

An efficient Algorithm for the Enabling Test of Colored Petri Nets

Sami Evangelista Jean-Francois Pradat-Peyre

CEDRIC - CNAM Paris

october 2004

Outline

- 1 The enabling test problem
- 2 Conflict and Causality relations
 - Definition
 - Implementation
- 3 Unification algorithm
 - Principle
 - Scheduling of the input tuples of a transition
- 4 Conclusions

Outline

- 1 The enabling test problem
- 2 Conflict and Causality relations
 - Definition
 - Implementation
- 3 Unification algorithm
 - Principle
 - Scheduling of the input tuples of a transition
- 4 Conclusions

The enabling test problem

- Colored Petri nets (CPN) is a high level language to express complex synchronization patterns.
- Major drawback : simulation and analysis of CPN is made difficult by the management of the colors of the net.
- The enabling test consists in finding under which assignments (or bindings) the transitions of a CPN are firable.
- For general CPN, i.e. with unstructured color domains and mappings, this problem is
 - undecidable if we consider infinite color domains
 - equivalent to unfold the net if the color domains are finite
- However, if we consider some restricted class of CPN, e.g. Well Formed Petri Nets we can define specific optimizations techniques.

A sub class of CPN

- Color domains are cartesian products of basic sets called color classes.
- Expressions that can appear in color mappings can be
 - 1 a variable of the corresponding transition, e.g. X, Y
 - 2 a constant, e.g., $0, true$
 - 3 a user defined mapping e.g., $f(X, 0)$
- Basic mappings are tuples of expressions, e.g., $\langle X, 0, f(X, 0) \rangle$.
- Color mappings are linear combinations of tuples, e.g., $2.\langle X, 0, f(X, 0) \rangle + 3.\langle X, Y, Z \rangle$.
- Transition guards can be any boolean expressions on the variables of the transition.

- Locality principle : the firing of a transition instance only affect the status (enabled / disabled) of its neighbor transitions.
- Simulation and search algorithms can benefit from this principle by :
 - 1 maintaining a set of enabled transitions
 - 2 updating this set when a transition instance is fired by only checking the transitions that share some places with the fired instance
- To update this set efficiently our algorithm relies on the relations of structural conflict and causality between transitions.
- **Conflict** relation enables to identify the disabled instances.
- **Causality** relation enables to identify the enabled instances.

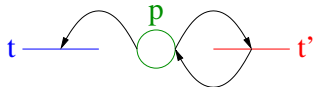
Outline

- 1 The enabling test problem
- 2 Conflict and Causality relations**
 - Definition
 - Implementation
- 3 Unification algorithm
 - Principle
 - Scheduling of the input tuples of a transition
- 4 Conclusions

Structural ordinary conflict

Structural conflict relation can help us to determine transitions instances **disabled** by a firing. t' is in structural conflict with t

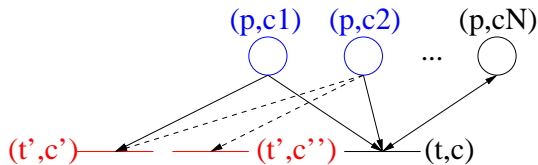
- t and t' share a same input place p
- t decrements the number of tokens in p



Structural colored conflict

(t', c') is in structural conflict with (t, c)

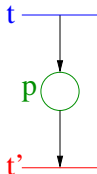
- t and t' share a same input place p
- $c' \in \overline{{}^t W^-(p, t')} \circ \overline{W^-(p, t) - W^+(p, t)}(c)$



Structural ordinary causality

Structural causality relation can help us to determine transitions instances **enabled** by a firing. t' is causally connected to t

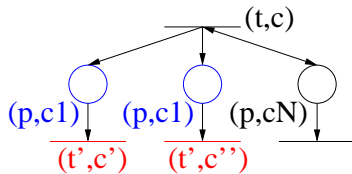
- t' has an input place p which is an output place of t
- t increments the number of tokens in p



Structural colored causality

(t', c') is causally connected to (t, c)

- t' has an input place which is an output place of t
- $c' \in \overline{{}^tW^-(p, t')} \circ \overline{W^+(p, t) - W^-(p, t)}(c)$



Implementing conflict and causality relations

- Problem : computing the conflict and causality relation is impossible for general colored Petri nets without unfolding.
- However, the class of net defined enables to deal with this problem in a symbolic way, i.e. without enumerating items of color domains.
- Idea : translate these relations into equivalent constraints systems before the search algorithm and solve these systems during the search to identify disabled / enabled transitions instances.

