Formal Models for Distributed Negotiations

Petri Nets

Place/Transition Petri nets [Petri 1962]
- Well-known model of concurrency
  - Theory / Applications
- Well-supported by a large community
  - Academy / Industry
- Well-developed tools
  - http://www.daimi.au.dk/PetriNets
- A media for conveying ideas to non-expert
  - Suggestive graphical presentation
Foundational Model of Concurrency

- Basic monoidal structure of states
  - A graph whose set of nodes is a monoid
  - Precursor of most modern models and calculi of concurrency
- Monoidal structure of computations
  - Essentially multiset rewriting
- Framework for studying issues of
  - Causality, concurrency, conflict
  - Event structures and domain
  - Deadlock, liveness, boundedness
  - Reachability, coverability, invariants

General-purpose Model of (Concurrent) Computation

- Algebraic representations of
  - Data, states, transitions, steps, computations
- Basic rewriting system later generalized by
  - Term rewriting, graph rewriting, term-graph rewriting, concurrent constraint models
- A framework where to mix concurrency with other features
  - Data types, time, probability, dynamic reconfiguration, read-without-consuming, negative preconditions, objects,
- Semantic framework for encoding other models and languages
  - Useful in studying and comparing expressiveness issues
Practical Specification Language

- Standard theory and notation
- Exploited in many heterogeneous areas
- Simple and natural graphical representation
- Supported by tools of “industrial quality”
- System design
- Refinement

Terminology and Notation

- Places a,b,c,... classes of resources
- Transitions t,t’,... basic activities
- Tokens • instances of a resource class
- Markings u,v,w,... multisets of resources
  - Multiset union \( u \oplus w \)
  - Empty marking \( \emptyset \)
  - Places are unary markings a
  - Multiset inclusion \( \subseteq \) (e.g. \( \emptyset \subseteq 2a \oplus 3c \subseteq 3a \oplus b \oplus 3c \))
- Pre-sets \( t \) resources necessary to execute \( t \)
  - multiset fetched by the execution of \( t \)
- Post-sets \( t^* \) resources produced by \( t \)
- We write \( t:u \rightarrow v \) for \( t=u \) and \( t^*=v \)
Formal Definition

A P/T Petri net is a graph $N=(S^\oplus,T,pre,post,u_0)$

- $S^\oplus$ is the set of markings
  - (is the free monoid over the set of places $S$)
  - Nodes of the graph
- $T$ is the set of transitions
  - Arcs of the graph
- $pre:T \rightarrow S^\oplus$ assigns pre-sets to transitions
  - $pre(t) = t^* \neq \emptyset$
  - Source map of the graph
- $post:T \rightarrow S^\oplus$ assigns post-sets to transitions
  - $post(t) = t^*$
  - Target map of the graph
- $u_0$ is the initial marking

Graphically

places are circles
transitions are boxes
weighted arcs model pre-/post-sets
Enabling and Firing

A transition \( t \) is enabled in the marking \( u \) if
- \( t \subseteq u \)
- Meaning that there exists \( u' \) such that \( u = u' \oplus t \)

If \( t \) is enabled, it means that the system has enough resources to execute \( t \) (called a firing)
- \( t \) can fetch the resources in its pre-set and then release fresh resources according to its post-set
- The system moves from the state \( u = u' \oplus t \) to the state \( v = u' \ominus t^* \)
- Usually written \( u[t]v \)

Steps

Several transitions that are concurrently enabled can fire concurrently
- A multiset of transition \( \bigoplus_i n_i t_i \) is enabled in the marking \( u \) if there exists \( u' \) such that
  - \( u = u' \oplus \bigoplus_i n_i t_i \)
- The concurrent execution of an enabled multiset of transitions is called a step
  - The system moves from \( u \) to \( v = u' \oplus \bigoplus_i n_i t_i^* \)
  - Usually written \( u[\bigoplus_i n_i t_i]v \)
Example: Step Sequences

\[ t_1: a_5 \rightarrow a_1 \]
\[ t_2: 2a_1 \oplus a_4 \rightarrow a_2 \oplus 2a_5 \]
\[ t_3: a_2 \rightarrow a_3 \]
\[ t_4: a_3 \rightarrow a_2 \]
\[ t_5: a_3 \rightarrow a_4 \]

\[ 4a_1 \oplus 2a_2 \]

---

Example: Step Sequences

\[ t_1: a_5 \rightarrow a_1 \]
\[ t_2: 2a_1 \oplus a_4 \rightarrow a_2 \oplus 2a_5 \]
\[ t_3: a_2 \rightarrow a_3 \]
\[ t_4: a_3 \rightarrow a_2 \]
\[ t_5: a_3 \rightarrow a_4 \]

\[ 4a_1 \oplus 2a_2 \text{ [2t_3]} \]
\[ 4a_1 \oplus 2a_3 \]
Example: Step Sequences

\[ t_1: a_5 \rightarrow a_1 \]
\[ t_2: 2a_1 \oplus a_4 \rightarrow a_2 \oplus 2a_5 \]
\[ t_3: a_2 \rightarrow a_3 \]
\[ t_4: a_3 \rightarrow a_2 \]
\[ t_5: a_3 \rightarrow a_4 \]

\[ 4a_1 \oplus 2a_2 \ [2t_3] \ 4a_1 \oplus 2a_3 \ [t_4 \oplus t_5] \ 4a_1 \oplus a_2 \oplus a_4 \]
Operational Semantics

