Where are we?

- SDL and MSC
- **Petri Nets**
  - Notation
  - Behavioral Properties
- **Symbolic Analysis methods of finite models**
- **Timed automata (real-time)**
  - Notation
  - Semantics
  - Analysis
- **Introduction to model checking?**
Petri nets – Motivation

• Invented by Carl Adam Petri in 1962 in his thesis “Kommunikation mit Automaten”

• In contrast to finite state machines, state transitions in Petri nets are executed *asynchronously*, but one at a time (DES).
  – The execution order of transitions is partly uncoordinated; it is specified by a partial order.

• Many flavors of Petri nets are in use, e.g.
  – PN with inhibitor arcs
  – Colored PN
  – PN extended with execution delays
    • Timed PN ↔ Timed Automata
    • Stochastic PN ↔ Markov chains
Petri net – Definition

- A Petri net is a bipartite, directed graph defined by a tuple 
  \((S, T, F, M_0)\), where

  - \(S\) is a set of places \(p_i\)
  - \(T\) is a set of transitions \(t_i\)
  - \(F\) is a set of edges (flow relations) \(f_i\)
    or connection relation:
    \[
    C \subseteq S \times T \cup T \times S
    \]

  - Pre set of \(t_i\) : \(\cdot t_i := \{p_i \mid (p_i, t_i) \in C\}\)
  - Post set of \(t_i\) : \(t_i \cdot := \{p_i \mid (t_i, p_i) \in C\}\)
  - analogously we can define pre- and post sets for each place \(p_i\)

- \(M_0 : S \rightarrow \mathbb{N}_0\); the initial marking: number of tokens for each place
Token marking

- Each place $p_i$ is marked with a certain number of tokens.
- $M(s)$ denotes the marking of a place $s$.
- The distribution of tokens on places defines the state of a PN, which can be described as a vector of size $|S|$
  $$\bar{s} := (m(p_1), m(p_1), \ldots, m(p_{|S|}))$$
- The initial distribution of the tokens is given by the initial state/marking often denoted $\bar{s}^0$ or $M_0$.
- The dynamics of a Petri net is defined by token game.
Token game of Petri nets

• A marking $M$ enables a transition $t_i \in T$ if all $p_k \in \cdot P_i$ contain at least one token. We write $M[> t_i$.

• If a transition $t$ is activated by $M$, it eventually fires
  – When a transition fires, it
    • consumes a token from each $p_i \in \cdot t_i$ (input place)
    • adds a token to each $p_i \in t_i \cdot$ (output place)
  – The firing gives one a state transition $M[> t_i M'$ with the new marking $M'$
  – The successive firing of all at a time enabled transitions, one at a time, allows one to visit sets of states
    • states reached on firing sequences of transitions are denoted as reachable
    • If the set of all reachable states ($[M_0>$) is finite, one speaks of a finite PN
Token game of PNs

Demo:

Non-Deterministic Evolution

- Any activated transactions might fire

Interleaving semantics: enabled transitions are executed sequentially (**unfolding all possible execution sequences**) => generates all possible behaviors
Co-operation, competition and concurrency

- PNs allow to model many-fold situations such as:
  - sequences
  - fork
  - join / synchronization
  - decision / conflict
  - concurrency
Basic types of PN (arc weights = 1)

- **State machine (SM):** A PN $P$ is denoted as SM iff $\forall t \in T: |\cdot t| = |t\cdot| \leq 1$

- **Marked Graph (MG):** A PN is denoted as MG iff $\forall p \in P: |\cdot p| = |p\cdot| \leq 1$
Basic types of PN (arc weights = 1)

• Free Choice net (FC-net): A PN is denoted as FC-net

iff \( \forall p, p' \in P: p \neq p' \Rightarrow p \cap p' \neq \emptyset \Rightarrow |p \cap p'| = |p' \cap p'| \leq 1 \)

For these simple classes many questions are decidable, e.g. can we reach a specific marking, etc.
A first extension: weighted edges

• Associating weights to edges:
  – Each edge \( f_k \) has an associated weight \( W(f_k) \) (defaults to 1)
  – A transition \( t_i \) is active if each place \( p_j \in \text{\textbullet}P_i \) contains at least \( W(f_k) \) tokens.
Token game in case of weighted edges

- A marking $M$ activates a transition $t_i \in T$ if each place $p_k \in \bullet P_i$ contains enough tokens:

$$\forall p_j \in \bullet t_i : m(p_j) \geq W(f(p_j, t_i))$$

- When a transition $t_i \in T$ fires, it
  - adds tokens to output places (1)
  - consumes tokens from input place (2)

