High-level Petri Nets - Concepts, Definitions and Graphical Notation

Committee Draft ISO/IEC 15909
October 2, 1997
Version 3.4

Editor's Foreword

Previous drafts of this standard have been discussed in WG11 meetings during 1995 and 1996, and the Working Draft was circulated to SC7 member bodies in February 1997. This draft incorporates comments from the WG11 meeting in Walnut Creek in June 1997 which considered member body comments and those of Petri net experts in the wider community. This document is being circulated as a first CD to SC7 member bodies for comment and vote. The next meeting of WG11 will be in London, 3-7 November 1997. Although the deadline for the ballot will be after this meeting, it would be valuable to obtain comments from member bodies before the meeting if at all possible. It would therefore be appreciated if comments could be forwarded to the editor by 20 October 1997.

National Bodies should send comments to the SC7 secretariat by email to sc7@QC.Bell.ca. WG11 experts may send their comments to the WG11 secretary, Mr Tony Williamson, at WILLIAMSONJA%AM5@mr.nawcad.navy.mil. It would be appreciated if these comments could also be copied to the editor, Jonathan Billington, at j.billington@unisa.edu.au.

In formulating their vote, member bodies may wish to consider the symbols being used in the definition (clause 7). Currently the standard has used the symbol S for the set of places in a high-level net. This is because it originally came from the German stellen. Several countries would prefer to use P for the set of places, as it is more suggestive in a standard written in English. If P was used, this would allow S to be used for the set of Sorts (instead of R currently). The symbol F is used for the set of arcs, as historically, it has been called the flow relation. The use of A for the set of arcs is more suggestive. The symbol C is used for the Typing function as historically it was called the 'Colour' function. It would be more suggestive to use $Type$ instead. The greek letter Sigma has been used for signatures, but it is also used for summation. Replacing Sigma by Sig for a signature, would avoid this overloading. A greek letter (omega) has been used for the set of operators, and use of O is probably preferred. The use of f for functions is probably better than what is currently used (an italic w subscripted by H). F can be used for the

set of functions. Script C is used to denote a set of Types, because historically it was a set of colour sets. It may be more suggestive to use Dom or D for a set of domains.

Jonathan Billington
Editor Project 1.7.19.3 Petri nets
Email: j.billington@unisa.edu.au

Contents

0 Introduction

Software is an increasingly important component of many large scale systems that will be developed as we mature as a global society in the information age. Currently the development of software as an engineering discipline is in its infancy. Techniques for improving software quality and reliability are thus being sought by governments and industry.

This International Standard aims to improve this situation by providing a well defined graphical technique for the specification and analysis of systems. The technique, High-level Petri nets, is mathematically defined, and may thus be used to provide unambiguous specifications and descriptions of applications. It is also an executable technique, allowing specification prototypes to be developed to test ideas at the earliest and cheapest opportunity. Specifications written in the technique may be subjected to analysis methods to prove properties about the specifications, before implementation commences, thus saving on testing and maintenance time.

The technique promises to have multiple uses. For example it may be used to define the semantics of data flow diagrams or directly to specify systems. The technique is particularly suited to parallel and distributed systems development as it supports concurrency.

This standard may be cited in contracts for the supply of software services, or used by application developers or Petri net tool vendors or users.

This International Standard provides an abstract mathematical syntax and a formal semantics for the technique. Conformance to the standard will be possible if a mathematical mapping is provided from the conformant technique to the standard technique, and the standard's semantics are adopted.

Clause 1 describes the scope, areas of application and the intended audience of this International Standard. Clause 2 provides normative references (none at present), while clause 3 provides a glossary of terms and defines abbreviations. The main mathematical apparatus required for defining the standard is developed in clause 4. The basic semantic model for High-level Petri Nets is given in clause 5, while the main concepts behind the graphical form are informally introduced in clause 6. Clause 7 defines the High-level Petri Net Graph, the form of the standard intended for industrial use. Clause 8 further describes syntactical conventions. Clause 9 relates the graphical form to the basic semantic model. The conformance clause is given in clause 10. Several informative annexes are provided: Annex A is a tutorial on the High-level Petri Net Graph; Annex B (not written yet) defines certain net classes as restrictions of the definition of Clause 7; and Annex C provides pointers to analysis techniques for High-level Petri Nets. A bibliography concludes the standard.

1 Scope

1.1 Intent

This International Standard defines a Petri net technique, called High-level Petri nets, including its syntax and semantics. It provides a reference definition that can be used both within and between organisations, to ensure a common understanding of: the technique; and the specifications written using the technique. The standard may be mandated in contracts for writing specifications, thereby facilitating international enterprize and trade. The standard will also facilitate the development and interoperability of Petri net computer support tools.

This International Standard, defines a mathematical semantic model, an abstract mathematical syntax and a graphical notation for High-level Petri nets.

This International Standard does not provide a concrete syntax nor a transfer syntax and it does not address techniques for modularity (such as hierarchies), augmentation of high-level Petri nets with time, and methods for analysis which may become the subject of future standardisation efforts.

1.2 Field of Application

This International Standard is applicable to a wide variety of concurrent discrete event systems and in particular distributed systems. Generic fields of application include:

- Requirements analysis;
- Development of specifications, designs and test suites;
- Descriptions of existing systems prior to re-engineering;
- Modelling business and software processes;
- Providing the semantics for concurrent languages;
- Simulation of systems to increase confidence;
- Formal analysis of the behaviour critical systems; and
- Development of Petri net support tools

The standard may be applied to a broad range of systems, including information systems, operating systems, databases, communication protocols, computer hardware architectures, security systems, manufacturing systems, defence command and control, business processes, banking systems, chemical processes, nuclear waste systems and telecommunications.

1.3 Audience

The standard is written as a reference for systems analysts, designers, developers, maintainers and procurers, and for Petri net tool developers and standards developers.

2 Normative References

None currently.

Editor's note 1: There was a suggestion to reference CD ISO/IEC 14481 Information Technology - Conceptual Schema Modelling Facilities (CSMF), in this clause. Unfortunately, as far as I understand it, the most useful part of CSMF is Annex D, Mathematical Conventions and Notation, which is informative, not normative. Hence referencing CSMF as Normative is not appropriate, but it could be included in the references in the bibliography (although text books (such as Truss) on discrete mathematics may be more suitable).

3 Glossary of Terms and Abbreviations

3.1 Glossary

Note: For notions of basic set theory including sets, functions, relations and λ expressions, see the book by Truss in the Bibliography.

Algebra: A mathematical structure comprising a set of sets, and a set of functions taking these sets as domains and co-domains.

Many-sorted Algebra: An Algebra in which the cardinality of the set of sets is greater than one.

Arc: A directed edge of a net which may connect a place to a transition or a transition to a place. Normally represented by an arrow.

Input Arc (of a transition): An arc directed from a place to the transition.

Output Arc (of a transition): An arc directed from the transition to a place.

Arc annotation: An expression that may involve constants, variables and operators used to annotate an arc of a net. The expression must evaluate (on variable substitution) to be a multiset over the type of the arc's associated place.

Arity: The input sorts and output sort for an operator.

Assignment: For a set of variables, the association of a value (of correct type) to each variable.

Basis set: The set of objects used to create a multiset.

Binding: see assignment

Carrier: A set of a many-sorted algebra.

Declarations: A set of statements which define the sets, constants, parameter values, typed variables and functions required for defining the inscriptions on a high-level net graph.

Enabling (a transition): A transition is enabled in a particular mode and net marking, when the following conditions are met:

The marking of each input place of the transition satisfies the demand placed on it by its arc expression evaluated for the particular transition mode. (The arc expression must evaluate to a multiset of tokens of the same type or subtype as the place.) The demand is satisfied when the place's marking contains (at least) the multiset of tokens indicated by the evaluated arc expression.

Remark: The determination of transition modes guarantees that the Transition Condition is satisfied (see Transition Mode).

High-level Net (High-level Petri Net): A algebraic structure comprising: a set of places; a set of transitions; a set of types; a function associating types to places, and modes (types) to transitions; *Pre* function determining token demands (multisets of tokens) on places for each transition mode; *Post* function determining output tokens (multisets of tokens) for places for each transition mode; and an initial marking.

High-level Petri Net Graph: A net graph and its associated annotations comprising Place Types, Arc Annotations, Transition Conditions, and their corresponding definitions in a set of Declarations, and an Initial Marking of the net.

Marking (of a net): The set of the place markings for all places of the net.

Initial Marking (of the net): The set of initial place markings given with the high-level net definition.

