Implementing Statecharts in PROMELA/SPIN

Erich Mikk

Christian Albrechts University Kiel, Germany

Joint work with

Gerard J. Holzmann, Yassine Lakhnech, Michael Siegel

WIFT98, Boca Raton, October 1998

Outline

1. Introduction
2. The Statecharts language
3. The STATEMATE semantics of statecharts
4. Extended hierarchical automata
5. Translating hierarchical automata to PROMELA
6. Conclusion

Introduction

WANTED: a general framework for translating statecharts to input language of any transition system based model-checker.

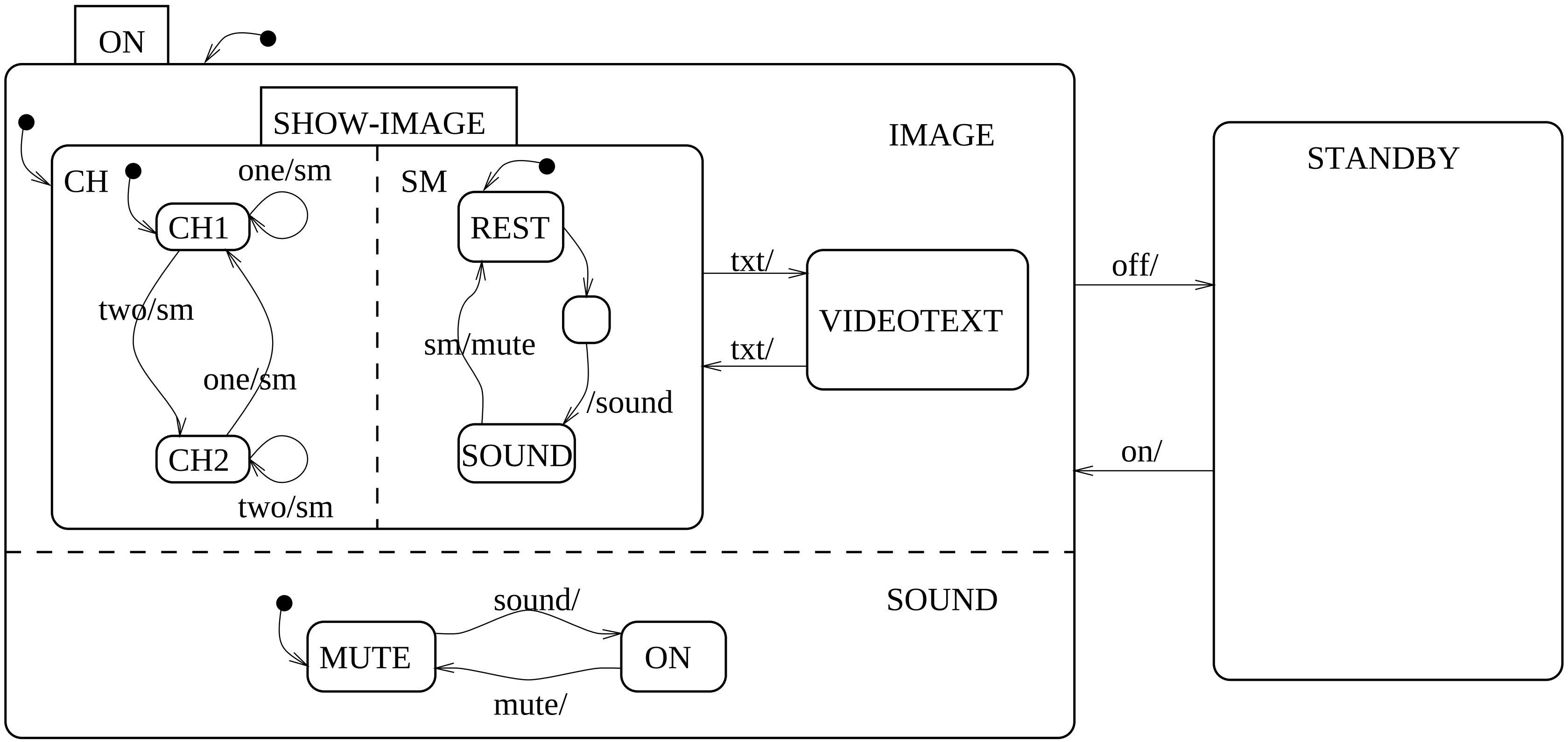
Given a model-checker MC, we look for a translation Tr such that:

- We can use MC to verify temporal properties of statecharts.
- Whenever a property is verified with MC, it must also hold for the source statechart.

