Reasoning About Discrete Event Systems
Reasoning About Discrete Event Systems

1. Introduction & basic concepts.
2. Planning, verification & diagnosis.
   - Relaxations and heuristics.
4. Reasoning about DES in logic.
5. Infinite-state systems.
Example: The Optical Telegraph Protocol

* Communications Protocol:
  – Message vocabulary.
  – Rules govern behaviour of protocol participants.
* Model protocol participants.
* Model system by composition.
Discrete Event Model of a Telegraph Operator
Synchronised Composition

* Discrete event system: \( S = (Q, L, q_I, T) \), where
  - \( Q \) is the set of states;
  - \( L \) is the set of event labels (alphabet);
  - \( q_I \in Q \) is the initial state; and
  - \( T \subseteq Q \times L \times Q \) the transition relation.

* \( S_1 \times S_2 = (Q_1 \times Q_2, L_1 \cup L_2, (q_{I_1}, q_{I_2}), T) \), where
  - \( T((q_1, q_2), (a, b), (q'_1, q'_2)) \) iff \( T_1(q_1, a, q'_1), T_2(q_2, b, q'_2) \)
    and \( aRb \);
  - \( T((q_1, q_2), (a, ), (q'_1, q'_2)) \) iff \( T_1(q_1, a, q'_1) \) and
    \( \neg \exists b \in L_2 : aRb \);
  - \( T((q_1, q_2), (, b), (q_1, q'_2)) \) iff \( T_2(q_2, b, q'_2) \) and
    \( \neg \exists a \in L_1 : aRb \); and
  - \( R \subseteq L_1 \times L_2 \) is the label synchronisation relation.
Introduction
24 states
215 states
1911 states
16984 states
150945 states

Introduction
Planning

“Planning is the art and practice of thinking before acting.”

* Given a world model, and initial world state, and a desired goal condition, find a (cheapest, fastest, etc.) course of actions that transforms the world from the initial state to a state where the goal condition holds.

* The Classical Planning Assumptions:
  – The (model of the) world is finite, discrete and deterministic.
  – The world model complete and correct.
  – There is no interference.

* In what situation could such assumptions possibly hold?

Planning, Verification & Diagnosis
Puzzles
Industrial Automation

Planning, Verification & Diagnosis
Airport Ground Traffic Control

(Hatzack & Nebel 2001)

(Trüg, Hoffmann & Nebel 2004)

Planning, Verification & Diagnosis
(Blanchette et al. 1999)

**Genome Edit Distance Computation**

Planning, Verification & Diagnosis
Narrative Generation

Planning, Verification & Diagnosis

(Chang & Soo 2009)

(Porteous, Charles & Cavazza 2013)
The Classical Planning World Model

* States are assignments of truth values to a finite set of atomic propositions (\( \mathcal{V} \)).
* Transitions are instances of actions, characterised by:
  - Preconditions: Must hold in the state for action to be applicable.
  - Effects: Will hold in the state after action application.
  - Assumption of inertia: Atoms not affected by the action remain as they were.
* There is a single initial state.
* The set of goal states is a defined by a formula over \( \mathcal{V} \).
Example: Sokoban

(empty s22) (empty s23)
(empty s32) (empty s33)
(box s42) (man s43) (empty s44) (empty s45)
(empty s52) (empty s53) (box s54) (empty s55)
(empty s62) (empty s63)
Verification

* Given a system model, decide if the system meets specified requirements:
  – Safety.
  – Liveness (absence of deadlock states).
  – Progress / responsiveness.

* Distributed & concurrent systems:
  – Set of finite-state processes, may be dynamic.
  – Inter-processes communication: shared memory; synchronised transitions; or message queues (pipes).
  – Non-deterministic: unpredictable relative speeds; or abstract model.
flag[0] = true;
turn = 1;
while (flag[1] && turn == 1) {
    // busy wait
}
// critical section
flag[0] = false;
Diagnosis & Diagnosability

* Given a system model, some knowledge of the initial system state and a series of observations of events from the running system, decide
  - what are possible current states of the system?
  - what unobservable events can / cannot have happened?
  - is system behaving according to the model?