Initial Marking of a Place: A special marking of a place, defined with the high-level net.

Marking of a place: A multiset of tokens associated with ('residing in') the place.

Reachable Marking: Any marking of the net that can be reached from the initial marking by the occurrence of transitions.

Reachability Set: The set of reachable markings of the net, including the initial marking.

Multiset: A collection of objects where (meaningful) repetition of objects is allowed.

Multiset cardinality: The cardinality of a multiset, is the sum of the multiplicities of each of the members of the multiset.

Net graph: A directed graph comprising a set of nodes of two different kinds, called places and transitions, and their interconnection by directed edges, such that only places can be connected to transitions, and transitions to places, but never transitions to transitions, nor places to places.

Node (of a net): A vertex of the net graph.

Operator: A symbol representing the name of a function.

Parameter: A constant that can take a range of values defined by a set.

Parameterized High-level Net Graph: A high-level net graph that contains parameters in its definition.

Place: A node of a net, taken from the place kind, normally represented by an ellipse in the net graph. A place is typed.

Input Place (of a transition): A place connected to the transition by an input arc.

Output Place (of a transition): A place connected to the transition by an output arc.

Place Type: A non-empty set of data items associated with a place. (This set can describe an arbitrarily complex data structure.)

Reachability Graph: A directed graph of nodes and edges, where the nodes correspond to reachable markings, and the edges correspond to transition occurrences.

Signature: An algebraic Structure comprising a set of sorts and a set of operators.

Boolean signature: A signature where one of the sorts is Boolean.

Many-sorted signature: A signature where the set of sorts has a cardinality greater than one.

Natural signature: A signature where one of the sorts corresponds to the Natural numbers.

Signature with Variables: A signature that includes a set of variable names, as well as the set of sorts and the set of operators.

Sort: A symbol representing the name of a set.

Argument sort: The sort of an argument of an operator.

Input sort: the same an argument sort

Output sort: The sort of an output of an operator.

Range sort: the same as an output sort

Term: An expression built from a signature and comprising constants variables and operators.

Closed term: A term comprising constants and operators but no variables.

Term evaluation: The result obtained after the binding of variables in the term, the computation of the results of the associated functions, and their reduction to simplest form.

Token: A data item associated with a place and chosen from the place's type.

Transition: A node of a net, taken from the transition kind, normally represented by a rectangle in the net graph.

Transition Condition: A boolean expression (one that evaluates to true or false) associated with a transition.

Transition Mode: an assignment of values to the transition's variables that satisfies the transition condition.

Transition occurrence (Transition rule): If a transition is enabled in a mode, it may occur in that mode. On the occurrence of the transition, the following actions occur indivisibly:

1. For each input place of the transition: the enabling tokens of the input arc with respect to that mode are subtracted from the input place's marking, and

2. For each output place of the transition: the multiset of tokens of the evaluated output arc expression is added to the marking of the output place.

Remark: A place may be both an input place and an output place of the same transition.

Transition Variables: All the variables that occur in the expressions associated with the transition. These are the transition condition, and the annotations of arcs surrounding the transition.

3.2 Abbreviations

HLPN: High-level Petri Net

HLPNG: High-level Petri Net Graph

4 Conventions and Notation

This clause defines the mathematical conventions required for the definition of High-level Petri nets.

4.1 Sets

- $N = \{0, 1, \ldots\}$ the natural numbers.
- $Z = \{\ldots, -1, 0, 1, \ldots\}$ the integers.
- $Boolean = \{true, false\}$

4.2 Multisets

A *multiset*, B, (also known as a bag) over a non-empty *basis* set, A, is the function

$$B : A \longrightarrow N$$

which associates a multiplicity, possibly zero, with each of the basis elements. The multiplicity of $a \in A$ in B, is given by $B(a)$. A set is a special case of a multiset, where the multiplicity of each of the basis elements is either zero or one.

The set of multisets over A is denoted by μA.

4.2.1 Sum representation

A multiset may be represented as a symbolic sum of basis elements scaled by their multiplicities (sometimes known as co-efficients).

$$B = \sum_{a \in A} B(a)a$$

If A is the set of rational numbers, then parentheses are used to separate the multiplicity $B(a)$ from the basis element a. If $B(a) = 1$, then it may be omitted and just a used in the sum.

4.2.2 Membership

Given a multiset, $B \in \mu A$, $a \in A$ is a member of B, denoted $a \in B$, if $B(a) > 0$, and conversely if $B(a) = 0$, then $a \notin B$.

4.2.3 Empty multiset

The empty multiset, $\emptyset$, has no members: $\forall a \in A$, $\emptyset(a) = 0$.

4.2.4 Cardinality and Finite Multiset

Multiset cardinality is defined in the following way. The cardinality $|B|$ of a multiset B, is the sum of the multiplicities of each of the members of the multiset.

$$|B| = \sum_{a \in A} B(a)$$

when $|B|$ is finite, the multiset is called a finite multiset.

4.2.5 Multiset Equality and Comparison

Two multisets, $B1, B2 \in \mu A$, are equal, $B1 = B2$, *iff* $\forall a \in A$, $B1(a) = B2(a)$.
$B1$ is less than or equal to (or contained in) $B2$, $B1 \leq B2$, *iff* $\forall a \in A$, $B1(a) \leq B2(a)$.

4.2.6 Multiset Operations

Addition and subtraction operations on multisets, $B1, B2 \in \mu A$, are defined as follows:

$B = B1 + B2 \quad iff \quad \forall a \in A \; B(a) = B1(a) + B2(a)$
$B = B1 - B2 \quad iff \quad \forall a \in A \; (B1(a) \geq B2(a)) \wedge (B(a) = B1(a) - B2(a))$

Scalar multiplication of a multiset, $B1 \in \mu A$, by a natural number, $n \in N$, is defined as

$B = nB1$ *iff* $\forall a \in A, B(a) = n \times B1(a)$

where $\times$ is arithmetic multiplication.

4.3 Concepts from Algebraic Specification

Editor's note 2: This clause does not as yet take into account the technical comments provided by Germany in the subdivision vote, due to word processing incompatibilities. However, it is believed that the current text is adequate as it stands.

In order to define the HLPN-graph we need concepts from algebraic specification. In the HLPN-graph, we shall inscribe arcs with multisets of terms involving variables, and transitions with Boolean expressions. Many-sorted signatures provide an appropriate mathematical framework for this representation. Signatures provide a convenient way to characterise many-sorted algebras at a syntactic level. This clause introduces the concepts of signatures, terms and many-sorted algebras that will be required for the definition of the HLPN-graph.

4.3.1 Signatures

Editor's Note 3: There is a suggestion to merge this and the next subclause, to form a new subclause entitled Signatures with Variables, to remove redundancy.

A *many-sorted* (or *R-sorted*) signature, Σ, is a pair:

$$\Sigma = (R, \Omega)$$

where

- R is a set of sorts (the **names** of sets, e.g. Int for the integers); and
- Ω is a set of operators (the **names** of functions) together with their *arity* in R which specifies the names of the domain and co-domain of each of the operators.

The arity is a function from the set of operator names to $R^* \times R$, where R^* is the set of finite sequences, including the empty string, ε, over R. Thus every operator in Ω is indexed by a pair (σ, r), $\sigma \in R^*$ and $r \in R$ denoted by $w_{(\sigma,r)}$. $\sigma \in R^*$ is known as the *input* or *argument* sorts, and r as the *output* or *range* sort of operator w. (The sequence of input sorts will define a cartesian product as the domain of the function corresponding to the operator and the output sort will define its co-domain - but this is jumping ahead to the many-sorted algebra.)

For example, if $R = \{Int, Bool\}$, then $w_{(Int.Int,Bool)}$ would represent a binary predicate symbol, such as *equality* $(=)$ or *less than* $(<)$. Using a standard convention, the sort of a constant may be declared by letting $\sigma = \varepsilon$. For example an integer constant would be denoted by $w_{(\varepsilon,Int)}$ or simply w_{Int}.

The sort of a variable may also be declared in the same way. This leads to the consideration of signatures with variables.

4.3.2 Signatures with Variables

A many-sorted signature with variables is the triple:

$$\Sigma = (R, \Omega, V)$$

where R is a set of sorts, Ω a set of operators with associated arity as before and V is a set of sorted variables, known as an *R-sorted set of variables*. It is assumed that R, Ω and V are disjoint. The sort of the variable is defined by the arity function, in a similar way to that of constants, from the set of variable names to $\{\varepsilon\} \times R$. A variable in V of sort $r \in R$ would be denoted by $v_{(\varepsilon,r)}$ or more simply by v_r. For example, if $Int \in R$, then an integer variable would be $v_{(\varepsilon,Int)}$ or v_{Int}.