Translating mappings to constraints systems

Definition (Constraints System)

A constraints system is a tuple $\langle V, D, I, O, P \rangle$ where

- V is a finite set of variables
- $D \in V \rightarrow \text{Type}$ is a type application
- $I \subseteq V$ is the set of input variables
- $O \subseteq V$ is the set of output variables (with $I \cap O = \emptyset$)
- $P \in V \rightarrow \{\text{true}, \text{false}\}$ is a predicate over V

Translating mappings to constraints systems

Definition (Valid predicate)

$$\begin{array}{l} \textit{Pred} ::= \textit{Pred} \vee \textit{Pred} \\ | \textit{Pred} \wedge \textit{Pred} \\ | \neg \textit{Pred} \\ | \textit{Expr} = \textit{Expr} \\ | \textit{Expr} \neq \textit{Expr} \end{array}$$

Translating mappings to constraints systems

Definition

Let $C = \langle V, D, I, O, P \rangle$ be a constraints system such that
 $I = \{I_1, \dots, I_n\}$, $O = \{O_1, \dots, O_m\}$, $V \setminus (I \cup O) = \{N_1, \dots, N_l\}$.
 The equivalent mapping ψ of C noted $\psi \equiv C$ is the mapping from
 $\mathcal{P}(D(I_1) \times \dots \times D(I_n))$ to $\mathcal{P}(D(O_1) \times \dots \times D(O_m))$ defined by :

$$\begin{aligned} \langle o_1, \dots, o_m \rangle \in \psi(\langle i_1, \dots, i_n \rangle) \\ \Leftrightarrow \\ \exists n_1 \in N_1, \dots, n_l \in N_l \mid P(i_1, \dots, i_n, n_1, \dots, n_l, o_1, \dots, o_m) \end{aligned}$$

Translating mappings to constraints systems

Tuple

Let $f = N.\langle ex_1, \dots, ex_n \rangle$ be a tuple and X_1, \dots, X_m be the variables of the corresponding transition.

$\bar{f} \equiv \langle V, D, I, O, P \rangle$ with

- $V = \{in_1, \dots, in_m\} \cup \{out_1, \dots, out_n\}$
- $\forall i \in \{1, \dots, m\}, D(in_i) = C(X_i)$, and
 $\forall i \in \{1, \dots, n\}, D(out_i) = C(ex_i)$
- $I = \{in_1, \dots, in_m\}$
- $O = \{out_1, \dots, out_n\}$
- $P = \bigwedge_{i=1}^n out_i = ex_i$

Translating mappings to constraints systems

Example

$$\begin{aligned} & \overline{2.\langle f(W), Y + 1, 4, Z \rangle} \\ & \equiv \\ & W' = f(W) \wedge X' = Y + 1 \wedge Y' = 4 \wedge Z' = Z \end{aligned}$$

Translating mappings to constraints systems

Sum

Let $f \in \text{Bag}(S) \rightarrow \text{Bag}(S')$ and $g \in \text{Bag}(S) \rightarrow \text{Bag}(S')$

If $\bar{f} \equiv \langle V_f, D_f, I, O, P_f \rangle$ and $\bar{g} \equiv \langle V_g, D_g, I, O, P_g \rangle$ then
 $\overline{f + g} = \bar{f} + \bar{g} \equiv \langle V, D, I, O, P \rangle$ with

- $V = V_f \cup V_g$
- $\forall v \in V_f, D(v) = D_f(v)$, and $\forall v \in V_g, D(v) = D_g(v)$
- $P = P_f \vee P_g$

Translating mappings to constraints systems

Sum

Example

$$\overline{\langle X, Y, Z \rangle} + 2 \cdot \overline{\langle f(X), Y + 1, 4 \rangle}$$