Tr should

- preserve the **parallel structure** of the statechart,
- exploit the **hierarchy** of the statechart,
- produce **efficient representation**.

Example



ON
SHOW-IMAGE
IMAGE
CH
CH1
CH2
one/sm
two/sm
one/sm
two/sm
SM
REST
sm/mute
/sound
SOUND
txt/
txt/
VIDEOTEXT
off/
on/
STANDBY
SOUND
MUTE
ON
sound/
mute/

The STATEMATE semantics of statecharts

D. Harel and A. Naamad
The STATEMATE Semantics of Statecharts. 1996

The semantics is given in two steps: first transitions are transformed into *full compound transitions* (full CTs), then the *step relation* is defined.

The step consists on synchronously firing all the transitions in a so-called *maximal non-conflicting set of transitions*.

E. Mikk, Y. Lakhnech, C. Petersohn, and M. Siegel
On formal semantics of Statecharts as supported by STATEMATE. 1997

We formalize the full CT's language and the step relation.
Now a *transition system* is associated to each statechart.

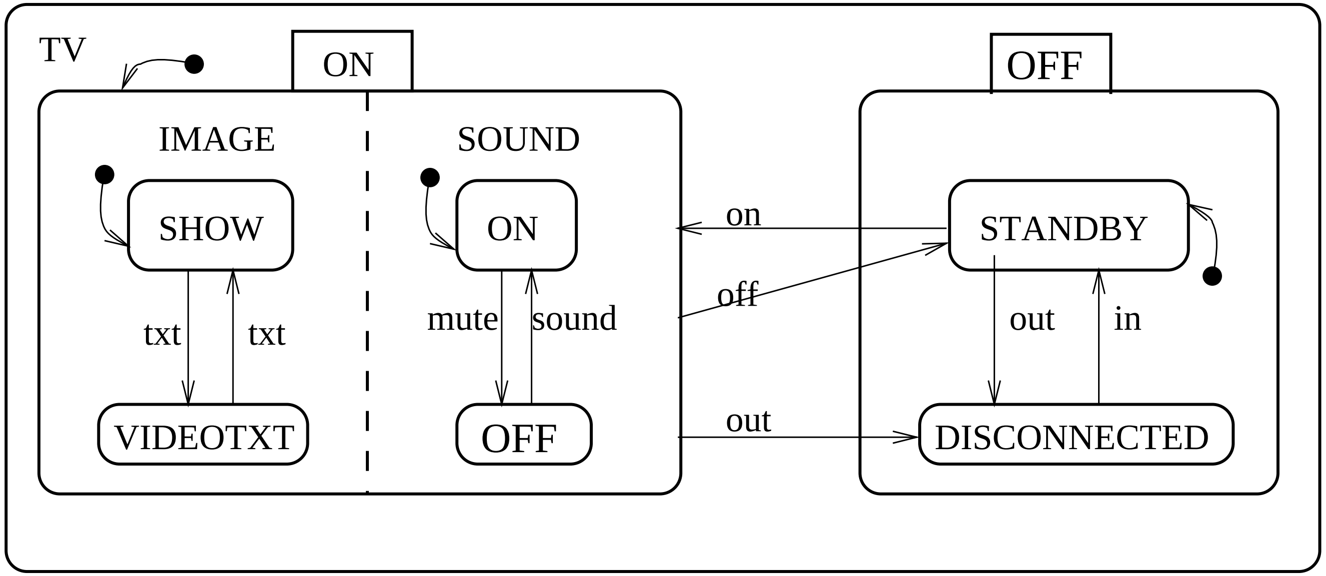
E. Mikk, Y. Lakhnech, and M. Siegel
Hierarchical automata as model for statecharts. 1997

"Simple" operational semantics of statecharts.