V may be partitioned according to sorts, where V_r denotes the set of variables of sort r (i.e. $v_a \in V_r$ *iff* $a = r$).

4.3.3 Natural and Boolean Signatures

The term *Boolean Signature* is used to mean a many-sorted signature where one of the sorts is Boolean. Similarly, the term *Natural Signature* is used when one of the sorts corresponds to the Naturals (N).

4.3.4 Terms of a Signature with Variables

Terms of sort $r \in R$ may be built from a signature $\Sigma = (R, \Omega, V)$ in the following way. We denote a term, e, of sort r by $e : r$ and the set of terms of sort r by $TERM(\Omega \cup V)_r$, and generate them inductively as follows. For $r, r_1, \ldots, r_n \in R$ $(n > 0)$

1. For all $w_{(\varepsilon,r)} \in \Omega$, $w_{(\varepsilon,r)} \in TERM(\Omega \cup V)_r$;
2. $V_r \subseteq TERM(\Omega \cup V)_r$; and
3. If $e_1 : r_1, \ldots, e_n : r_n$ are terms and $w_{(r_1 \ldots r_n, r)} \in \Omega$, is an operator, then $w_{(r_1 \ldots r_n, r)}(e_1, \ldots, e_n) \in TERM(\Omega \cup V)_r$

Thus if Int is a sort, integer constants and variables, and operators (with appropriate arguments) of output sort Int are terms of sort Int.

We denote the set of all terms of a signature with variables by $TERM(\Omega \cup V)$, the set of all *closed* terms (those not containing variables, also known as *ground* terms) by $TERM(\Omega)$. Thus

$$TERM(\Omega \cup V) = \bigcup_{r \in R} TERM(\Omega \cup V)_r$$

4.3.5 Multisets of Terms

Multisets or *bags* of terms can also be built inductively from the signature if we assume that we have a Natural signature. We define multisets of terms this way to allow the multiplicities to be terms of sort Nat, rather than just the Naturals themselves. (This allows, for example, the introduction of conditions into arc expressions.)

Let $BTERM(\Omega \cup V)$ denote the set of multisets of terms, defined inductively as follows.

- $TERM(\Omega \cup V) \subset BTERM(\Omega \cup V)$;

- if $b1, b2 \in BTERM(\Omega \cup V)$, then $(b1 + b2) \in BTERM(\Omega \cup V)$; and
- if $i \in TERM(\Omega \cup V)_{Nat}$ and $b \in BTERM(\Omega \cup V)$, then $i \times b \in BTERM(\Omega \cup V)$ where '$\times$' represents scalar multiplication.

Where there is no confusion the '$\times$' will be dropped and juxtaposition will be used for scalar multiplication (e.g. '$3 \times x$ can be replaced by $3x$ and $4 \times 3 \times x$ by $4 \times 3x$ which is distinctly different from $43x$). When the multiplicity of a term is one (eg $1x$) we drop the one, so that $1x$ is represented by x. Likewise we interpret any term that does not have a multiplicity to have a multiplicity of one.

4.3.6 Many-sorted Algebras

A many-sorted algebra, (or Σ-Algebra), H, provides an interpretation (meaning) for the signature Σ. For every sort, $r \in R$, there is a corresponding set, H_r, known as a *carrier* and for every operator $w_{(r_1 \ldots r_n, r)} \in \Omega$, there is a corresponding function

$$w_H : H_{r_1} \times \ldots \times H_{r_n} \rightarrow H_r.$$

In case an operator is a constant, w_r, then there is a corresponding element $w_H \in H_r$. They may be considered as functions of arity zero.

Definition: A many-sorted Algebra, H, is a pair

$$H = (R_H, \Omega_H)$$

where $R_H = \{H_r | r \in R\}$ is the set of carriers, with for all $r \in R, H_r \neq \emptyset$ and
$\Omega_H = \{w_H | w_{\sigma,r} \in \Omega, \sigma \in R^*$and $r \in R\}$ the set of corresponding functions.

For example, if $\Sigma = (\{Int, Bool\}, \{<_{(Int.Int, Bool)}\})$ then a corresponding many-sorted algebra would be

$$H = (Z, Boolean; lessthan)$$

where Z is the set of integers: $\{\ldots, -1, 0, 1, \ldots\}$
$Boolean = \{true, false\}$
and $lessthan : Z \times Z \rightarrow Boolean$ is the usual integer comparison function.

It could also be

$$\mathcal{B} = (N, Boolean; lessthan)$$

where N is the set of non-negative integers: $\{0, 1, \ldots\}$
$Boolean = \{true, false\}$
and $lessthan : N \times N \rightarrow Boolean$.

For signatures with variables, variables are R-sorted. In the algebra, the variable is typed by the carrier corresponding to the sort.

4.3.7 Assignment and Evaluation

Given an R-sorted algebra, H, with variables in V, an *assignment* [1] for H and V is a set of functions α, comprising an assignment function for each sort $r \in R$,

$$\alpha_r : V_r \to H_r.$$

This function may be extended to terms by considering the family of functions $assign$ comprising

$$assign_r : TERM(\Omega \cup V)_r \to H_r$$

for each sort $r \in R$. The values are determined inductively as follows. For $\sigma \in R^* \setminus \{\varepsilon\}$, $\sigma = r_1 r_2 \ldots r_n$, with $r, r_1, \ldots, r_n \in R$ and $e, e_1, \ldots, e_n \in TERM(\Omega \cup V)$,

- If $e \in V_r$ is a variable, then $assign_r(e) = \alpha_r(e)$
- For a constant, $w_r \in \Omega$, $assign_r(w_r) = w_H \in H_r$.
- If $e = w_{(\sigma,r)}(e_1, \ldots, e_n)$, then
  $assign_r(e) = w_H(assign_{r_1}(e_1), \ldots, assign_{r_n}(e_n)) \in H_r$, where $e_1 : r_1 \ldots e_n : r_n$.

Knowing the values of terms we can determine the value of multisets of terms by expanding the multiset into a sum of scaled terms and evaluating each scalar and term for a particular assignment to variables. This is defined inductively as follows for $a \in TERM(\Omega \cup V)$, $i \in TERM(\Omega \cup V)_{Nat}$ and $b1, b2 \in BTERM(\Omega \cup V)$

- $Val_\alpha(i \times a) = assign(i) \times assign(a)$
- $Val_\alpha(b1 + b2) = Val_\alpha(b1) + Val_\alpha(b2)$

5 Semantic Model for High-level Petri Nets

This clause provides the basic semantic model for High-level nets.

5.1 Definition

A **HLP-net** is a structure $HLPN = (S, T, \mathcal{C}; C, Pre, Post, M_0)$ where

- S is a finite set of elements called Places
- T is a finite set of elements called Transitions disjoint from S $(S \cap T = \emptyset)$
- $\mathcal{C}$ is a non-empty finite set of types
- $C : S \cup T \longrightarrow \mathcal{C}$ is a function used to type places and determine transition modes

[1] The terms *binding* and *valuation* are also used in this context.

- $Pre, Post : TRANS \longrightarrow \mu PLACE$ are the pre and post mappings with

$$TRANS = \{(t, m) \mid t \in T, m \in C(t)\}$$

$$PLACE = \{(s, g) \mid s \in S, g \in C(s)\}$$

- $M_0 \in \mu PLACE$ is a multiset known as the initial marking of the net

5.2 Marking of HLP-net

A **Marking** of the HLP-net is a multiset, $M \in \mu PLACE$.

5.3 Enabling of Transition Modes

A finite multiset of transition modes, $T_\mu \in \mu TRANS$, is *enabled* at a marking M *iff*

$$Pre(T_\mu) \leq M$$

where the linear extension of Pre is given by

$$Pre(T_\mu) = \sum_{tr \in TRANS} mult(tr, T_\mu) Pre(tr)$$

.

Thus a multiset of transition modes is enabled if there are enough tokens on the input places to satisfy the linear combination of the pre maps for each transition mode in T_μ.

5.4 Transition Rule

Given that a multiset of transition modes, T_μ, is enabled at a marking M, then a *step* may occur resulting in a new marking M' given by

$$M' = M - Pre(T_\mu) + Post(T_\mu).$$

where the linear extension of $Post$ is used.

A step is denoted by $M[T_\mu\rangle M'$ or $M \xrightarrow{T_\mu} M'$.