* Given a system model, decide
  - what unobservable (failure) events, if any, can go undiagnosed infinitely?
  - where to place additional sensors to make important (e.g., failure) events diagnosable, at minimum cost?
Example: The Vadstena Water Plant
Industrial Automation

Planning, Verification & Diagnosis
Autonomous Systems

(Williams & Nayak 1996)
client1.connect(server)
cclient1.high_NX_volume
cserver.connect(client1)
cserver.port_scanning(client2)
cserver.port_scanning(client3)
cclient3.connect(server)
cclient2.connect(server)
cclient2.many_requests_to_one_domain
cclient3.many_requests_to_one_domain

(Sohrabi, Udrea & Riabov 2013)

Security

Planning, Verification & Diagnosis
Example: WITAS UAV

10877070000 NAV.0.0 Created { { MissionExecutor { { Position { 265.0 -66.0 20.0 ... 
10893800000 NAV.0.0 CheckPt 4 { { SegmentSeq { { } } } } 
10895050000 F3D.0.0 Created { NAV.0.0 { { SegmentSeq { { } } } } } 
10895770000 F3D.0.0 CheckPt 11 { { double 19.1655151302 } } } } 
10896280000 F3D.0.0 CheckPt 21
10990120000 Heli 0 FINISHED 
10990270000 F3D.0.0 CheckPt 24
10990380000 F3D.0.0 Mode:done
10990660000 F3D.0.0 Destroyed

<state name="wait_trajectory_end">
  <reaction event="fcl_finished">
    <checkpoint id="24">
      <send_event/>
    </checkpoint>
    if (final_segment(path, segment_index)) {
      <exit message="path complete"/>
    } 
    else {
      <jump state="turn_to_goal_heading"/>
    }
  </reaction>
</state>

Planning, Verification & Diagnosis
Planning, Verification & Diagnosis
Planning, Verification & Diagnosis
Planning, Verification & Diagnosis
Diagnosability

* Let Mod be the set of all event sequences allowed by the system model.

* A fault $f$ is diagnosable iff $\exists n$
  \[
  \forall s \in \text{Mod} : f \in s \\
  \forall s' \in \text{Mod} : s' = s, t; |\text{obs}(t)| \geq n \\
  \forall s'' \in \text{Mod} : \text{obs}(s'') = \text{obs}(s') \\
  f \in s'',
  \]
i.e., $f$ is detectable after at most $n$ observations.

* Twin-plant construction:
  – Compose $S$ with a copy $S'$ of itself, synchronising on observations.
  – If no infinite (i.e., looping) trajectory in $S \times S'$ such that $f$ occurs in $S$ but not in $S'$ exists, $f$ is diagnosable.
Problem Reductions

* Planning-as-model checking
  – Create a deadlock when goal achieved.

* Diagnosis-as-planning
  – Goal is to generate the given observations.
  – Need 0/1 action costs to minimise fault events.

* Planning-as-diagnosis
  – Explain observation “goal achieved”, with/without “cheating” (fault).

* Model checking-as-planning
  – Encode acceptance criteria.
  – “Mark & return” for looping path.
Domain Independence

- System models required for reasoning typically given in a formal description language.
- A reasoner/algorithm is domain-independent if it is applicable to a class of models that is defined by what the reasoners input language can express.
- Why, or why not, domain-independence?
  - Generality of algorithms & underlying principles.
  - Improve reuse, rapid prototyping.
  - Opportunities for improvement (efficiency, reasoning capability) by specialisation to a domain / application.
- No reasoning systems are *truly* domain-independent.
* Trying to find a solution for ~ 3 days.
* Writing a PDDL model took ~ 45 minutes.
* Planner to find an optimal plan: 0.1 seconds.
State-Space Exploration

* Given an *implicit representation* of $S = S_1 \times \ldots \times S_n$, generate (and store) only as much of $S$ as is needed to solve the problem.