E. Mikk, Y. Lakhnech, M. Siegel, and G. Holzmann
Implementing Statecharts in PROMELA/SPIN

The STATEMATE semantics of statecharts

DIFFICULTY: how to reason *structurally* in presence of inter-level transitions?



Harel&Naamad associate *scope* to every transition.
Our idea is to lift transitions to the uppermost states that are affected when the transition is taken.

Extended hierarchical automata, motivation

Automata-theoretic model for statecharts.

Concurrency and communication: A parallel composition operator with maximal parallelism semantics, broadcast communication.

Hierarchy: An automaton state can be refined to another automaton or parallel composition of automata.

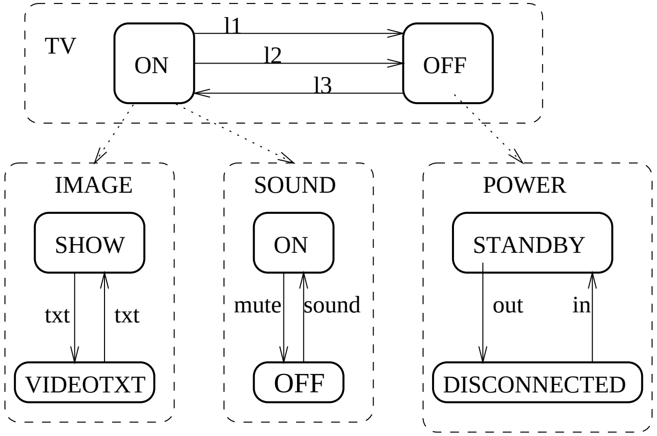
Simple transition syntax: transitions have single source and single target states; inter-level transitions are not allowed.

ALSO

Powerful modeling: Capable to model *inter-level transitions* and transitions with *multiple sources and multiple targets.*

Avoid explosion of *states, transitions or conditions* while translating from statecharts to hierarchical automata.

Extended hierarchical automata, example



l1 = <{},off,{},{STANDBY}>

l2 = <{},out,{},{DISCONNECTED}>

l3 = <{STANDBY},on,{}{SHOW,SOUND.ON}>

Extended hierarchical automaton HA is a triple (F, E, γ) :

A set of *mutually distinct sequential automata*:
$F = \{TV, IMAGE, SOUND, POWER\}$.

Sequential automaton $A \in F$ is a 4-tuple (Σ, s_0, L, δ).

- Σ is the set of states of A,
- $s_0 \in \Sigma$ is the initial state of A,
- L is the set of transition labels,
- $\delta \subseteq \Sigma \times L \times \Sigma$ are of transitions of A.

Composition function γ on F to express *refinement and parallelism*:
$\{ON \mapsto \{IMAGE, SOUND\}, OFF \mapsto \{POWER\}\} \cup \ldots$

Configuration C of γ
is a maximum set of states that can be simultaneously active, e.g.,
$C = \{ON, SHOW, SOUND.OFF\}$.

Transition labels $l \in L$ are tuples $l = (sr, ex, ac, td)$, where

sr is sub-configuration below source (source restriction),

td is sub-configuration below target (target determinator),

ex is a proposition over $E \times \bigcup \Sigma_A$,

$ac \subseteq E$ is a set of events.

Semantics of EHA

In the following let $EHA = (F, E, \gamma)$ be an extended hierarchical automaton. The semantics of EHA is a Kripke structure $\mathbf{K} = (\mathbf{S}, \mathbf{s_0}, \stackrel{STEP}{\longrightarrow})$, where

- $\mathbf{S} = Conf(\gamma) \times E$ is the set of states of $\mathbf{K}$,
  a state of $\mathbf{K}$ is also called *status*,
- $\mathbf{s_0} \in \mathbf{S}$ is the initial state
- $\stackrel{STEP}{\longrightarrow} \subseteq \mathbf{S} \times \mathbf{S}$ is the *transition relation* of $\mathbf{K}$ that will be defined in the sequel.