6 Concepts Required for the High-level Petri Net Graph

This clause introduces the concepts that are needed in the definition of the High-level Petri net graph. Readers interested in a tutorial exposition on High-level Petri nets are referred to Annex B.

6.1 High-level Petri Net Graph components

A High-level Petri net graph comprises:

- *A Net Graph*, consisting of sets of nodes of two different kinds, known as *places* and *transitions*, and *arcs* connecting places to transitions, and transitions to places.
- *Place Types*. Non-empty sets. One type is associated with each place.
- *Place Marking*. A collection of elements (data items) chosen from the place's type and associated with the place. Repetition of items is possible. The items associated with places are called *tokens*.
- *Arc Annotations*: Arcs are inscribed with expressions which may comprise constants, variables and function images (eg $f(x)$). The expressions are evaluated by substituting values for the variables. When an arc's expression is evaluated, it must result in a collection of items taken from the arc's place's type. The collection may have repetitions.
- *Transition Condition*: A boolean expression (eg $x < y$) inscribing a transition.
- *Declarations*: comprising definitions of Place Types, typing of variables, and function definitions.

Note: A collection of items which allows repetitions is known in mathematics as a multiset.

6.2 Net execution

HLPN-graphs are executable, allowing the flow of tokens around the net to be visualised. This can illustrate flow of control and flow of data within the same model. Key concepts governing this execution are *enabling* of transitions and the *occurrence* of transitions defined by the *Transition Rule*.

6.2.1 Enabling

A transition is enabled with respect to a *net marking*. A net marking comprises the set of all place markings of the net.

A transition is also enabled in a particular *transition mode*. A transition mode is an assignment or substitution of values for the *transition's variables*, that satisfies the transition condition (ie the transition condition is true). The transition's variables are all those variables that occur in the expressions associated with the transition. These are the transition condition, and the annotations of arcs involving the transition.

Enabling a transition involves the marking of its *input places*. An input place of a transition is a place which is connected to the transition by an arc leading from that place to the transition. An arc that leads from an input place to a transition is called an *input arc* of the transition.

A transition is enabled in a specific mode, for a particular net marking. Each input arc expression is evaluated for the transition mode, resulting in a multiset of tokens of the same type as that of the input place. If each input place's marking contains at least its input arc's multiset of tokens (resulting from the evaluation of the input arc's expression in the specific mode), then the transition is enabled in that mode.

An example is given in subclause 6.3.

The input arc's multiset of tokens resulting from the evaluation of the input arc's expression in a specific mode is called the input arc's *enabling tokens*, with respect to that mode.

Two transition modes are *concurrently enabled* for a particular marking, if for the associated transitions, each input place's marking contains at least the sum of the enabling tokens (with respect to both modes) of each input arc associated with that input place.

6.2.2 Transition Rule

Single Transition Mode

Enabled transitions can *occur*. When a transition occurs, tokens are removed from its input places, and tokens are added to its *output places*. An output place of a transition is a place which is connected to the transition by an arc directed from the transition to the place. An arc that leads from a transition to a place (an output place of the transition) is called an *output arc* of the transition.

If a transition is enabled in a mode, it may occur in that mode. On the occurrence of the transition in a specific mode, the following actions occur atomically:

1. For each input place of the transition: the enabling tokens of the input arc with respect to that mode are subtracted from the input place's marking, and

2. For each output place of the transition: the multiset of tokens, resulting from the evaluation of the output arc expression for the mode, is added to the marking of the output place.

Remark: A place may be both an input place and an output place of the same transition.

Step of Concurrently enabled Transition Modes

Several concurrently enabled transition modes may occur in one *step*, that is in one atomic action. The change to the marking of the net when a step occurs is given by the sum of all the changes that occur for each transition mode, as described above.

An example is given in the next subclause.

6.3 Examples

6.3.1 Simple Example

A simple example of an HLPN graph is given in figure 1.

Declarations

$A = \{1, 2, 3, 4\}$
$B = \{3, 4, 5, 7\}$
$<: Z \times Z \rightarrow Boolean$ arithmetic 'less than'
$x : A, y : B$

Graph

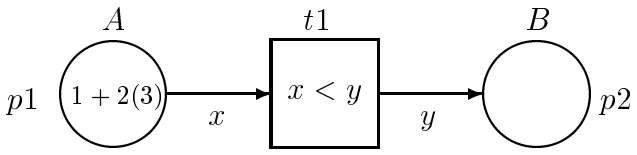


Figure 1: HLPN-Graph with a Transition Condition

This example comprises two places, named $p1$ and $p2$, one transition, $t1$, and arcs from $p1$ to $t1$, and $t1$ to $p2$. The declarations define two types, A and B that are different subsets of the positive integers. Variable x is of type A, and variable y is of type B. The transition is inscribed with the boolean expression $x < y$, where the *less than* operator is defined in the declarations. Arc $(p1, t1)$ is annotated with the variable x, while arc $(t1, p2)$ is annotated with y.

Place $p1$ is typed by A and has an initial marking $1 + 2(3)$, representing the multiset $M_0(p1) = \{(1, 1), (2, 0), (3, 2), (4, 0)\}$. Place $p2$ is typed by B, and is empty representing the empty multiset, $M_0(p2) = \emptyset$.

In the initial marking, $t1$ can be enabled in the following modes

$\{(1, 3), (1, 4), (1, 5), (1, 7), (3, 4), (3, 5), (3, 7)\}$

where the first element of each pair represents a substitution for x, and the second, a substitution for y which satisfies $x < y$.

It can be seen that the multiset of modes, $(1, 3)+2(3, 5)$ are concurrently enabled. Another example of the concurrent enabling of modes is the multiset $(1, 5) + (3, 4)$ and yet another is $(1, 7) + (3, 6) + (3, 7)$.

If transition $t1$ occurred in mode (3,5), then the resultant marking would be:

$M(p1) = \{(1, 1), (2, 0), (3, 1), (4, 0)\}$
$M(p2) = \{(3, 0), (4, 0), (5, 1), (7, 0)\}$.

Alternatively, if the multiset of modes $(1, 3) + 2(3, 5)$ occurred concurrently, the resultant marking would be

$M(p1) = \emptyset$
$M(p2) = \{(3,1),(4,0),(5,2),(7,0)\}$.

6.3.2 Conditionals in Arc Expressions, and Parameters

The example uses a variant of the readers/writers problem to illustrate many of the features of a HLPN graph including the use of conditionals in arc expressions.

A number (N) of agents (processes) wish to access a shared resource (such as a file). Access can be in one of two modes: shared (s), where up to L agents may have access at the same time (e.g. reading); and exclusive (e), where only one agent may have access (e.g. writing). No assumptions are made regarding scheduling. An HLPN graph model is given in figure 2. This illustrates the use of two parameters, L and N, both of which are positive integers. This is therefore a parameterized HLPNG, which represents infinitely many readers/writers systems. Each instantiation of N and L would produce a HLPNG, which could then be executed.

It has been assumed that the initial state is when all the agents are waiting to gain access to the shared resource (with no queueing discipline assumed). In this example, the initial markings are given in the declaration. Place *Wait* is marked with all agents; the *Control* place contains L ordinary tokens and *Access* is empty. An agent can obtain access in one of two modes: if shared (m=s), then a single token is removed from *Control* (as m=e is false) when *enter* occurs in a single mode; if exclusive (m=e), then all L tokens are removed preventing further access until the resource is released (transition *Leave*). Shared access is limited to a maximum of L agents as transition *enter* is disabled when *Control* is empty.

Outfix notation has been used for the function $Bool \rightarrow \{0,1\}$ and this will be used as a standard convention. It is assumed that integer addition and subtraction and the equality predicate are primitive and do not need to be defined in the Declaration.

7 Definition of the High-level Petri Net Graph

7.1 Introduction

High-level Petri nets can be defined in a number of ways. Clause 5 provides the definition of the basic mathematical semantic model. The basic semantic model is not what is used by practitioners. This clause provides a formal definition for the graphical form of high-level nets. This approach is taken, as it is the graphical form of HL nets that is most appropriate for industrial use. We will refer to the graphical form as a High-level Petri net graph (HLPN-graph). It provides a mathematical syntax for inscribing the graphical elements. The concepts of marking, enabling and transition rule are also formally defined.

