Problem Solving with Model Checking Techniques

Jaco van de Pol and Michael Weber

June 12, 2011
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Model Checking in a Nutshell</td>
</tr>
<tr>
<td>2</td>
<td>Planning Example: Sokoban</td>
</tr>
<tr>
<td>3</td>
<td>LTSmin Tool Architecture</td>
</tr>
<tr>
<td>4</td>
<td>Symbolic Algorithms</td>
</tr>
<tr>
<td>5</td>
<td>Multi-Core Algorithms</td>
</tr>
</tbody>
</table>
# Table of Contents

1. Model Checking in a Nutshell
2. Planning Example: Sokoban
3. LTSmin Tool Architecture
4. Symbolic Algorithms
5. Multi-Core Algorithms
Model Checking in a Nutshell

- Specification of system: logical formula $\varphi$
- Implementation of system: Kripke structure
- Question: Does the system meet its specification?

- Applications: hardware, software, wetware
- Method: (Variations of) Graph Reachability
What is Model Checking?

Model Checking

- Check if a given model satisfies a given property
- Promise: Automatic answer to combinatorial questions
What is Model Checking?

**Model Checking**

- Check if a **given model** satisfies a **given property**
- **Promise**: Automatic answer to combinatorial questions

**Models:**

**discrete dynamics**

- software / hardware / embedded systems
- communicating concurrent components
- biological systems, intra/inter cell level
What is Model Checking?

Model Checking

- Check if a **given model** satisfies a **given property**
- **Promise**: Automatic answer to combinatorial questions

**Models:**
- **discrete dynamics**
  - software / hardware / embedded systems
  - communicating concurrent components
  - biological systems, intra/inter cell level

**Properties:**
- of transition graphs
  - invariants, assertions, absence of errors
  - absence / presence of event orderings
  - complicated fairness restrictions possible
Applications

Impressive Applications ............. Turing Award 2007

- Numerous case studies have been published:
  - communication and security protocols
  - embedded controllers, e.g. elevators, railways, cars
  - concurrent and distributed algorithms
  - Biology: signaling pathways, gene regulation, differentiation
Applications

Impressive Applications ............... Turing Award 2007

- Numerous case studies have been published:
  - communication and security protocols
  - embedded controllers, e.g. elevators, railways, cars
  - concurrent and distributed algorithms
  - Biology: signaling pathways, gene regulation, differentiation

- Leading industries rely on model checking for quality control:
  - Intel/IBM’s processors go through extensive model checking (they report that it replaced a considerable amount of testing)
  - Microsoft’s Static Device Verifier is part of the WDK (3rd party device drivers are checked for interface compliance)
Impressive Applications ............... Turing Award 2007

- Numerous case studies have been published:
  - communication and security protocols
  - embedded controllers, e.g. elevators, railways, cars
  - concurrent and distributed algorithms
  - Biology: signaling pathways, gene regulation, differentiation

- Leading industries rely on model checking for quality control:
  - Intel/IBM’s processors go through extensive model checking (they report that it replaced a considerable amount of testing)
  - Microsoft’s Static Device Verifier is part of the WDK (3rd party device drivers are checked for interface compliance)

But still, model checking is...

- Not built into CASE tools, despite its “push-button” nature
- Not available to the average (SME-type) software engineer
### Table of Contents

1. Model Checking in a Nutshell
2. Planning Example: Sokoban
3. LTSmin Tool Architecture
4. Symbolic Algorithms
5. Multi-Core Algorithms
Sokoban as you know it
States and Transitions

- **States:**
  - View each location on the board as a variable $x_{i,j}$
  - Possible values: $x_{i,j} \in \{wall, man, block, empty\}$

- **Transitions** — distinguish moves and pushes in four directions
  - **Move right:**
    - if $x_{i,j} = man$ and $x_{i,j+1} = empty$ then set $x_{i,j} := empty$ and $x_{i,j+1} := man$.
  - **Push down:**
    - if $x_{i,j} = man$ and $x_{i,j+1} = block$ and $x_{i+2,j} = empty$ then set:
      - $x_{i,j} := empty$
      - $x_{i+1,j} := man$
      - $x_{i+2,j} := block$. 
States and Transitions

- **States:**
  - View each location on the board as a variable $x_{i,j}$
  - Possible values: $x_{i,j} \in \{\text{wall, man, block, empty}\}$

- **Transitions – distinguish moves and pushes in four directions**
  - Move right:
    - if $x_{i,j} = \text{man}$ and $x_{i,j+1} = \text{empty}$
    - then set $x_{i,j} := \text{empty}$ and $x_{i,j+1} := \text{man}$.
  - Push down:
    - if $x_{i,j} = \text{man}$ and $x_{i+1,j} = \text{block}$ and $x_{i+2,j} = \text{empty}$
    - then set: $x_{i,j} := \text{empty}$, $x_{i+1,j} := \text{man}$ and $x_{i+2,j} := \text{block}$. 
Modelling Sokoban as a Transition System

States and Transitions

- **States:**
  - View each location on the board as a variable \( x_{i,j} \)
  - Possible values: \( x_{i,j} \in \{wall, man, block, empty\} \)

- **Transitions – distinguish moves and pushes in four directions**
  - **Move right:**
    - if \( x_{i,j} = man \) and \( x_{i,j+1} = empty \)
    - then set \( x_{i,j} := empty \) and \( x_{i,j+1} := man \).
  - **Push down:**
    - if \( x_{i,j} = man \) and \( x_{i+1,j} = block \) and \( x_{i+2,j} = empty \)
    - then set: \( x_{i,j} := empty \), \( x_{i+1,j} := man \) and \( x_{i+2,j} := block \).