Preliminary definitions

Let (C, E) be a status of $\mathbf{K}$.
A transition $t = (s, (sr, ex, ac, td), s')$ of a sequential automata $A \in F$ is *enabled* in the status (C, E):

$$enabled_{(\mathsf{C},\mathsf{E})}(t) \Leftrightarrow (s \in \mathsf{C} \wedge sr \subseteq \mathsf{C} \wedge (\mathsf{C}, \mathsf{E}) \models ex).$$

Start rule

CLOSED SYSTEMS:

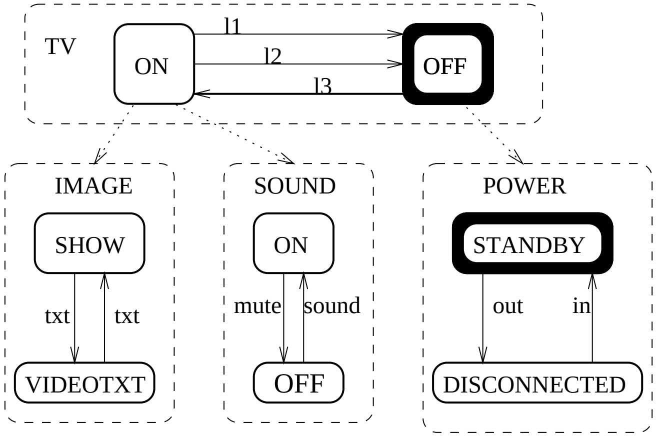
$$\frac{\gamma_{root} :: (\mathsf{C}, \mathsf{E}) \rightarrow (\mathsf{C}', \mathsf{E}')}{(\mathsf{C}, \mathsf{E}) \stackrel{STEP}{\longrightarrow} (\mathsf{C}', \mathsf{E}')}$$

OPEN SYSTEMS:

$$\frac{\gamma_{root} :: (\mathsf{C}, \mathsf{E}) \rightarrow (\mathsf{C}', \mathsf{E}') \wedge \mathsf{E}' \subseteq \mathsf{E}''}{(\mathsf{C}, \mathsf{E}) \stackrel{STEP}{\longrightarrow} (\mathsf{C}', \mathsf{E}'')}$$

Progress rule

$$\frac{\{s\} = \mathsf{C} \cap \Sigma_A \qquad \exists\, tr \in \delta_A . enabled_{(\mathsf{C},\mathsf{E})}(tr) \wedge tr = (s, (sr, ex, ac, td), s')}{A :: (\mathsf{C}, \mathsf{E}) \to (\{s'\} \cup td, ac)}$$



l1 = <{},off,{},{STANDBY}>

l2 = <{},out,{},{DISCONNECTED}>

l3 = <{STANDBY},on,{}{SHOW,SOUND.ON}>

Status (C, E) at the beginning of the step:
$(\{OFF, STANDBY\}, \{on, out\})$

Rule instantiation for TV:
active state $s = OFF$
$OFF \xrightarrow{l3} ON$ is enabled in (C, E)

Result $(\mathsf{C}', \mathsf{E}')$:
$\mathsf{C}' = \{TV.ON, SHOW, SOUND.ON\}$
$\mathsf{E}' = \emptyset$

Composition rule

$\{s\} = \mathsf{C} \cap \Sigma_A$

$\forall\, tr \in \delta_A . tr = (s, l, s') \Rightarrow \neg\, enabled_{(\mathsf{C},\mathsf{E})}(tr)$

