encodings represented by a given boolean expression. The action taken by an active transition will represented by the '$\rightarrow$' operator.*

LEMMA 1: $(A \vee B) \rightarrow T = (A \rightarrow T) \vee (B \rightarrow T)$

LEMMA 2: $(A \wedge B) \rightarrow T = (A \rightarrow T) \wedge (B \rightarrow T)$

LEMMA 3: $\neg(A \rightarrow T) = \neg A \rightarrow \neg T$

DEFINITION 7: ***Passive transitions** represent guards. The transition expression for a guard may be any boolean expression. For a passive transition T to fire given the current state S_i, $S_i \wedge T \neq false$ meaning there is at least one state encoding represented by the predicate of S_i that satisfies the condition given by T. This also means that a passive transition may be enabled, but is unable to fire. When a passive transition fires, the predicate for the new state, $S_{i+1} = S_i \wedge T$. If T has multiple minterms T_j such that $S_i \wedge T_j \neq false$, then each of those terms must be considered in independent executions.*

DEFINITION 8: *A transition T with input places P_{in} and output places P_{out} is **definitely vacuous** if*

$$\bigwedge_{P_o \in P_{out}} P_o = \bigwedge_{P_i \in P_{in}} P_i \tag{1}$$

DEFINITION 9: *A transition T with input places P_{in} and output places P_{out} is **possibly vacuous** if*

$$\bigwedge_{P_o \in P_{out}} P_o \wedge \bigwedge_{P_i \in P_{in}} P_i \neq false \tag{2}$$

LEMMA 4: *All passive transitions are either possibly or definitely vacuous.*

By definition of passive transition, the predicate of the output places of a passive transition is always a subset of the predicate of the input places.

LEMMA 5: *All of the output transitions of a conditional split will be passive.*

The only case in which a conditional would not have a guard is when the guard is $true$. However if the guard is $true$, then there can only be one guard and therefore it is not a conditional split.

There are also some restrictions that must be maintained in order for this to work. First, a transition with multiple incoming arcs or multiple outgoing arcs must be definitely vacuous (preferably $T \equiv true$). Second, a transition cannot have both multiple incoming arcs and multiple outgoing arcs. This means that you cannot have a parallel merge in the same transition that you also have a parallel split. These two requirements guarantee that all conflicting state pairs are detectable and that it is possible to insert a state variable transition between any two transitions, splits, or merges by cutting exactly one arc. Finally, proper nesting is required for conditional splits and merges, but not required for parallel splits and merges, and the initial marking must be a state that is not on a conditional branch. This requirement allows for correct simulation of the hand shaking expansion.

II. STATE ELABORATION

A. The basics

The state elaboration algorithm at its core is just a full exploration of all possible executions of a set of petri nets.

DEFINITION 10: *A **program counter** is a token and a boolean minterm that represents a set of state encodings.*

DEFINITION 11: *A **program state** is a set of program counters such that their tokens form a state across all petri nets in the execution.*

DEFINITION 12: *An **execution** is a single trace of a program state through the set of petri nets.*

The elaboration algorithm consists of a set of executions E initialized with one execution at the reset state. Given the set of transitions T which are allowed to fire given the current program state of an execution $E_i \in E$, E_i is duplicated into a set $E_i \subset E$ of $|T|$ executions equivalent to the original E_i. For each of the executions $E_{i_j} \in E_i$, the transition T_j fires. If T_j is passive and has multiple minterms that are allowed to fire, then the execution E_{i_j} will be duplicated again into a set $E_{i_j} \subset E_i$ of executions with one execution $E_{i_{j_k}}$ for every minterm $T_{j_k} \in T_j$. The set of program counters, PC, that enable T_{j_k} are merged into one, such that it's minterm is:

$$\bigwedge_{PC_i \in PC} PC_i \tag{3}$$

Then the minterm of that program counter, PC, will be modified to apply the transition:

- T_j is passive: $PC = PC \wedge T_{j_k}$
- T_j is active: $PC = PC \rightarrow T_j$

If T_j is active, then the minterm of all of the other program counters, PC_x, in the execution are modified to include the transition event:

$$PC_x = PC_x \vee (PC_x \rightarrow T_j) \tag{4}$$

Finally, PC will be duplicated to all of the output places of T_j.

Every time a program counter PC passes over a place P, the minterm of that program counter is ORed into the predicate of that place: $P = P \vee PC$. This creates the predicates for all of the places.