- **Initial state:** an assignment to all the \( x_{i,j} \)

- **Goal state:** If for all goal positions \((i,j): x_{i,j} = block\) then do special action \texttt{finish}
Next: Solve Sokoban with Brute Force

Reachability

- This is just the reachability problem
- Property: “Finish action is not reachable”
- Counter-example: trace to a finish-action
Next: Solve Sokoban with Brute Force

Reachability

- This is just the reachability problem
- Property: “Finish action is not reachable”
- Counter-example: trace to a finish-action

Brute Force Exploration

- On-the-fly:
  - start with the initial state
  - expand newly encountered states
  - stop when the goal is reached
- Breadth-first strategy guarantees the shortest solution
- Limitations:
  - Only feasible for about $10^9$ states
  - General: no utilization of specific structure of Sokoban
### Small Problem: Exponential Growth

<table>
<thead>
<tr>
<th>screen</th>
<th>states</th>
</tr>
</thead>
<tbody>
<tr>
<td>107</td>
<td>10,165</td>
</tr>
<tr>
<td>1001</td>
<td>127,509</td>
</tr>
<tr>
<td>387</td>
<td>1,235,214</td>
</tr>
<tr>
<td>372</td>
<td>10,992,856</td>
</tr>
<tr>
<td>792</td>
<td>117,434,655</td>
</tr>
<tr>
<td>747</td>
<td>1,307,942,326</td>
</tr>
<tr>
<td>38</td>
<td>12,197,960,188</td>
</tr>
<tr>
<td>754</td>
<td>308,479,382,084</td>
</tr>
<tr>
<td>2</td>
<td>4,748,854,893,784</td>
</tr>
</tbody>
</table>
Small Problem: Exponential Growth...

<table>
<thead>
<tr>
<th>screen</th>
<th>states</th>
</tr>
</thead>
<tbody>
<tr>
<td>107</td>
<td>10,165</td>
</tr>
<tr>
<td>1001</td>
<td>127,509</td>
</tr>
<tr>
<td>387</td>
<td>1,235,214</td>
</tr>
<tr>
<td>372</td>
<td>10,992,856</td>
</tr>
<tr>
<td>792</td>
<td>117,434,655</td>
</tr>
<tr>
<td>747</td>
<td>1,307,942,326</td>
</tr>
<tr>
<td>38</td>
<td>12,197,960,188</td>
</tr>
<tr>
<td>754</td>
<td>308,479,382,084</td>
</tr>
<tr>
<td>2</td>
<td>4,748,854,893,784</td>
</tr>
<tr>
<td>376</td>
<td>1,066,317,080,868,510,000</td>
</tr>
<tr>
<td>45</td>
<td>1,195,159,588,147,710,000,000</td>
</tr>
<tr>
<td>71</td>
<td>1,238,482,287,687,790,000,000,000</td>
</tr>
<tr>
<td>551</td>
<td>1,654,592,279,840,460,000,000,000</td>
</tr>
<tr>
<td>48</td>
<td>40,215,766,407,984,300,000,000,000,000</td>
</tr>
<tr>
<td>22</td>
<td>3,531,895,015,833,180,000,000,000,000,000</td>
</tr>
<tr>
<td>10</td>
<td>12,864,741,234,813,200,000,000,000,000,000,000,000,000,000,000</td>
</tr>
<tr>
<td>558</td>
<td>218,612,674,527,952,000,000,000,000,000,000,000,000,000,000,000,000,000</td>
</tr>
<tr>
<td>778</td>
<td>9,341,745,200,574,070,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000</td>
</tr>
</tbody>
</table>
A Glimpse of Hope: Transition Locality

**Locality**

- Transitions may depend on a part of the state vector only.
- In Sokoban, every transition depends on 2 or 3 variables; independent of the size of the Sokoban screen.
- In general, if you learn one transition and (statically) know the dependency matrix, then you can infer many more transitions.
A Glimpse of Hope: Transition Locality

Locality

- Transitions may depend on a part of the state vector only
- In Sokoban, every transition depends on 2 or 3 variables; independent of the size of the Sokoban screen
- In general, if you learn one transition and (statically) know the dependency matrix, then you can infer many more transitions

Example matrix for a Sokoban instance (fragment)