\equiv

$$X' = X \wedge Y' = Y \wedge Z' = Z \vee X' = f(X) \wedge Y' = Y + 1 \wedge Z' = 4$$

Translating mappings to constraints systems

Transposition

Let $f \in \text{Bag}(S) \rightarrow \text{Bag}(S')$

If $\bar{f} \equiv \langle V, D, I, O, P \rangle$ then

$${}^t\bar{f} = {}^t\bar{f} \equiv \langle V, D, O, I, P \rangle$$

Translating mappings to constraints systems

Composition

Let $f \in \text{Bag}(S'') \rightarrow \text{Bag}(S')$ and $g \in \text{Bag}(S) \rightarrow \text{Bag}(S'')$

If $\bar{f} \equiv \langle V_f, D_f, I_f, O_f, P_f \rangle$ with $I_f = \{I_f^1, \dots, I_f^n\}$ and
 $\bar{g} \equiv \langle V_g, D_g, I_g, O_g, P_g \rangle$ with $O_g = \{O_g^1, \dots, O_g^n\}$ then
 $\overline{f \circ g} = \bar{f} \circ \bar{g} \equiv \langle V, D, I, O, P \rangle$ with

- $V = V_f \cup V_g$
- $\forall v \in V_f, D(v) = D_f(v)$, and $\forall v \in V_g, D(v) = D_g(v)$
- $I = I_g$
- $O = O_f$
- $P = P_f \wedge P_g \wedge_{i=1}^n I_f^i = O_g^i$

Translating mappings to constraints systems

Composition

Example

$$\overline{\langle X, Y \rangle \circ \langle W, 4 \rangle}$$

\equiv

$$X' = X \wedge Y' = Y \wedge W' = W \wedge Z' = 4 \wedge X = W' \wedge Y = Z'$$

Translating mappings to constraints systems

Difference

Let $f \in \text{Bag}(S) \rightarrow \text{Bag}(S')$ and $g \in \text{Bag}(S) \rightarrow \text{Bag}(S')$

If $\bar{f} \equiv \langle V_f, D_f, I, O, P_f \rangle$ and $\bar{g} \equiv \langle V_g, D_g, I, O, P_g \rangle$ then
If $\overline{f - g} = \bar{f} - \bar{g}$ then $\overline{f - g} \equiv \langle V, D, I, O, P \rangle$ with

- $V = V_f \cup V_g$
- $\forall v \in V_f, D(v) = D_f(v)$, and $\forall v \in V_g, D(v) = D_g(v)$
- $P = P_f \wedge \neg P_g$

Translating mappings to constraints systems

Difference

Example

$$\begin{aligned}
 & \overline{\langle X, 7 \rangle - \langle X, f(Y) \rangle} \\
 & \equiv \\
 & X' = X \wedge Y' = 7 \wedge \neg(X' = X \wedge Y' = f(Y)) \\
 & \equiv \\
 & X' = X \wedge Y' = 7 \wedge Y' \neq f(Y)
 \end{aligned}$$

Counter example

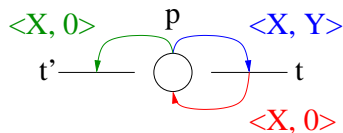
$ \begin{aligned} & \overline{2.\langle X \rangle - \langle X \rangle} \\ & = \\ & \overline{\langle X \rangle} \\ & \equiv \\ & X' = X \end{aligned} $	$ \begin{aligned} & \overline{2.\langle X \rangle - \langle X \rangle} \\ & = \\ & \emptyset \\ & \equiv \\ & \text{false} \end{aligned} $
---	--

Translating mappings to constraints systems

An example

Computation of the conflict relation

$$\frac{CSCO(t, t') =}{{}^tW^-(p, t') \circ W^-(p, t) - W^+(p, t)}$$

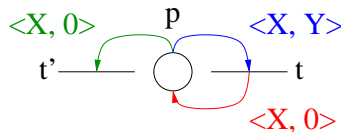


Translating mappings to constraints systems

An example

Computation of the conflict relation

$$\begin{aligned} & \overline{W^-(p, t)} \\ & \equiv \\ & X_p = X_t \wedge Y_p = Y_t \end{aligned}$$