Remark: $m(p_j)'$ is the next value, i.e. the next marking of place $p_i$

1. $\forall p_j \in t_i \bullet : m(p_j)' := m(p_j) + W(f(t_i, p_j))$
2. $\forall p_j \in \bullet t_i : m(p_j)' := m(p_j) - W(f(p_j, t_i))$
Properties

- **Reachability**: A marking $M'$ is reachable $\iff$ there exists a sequence of transitions $\{t_{10}, t_5, \ldots, t_k\}$ the seq. execution of which delivers $M'$

$$M_n = (((M_0 [> t_{10}) [> t_5]), \ldots, [> t_j)$$

Decidable (exponential space and time) for standard PNs only)

- **K-Bounded**: A Petri net $(N, M_0)$ is $K$-bounded $\iff \forall m \in [M_0] > : m(p) \leq K$ (finite PNs are trivially $k$-bounded & vice-versa).

- **Safety**: 1-Boundedness (every node holds $\leq 1$ token (always)

- **Liveness**: A PN is (strongly) live iff for any reachable state all transitions can be eventually fired.

- **Deadlock-free**: A PN is deadlock-free or weakly live iff for each of its reachable states at least one transition is enabled.

These questions are solely decidable for standard PNs only!
Analysis Methods

1. **Analytic methods** (smart methods), *e.g. based on linear algebra*:
   solution of a system of linear equation is a necessary condition for reachability; only applicable for basic types of PNs, since PNs with more than 2 inhibitor arcs have Turing-power => most questions (deadlock-freeness, etc.) not decidable anymore.

2. **Methods based on state space exploration (brute-force):**
   1. **State Space exploration for finite PNs**:
      Enumeration of all reachable markings.

   2. **Simulation for finite and in-finite PNs**:
      Play token game by solely executing one of the enabled transitions (gives single trace of possible executions (= run))

   3. **State Space exploration of infinite PNs (Coverability tree)**:
      Enumeration of all classes of reachable markings
Method 1: Incidence Matrix

- Goal: Describe a Petri net through equations
- The incidence matrix $A$ describes the token-flow according for the different transitions
- $A_{ij} = \text{gain of tokens at node } i \text{ when transition } j \text{ fires}$
- A marking $M$ is written as a $m \times 1$ column vector

\[
A = \begin{bmatrix}
-2 & 1 & 1 \\
1 & -1 & 0 \\
1 & 0 & -1 \\
0 & -2 & 2 \\
\end{bmatrix}
\]

\[
M_0 = \begin{bmatrix}
2 \\
0 \\
1 \\
0 \\
\end{bmatrix}
\]
Method 1: State Equation

- The firing vector $u_i$ describes the firing of transition $i$. It consists of all ‘0’, except for the $i$-th position, where it has a ‘1’.

  \[ u_i = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \]

  E.g.  \[ t1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad t2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad t3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \]

- A transition $t$ from $M_k$ to $M_{k+1}$ is written as

  \[ M_{k+1} = M_k + A \cdot u_i \]

$M_1$ is obtained from $M_0$ by firing $t3$

\[
\begin{bmatrix} 3 \\ 0 \\ 0 \\ 2 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 1 \\ 0 \end{bmatrix} + \begin{bmatrix} -2 & 1 & 1 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \\ 0 & -2 & 2 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}
\]
Method 1: Condition for reachability

• A marking $M_k$ is reachable from $M_0$ if there is a sequence of transitions $\{t_1, t_2, \ldots, t_k\}$ such that $M_k = M_0 \cdot t_1 \cdot t_2 \cdot \ldots \cdot t_k$.

• Expressed with the incidence matrix:

$$M_k = M_0 + A \cdot \sum_{i=1}^{k} u_i \quad (1)$$

which can be rewritten as

$$M_k - M_0 = \Delta M = A \cdot \bar{x} \quad (2)$$

If $M_k$ is reachable from $M_0$, equation (2) must have a solution where all components of $\bar{x}$ are positive integers.

(This is a necessary, but not sufficient condition for reachability.)
Method 2: State Space Exploration (finite PN)

- If the set of reachable states is finite, one may execute each enabled transition for each marking of the net.