7.2 Definition

A **HLPN-graph** is a structure

Declarations

Set of Agents:A = $\{a_1, \ldots, a_N\}$
Set of Access Modes:M = {s,e}
Control: C = {•}
Positive integer constants: N,L
Variables x:A ; m:M
Function [ ]:*Bool* → {0, 1} where
$[true] = 1$ and $[false] = 0$
M_0(Wait) =A
M_0(Control) =L<•>
M_0(Access) = ∅

Graph

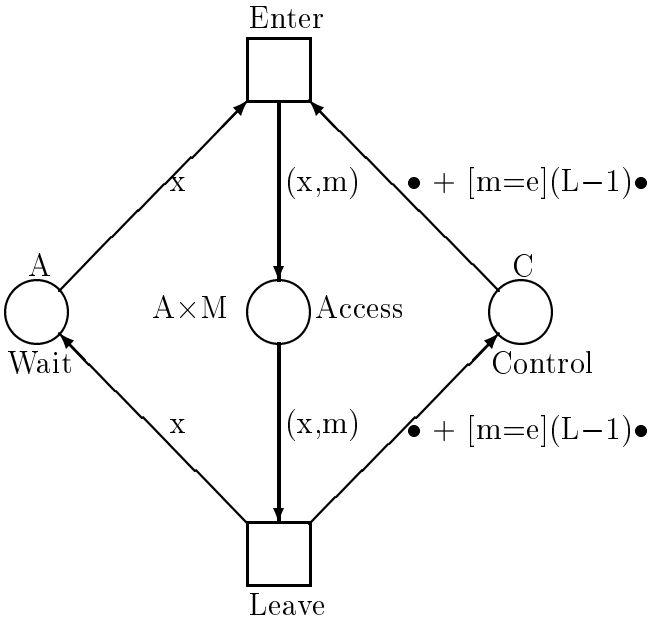


Figure 2: HLPN Graph of Resource Management

$$\text{HLPNG} = (NG, \Sigma, C, AN, M_0)$$

where

- $NG = (S, T; F)$ is called a net graph, with
  - S a finite set of nodes, called Places;
  - T a finite set of nodes, called Transitions, disjoint from S ($S \cap T = \emptyset$); and
  - $F \subseteq (S \times T) \cup (T \times S)$ a set of arcs, known as the flow relation;
- $\Sigma = (R, \Omega, V)$ is a Natural-Boolean signature with variables. It has a corresponding Σ-Algebra, $H = (R_H, \Omega_H)$.
- $C : S \to R_H$ is a function which types places.
- $AN = (A, TC)$ is a pair of net annotations.
  - $A : F \to BTERM(\Omega \cup V)$ such that for $C(s) = H_r$ and for all $(s,t), (t',s) \in F$, $A(s,t), A(t',s) \in BTERM(\Omega \cup V)_r$. A is a function that annotates arcs with a multiset of terms of the same sort as the carrier associated with the arc's place.
  - $TC : T \to TERM(\Omega \cup V)_{Bool}$ is a function that annotates transitions with Boolean expressions.
- $M_0 : S \to \bigcup_{s \in S} \mu C(s)$ such that $\forall s \in S,\ M_0(s) \in \mu C(s)$, is the initial marking function which associates a multiset of tokens of correct type which each place.

In summary, a HLPN-graph consists of a net graph where the arcs are annotated by multisets of terms. The multiplicities of the multisets are non-negative integer terms. Transitions are annotated by Boolean terms. The terms are built from a Natural-Boolean signature which has an associated many-sorted algebra. A typing function associates a carrier of the many-sorted algebra with each place. A place can only hold tokens of the same type as the place and hence the initial marking is a multiset over the place's type.

Editor's Note 4: In defining HLPN-graphs, we have intentionally associated a type with each place. This type is a carrier of the chosen many-sorted algebra, H. This allows us to specify concrete systems where the sets and functions have already been determined.

There is also a need for a more abstract or syntactic form that allows classes of systems to be specified. In this case, the places become R-sorted, i.e. a sort (rather than a type) is associated with each place. This leads us to the notion of a HLPN-graph schema which may be considered for future standardisation.

Editor's Note 5: When generating multisets of terms for the arc inscriptions, we allow the multiplicities to be natural number terms, so that the value can depend on the values of variables and operators of other types. This allows there to be conditionals in arc expressions. There may be other useful extensions.

7.3 Marking

We define a marking, M, of the HLPN-graph in the same way as the initial marking.

$M : S \rightarrow \bigcup_{s \in S} \mu C(s)$ such that for all $s \in S$, $M(s) \in \mu C(s)$.

7.4 Enabling

A transition $t \in T$ is enabled in a Marking, M, for a particular assignment to its variables, α, that satisfies the transition condition, $assign_{bool}(TC(t)) = true$, known as a *mode* of t, *iff*

$$\forall s \in S \quad Val_\alpha(\overline{s,t}) \leq M(s)$$

where for $(u, v) \in (S \times T) \cup (T \times S)$,

- $\overline{u,v} = A(u, v)$, for $(u, v) \in F$,
- $\overline{u,v} = \Phi$, for $(u, v) \notin F$

and $Val_\alpha(\Phi) = \emptyset$, the empty multiset.

7.5 Transition Rule

If $t \in T$ is enabled in *mode* α, for marking M, t may *occur in mode* α. When t occurs in mode α, the marking of the net is transformed to a new marking M', denoted $M[t, \alpha\rangle M'$, according to the following rule:

$$\forall s \in S \quad M'(s) = M(s) - Val_\alpha(\overline{s,t}) + Val_\alpha(\overline{t,s})$$

Editor's Note 6: Do we need to include concurrent enabling and step?

8 Notation for High-level Petri Net Graphs

8.1 General

The graphical form comprises two parts: a *Graph* which represents the net elements graphically and carries textual inscriptions; and a *Declaration*, defining all the types, variables, constants and functions that are used to annotate the Graph part. The declaration may also include the initial marking and the typing function if these cannot be inscribed on the graph part due to lack of space. There needs to be a visual association between an inscription and the net element to which it belongs.

The width, colour and patterns of the lines used to draw the graph are not mandated by this standard.

8.2 Places

Places are represented by ellipses (often circles). ◯

Three annotations are associated with a place s:

- the place name;
- the name of the type ($C(s)$) associated with the place; and
- the initial marking, $M_0(s)$.

A mechanism must be provided to remove any ambiguity regarding the association of these annotations with the correct place.

If the initial marking is empty, then it may be omitted.

8.3 Transitions

A Transition is represented by a rectangle and is annotated by a name and a boolean expression, the *Transition Condition*. If the Transition Condition is true ($TC(t) = true$), it may be omitted.

For example,

$\boxed{x < y}$ $t1$

represents a transition with a name $t1$, and a transition condition, $x < y$, where both variables, x and y, and the operator less than, $<$, are defined in the declarations.

A mechanism must be provided to remove any ambiguity regarding the association of these annotations with the correct transition.

8.4 Arcs

An arc is represented by an arrow: ⟶

For $(s,t) \in F$, an arrow is drawn from place s to transition t and vice versa for $(t,s) \in F$. If (s,t) and (t,s) have the same annotations (s is a side place of t), $A(s,t) = A(t,s)$, then this may be shown by a single arc with an arrowhead at both ends and annotated by a single inscription.

Arcs are annotated with multisets of terms. Multisets are represented by the symbolic sum representation defined in the Conventions (clause 4.2.1). In order to distinguish multiplicities from terms, the convention is adopted that terms are enclosed in parentheses.

8.5 Markings and Tokens

A token is a member of $\bigcup_{s \in S} C(s)$. A Marking of the net may be shown graphically by annotating a place with its multiset of tokens $M(s)$ using the symbolic sum representation.

Parentheses should be used to distinguish token multiplicities (Natural numbers) from token values (e.g. Integers) when required.

9 Semantics of HLPN Graph

The HLPN-graph may be given an interpretation as a HLP-net (see clause 5) in the following way.

1. Places: S is the set of places in the HLP-net.

2. Transitions: T is the set of transitions in the HLP-net.

3. Set of Types: The set of modes for a transition is determined by the types of the variables occurring in the surrounding arc annotations restricted by its transition condition.

   Let there be n_t free variables associated with the arcs surrounding a transition $t \in T$. Let these have names $v_{r_1}(t), \ldots, v_{r_{n_t}}(t) \in V$. In the Σ-Algebra, H, for all $i \in \{1, 2, \ldots, n_t\}$, let the carrier corresponding to r_i, H_{r_i}, be denoted by G_i with typed variables $v_i(t) : G_i$. For all i, let $g_i \in G_i$, then

   $$C(t) = \{(g_1, \ldots, g_{n_t}) \mid TC'(t)\} \text{ where}$$

   $TC'(t) = \lambda(v_1(t), \ldots, v_{n_t}(t)).TC(t)(g_1, \ldots, g_{n_t})$.
   Tuples which satisfy $TC(t)$ are included in $C(t)$. (The λ-expression provides a means for formally substituting values for the variables in the Transition Condition.)