* Forward & Regression search.

* Graph search algorithms:
  - Best-First Search.
  - Memory-limited search.
State-Space Exploration
State-Space Exploration
Regression (backward search)

\[ s_n \xrightarrow{\text{regression through } a_n} s_{n-1} \xrightarrow{\text{satisfies}} s_1 \xrightarrow{\text{regression through } a_1} s_0 = G \]

- \( I = w_0 \xrightarrow{\text{execute } a_n} w_1 \xrightarrow{\text{satisfies}} w_{n-1} \xrightarrow{\text{execute } a_1} w_n \)

\[ \varphi \text{ holds in } \text{apply}(a, s) \iff \text{regress}(\varphi, a) \text{ holds in } s. \]

- \( \text{apply}(a, s) \models \varphi \iff s \models \text{regress}(\varphi, a). \)

\* In STRIPS notation:

\[
\text{regress}(c, a) = \begin{cases} 
(c - \text{add}(a) \cup \text{pre}(a)) & \text{if } \text{del}(a) \cap c = \emptyset \\
\text{FALSE} & \text{otherwise}
\end{cases}
\]
State-Space Exploration
State-Space Exploration
State-Space Exploration
(push-down s22 s32 s42)

(empty s42),
(man s22),
(box s32),
(box s23)

(move-down s42 s52)

(empty s52),
(man s42),
(man s22),
(box s32),
(box s23)

(move-up s42 s32)

(man s42),
(empty s32),
(man s22),
(box s32),
(box s23)

(move-up s32 s22)

(empty s42),
(man s32),
(empty s22),
(box s32),
(box s23)
Relaxations and Heuristics

- A relaxation is a problem transformation that preserves solution existence and cost.
- If the relaxed problem is easier to solve (e.g., polynomial), optimal solution cost for the relaxed problem is usable as an admissible heuristic for the original problem.
- Relaxations of the reachability problem:
  - Abstraction.
  - Monotonic relaxation.
Abstraction

* An abstraction is a mapping from (states of) $S$ to some abstract space $S'$, which preserves labelled paths and goal states.
Monotonic Relaxation

* State variables/components accumulate values.
* Propositionally, facts, once true, never become false.
  – In planning, this is known as the delete relaxation ($P^+$).
* Relaxation: any plan for $P$ is also valid for $P^+$.
* Solving $P^+$ is in $P$.
  – Applying an action (transition) cannot disable any other action.
  – No action needs to be applied more than once.
* Solving $P^+$ optimally is NP-hard.
Example: Monotonic Relaxation

State-Space Exploration: Relaxations and Heuristics
Reasoning About DES in Logic

* Proving invariance.
* Bounded reachability and propositional logic.
* Tense-modal logic
  - Logic as a specification language.
  - Linear Temporal Logic (LTL)
Proving Invariance (for Planning)

* A formula \( \varphi \) is an invariant iff \( s \models \varphi \) and enabled(\( a, s \)) implies apply(\( a, s \)) \( \models \varphi \), for all \( s \) and \( a \).
  
  - If \( \varphi \) is invariant and holds in \( s_0 \), \( \varphi \) holds in every reachable state.

* Testing invariance w.r.t. action schemas:
  
  - Regression: apply(\( a, s \)) \( \models \varphi \) iff \( s \models \text{regress}(\varphi, a) \).
  
  - \( \varphi \) is invariant iff for each \( a \)

\[
\forall \vec{x} (\varphi \land \text{pre}(a(\vec{x}))) \rightarrow \text{regress}(\varphi, a(\vec{x}))
\]

is a tautology.
Regression over Action Schemas

* Regressing an atomic formula: apply\( (a(\vec{y}), s) \models p(\vec{x}) \) iff
  - \( p(\vec{z}) \in \text{effects}(a(\vec{y})) \) and \( \vec{x} = \vec{z} \); or
  - \( s \models p(\vec{x}) \) and for any \( \neg p(\vec{z}) \in \text{effects}(a(\vec{y})) \), \( \vec{x} \neq \vec{z} \)

* Regressing any formula: substitute condition for atoms.