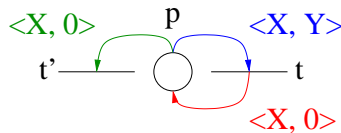


Translating mappings to constraints systems

An example

Computation of the conflict relation

$$\begin{aligned} & \overline{W^+(p, t)} \\ & \equiv \\ X_p = X_t \wedge Y_p = 0 \end{aligned}$$



Translating mappings to constraints systems

An example

Computation of the conflict relation

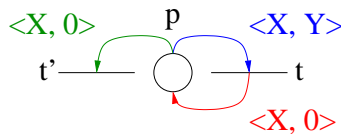
$$\overline{W^-(p, t) - W^+(p, t)}$$

$$\equiv$$

$$X_p = X_t \wedge Y_p = Y_t \wedge \neg(X_p = X_t \wedge Y_p = 0)$$

$$\equiv$$

$$X_p = X_t \wedge Y_p = Y_t \wedge Y_p \neq 0$$

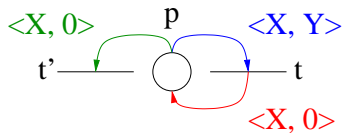


Translating mappings to constraints systems

An example

Computation of the conflict relation

$$\begin{aligned} & \overline{{}^tW^-(p, t')} \\ & \equiv \\ & X_{t'} = X_p \wedge Y_p = 0 \end{aligned}$$

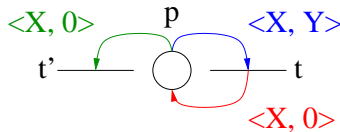


Translating mappings to constraints systems

An example

Computation of the conflict relation

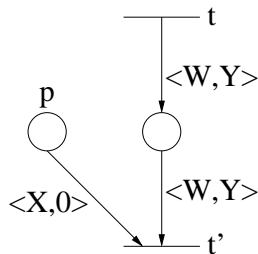
$$\begin{aligned}
 & \overline{{}^tW^-(p, t')} \circ \overline{W^-(p, t) - W^+(p, t)} \\
 & \equiv \\
 & X_{t'} = X_p \wedge Y_p = 0 \\
 & \quad \wedge \\
 & X_p = X_t \wedge Y_p = Y_t \wedge Y_p \neq 0 \\
 & \equiv \\
 & \textit{false}
 \end{aligned}$$



Translating mappings to constraints systems

Limits of the approach

The causality relation indicates that an instance $[t, (W = w, Y = y)]$ can potentially enable the firing of an instance $[t', (W = w, X = ?, Y = y)]$. However, we have to check p to complete the assignment.



Outline

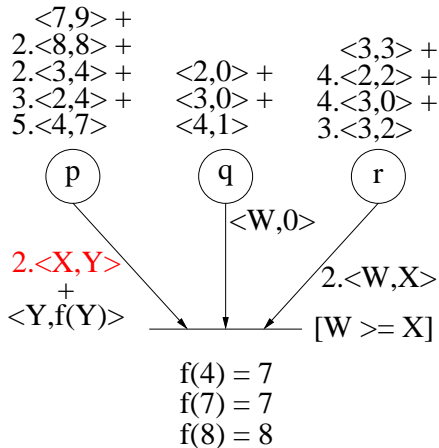
- 1 The enabling test problem
- 2 Conflict and Causality relations
 - Definition
 - Implementation
- 3 **Unification algorithm**
 - Principle
 - Scheduling of the input tuples of a transition
- 4 Conclusions

Unification algorithm for a transition

- Start the unification process by leaving all the variables undefined.
- Treat input tuples of the transition in a predefined static order.
- For each tuple we iterate on all the tokens present in the corresponding place. If a token match the tuple, we unify the variables which appear in the tuple, and
 - if it is the last tuple to treat we add the assignment to the set of enabled bindings
 - else we treat the next tuple
- The guard is treated as soon as all the variables which appear in it have been unified. If it is evaluated to *false* the assignment is discarded, else the algorithm can pursue normally.