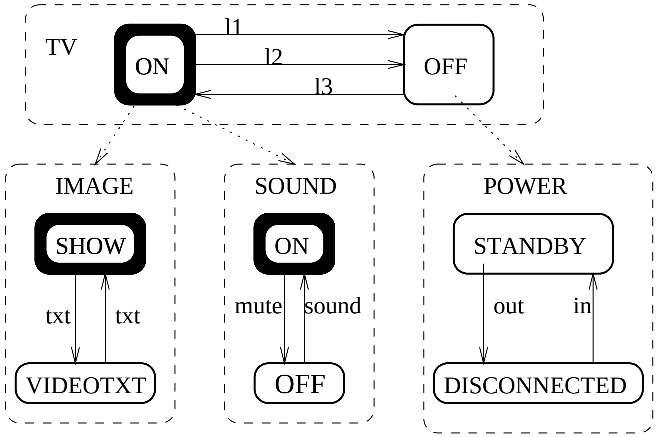
$\gamma(s) = \{A_1, \ldots, A_m\} \neq \emptyset$

$A_1 :: (\mathsf{C},\mathsf{E}) \rightarrow (\mathsf{C}'_1, \mathsf{E}'_1)$

$\ldots$

$A_m :: (\mathsf{C},\mathsf{E}) \rightarrow (\mathsf{C}'_m, \mathsf{E}'_m)$

$A :: (\mathsf{C},\mathsf{E}) \rightarrow (\{s\} \cup \mathsf{C}'_1 \cup \ldots \cup \mathsf{C}'_m, \mathsf{E}'_1 \cup \ldots \cup \mathsf{E}'_m)$



Status (C,E) at the beginning of the step:
$(\{TV.ON, SHOW, SOUND.ON\}, \{txt, mute\})$

Rule instantiation for TV:

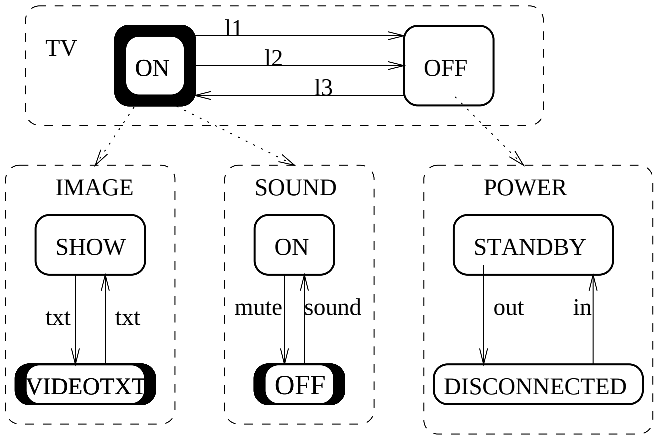
$IMAGE :: (\{TV.ON, SHOW, SOUND.ON\}, \{txt, mute\}) \rightarrow (\{VIDEOTXT\}, \emptyset)$

$SOUND :: (\{TV.ON, SHOW, SOUND.ON\}, \{txt, mute\}) \rightarrow (\{OFF\}, \emptyset)$

Result $(\mathsf{C}', \mathsf{E}')$:
$\mathsf{C}' = \{TV.ON, VIDEOTXT, OFF\}$
$\mathsf{E}' = \emptyset$

Stuttering rule

$$\frac{\{s\} = \mathsf{C} \cap \Sigma_A \quad Basic_\gamma(s) \quad \forall\, tr \in \delta_A . tr = (s, l, s') \Rightarrow \neg\, enabled_{(\mathsf{C},\mathsf{E})}(tr)}{A :: (\mathsf{C}, \mathsf{E}) \to (\{s\}, \emptyset)}$$



l1 = <{},off,{},{STANDBY}>

l2 = <{},out,{},{DISCONNECTED}>

l3 = <{STANDBY},on,{}{SHOW,SOUND.ON}>

Status (C, E) at the beginning of the step:
$(\{TV.ON, VIDEOTXT, OFF\}, \{txt\})$

Rule instantiation for $SOUND$:
$SOUND :: (\{TV.ON, VIDEOTXT, OFF\}, \{txt\}) \to$
$(\{OFF\}, \emptyset)$

Composition rule for TV:
$TV :: (\{TV.ON, SHOW, SOUND.ON\}, \{txt\}) \to$
$(\{TV.ON, SHOW, OFF\}, \emptyset)$

Result $(\mathsf{C}', \mathsf{E}')$:
$\mathsf{C}' = \{TV.ON, SHOW, OFF\}$
$\mathsf{E}' = \emptyset$

The model-checker SPIN

Process meta language PROMELA

Concurrency by interleaving.