DEFINITION 13: *A **half synchronization** event is where a set of program counters C must wait at an enabled passive transition T_p until the program state $S \supset C$ has fired an active transition T_a that fills the condition for T_p and allows it to fire. It is a directional guarantee that if T_p has fired, then the program state has passed the transition T_a.*

DEFINITION 14: *A **full synchronization** event consists of two opposing half synchronization events, guaranteeing that if one set of program counters C_0 in the program state S has passed a set of transitions T_0, then another set of program counters C_1 in the program state S has also passed a set of transitions T_1.*

LEMMA 6: *Parallel composition of two places P_0 and P_1 structurally does not imply that a state S exists such that both P_0 and P_1 are in S.*

It is possible for a half synchronization event to guarantee that if a program counter is at P_1, then another program counter must have already passed P_0.

LEMMA 7: *Given a place P and the set of states S that contain P:*

$$P = \bigvee_{S_i \in S} S_i \tag{5}$$

By Construction.

B. Early exit

Given two partial executions E_0 and E_1 for which different choices were made for ordering of events or for conditional splits, if they end up at the same program state after all of those choices, then one of those executions may be dropped and only one needs to continue.

C. Scheduling for fast early exit

D. Scheduling for memory management

E. Independant Parallelism

III. Effective Predicate

A. Why not just a special case

During channel reset when all data lines are being cleared, all of those transitions are possibly vacuous. When combined with the Guard before it which is always at least possibly vacuous, you can start to get some decently complicated structures in the simplest of processes. For example, every reshuffling for a one bit fifo process already has a possibly vacuous set of transitions composed in a combination of parallel and sequence. In these examples I will only highlight the more complicated structures.

Listing 1: PCHB One Bit FIFO

```
*[
  [ R.e ∧ L.t → R.t ↑
  []R.e ∧ L.f → R.f ↑
  ]; L.e ↓; [¬R.e]; R.t ↓, R.f ↓; [¬L.t ∧ ¬L.f]; L.e ↑
 ]
```

Listing 2: WCHB One Bit FIFO

```
*[
  [ R.e ∧ L.t → R.t ↑
  []R.e ∧ L.f → R.f ↑
  ]; L.e ↓; [¬R.e ∧ ¬L.t ∧ ¬L.f]; R.t ↓, R.f ↓; L.e ↑
 ]
```

Listing 3: PCFB One Bit FIFO

```
*[
  [ R.e ∧ L.t → R.t ↑
  []R.e ∧ L.f → R.f ↑
  ]; L.e ↓; en ↓;
  (
    [¬R.e → R.t ↓, R.f ↓]||
    [¬L.t ∧ ¬L.f → L.e ↑]
  ); en ↑
 ]
```

Now you start to introduce things like skips into the HSE and the problem gets much worse. Here is the HSE for a pcfb reshuffling of a constant time accumulator process:

Listing 4: PCFB One Bit Constant Time Accumulator

```
*[
  [ D.e ∧ A.f ∧ B.f → D.f ↑
  []D.e ∧ A.f ∧ B.t → D.t ↑
  []A.t → skip
  ]||
  [ S.e ∧ A.f → S.f ↑
  []S.e ∧ A.t ∧ B.t → S.t ↑
  []A.t ∧ B.f → skip
  ]||
  [ A.e ∧ (A.f ∨ A.t ∧ B.t) → T.f ↑
  []A.e ∧ A.t ∧ B.f → T.t ↑
  ]||
  [ Ne ∧ (A.t ∧ B.f) → Nr ↑
  []A.f ∨ B.t → skip
  ];
  A.e ↓; en ↑;
  (
    [¬A.f ∧ ¬A.t → A.e ↑]||
    [ (D.f ∨ D.t) ∧ ¬D.e → D.f ↓, D.t ↓
    []¬D.f ∧ ¬D.t → skip
    ]||
    [ (S.f ∨ S.t) ∧ ¬S.e → S.f ↓, S.t ↓
    []¬S.f ∧ ¬S.t → skip
    ]||
    [ Nr ∧ ¬Ne → Nr ↓
    []¬Nr → skip
    ]||
```

```
    [ T.f → B.t ↓; B.f ↑; T.f ↓
    []T.t → B.f ↓; B.t ↑; T.t ↓
    ]
  )
]
```

B. Current Implementation

Given a marking M_0 such that there is at least one token on every input place of a group of transitions T and every transition $T_i \in T$ is vacuous for that marking's state encoding, and a marking M_1 that is equal to M_0 after all of the transitions $T_i \in T$ have fired, M_0 is equivalent to M_1. This means that when searching for conflicting markings, M_0 cannot conflict with M_1.