An example

- Before :
 $X = *, Y = *, W = *$
- After :
 $X = 8, Y = 8, W = *$
 $X = 3, Y = 4, W = *$
 $X = 2, Y = 4, W = *$
 $X = 4, Y = 7, W = *$



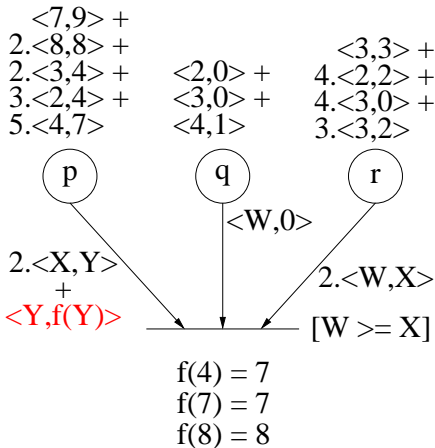
An example

- Before :

$X = 8, Y = 8, W = *$
 $X = 3, Y = 4, W = *$
 $X = 2, Y = 4, W = *$
 $X = 4, Y = 7, W = *$

- After :

$X = 3, Y = 4, W = *$
 $X = 2, Y = 4, W = *$



An example

- Before :

$X = 3, Y = 4, W = *$

$X = 2, Y = 4, W = *$

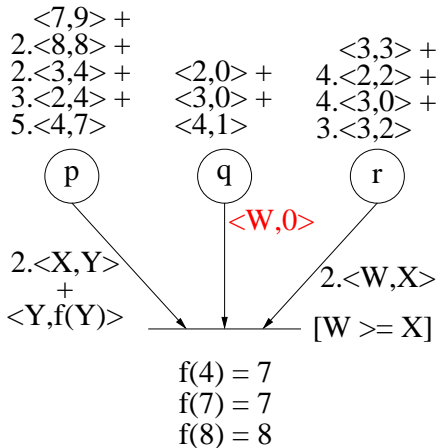
- After :

$X = 3, Y = 4, W = 2$

$X = 2, Y = 4, W = 2$

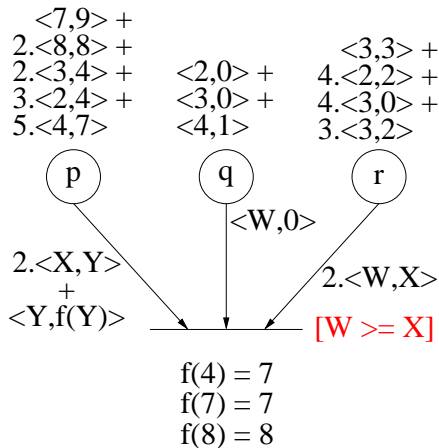
$X = 3, Y = 4, W = 3$

$X = 2, Y = 4, W = 3$



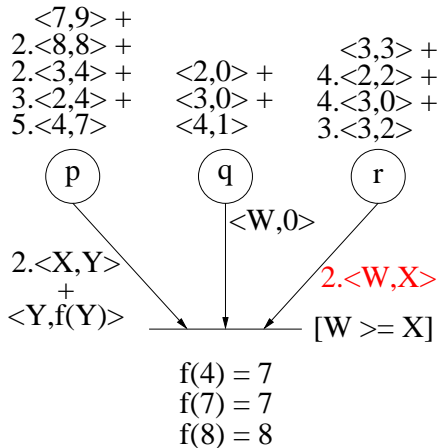
An example

- Before :
 - $X = 3, Y = 4, W = 2$
 - $X = 2, Y = 4, W = 2$
 - $X = 3, Y = 4, W = 3$
 - $X = 2, Y = 4, W = 3$
- After :
 - $X = 2, Y = 4, W = 2$
 - $X = 3, Y = 4, W = 3$
 - $X = 2, Y = 4, W = 3$



An example

- Before :
 $X = 2, Y = 4, W = 2$
 $X = 3, Y = 4, W = 3$
 $X = 2, Y = 4, W = 3$
- After :
 $X = 2, Y = 4, W = 2$
 $X = 2, Y = 4, W = 3$



Optimizing the search : scheduling the tuples

- The efficiency of the unification algorithm highly depends on the order in which tuples are treated.
- We use Makela's idea : use a cost function to estimate the consequence of a tuples scheduling on the search tree.
- Our cost function has two objectives :
 - ① minimize the size of the search tree
 - ② try to limitate repetitive application of user defined mappings
- For each transition, the scheduling which minimizes the cost function is selected prior the search.
- During the search algorithm, tuples are always processed according this scheduling.