Communication between processes via:

- Shared variables.
- Synchronous and asynchronous channels.

Atomic statement for resolving race conditions.

Properties can be expressed as LTL formula or omega automata.

Translating hierarchical automata to PROMELA

Events are implemented as boolean variables,

States are implemented as enumeration type variables,

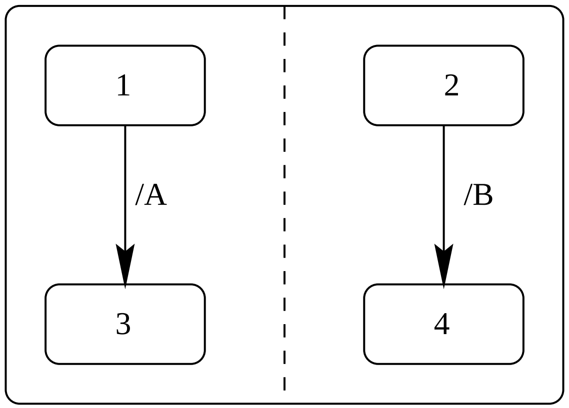
Transitions are implemented as conditions over state and event variables and as assignments to the same variables.

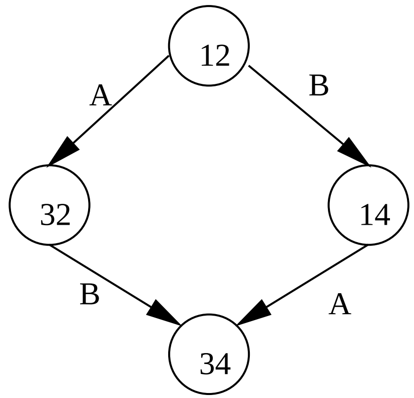
Hierarchy is implemented by recursive descend from root down to the first enabled transition; if one is found then the lower part of the state hierarchy is not considered in the current step.

Concurrency

- *Simultaneous access* of events and states by parallel components of the hierarchical automata is implemented by two copies of state and event variables: one copy to fix the beginning of the step execution and another copy to collect (partial) results during the step execution.
- *Communication.* Since parallel components of the hierarchical automata do not communicate with each other during a step, the compiler may choose the execution order for parallel components.