- Starting with the initial marking $M_0$ and until a fixed point is reached gives one the set of all reachable states and the transitions among them (details on reachability algorithms will follow).
A labelled transition system (LTS) is tuple \((S, Act, \rightarrow, I)\) where

- \(S\) is the set of reachable states (markings of the PN)
- \(I\) is the set of initial states (the initial marking \(M_0\) of the PN)
- \(Act\) is the set of activity/action labels (transition identifier of the PN)
- \(\rightarrow \subseteq S \times Act \times S\) is a transition relation

Via state space exploration each finite PN can be mapped to its (underlying) transition system, also often denoted as state graph (SG).

What does set of reachable states means? The set of reachable states is the set of those markings of a PN, which can be obtained by executing all enabled (activated transitions) within each state, starting from the initial marking \(M_0\). In the following we will denote such sets \(\text{Reach}(M)\) with respect to a model \(M\).
Method 2: State Space Exploration (finite PN)

Properties to be directly answered on the level of the finite SG:

– Does the model has finite executions only (termination)?
– Is the model deadlock-free?
– Is the model alive, i.e. each path contains every transition? (strongly connected component with all transition labels included)
– Is the model weakly alive: each transition occurs within the SG.
– Is the model reversible, i.e. from every reachable marking there is a way back to the initial state.
– Is $m(p_i)$ of a place $p_i$ bounded?
Method 3: Coverability Tree/Graph (CG) (non-finite PNs)

- A PN can be infinite, i.e. its set of reachable states is un-bounded. What can we do?
- Detect & handle infinite cycles (CG is not unique)
- What kind of questions can we answer?
  - is the PN finite?
  - which are the bounded/un-bounded places
  - is there a marking reachable s.t. \( t_i \) is enabled?

\[
\begin{align*}
M_0 &= [1 \ 0 \ 0] \\
M_1 &= [0 \ 0 \ 1] \\
M_3 &= [1 \ \omega \ 0] \\
M_4 &= [0 \ \omega \ 1] \\
M_5 &= [0 \ \omega \ 1] \\
M_6 &= [1 \ \omega \ 0]
\end{align*}
\]

\( \omega \) denotes an arbitrary number of tokens.
Coverability Graph – the Algorithm

Special symbol $\omega$, similar to $\infty$: $\forall n \in \mathbb{N}$: $\omega > n$; $\omega = \omega + n$; $\omega \geq \omega$

- Label initial marking $M_0$ as root and tag it as new
- while new markings exist, pick one, say $M$
  1. If $M$ is identical to a marking on the way from the root to $M$, mark it as old; continue;
  2. If no transitions are enabled at $M$, tag it as deadend;
  3. For each enabled transition $t$ at $M$ do
     a) Obtain marking $M' = M[t$
     b) If there exists a marking $M''$ on the way from the root to $M$ s.t. $M'(p) \geq M''(p)$ for each place $p$ and $M' \neq M''$, replace $M'(p)$ with $\omega$ for $p$ where $M'(p) > M''(p)$.
     c) Introduce $M'$ as a node, draw an arc with label $t$ from $M$ to $M'$ and tag $M'$ new.
Results from the Coverability Tree $T$

- The net is **bounded** iff $\omega$ does not appear in any node label of $T$
- The net is **safe** iff only ‘0’ and ‘1’ appear in the node labels of $T$
- A transition $t$ is **dead** iff it does not appear as an arc in $T$
- If $M$ is **reachable** from $M_0$, then there exists a node $M'$ s.t. $M \leq M'$.
  (This is a necessary, but not sufficient condition for reachability.)

- For **bounded** Petri nets, this tree is also called **reachability tree**, as all reachable markings are contained in it.
Compositionality

• When it comes to the modelling of complex systems, PN tend to become very large and unclear.

• Concepts developed in the context of Process Algebra have been taken over in the world of PN.

• This allows to construct PNs in a compositional manner, where we will only roughly touch:
  – Composition via sharing of places
  – Composition via synchronization

Remark:
As it turns out, compositionality can also often be exploited when analysing high-level models.
Synchronisation

• Dedicated activities have to be executed jointly:

  • \( T_i \) is the transition system of submodel \( i \),

  • \( \mathcal{Act}_S \) the set of synchronizing transitions

  • \( \mathcal{Act}_S \) the set of non-synchronizing transitions

• We have the following rules (modus ponens) on the level of LTS (labelled transition systems)

  – Synchronizing activities:

    \[
    \frac{T_1 : \vec{x} \xrightarrow{\alpha} \vec{y} \land T_2 : \vec{q} \xrightarrow{\alpha} \vec{r}}{T_1 \times T_2 : (\vec{x}, \vec{a}) \xrightarrow{\alpha} (\vec{y}, \vec{r})} \quad \alpha \in \mathcal{Act}_S
    \]