   The types of places are obtained directly from the HLPN graph definition. Thus the set of types is given by $\mathcal{C} = \{C(x) | x \in S \cup T\}$.

4. The Typing Function: The typing function restricted to places is defined in the HLPN-graph and $C(t)$ is given above.

5. Pre and Post Maps.

   The pre and post maps are given, for all $(s,t), (t,s) \in F$, by the following family of mappings from $C(t)$ into $\mu C(s)$

   $$Pre(s,t) = \lambda(v_1(t), \ldots, v_{n_t}(t)).A(s,t)$$

   $$Post(s,t) = \lambda(v_1(t), \ldots, v_{n_t}(t)).A(t,s)$$

   For $(s,t) \notin F$ and $\forall m \in C(t)$, $Pre(s,t;m) = \emptyset$ and for $(t,s) \notin F$ and $\forall m \in C(t)$, $Post(s,t;m) = \emptyset$.

   Thus for all $t \in T$ and for all $m \in C(t)$

   $$Pre(t,m) = \{(s,b) \mid s \in S, b \in Pre(s,t;m)\}$$

   $$Post(t,m) = \{(s,b) \mid s \in S, b \in Post(s,t;m)\}$$

6. Initial Marking.

   For all $s \in S$, $M_0(s)$ is as defined in the HLPN-graph.

10 Conformance

Conformance to this International Standard may be at several levels.

10.1 Level 1 Conformance

To claim Level 1 conformance to this International Standard an implementation shall demonstrate that it has the semantics defined in clause 5, by providing a mapping from the implementation's syntax to the semantic model in a similar way to that defined in clause 9.

10.2 Level 2 Conformance

To claim Level 2 conformance to this International Standard an implementation shall have satisfied the requirements of Level 1 conformance and in addition shall provide a mapping from the implementation's syntax to that of the HLPNG defined in clause 7.

10.3 Level 3 Conformance

To claim Level 3 conformance to this International Standard an implementation shall have satisfied the requirements of Level 1 conformance and in addition shall adopt the syntax of the HLPNG defined in section 7 and the notational conventions of clause 8.

Annex A
(informative)

Tutorial

Editor's Note 7: Proposals for further examples in the tutorial have been received (Jensen, Lilius), by the editor.

A.1 Introduction

High level Petri net graphs (HLPNGs) are used to model discrete event systems. A discrete event system comprises

- collections of real or abstract objects and
- discrete actions which
  - modify or consume objects from some collections and
  - create objects in other collections.

The created objects may be related to objects that are consumed. It is assumed that the collections considered have some permanent identity, irrespective of varying contents. Take, for example, the collection of coins in someone's purse, or a data base. Generally, several instances of the same object can be contained in a collection.

A.2 Net Graphs

In HLPNGs, an action is modelled by a *transition*, which is graphically represented by a rectangle. A collection is modelled by a *place*, which is graphically represented by a circle or an ellipse. Places and transitions are called the *nodes* of a *net graph*. Arrows, called *arcs*, show which places a transition operates on. Each arc connects a place and a transition in one direction. Arcs never connect a place with a place nor a transition with a transition. The graphical representation of a net graph is shown in figure 3.

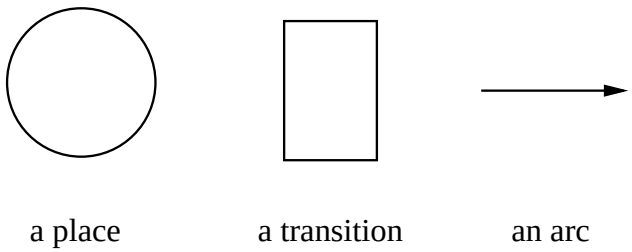


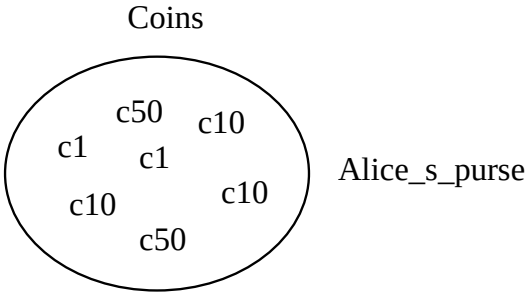
Figure 3: Graphic conventions.

A.2.1 Places and tokens

The objects of the system are modelled by (arbitrarily complex) data items called *tokens*. Tokens reside in places. The contents (i.e. the tokens) of a place is called the *marking* of the place. The tokens form a collection (known in mathematics as a multiset) i.e. several instances of the same token can reside in the place. A *marking* of a net consists of the markings of each place.

Example_A in figure 4 consists of a single place, Alice_s_purse, which models that Alice's purse contains two 1-cent, three 10-cent and two 50-cent coins. The set of coins is defined in a textual part of the HLPNG called the Declarations.

The place, Alice_s_purse, is typed by the set, Coins. This means that only coins (belonging to Coins) can reside in Alice's purse. In this example, the tokens correspond to coins.



Coins = {c1, c2, c5, c10, c20, c50}

Figure 4: *Example_A*.

Example_A is a net graph. It has neither transitions nor arcs. As no actions are modelled, nothing ever happens and nothing ever changes in this system.

When a particular instance of a HLPNG is defined, each place is defined with a special marking, called the *initial marking*, because other markings will usually evolve, once a net is executed. As a place can be marked with a large number of tokens, the initial marking may be declared textually instead of pictorially. Thus, Alice's present coin collection can be written as the initial marking,

$M_0(\text{Alice\_s\_purse}) = 2c1 + 3c10 + 2c50$

and the net graph is then drawn (admittedly in a less illustrative way) without tokens.

A.2.2 Transitions

Example_B in figure 5 models the dripping of a tap. Transition *drip* can always happen, any number of times. *Example_B* is also a net, even though it has neither places nor arcs.

A.2.3 Arcs

An arc from a place to a transition indicates that this transition consumes objects from the place. An arc in the opposite direction indicates that this transition produces tokens

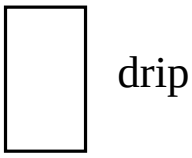
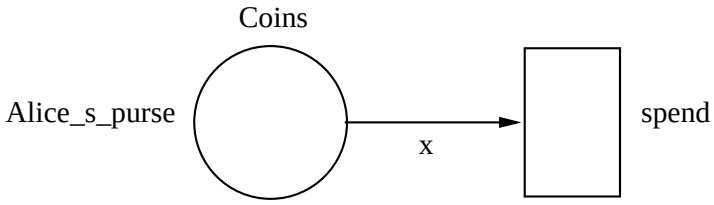


Figure 5: *Example_B.*

on the place. In figure 6, *Example_C*, Alice has a smaller coin collection. She may spend any number of coins at a time.



Coins = {c1, c2, c5, c10, c20, c50}

x : Coins

M_0(Alice_s_purse) = c10 + 2c50

Figure 6: *Example_C.*

Arc annotations determine the kinds and numbers of tokens that are produced or consumed. Here, the annotation "x" indicates that any coin (from Alice's purse) can be spent. However, it has to be declared in the textual part of *Example_C* that "x" denotes a variable for coins. Alice could spend: a ten cent coin; a fifty cent coin; a ten cent and a fifty cent coin; two fifty cent coins; and all her coins in one transaction, that is by the occurrence of transition spend.

Editor's Note 8: Need to include key concepts of occurrence modes, evaluation of arc expressions, enabling, concurrency and transition rule in the above discussion ?

A.2.4 The net graph

The size and position of the nodes, as well as the size and shape of the arcs, though often important for readability, are irrelevant to the mathematical description of a net, i.e. the places, transitions, and arcs of the net, the *net graph*. Informally, one might say, the net has one place, called Alice_s_purse, one transition, spend, and one arc from Alice_s_purse to spend. Formally this can be expressed as:

S={Alice_s_purse}

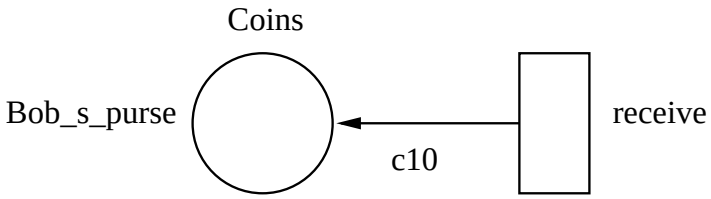
T={spend}

F={(Alice_s_purse, spend)}

Traditionally, S denotes the set of places (the second most popular name being P), T denotes the set of transitions, and F denotes the set of arcs. These letters are the initials of the German words *Stelle, Transition, Flussrelation.* Each arc is thus described as the pair consisting of its origin node and its target node.