DEFINITION 15: *The **effective predicate** is a restriction of the predicate of a state S_0 relative to another state of places S_1 that removes the state encodings from S_0 for which all transitions between S_0 and S_1 are vacuous.*

DEFINITION 16: *The **effective restriction** is the restriction placed on the predicate of a state in order to get the effective predicate: $effective(S_0, S_1) = S_0 \wedge restrict(S_0, S_1)$*

LEMMA 8: *Given two partial states S_0 and S_1 and a set of transitions T from S_0 to S_1 in parallel or sequence, the effective restriction from S_0 to S_1 is:*

$$restrict(S_0, S_1) = \bigvee_{T_i \in T} \neg T_i \tag{6}$$

If a state encoding in S_0 causes all of transtions in T to be vacuous, then that state encoding should also be in S_1. This state encoding should be ignored since the transitions for which S_1 is an implicant are allowed to fire in that case. This means that in order for a state encoding to not be ignored, at least one of the transitions in T must be non-vacuous.

LEMMA 9: *Given two partial states S_0 and S_1 such that there are multiple conditional branches B from S_0 to S_1, the effective restriction from S_0 to S_1 across all branches is:*

$$restrict(S_0, S_1) = \bigwedge_{B_i \in B} restrict(S_0, S_1, B_i) \tag{7}$$

Since there are multiple conditional branches that could be taken in order to get from S_0 to S_1, then at least one transition in each branch must be non-vacuous given a state encoding in S_0 for that state encoding to not also show up in S_1 as a valid implicant for the transitions after S_1. If all of the transitions down one of the branches were vacuous then that branch was taken and the transitions for which S_1 is an implicant are allowed to fire.

Given two partial states with a set of transitions T, if T has a pair of transitions in different directions on the same variable, then the restriction comes out to 1 meaning that there are no state encodings that we can throw away. If all of the transitions in T are definitly vacuous, then the restriction will come out to 0 meaning we can completely ignore that entire state. If all of the transitions in T are non-vacuous, then the state encoding will fit perfectly within the restriction, having the same effect as a restriction of 1.

C. What I think is missing

I am missing some kind of interaction with the environment. If we look at a simple one bit pcfb fifo:

Listing 5: PCFB One Bit FIFO

$$
\begin{array}{l}
*[\\
\quad [\ R.e \wedge L.t \rightarrow R.t\uparrow \\
\quad []R.e \wedge L.f \rightarrow R.f\uparrow \\
\quad]; L.e\downarrow; en\downarrow; \\
\quad (\\
\quad\quad \mathbf{A}[\neg R.e \rightarrow R.t\downarrow, R.f\downarrow]|| \\
\quad\quad [\neg L.t \wedge \neg L.f \rightarrow L.e\uparrow]\mathbf{A} \\
\quad)\mathbf{B}; en\uparrow \\
\ ]
\end{array}
$$

Using the current implementation, the effective restriction from the state A to the state B comes out to be $R.f \vee R.t \vee R.e$. However, at B, the environment is allowed to change the value of $R.e$. This means that the transition on $R.e$ cannot be used in the effective restriction calculation. The effective restriction should really be 1.

D. Alternative Method

If a state encoding in S enables a transition T, then that state encoding is a duplicate of a state encoding that was really just passing through on its way to the next state.

DEFINITION 17: *The **effective predicate** of a state S is a restriction of the predicate that removes the state encodings for which any transition in the set of enabled output transitions of S, T, is vacuous.*

$$effective(S) = S \wedge \bigwedge_{T_i \in T} \neg T_i \tag{8}$$

DEFINITION 18: *The **effective predicate** of a partial state S_p representing the set of states S is:*

$$effective(S_p) = \bigvee_{S_i \in S} effective(S_i) \tag{9}$$

LEMMA 10: *Given the set of states S for which S_p is a partial state and S_{np} is a set of partial states such that each partial state $S_{npi} \in S_{np}$ contains the places in S_i and not S_p, the set of enabled output transitions T_i of $S_i \in S$, T_p of S_p, T_{npi} of $S_{npi} \in S_{np}$, and T_{ni} such that $T_{ni} \subset T_i$ but $T_{ni} \cap (T_p \cup T_{npi}) = \emptyset$.*