  – Non-synchronizing activities

    \[
    \frac{T_1 : \vec{x} \xrightarrow{\alpha} \vec{y} \land T_2 : \vec{q} \xrightarrow{\alpha} \vec{r}}{T_1 \times T_2 : (\vec{x}, \vec{a}) \xrightarrow{\alpha} (\vec{y}, \vec{a}) \land (\vec{x}, \vec{a}) \xrightarrow{\alpha} (\vec{x}, \vec{r})} \quad \alpha \in \mathcal{Act}_S
    \]
Synchronisation (example)

For PNs synchronization can be realized by merging effected transitions;

Remark:
Cross-product computation can also be executed on the level of local state graphs.

Sharing of places (variables)

- Dedicated places have to hold same number of tokens:

\[
T_1 : (x_1, \ldots, x_m) \xrightarrow{\alpha} (x'_1, \ldots, x'_k) \wedge T_2 : (y_1, \ldots, y_m) \xrightarrow{\beta} (y'_1, \ldots, y'_m)
\]

\[
T_1 \times T_2 : ((x_1, \ldots, x_m); (y_1, \ldots, y_m)) \xrightarrow{\alpha} ((x'_1, \ldots, x'_m); (y'_1, \ldots, y''_m, \ldots, y_m)) \wedge ((x_1, \ldots, x_m); (y_1, \ldots, y_m)) \xrightarrow{\beta} ((x_1, \ldots, x''_1, \ldots, x_m); (y'_1, \ldots, y'_m))
\]

where for \( x_i = y_j, x'_i = y''_j \wedge x''_i = y'_j \) for \( p_i, p_j \in P_{Sh} \) holds

- \( P_{Sh} \) is the set of shared places
- \( x_i, y_j \) their values in a source and
- \( x'_i, y'_j, x''_i, y''_j \) in a target state
Common Extensions

- **Colored Petri nets**: Tokens carry values (colors)
  Any Petri net with finite number of colors can be transformed into a regular Petri net.

- **Continuous Petri nets**: The number of tokens can be real.
  Cannot be transformed to a regular Petri net.

- **Inhibitor Arcs**: Enable a transition if a place contains no tokens
  Cannot be transformed to a regular Petri net, as soon as we have more than 2 inhibitor arcs (for 2 inhibitor arcs this depends on the structure of the PN)
Literature

- W. Reisig: Petri Netze – Eine neue EInfuehrung, November 2007,
  http://www2.informatik.hu-berlin.de/top/pnene_buch/pnene_buch.pdf

  http://ls4-www.informatik.uni-dortmund.de/QM/MA/fb/spnbook2.html


Analysis of finite high-level models

• Common high-level model description techniques have Turing-power =>
  most questions are not decidable.

• If a dynamic model, e.g. a PN, is bounded or finite one may generate its
  underlying reachability graph, also commonly denoted as state graph (SG).
  Its inspection may answer the questions of interest such as deadlock-
  freeness, liveness, …

• In the following we will discuss two techniques for analyzing such systems
  – Standard approach: Reachability analysis, based on hash table
  – Symbolic approaches:
    Reachability analysis, based on “symbolic” data structures
Standard reachability analysis technique

1) \( \text{Stack} := \emptyset, \text{HashTable} := \emptyset \) \( s_0 := \text{initialState} \)
2) \( \text{push}(s_0, \text{Stack}) \)
3) \( \text{insert}(s_0, \text{HashTable}) \)
4) Call DFS()

5) Function DFS()
6) While (Stack != \( \emptyset \))
7) \( S := \text{pop}(\text{Stack}) \)
8) Forall succ \( s' \) of \( s \) do
9) If \( (s' \) HashTable)
10) \( \text{push}(s', \text{Stack}) \)
11) \( \text{insert}(s', \text{HashTable}) \)
12) endif
13) endwhile
14) endwhile
15) endfunction
Computer-assisted validation

What’s the obstacle?
Interleaving semantics gives that the number of states grows exponential with the number of independent transitions and or with the number of tokens (concurrent processes).
Symbolic techniques

In the following we will have a look on so called symbolic techniques for the analysis of finite systems:

1. Techniques based on Binary Decision Diagrams
2. SAT-Solvers for $k$-bounded transition systems
Binary Decision Diagrams

• A Binary Decision Diagram (BDD) is a directed non-cyclic graph for representing Boolean functions ($\{1,0\}^n \rightarrow \{0,1\}$).