A.3 Transition conditions

Example_D in figure 7 models that Bob starts with an empty purse and collects 10-cent coins.

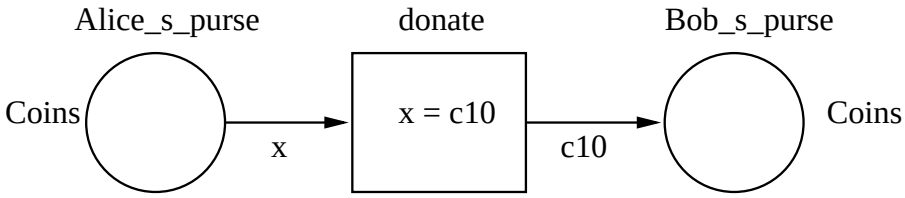


Coins = {c1, c2, c5, c10, c50}
M_0(Bob_s_purse) = $\emptyset$

Figure 7: *Example_D.*

Example_D does not model where the coins may come from. It only shows what happens to Bob's purse as a consequence of an arbitrary number of occurrences of receive.

In the next example, depicted in figure 8, Alice is ready to give Bob any of her coins. Bob, however, accepts only 10c coins from Alice.

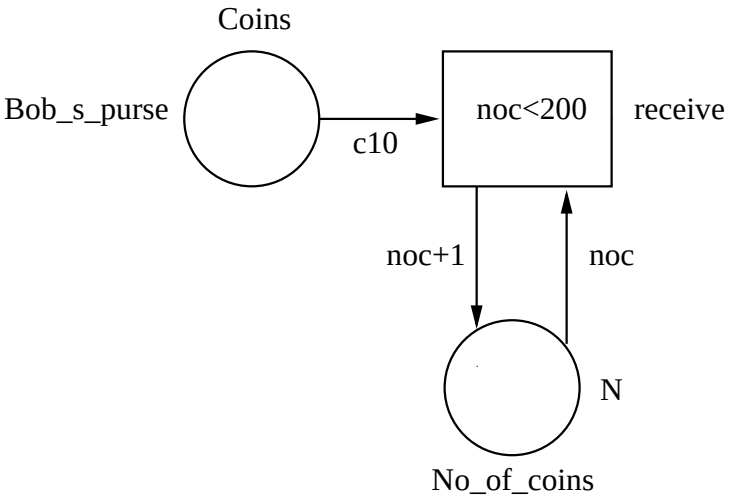


Coins = {c1, c2, c5, c10, c50}
x : Coins
M_0(Alice_s_purse) = c10 + 2c50
M_0(Bob_s_purse) = $\emptyset$

Figure 8: *Example_E.*

Now we have added a *transition condition*, requiring that x=c10. The transition **donate** can occur only with such variable values as fulfill the condition. If there are no appropriate,

i.e. c10, tokens in Alice_s_purse, then **donate** cannot occur. In a more realistic variant of *Example_D*, Bob cannot put arbitrarily many coins into his purse. *Example_F* in figure 9 limits the number of 10 cent coins that Bob can receive to 200.



Declarations

N set of natural numbers
Coins = {c1, c2, c5, c10, c50}
noc : N
< :N×N→ *Bool* usual "less than" predicate
+ :N×N→N arithmetic addition

M_0(Bob_s_purse) = ∅
M_0(No_of_coins) = 0

Figure 9: *Example_F.*

A.4 Net Dynamics

Example_C can be used to illustrate the modelling of system dynamics in HLPNs. In her first action, Alice can spend any number of her coins. Let her spend a 10-cent coin. Then in the net, spend occurs, with x having the value c10. This *occurrence* of spend is denoted by (spend, {(x,c10)}). This indicates which transition occurs (spend), and to which value each of the variables, appearing in the arc annotations around the transition, is bound. In this case only x appears, and is bound to c10.

By the occurrence (spend, {(x,c10)}) a new marking of the net is created: M_1(Alice_s_purse) = 2c50.

The new marking is called a *reachable marking* of *Example_C*. A different marking would be reached by Alice spending a fifty cent coin. As long as Alice's purse contains coins, she can spend any of them. In the net, as long as *Alice_s_purse* is marked with a non-empty multiset of tokens, spend can occur with x bound to any one of the tokens in the marking of Alice_s_purse. Markings reachable from reachable markings are also called reachable.

The dynamics of *Example_ C*, i.e.

- the markings reachable in *Example_C*, as well as
- the transition occurrences performed to reach each one,

are depicted in the *reachability graph* in figure 10.

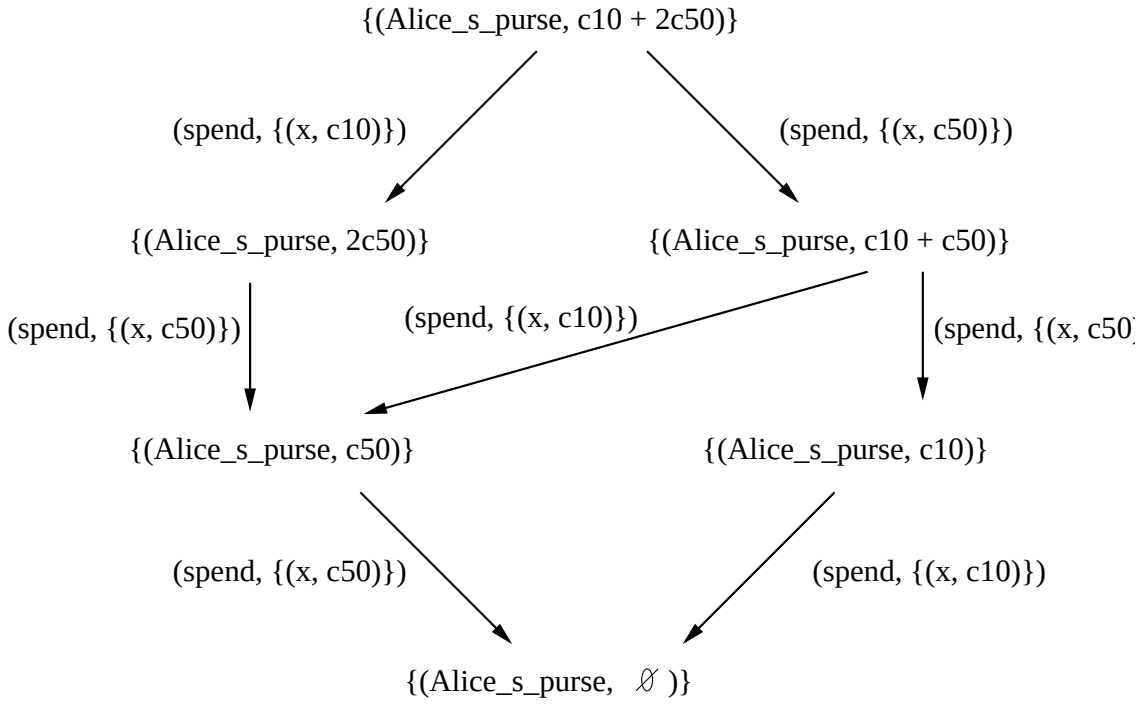


Figure 10: Reachability graph of *Example_C*.

From this diagram, one can read, for example, the following facts about the dynamics of *Example_C*:

- If Alice spends first 10 cents and then 50 cents, or if she does it in the reverse order, then she will have 50 cents left.
- Alice can perform at most three actions.
- Every sequence of 3 actions ends with an empty purse.
- No sequence of actions (save, trivially, the empty sequence) will allow Alice to restore the contents of her purse.

All of this holds, of course, only within the range of actions considered in *Example_C*. In the reachability graph, set braces {...} are written around the pair parentheses (...) wherever usually an entire set appears in this position. Generally there are sets of place markings and sets of variable bindings to be described. It just happens that in our very simple example these are one-element sets and the {(...)} looks unnecessarily complicated.

The reachability graph for *Example_D* consists of a single infinite chain, as indicated in figure 11. The occurrence (receive, $\emptyset$) does not mean that Bob receives no coins, but that no variable is assigned a value.

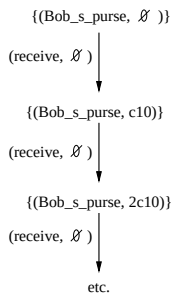


Figure 11: Reachability graph of *Example_D*.