* move-right\((f, t)\)
  
  precond: right-of\((f, t)\), man\((f)\), empty\((t)\)
  
  effects: \( \neg \text{man}(f) \), empty\((f)\), man\((t)\), \( \neg \text{empty}(t) \)

* regress\((\text{man}(x) \lor \text{box}(x) \lor \text{empty}(x)), \text{move-right}(f, t)\)\)
  \[ = (((x = t) \lor (\text{man}(x) \land x \neq f)) \lor \text{box}(x) \lor ((x = f) \lor (\text{empty}(x) \land x \neq t))). \]
Example: Invariants (?)

1. $\forall x (\text{man}(x) \lor \text{box}(x) \lor \text{empty}(x))$
2. $\forall x (\neg \text{man}(x) \land \neg \text{box}(x)) \lor (\neg \text{man}(x) \land \neg \text{empty}(x)) \lor (\neg \text{box}(x) \land \neg \text{empty}(x))$
3. $\forall x, y (\text{man}(x) \land \text{man}(y) \rightarrow (x = y))$
4. $\forall x, y, z (\text{box}(x) \land \text{box}(y) \land \text{box}(z) \rightarrow ((x = y) \lor (x = z) \lor (y = z)))$
5. $\exists x, y (\text{box}(x) \land \text{box}(y) \land (x \neq y))$
**Horizon-\(k\) Planning as SAT**

| \(p_1@0\) | \(a_1@1\) | \(p_1@1\) | \(a_1@2\) | \(\ldots\) | \(a_1@k\) | \(p_1@k\) |
| \(p_2@0\) | \(a_2@1\) | \(p_2@1\) | \(a_2@2\) | \(\ldots\) | \(a_2@k\) | \(p_2@k\) |
| \(\vdots\) | \(\vdots\) | \(\vdots\) | \(\vdots\) | \(\ldots\) | \(\vdots\) | \(\vdots\) |
| \(p_n@0\) | \(a_m@1\) | \(p_n@1\) | \(a_m@2\) | \(\ldots\) | \(a_m@k\) | \(p_n@k\) |

* The set of valid plans up to (parallel) length \(k\) can be encoded as a propositional logic formula \(\phi\).
* $a@i \rightarrow \bigwedge_{p \in \text{pre}(a)} p@(i - 1)$.  
  - If $a$ takes place at $i$, $\text{pre}(a)$ must hold at $i - 1$.
* $a@i \rightarrow \bigwedge_{p \in \text{add}(a)} p@i$.  
* $a@i \rightarrow \bigwedge_{q \in \text{del}(a)} \neg q@i$.  
  - If $a$ takes place at $i$, it’s effects hold at $i$.
* $(p@i \land \neg p@(i - 1)) \rightarrow \bigvee \{a \mid p \in \text{add}(a)\} a@i$.  
* $(\neg p@i \land p@(i - 1)) \rightarrow \bigvee \{a \mid p \in \text{del}(a)\} a@i$.  
  - If $p$ changes from $i - 1$ to $i$, some action must have caused the change.
* $p@0$ if $p \in s_0$.
* $\neg p@0$ if $p \notin s_0$.
  - The first atom layer is the initial state.
* $p@k$ if $p \in G$.
  - The last atom layer satisfies the goal.
* $\neg a@i \lor \neg a'@i$ if $\text{del}(a) \cap (\text{pre}(a') \cup \text{add}(a')) \neq \emptyset$ or $\text{del}(a') \cap (\text{pre}(a) \cup \text{add}(a)) \neq \emptyset$.
  - More than one action can take place at $i$ as long as they do not interfere.
  - Weaker requirement: actions at $i$ can be performed in some sequence.
SAT-based planning

* How to find the right $k$?
* Test $\Phi_k$, $k = 1, 2, \ldots$, until a satisfying assignment (plan) is found.
* Can be very inefficient.
* This is what SATPLAN does.