• Thus they can be used for encoding sets and transition relations, i.e. a BDD may represent a characteristic function of a set $S$

$$\chi(x) := \begin{cases} 1 & \iff x \in S \\ 0 & \text{else} \end{cases}$$
Reduced ordered Binary Decision Diagram

A BDD consists of the following sets:

- $\mathcal{K}_T$ (terminal nodes)
- $\mathcal{K}_{NT}$ (non-terminal nodes)
- $\mathcal{V} := \{x_1, \ldots, x_n\}$
  (boolean input or function variables)

and we have the following functions:

- $\text{var}: \mathcal{K}_{NT} \rightarrow \mathcal{V}$
- $\text{value}: \mathcal{K}_T \rightarrow \{0, 1\}$
- then- and else-function: $\mathcal{K}_{NT} \rightarrow \mathcal{K}_T \cup \mathcal{K}_{NT}$.
  - dashed line else- or 0-successor: $\text{else}(n) = l$
  - solid line then- or 1-successor: $\text{then}(n) = k$
Reduced ordered Binary Decision Diagram

A BDD is called reduced iff there exist

- no don’t care node: \( \forall n \in \mathcal{K}_{NT} : \text{else}(n) = \text{then}(n) \).
- no isomorphic nodes: \( \forall n, k \in \mathcal{K}_{NT} : k \equiv n \)

A BDD is called ordered iff there exists no node the children of which are labelled with a large/smaller variable with respect to an ordering relation:

\( \forall n \in \mathcal{K}_{NT} : \text{var(else}(n)) < \text{var}(n) \lor \text{var(then}(n)) < \text{var}(n). \)
Reduced ordered Binary Decision Diagram

Merge redundant terminal nodes!
Reduced ordered Binary Decision Diagram

Merge redundant non-terminal nodes!
Reduced ordered Binary Decision Diagram

Merge redundant non-terminal nodes!
Reduced ordered Binary Decision Diagram

Eliminate don’t-care nodes, re-direct incoming arcs to successor
Shannon-Expansion

A RO-BDD is a graph-based representation of a Boolean function, its interpretation is based on the Shannon-expansion:

\[
f(a, b) := \neg a f|_{a=0}(b) + a f|_{a=1}(b)
\]

\[
= \neg a \neg b f|_{ab=00} + \neg ab f|_{ab=01} + a \neg b f|_{ab=10} + ab f|_{a=11}
\]

\(f|_{a=0}\) is denoted as 0-cofactor and \(f|_{a=1}\) as 1-cofactor of \(f\) with respect to variable \(a\). Note: \(f|_{a=0} = 1\).
A RO-BDD is a graph-based representation of a Boolean function, its interpretation is based on the Shannon-expansion:

\[
\begin{align*}
&= \neg a \cdot b \cdot f |_{ab=00} + \neg a \cdot b \cdot f |_{ab=01} \\
&+ a \cdot b \cdot f |_{ab=10} + a \cdot b \cdot f |_{a=11} \\
&= a \cdot b
\end{align*}
\]

Remove terms which evaluates to 0.
Operations on RO-BDDs: The Apply algorithm (1)

A binary operator can be applied to two BDDs by making use of Bryant’s recursive Apply-algorithm:

\[
\text{Apply}(\text{op}, n, m) \\
(0) \text{ node } e, t, res; \\
\]

Reached terminal nodes, end of recursion

(1) If \( n, m \in \mathcal{K}_T \) Then
(2) \( v := \text{value}(n) \text{ op value}(m); \)
(3) Return \( \text{getTerminal}(v); \)

Check \( \text{op cache if result is already known} \)

(4) \( res = \text{cacheLookup}(\text{op}, n, m); \)
(5) If \( res \neq \epsilon \) Then Return \( res; \)
Operations on RO-BDDs: The Apply algorithm (2)

Depending on the ordering, branch into resp. recursion

(6) If $\var(n) = \var(m)$ Then
(7) \quad v := \var(n);
(8) \quad e := \text{Apply}(\text{op}, \text{else}(n), \text{else}(m));
(9) \quad t := \text{Apply}(\text{op}, \text{then}(n), \text{then}(m));

(10) Else if $\var(n) < \var(m)$ Then
(11) \quad v := \var(n);
(12) \quad e := \text{Apply}(\text{op}, \text{else}(n), m);
(13) \quad t := \text{Apply}(\text{op}, \text{then}(n), m);