Principles of our cost function

- 1 Tuples in which constant appear should be considered as early as possible.
- 2 If a variable is already unified it should be considered as a constant.
- 3 When all the variables appearing in the tuples are already unified the unification is equivalent to the search of an item in a set. By using balanced tree, this search has a logarithmic complexity.
- 4 Scheduling which make the guard evaluatable earlier should be prioritized.
- 5 If all the parameters which appear in a function call are unified variables the call can be made a single time for all the tokens in the place.

Principles of our cost function

- 1 Tuples in which constant appear should be considered as early as possible.
- 2 If a variable is already unified it should be considered as a constant.
- 3 When all the variables appearing in the tuples are already unified the unification is equivalent to the search of an item in a set. By using balanced tree, this search has a logarithmic complexity.
- 4 Scheduling which make the guard evaluatable earlier should be prioritized.
- 5 If all the parameters which appear in a function call are unified variables the call can be made a single time for all the tokens in the place.

Principles of our cost function

- 1 Tuples in which constant appear should be considered as early as possible.
- 2 If a variable is already unified it should be considered as a constant.
- 3 When all the variables appearing in the tuples are already unified the unification is equivalent to the search of an item in a set. By using balanced tree, this search has a logarithmic complexity.
- 4 Scheduling which make the guard evaluatable earlier should be prioritized.
- 5 If all the parameters which appear in a function call are unified variables the call can be made a single time for all the tokens in the place.

Principles of our cost function

- 1 Tuples in which constant appear should be considered as early as possible.
- 2 If a variable is already unified it should be considered as a constant.
- 3 When all the variables appearing in the tuples are already unified the unification is equivalent to the search of an item in a set. By using balanced tree, this search has a logarithmic complexity.
- 4 Scheduling which make the guard evaluatable earlier should be prioritized.
- 5 If all the parameters which appear in a function call are unified variables the call can be made a single time for all the tokens in the place.

Principles of our cost function

- 1 Tuples in which constant appear should be considered as early as possible.
- 2 If a variable is already unified it should be considered as a constant.
- 3 When all the variables appearing in the tuples are already unified the unification is equivalent to the search of an item in a set. By using balanced tree, this search has a logarithmic complexity.
- 4 Scheduling which make the guard evaluatable earlier should be prioritized.
- 5 If all the parameters which appear in a function call are unified variables the call can be made a single time for all the tokens in the place.




Outline

- 1 The enabling test problem
- 2 Conflict and Causality relations
 - Definition
 - Implementation
- 3 Unification algorithm
 - Principle
 - Scheduling of the input tuples of a transition
- 4 Conclusions

Conclusions

The algorithm we proposed has two main components.

- 1 It implements the relations of conflict and causality between transitions to efficiently manage a set of enabled transitions. This set is updated by the resolution of some constraints systems built before the search algorithm.
- 2 Incomplete transitions bindings are completed by the unification algorithm which treat tuples in a predefined order. This order is computed by a cost function which try to minimize
 - 1 the size of the search tree
 - 2 the calls to user defined mappings which may slow the unification process

-  Dutheillet C. and Haddad S.
Conflict sets in colored petri nets.
In 5th Int. Workshop on Petri Nets and Performance Models, Toulouse (F) 19.-22., pages 76–85, 1993.
-  Mäkelä M.
Optimising enabling tests and unfoldings of algebraic system nets.
In ICATPN 2001, number 2075 in LNCS, pages 283–302. Springer, 2001.
-  Evangelista S.
Syntactical rules for colored petri nets manipulation.
Technical report, CNAM / Cedric, <http://cedric.cnam.fr>, 2004.