We can describe concurrent computations by means of three simple inference rules:

- \( a \in S \) \( \rightarrow \) \( Na \) \[reflexivity\]
- \( t:u \rightarrow v \in T \) \( \rightarrow \) \( u \rightarrow_N v \) \[firing\]
- \( u \rightarrow_N v \) \( \rightarrow \) \( u' \rightarrow_N v' \) \[parallel composition\]
- \( u \rightarrow_N v \) \( \rightarrow \) \( v \rightarrow_N w \) \[sequential composition\]
- \( u \rightarrow_N w \) \[sequential composition\]

Basic Properties

- **Proposition**
  - There is a step sequence leading from \( u \) to \( v \) iff \( u \rightarrow_N^* v \)
- **Decidable properties**
  - termination
  - reachability
  - coverability
An Algebra of Computations

We can use proof terms to denote computations:

\[ \alpha: u \rightarrow Nv \quad \beta: u' \rightarrow Nv' \]

[parallel composition]  
assosiative  
commutative  
unit  
\[ \alpha \oplus \beta: u \oplus u' \rightarrow Nv \oplus v' \]

[sequential composition]  
monoid homomorphism  
identities  
\[ \alpha \beta: u \rightarrow Nw \]

Example: Step Sequences

\[ t_1: a_5 \rightarrow a_1 \]
\[ t_2: 2a_1 \oplus a_4 \rightarrow a_2 \oplus 2a_5 \]
\[ t_3: a_2 \rightarrow a_3 \]
\[ t_4: a_3 \rightarrow a_2 \]
\[ t_5: a_3 \rightarrow a_4 \]

idle resources  
activities  
\[ 4a_1 \oplus 2a_2 [2t_3] \quad 4a_1 \oplus 2a_3 [t_4 \oplus t_5] \quad 4a_1 \oplus a_2 \oplus a_4 [t_2] \quad 2a_1 \oplus 2a_2 \oplus 2a_5 \]

\[ (4a_1 \oplus 2t_3); (4a_1 \oplus t_4 \oplus t_5); (2a_1 \oplus a_2 \oplus t_2): 4a_1 \oplus 2a_2 \rightarrow 2a_1 \oplus 2a_2 \oplus 2a_5 \]
Basic Facts About Concurrency

- Suppose \( t : u \rightarrow v \) and \( t' : u' \rightarrow v' \)
  - \( t ; v = t = u ; t \)
    - Idle steps are immaterial
  - \( t \oplus t' : u \oplus u' \rightarrow v \oplus v' \) // concurrent execution
  - \( t \oplus t' = (t ; v) \oplus (u ; t') = (t \oplus u') ; (v \oplus t') \) // \( t \) precedes \( t' \)
  - \( t \oplus t' = (u ; t) \oplus (t' ; v) = (u \oplus t') ; (t \oplus v') \) // \( t' \) precedes \( t \)

- If two activities can be executed concurrently, they can be executed in any order
- The vice versa is not true
  - Take \( t : a \rightarrow a \) and \( t' : a \rightarrow a \)
  - \( t ; t' \) and \( t' ; t \) are very different from \( t \oplus t' \)

Token Philosophies

- This semantics follows the so-called Collective Token Philosophy (CTPh)
  - Any two tokens in the same place are
    - indistinguishable one from the other
    - computationally equivalent

- Other semantics follow the Individual Token Philosophy (ITPh)
  - Any token carries its own history
    - tokens have unique origins
    - fetching one token makes an activity causally dependent from the activities that produced it
    - such analysis can be important for recovery purposes, detecting intrusions, increase parallelism, ...
Process Semantics

- Non-sequential behaviour of nets
  - Causality and concurrency within a single run
- Runs are described by Processes
  - A process net $P$
    - acyclic net
    - pre-/post-sets are just sets, not multisets
    - transitions have disjoint pre-sets
    - transitions have disjoint post-sets
- A net morphism $\pi : P \rightarrow N$
  - places to places
  - transitions to transitions

Example

Graphically $\pi$ is rendered by a suitable labeling
Example

Graphically $\pi$ is rendered by a suitable labeling.

Formal Models for Distributed Negotiations
Graphically \( \pi \) is rendered by a suitable labeling.