$$
\begin{aligned}
effective(S_p) &= \bigvee_{S_i \in S} effective(S_i) && (10)\\
&= \bigvee_{S_i \in S} \left(S_i \wedge \bigwedge_{T_{ij} \in T_i} \neg T_{ij} \right) && (11)\\
&= \bigvee_{S_{npi} \in S_{np}} \left(S_p \wedge S_{npi} \wedge \left(\bigwedge_{T_{pi} \in T_p} \neg T_{pi} \right) \wedge \left(\bigwedge_{T_{npij} \in T_{npi}} \neg T_{npij} \right) \wedge \left(\bigwedge_{T_{nij} \in T_{ni}} \neg T_{nij} \right) \right) && (12)\\
&= \left(S_p \wedge \bigwedge_{T_{pi} \in T_p} \neg T_{pi} \right) \wedge \bigvee_{S_{npi} \in S_{np}} \left(\left(S_{npi} \wedge \bigwedge_{T_{npij} \in T_{npi}} \neg T_{npij} \right) \wedge \bigwedge_{T_{nij} \in T_{ni}} \neg T_{nij} \right) && (13)
\end{aligned}
$$

If a transition $T_{npi} \in T_{np}$ is at least possibly vacuous, then the state encoding for which T_{npi} is vacuous exists in both the predicates for the input and output places by definition of vacuous. Given that S_i is an input state of T_{npi} and the state S_o is the result of the transition T_{npi} out of S_i and that state encoding is not vacuous for any output transition of S_o, then the above equation will remove a state encoding and

leave its duplicate, essentially doing nothing. Therefore, we do not need to remove these state encodings in the first place.

$$= \left(S_p \wedge \bigwedge_{T_{pi} \in T_p} \neg T_{pi} \right) \wedge \bigvee_{S_{npi} \in S_{np}} \left(S_{npi} \wedge \bigwedge_{T_{nij} \in T_{ni}} \neg T_{nij} \right) \tag{14}$$

DEFINITION 19: *The* ***effective predicate*** *of a partial state S_p given its output transitions T and the set of states for which one of those transitions have fired S_o:*

$$effective(S_p) = S_p \wedge \bigwedge_{S_{oi} \in S_o} \neg S_{oi} \tag{15}$$

The reason for using the output partial states instead of the output transitions in this definition is because of parallel merges. Given a set of states S represented by the partial state S_p such that states exists that both enable and don't enable a parallel merge transition T, then T is only vacuous some of the time given S_p. So it would be wrong to restrict the predicate of S_p using $\neg T$ because that could cut out state encodings for which T is not enabled. However if we instead look at the output partial states, we cannot eliminate a state encoding for which T was not actually vacuous because a state encoding will only exist in the predicate of an output partial state if T was enabled and fired.

IV. CHECKING FOR CONFLICTS

A. Conflicting States

DEFINITION 20: *A state S_0 with output transitions T_0 is* ***indistinguishable*** *from a state S_1 with output transitions T_1 for an active transition $T_{0_i} \in T_0$ given the following conditions:*

- *S_0 and S_1 are not the same state.*
- *$effective(S_1) \wedge hide(S_0, T_{0_i}) \neq 0$*

We need to look at the effective predicate of S_1 to eliminate any cases where all of the transitions in between S_1 and S_0 are vacuous, and we need to hide the node affected by the active transition from the predicate of S_0 because we cannot use that node to create a production rule for that transition.

DEFINITION 21: *A state S_0 with output transitions T_0* ***conflicts*** *with a state S_1 with output transitions T_1 for an active transition $T_{0_i} \in T_0$ given the following conditions:*

- *S_0 and S_1 are indistinguishable.*
- *T_{0_i} is not a vacuous transition for the state S_1.*
- *There is no transition $T_{1_j} \in T_1$ such that $T_{1_j} = T_{0_i}$.*

B. An Optimization: Conflicting Places

We can bring the conflict check from $O(k^{2n})$ down to $O(n^2)$ where n is the number of places by taking advantage of some simple properties.

DEFINITION 22: *A place P_0 with output transitions T_0 is* ***indistinguishable*** *from a place P_1 with output transitions T_1 for an active transition $T_{0_i} \in T_0$ given the following conditions:*

- *P_0 and P_1 are not the same place.*
- *P_0 and P_1 are not in parallel.*
- *$effective(P_1) \wedge hide(P_0, T_{0_i}) \neq 0$*

DEFINITION 23: *A place P_0 with output transitions T_0* ***conflicts*** *with a place P_1 with output transitions T_1 for an active transition $T_{0_i} \in T_0$ given the following conditions:*

- P_0 *and* P_1 *are indistinguishable.*
- T_{0_i} *is not a vacuous transition for the place* P_1*.*
- *There is no transition* $T_{1_j} \in T_1$ *such that* $T_{1_j} = T_{0_i}$*.*