(Rintanen, Heljanko & Niemelä, 2006)
SAT-based planning

* Try many values in parallel, allocating geometrically less time to higher values.
* As SAT formulas proved unsolvable, the frontier moves up.
* Limited by memory.

(Rintanen, Heljanko & Niemelä, 2006)
LTL, CTL and CTL* are tense modal logics.
- LTL is interpreted over paths (sequences of states).
- CTL and CTL* are interpreted over trees.

Property specification language:
- Safety: □¬deadlock (LTL), ∀□¬deadlock (CTL)
- Responsiveness: □(p → ◊q) (LTL), ∀□(p → ∀◊q) (CTL)
- Possibility: ∀□∃◊goal (CTL)
- ◊□p (LTL)
Linear Temporal Logic (LTL)

* LTL is interpreted over infinite sequences of states.
* \( \omega = s_0, s_1, \ldots \):
  - \( \omega \models \rho \) iff \( s_0 \models \rho \).
  - Interpretation of logical connectives (\( \neg, \land, \lor \)) is standard.
  - \( \omega \models \Box \alpha \) iff \( \omega^1 = s_1, s_2, \ldots \models \alpha \).
  - \( \omega \models (\alpha U \beta) \) iff \( \exists k. \omega^k \models \beta \) and \( \forall 0 \leq i < k. \omega^i \models \alpha \).
* Common abbreviations;
  - \( \Diamond \alpha \equiv (\text{TRUE} U \alpha) \)
  - \( \square \alpha \equiv (\alpha U \text{FALSE}) \)
  - \( \alpha W \beta \equiv (\alpha U \beta) \lor \Box \alpha \)
Planning as LTL-SAT

The set of valid plans can be encoded as an LTL formula:

- $\Box (a \rightarrow \bigwedge_{p \in \text{pre}(a)} p)$ (action implies prec.)
- $\Box (a \rightarrow \bigwedge_{p \in \text{add}(a)} \bigcirc p)$ (action implies eff.)
- $\Box (a \rightarrow \bigwedge_{p \in \text{del}(a)} \bigcirc \neg p)$ (frame axioms)
- $\Box (\neg p \land \bigwedge\{a \mid p \in \text{add}(a)\} \neg a \rightarrow \bigcirc \neg p)$ (single action per step)
- $\Box (p \land \bigwedge\{a \mid p \in \text{del}(a)\} \neg a \rightarrow \bigcirc p)$ (init state)
- $\Diamond \bigwedge_{p \in \text{G}} p$ (eventually goal)
Infinite-State Systems

* The undecidability problem.
  - Counter machines.
* Petri nets.
* Timed & hybrid automata.
$k$-Counter Machines

* Deterministic finite program with $k$ counters of infinite capacity.

* Typical instruction set:
  - Set $c_i$ to 0.
  - Increment or decrement $c_i$ by 1.
  - goto $n$.
  - if $c_i = 0$ goto $n$ else goto $n'$.
  - Halt.

* Halting of $k$-CMs is undecidable for $k \geq 2$.

1. clear $c_2$
2. if $c_1 = 0$ goto 7
3. dec $c_1$
4. inc $c_2$
5. inc $c_2$
6. goto 2
7. if $c_2 = 0$ goto 11
8. dec $c_2$
9. inc $c_1$
10. goto 7
11. …
Petri Nets

* Directed bipartite graph of places and transitions.
  - A marking assigns each place \( n \in \mathbb{N} \) tokens.
  - Transition \( t \) applicable at marking \( n \) iff \( \forall p \in \bullet t, m(p) > 0 \) and leads to marking \( m' = m - \bullet t + t^* \).

* Marking reachability for Petri nets is decidable.
Timed Automata

* Non-deterministic finite automata with real-valued clocks.
  - Transition guards: Linear conditions on clocks and clock differences.
  - Transitions can reset clocks.
  - Clocks evolve at uniform rate.
* Reachability is PSPACE-complete for a single automaton.