Now there are two disjoint activities!
Concatenable Processes

- Each process $\pi$ has an initial marking $u$
  - places with no antecedents – minimal places
- and a final marking $v$
  - places with no successors – maximal places
- $\pi: u \rightarrow v$
- Can processes be composed analogously to CTPh runs?
  - In general there is some ambiguity
  - The correspondence between final places of the first process and initial places of the second process must be fixed
- Concatenable processes come equipped with suitable orders on minimal / maximal places of $P$
  - The orders concern places that are mapped to the same place of $N$

Concatenable Processes: Graphically

Superscripts denote order on minimal places
Subscripts denote order on maximal places

$$\pi: na \oplus mb \oplus ... \oplus kd \rightarrow n'a' \oplus m'b' \oplus ... \oplus k'd'$$
Composing Concatenable Processes

- Idle computations
  - any place is both minimal and maximal (no transitions)
  - minimal and maximal orders coincide
- Parallel composition $\pi_1 \otimes \pi_2$
  - juxtaposition (NOT COMMUTATIVE)
  - the orders in the result are obtained by assuming that places of the first process precede places of the second process
- Sequential composition $\pi_1; \pi_2$
  - maximal places of $\pi_1$ are merged with minimal places of $\pi_2$ according to their orders

Symmetries

- Special concatenable processes allow to rearrange the orders of minimal and maximal places
  - Called Symmetries
  - No transitions
  - The order of minimal places differs from that of maximal places
  - Symmetries are important to generate all possible causal dependencies arising from different combination of minimal and maximal places during composition
Unfolding Semantics

- Instrumental in giving denotational semantics to nets
  - a unique prime event structure that faithfully represent causality, concurrency and conflict between all possible events that can be generated from the net
  - Unfolding approximations can be used for verification
- Unfolding combines all processes in a unique structure
  - Non-deterministic exploration of computation space
  - Define a nondeterministic net $U(N)$ together with a net morphism from $U(N)$ to $N$
    - acyclic, no backward conflicts, pre-/post-sets are sets
    - places are tokens, transitions are events

Example: Three Processes
Example: Unfolding

Formal Models for Distributed Negotiations

Three relations:
- ≤ : Causality
- co : Concurrency
- # : Conflict
Unfolding Construction I

- Immediate precedence
  \[ <_0 = \{ (a,t) \mid a \in \text{t} \} \cup \{(t,a) \mid a \in \text{t} \} \]

- Causal dependence
  \[ \leq \text{ is the transitive closure of } <_0 \]
  Note that \( \leq \) and \( # \) have empty intersection

- Binary Conflict
  \[ # \text{ is the minimal symmetric relation that} \]
  is hereditary w.r.t. \( \leq \) and
  contains \( #_0 \) defined by: \( s#_0 t \iff s \neq t \wedge s \cap *t \neq \emptyset \)

- Concurrency
  \[ \text{co}(x,y) \iff \text{not}(x<y \lor y<x \lor x#y) \]
  we also write \( \text{co}(X) \iff \text{for all } x,y \in X \text{ we have } \text{co}(x,y) \)
Unfolding Construction III

- The net \( U(N) \) is the minimal net generated by the two rules below

\[
\begin{align*}
ka & \subseteq u_0 \\
\langle a, k, \emptyset \rangle & \in S_{U(N)} \quad \text{initial marking of } U(N) \\
t : a_i \rightarrow \oplus j n_j & \in T \\
\Theta &= \{ \langle a, k, H_i \rangle \} \subseteq S_{U(N)} \\
e &= \langle t, \Theta \rangle \in T_{U(N)} \\
\Psi &= \{ \langle b, m, \{ e \} \} | 1 \leq m \leq n_j \} \subseteq S_{U(N)} \\
\text{pre}(e) &= \Theta \\
\text{post}(e) &= \Psi
\end{align*}
\]

Unfolding Construction IV

- The condition \( \text{co}(\Theta) \) depends exclusively on the histories \( H_i \) and cannot be altered by successive firings
  - Histories can be completely cabled inside the tokens so that it is not necessary to recompute them at every firing (as in memoizing or dynamic programming)
  - Histories retain concurrent information, not just sequential
  - Each token / event is generated exactly once
    - It can be referred several times successively
- Several occurrences of the second rule can be applied concurrently
  - The unfolding can be implemented as a distributed algorithm
Recap

- We have seen
- Basic theory of Petri nets
  - Formal definition
  - Graphical representation
  - Step semantics
  - Process semantics
  - Unfolding semantics

References I

  - C.A. Petri
- Petri nets: an introduction (EATCS Monograph on TCS, Springer Verlag 1985)
  - W. Reisig
- Petri nets are monoids (Information and Computation 88(2):105-155, Academic Press 1990)
  - J. Meseguer, U. Montanari
References II

  - U. Goltz, W. Reisig
  - M. Nielsen, G. Plotkin, G. Winskel
- Configuration structures (Proc. LICS’95, IEEE, pp.199-209)
  - R.J. van Glabbeek, G.D. Plotkin