A.5 A larger example: flow control

Two companies, A_Co and B_Co, reside in different cities. A_Co packs and sends big crates of equal size to B_Co, one by one. B_Co has a room where the crates are stored. Crates may be taken from the store-room for processing (for example, distributed to retailers, or opened and the contents consumed - it does not matter here). This procedure is modelled in figure 12.

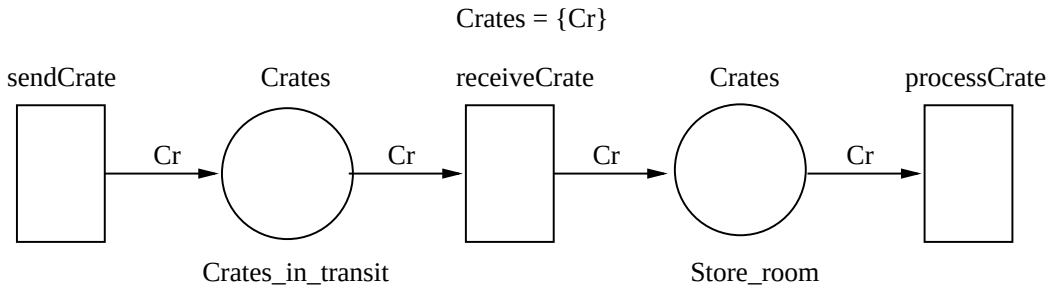


Figure 12: Crate procedure.

The people from B_Co have a problem. The store-room of B_Co can only hold a certain number, say, MAX, of these crates. In order to avoid being forced either to leave crates in the street or to rent another store-room, B_Co agrees with A_Co on a "flow control protocol".

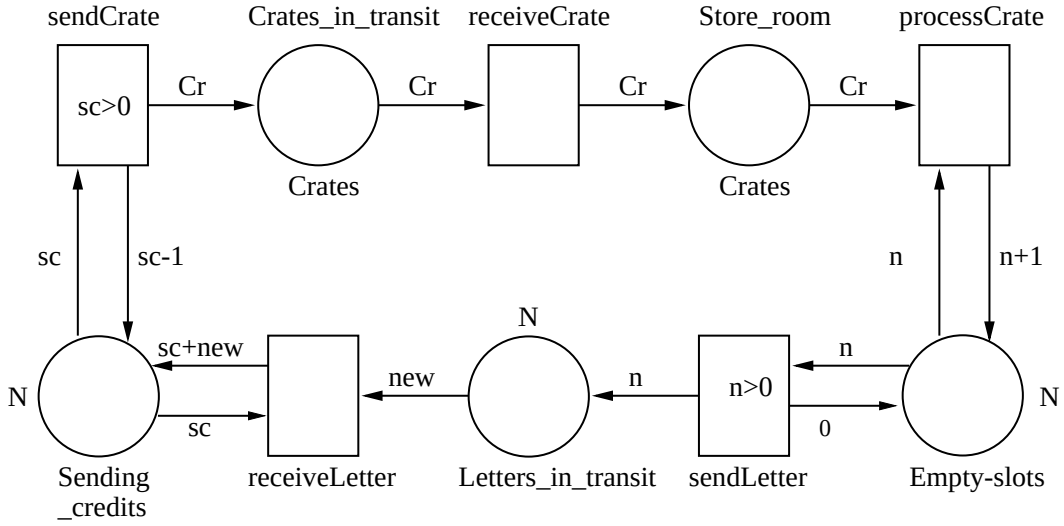
To implement the protocol, A_Co keeps a record of *SendingCredits*, while B_Co keeps a record of empty "slots" available for placing crates in the store. Any time that there are empty slots, B_Co may give the number of empty slots as sending credits for crates to

A_Co. B_Co does this by sending a letter with this number and setting the number of empty slots to 0. Whenever A_Co receives such a letter, it increases *SendingCredits* by the number written in it.

Sending a crate, which is only possible if *SendingCredits* is 1 or more, lowers *SendingCredits* by 1, and processing a crate raises the number of empty slots by one.

Initially, the situation is as follows: no crate or letter is in transit; the store-room is empty; there is no sending credit; and all slots are empty.

This distributed system is modelled by figure 13.



$$M_0(\text{Crates\_in\_transit}) = \text{Mo}(\text{Letters\_in\_transit}) = \text{Mo}(\text{Store\_room}) = \emptyset$$
$$M_0(\text{SendingCredits}) = 0$$
$$M_0(\text{Empty-slots}) = \text{MAX}$$

Declarations

Crates = {Cr}
N = {0, 1, 2, ...}
Z is the set of integers
n, new, sc : N
MAX : N
$+ : Z \times Z \to Z$ is arithmetic addition
$- : Z \times Z \to Z$ is arithmetic substraction

Figure 13: Example_G.

Note that this net models infinitely many different systems. It is a *parameterized* HLPNG with a parameter, MAX, that may take any natural number as a value. Each such value *val*, substituted for MAX instantiates *Example_G* to an "ordinary" HLPNG without parameters,*Example_G(val)*.

Annex B
(informative)

Net Classes

Editor's Note 9:

It is intended to define various classes of nets here. In particular there is interest in defining Petri nets (Place/Transition systems) as a subclass of the HLPNG. Other subclasses may include Elementary Net systems and other high-level nets.

Annex C
(informative)

Analysis Techniques

There are a large number of analysis techniques for Petri nets, including reachability analysis (in many forms), structural analysis and invariants analysis that may be used to investigate properties of systems modelled by nets. This annex is just to alert readers of this standard to these possibilities. The Petri net community plan to publish a 3 volume set in the Lecture Notes in Computer Science series as a handbook on Petri nets in 1998. This will be based on lectures given at a two week advanced course in Petri nets held in Germany in 1996. Readers are also referred to the bibliography, for example, the second volume of Kurt Jensen's book on Coloured Petri Nets.

Bibliography

1. J. L. Peterson, "Petri Net Theory and the Modeling of Systems", Prentice-Hall, N.J., 1981.

2. W. Reisig, "Petri Nets, An Introduction", EATCS, Monographs on Theoretical Computer Science, W.Brauer, G. Rozenberg, A. Salomaa (Eds.), Springer Verlag, Berlin, 1985.

3. T. Murata, "Petri nets: properties, analysis and applications", Proc IEEE, Volume 77, No. 4, pp. 541-580, 1989.

4. B. Baumgarten, "Petri-Netze, Grunlagen und Anwendungen", Wissenschaftsverlag, Mannheim, 1990.

5. K. Jensen, "Coloured Petri Nets", Volume 1: Basic Concepts, Springer-Verlag 1992.

6. K. Jensen, "Coloured Petri Nets", Volume 2: Analysis Methods, Springer-Verlag 1994.

7. K. Jensen, "Coloured Petri Nets", Volume 3: Practical Use, Springer-Verlag 1997.

8. J. Desel and J. Esparza, "Free Choice Petri Nets", Cambridge Tracts in Theoretical Computer Science 40, Cambridge University Press, 1995

9. R. David and H. Alla, "Petri nets and Grafcet", Prentice Hall, 1992.

10. A. A. Desrochers and R.Y. Al'Jaar, "Applications of Petri nets in Manufacturing Systems: Modelling, Control and Performance Analysis", IEEE Press 1995.

11. Advanced Course on Petri Nets, Bad Honnef, West Germany, September 1986. Published in 'Advances' series, LNCS Vols 254, 255, 1987.

12. E. Best and C. Fernandez, "Notations and Terminology on Petri Net Theory", Arbeitspapiere der GMD 195, March 1987.

13. J. Billington, "Extensions to Coloured Petri Nets", Proceedings of the Third International Workshop on Petri Nets and Performance Models, Kyoto, Japan, 11-13 December, 1989, pp. 61-70.

14. J. Billington, "Many-sorted High-level Nets", invited paper in Proceedings of the Third International Workshop on Petri Nets and Performance Models, Kyoto, Japan, 11-13 December, 1989, pp. 166-179, also reprinted in K. Jensen, G. Rozenberg (Eds.) *High-Level Petri Nets: Theory and Application*, Springer-Verlag, 1991.

15. J. Billington, "Extensions to Coloured Petri Nets and their Application to Protocols", University of Cambridge Computer Laboratory Technical Report No. 222, May 1991.

16. W. Reisig, "Petri nets and algebraic specifications", TCS, Vol. 80, pp. 1-34, May, 1991.

17. J.K. Truss, "Discrete Mathematics for Computer Scientists", Addison-Wesley, 1991.