Implementation of parallelism

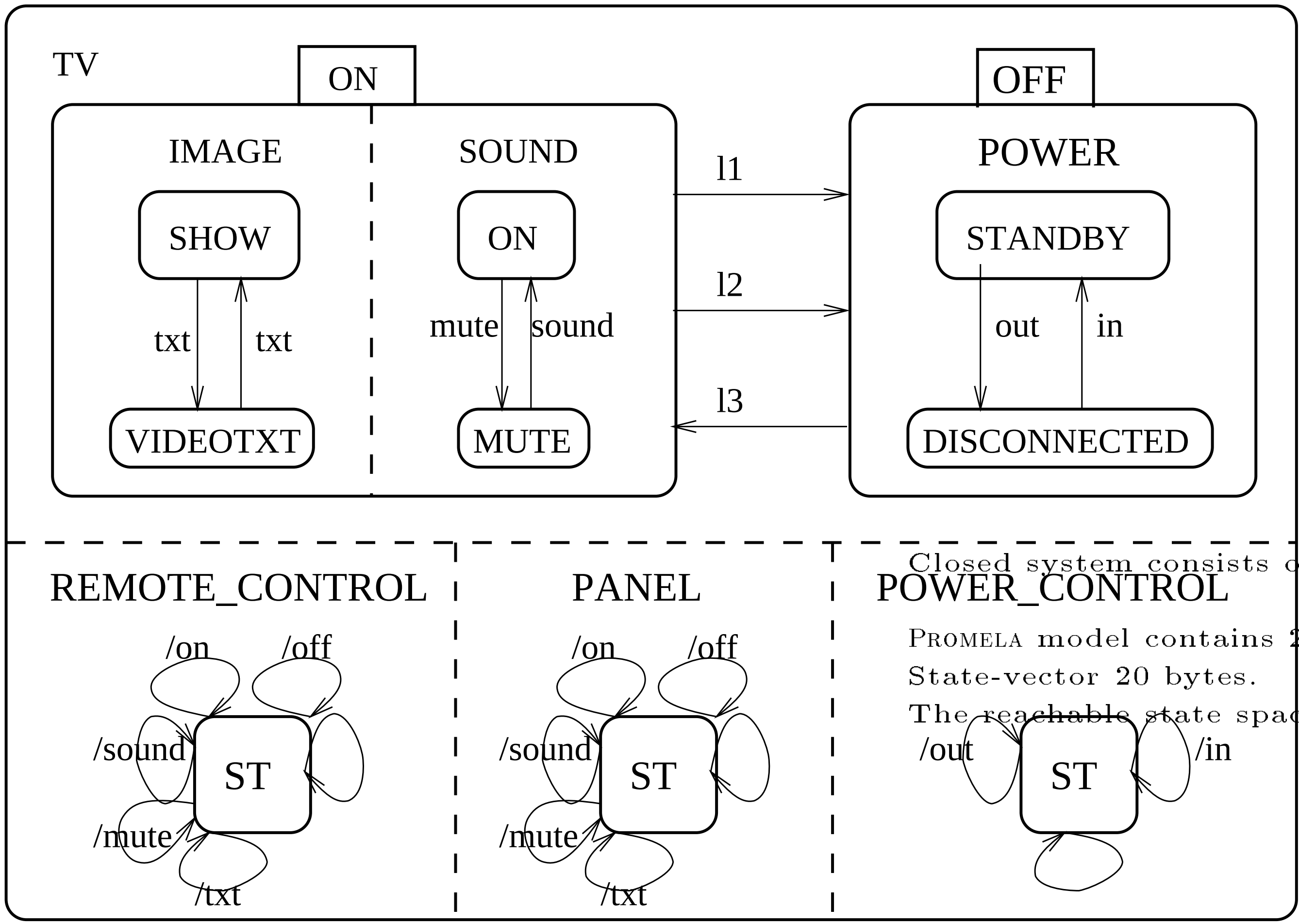




({1,2},0) $\xrightarrow{\text{STEP}}$ ({3,4},{A,B})

Modeling statecharts parallelism using interleaving.

TV-set case study



l1 = <{},off,{},{STANDBY}>

l2 = <{},out,{},{DISCONNECT

l3 = <{STANDBY},on,{},{SHOV

Closed system consists of 7 automata and 7 events.

Promela model contains 24 boolean variables.

State-vector 20 bytes.

The reachable state space is 80 states.

There is counterintuitive behaviour, the following properties do not hold:

- $\Box(out \Rightarrow \odot\ disconnected)$,
- $\Box(off \wedge tv.on \Rightarrow \odot\ standby)$.

Production cell case study

German KORSO project initiative, 36 contributions, 2 with STATEMATE.

We use a specification provided by University of Oldenburg (W.Damm group).

The statecharts specification contains 30 parallel automata, 30 events.

PROMELA model contains 60 boolean and 77 enumeration type variables.

The reachable state space is $3.08191e + 06$ states.

- Default search: 266MB memory, 1:37:21h,
- Graph encoding technique: 111MB memory, 2:34:08h.

Conclusion

Hierarchical automata allow for:

- **Structure preserving translation** from statecharts to PROMELA, a prototype compiler exists.
- **Generic interpretation.** Our semantics can be used to translate statecharts to other languages, too.

Verification with SPIN **is feasible,** however partial order reduction techniques of SPIN do not help to avoid the state explosion problem in statecharts.

Further applications of our semantics are simulation, code generation and test sequence generation.

Further work: **Language extension:** include timers, data variables, history concept, activity charts.

- **Compositional** semantics and compositional reasoning,
- **Application** of abstraction and symmetry reduction to statecharts,
- **Comparison** with other model checkers, e.g. COSPAN, SMV.