(14) Else
(15) \quad v := \var(m);
(16) \quad e := \text{Apply}(\text{op}, n, \text{else}(m));
(17) \quad t := \text{Apply}(\text{op}, n, \text{then}(m));
Operations on RO-BDDs: The Apply algorithm (3)

Allocate new node, (unique and non-dnc-node)
(18) \( res := \text{getUniqueDNCFreeNode}(v, t, e); \)

Insert result into op cache and terminate recursion
(19) \( \text{cacheInsert}(op, n, m, res); \)
(20) Return \( res; \)
Binary Decision Diagrams

- BDDs can be used for encoding sets and transition relations, i.e. a BDD may represent a characteristic function of a set $S$ or a transition relation $T$

A detailed example on BDDs and finite PNs will follow in the exercise class!

DNC-nodes kept solely for illustration purpose!
Standard Symbolic Reachability Analysis (bfs)

ReachabilityAnalysis()
(0) \( Z_U := \text{Encode}(\mathcal{S}^\epsilon, \mathcal{S}) \);
(1) \( Z^p := \text{transition function} \);
(2) Do
(3) \( Z_{\text{tmp}} := Z^p \cap Z_U \);
(4) \( Z_{\text{tmp}} := \text{Abstract}(Z_{\text{tmp}}, \mathcal{S}, +) \);
(5) \( Z_{\text{tmp}} := Z_{\text{tmp}} \setminus Z_R \);
(6) \( Z_R := Z_R \cup Z_{\text{tmp}} \{ \mathcal{S} \leftarrow \mathcal{T} \} \);
(7) \( Z_U := Z_{\text{tmp}} \{ \mathcal{S} \leftarrow \mathcal{T} \} \);
(8) Until \( Z_U = \emptyset \)
(9) Return \( Z_R \);

The Abstract algorithm delivers
- the existential-quantification for +,
- and the all-quantification for *
- with respect to a set of variables

The operation \( a \leftarrow b \) re-labels each occurrence of variable \( a \) with variable \( b \), can be applied to set of variables (interleaved orderings!).

How-can we check now for a deadlock?
How-to exploit compositionality, when using BDDs?

- Synchronization of activities:

\[
\prod_{\alpha_i \in \text{Act}_S} Z_{\alpha_i} \cdot Z_i \cdot 1_i + \sum_{\beta_j \in \text{Act}_S} Z_{\beta_j} \cdot Z_j \cdot 1_j
\]

- Sharing of variables

\[
\sum_{\beta_j \in \text{Act}} Z_j \cdot 1_j
\]

- \(Z_\alpha\) is the BDD representing the transition name (index),
- \(Z_i\) is the BDD representing the transition system of submodel \(i\),
- \(1_i\) are identity structures of appropriate dimension.
Common Extensions of BDDs

• **Multi-terminal BDDs**: Terminal nodes hold values from a finite domain (pseudo-boolean functions)

• **Zero-suppressed (MT) BDDs**: Instead of dnc-nodes one eliminates nodes the out-going 1-one edge of which leads to the 0-sink.

• **Multi-valued DD**: Nodes contain set of numbers and have more than 2-successors only
Literature


  http://www.hpi.uni-potsdam.de/fileadmin/hpi/FG_ITS/books/OBDD-Book.pdf


Outlook: From BDDs to SAT-solvers

- **State-of-the-art**
  - State machines, i.e. their transition relation (TR) can be represented by Binary Decision Diagram (= directed, acyclic graph for rep. Boolean functions)
  - complex procedures for deriving BDD from high-level model description
  - BDD encodes one-step TR, two sets of boolean variables:
    - $x$-variables holding source states
    - $y$-variables for holding target states $t$

Depending on the modelled system BDD may explode in the number of allocated nodes (add-function $y := x + p$)

- **New trends:** SAT-Solvers have shown to be of value in such cases
SAT-Solvers at glance (1)

- Satisfiability: Does there exists an assignment to the variables of a formula $\alpha$ of propositional logics, so that the formula evaluates to true.

- 3-SAT: In general this problem is NP-complete (Cook 1972).

  $\Rightarrow$ One may not expect always efficient computations. But in practice SAT-solver have shown to be very powerful, outperform BDDs.

- Employing SAT-based MC:
  
  - Encode TR as boolean formula (unfolding of loops, each step in TR is encoded by a new set of variables, $k$-steps within TR? 

\[
I(s_0) \land \bigwedge_{i=0}^{k-1} T(s_i, s_{i-1})
\]

  - Encode properties to be checked as boolean equation.
  - Check if the obtained overall formula is satisfiable.
Literature