<p>| +++-----  |
| +-------  |
| +-------  |</p>
<table>
<thead>
<tr>
<th>+-------</th>
</tr>
</thead>
<tbody>
<tr>
<td>--------</td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td>---+++--</td>
</tr>
<tr>
<td>---++---</td>
</tr>
</tbody>
</table>

- The dependency matrix is sparse (good)
- The +'s are often close together
- Some +'s are far apart (2D board)
- Heuristic regrouping can help a lot (cf. BDD variable reordering)
**Exploiting Locality**

By locality, successor states are “much alike”:

**Locality helps Implementation**

- Compression schemes for storing sets of states
- Incremental hashing / storage / compression / etc.
- Communicate diffs only, save bandwidth on clusters
Exploiting Locality

By locality, successor states are “much alike”:

### Locality helps Implementation

- Compression schemes for storing sets of states
- Incremental hashing / storage / compression / etc.
- Communicate diffs only, save bandwidth on clusters

### Locality helps Algorithms (orders of magnitude!)

- Cache intermediate results to save computations
- Store sets of states in **Binary Decision Diagrams**
- Apply transitions to **sets of states** symbolically
Mentioned later

- Multi-core and Grid implementations
- State space reduction (e.g. partial-order reduction)
- Symbolic Model Checking using BDDs
- Adapting the search order of reachability
- Effects of changing the search order:
  - the peak memory of intermediate BDDs is reduced
  - however, a shortest solution is not guaranteed
Alternatives

- Bounded model checking, based on SAT solving
  - Runs of increasing, fixed length, are encoded into one big satisfiability problem
  - Huge potential, but Sokoban runs are very long
## Alternatives

- **Bounded model checking, based on SAT solving**
  - Runs of increasing, fixed length, are encoded into one big satisfiability problem
  - Huge potential, but Sokoban runs are very long

- **Directed Model Checking**
  - Gives priority to transitions towards a “promising” direction
  - Here maybe: number of blocks already in correct position
### Alternatives

- **Bounded model checking, based on SAT solving**
  - Runs of increasing, fixed length, are encoded into one big satisfiability problem
  - Huge potential, but Sokoban runs are very long

- **Directed Model Checking**
  - Gives priority to transitions towards a “promising” direction
  - Here maybe: number of blocks already in correct position

- **Ad Hoc techniques for (classes of) puzzles**
  - Avoid “deadlock” situations
  - Use high-level planning, e.g. recognize rooms
Alternatives

- Bounded model checking, based on SAT solving
  - Runs of increasing, fixed length, are encoded into one big satisfiability problem
  - Huge potential, but Sokoban runs are very long
- Directed Model Checking
  - Gives priority to transitions towards a “promising” direction
  - Here maybe: number of blocks already in correct position
- Ad Hoc techniques for (classes of) puzzles
  - Avoid “deadlock” situations
  - Use high-level planning, e.g. recognize rooms
- Here is where model checking might learn from planning!
Beyond Reachability

More complicated planning problems

- Avoid unwanted situations (still a reachability problem)
  - in LTL logic: $F \text{Good}$ becomes $(\neg \text{Bad}) \text{Until Good}$
  - alternatively: restrict the transition relation

- Associate costs with transitions (e.g.: minimal pushes)
  - this requires an adapted search strategy
Beyond Reachability

More complicated planning problems

- Avoid unwanted situations (still a reachability problem)
  - in LTL logic: $F \text{Good}$ becomes $(\neg \text{Bad}) \text{Until Good}$
  - alternatively: restrict the transition relation
- Associate costs with transitions (e.g.: minimal pushes)
  - this requires an adapted search strategy
- Take into account real time
  - UPPAAL: real-time model checker; used for HRT scheduling
- Compute (optimal) cyclic schedules (mean pay-off games)
  - Liveness in logic: $GF \text{Good}$, reachability infinitely often
Beyond Reachability

More complicated planning problems

- Avoid unwanted situations (still a reachability problem)
  - in LTL logic: \( F \text{ Good} \) becomes \((\neg \text{Bad}) \text{ Until Good}\)
  - alternatively: restrict the transition relation
- Associate costs with transitions (e.g.: minimal pushes)
  - this requires an adapted search strategy
- Take into account real time
  - UPPAAL: real-time model checker; used for HRT scheduling
- Compute (optimal) cyclic schedules (mean pay-off games)
  - Liveness in logic: \( GF \text{ Good} \), reachability infinitely often
- Plan in the presence of uncertainty (two-player games)
  - and/or graphs with loop restrictions: parity games, \( \mu \)-calculus
  - can even take into account stochastic environment (MDP)
Table of Contents

1. Model Checking in a Nutshell
2. Planning Example: Sokoban
3. LTSmin Tool Architecture
4. Symbolic Algorithms
5. Multi-Core Algorithms
LTSmin Tool Architecture
Great Abstractions are Cheap!

- Automatic model translations are not good enough
- Separate languages and tools via a clean interface (API)
- API should be simple: allow many different languages
- API should be rich: expose structure, enable algorithms
Great Abstractions are Cheap!

- Automatic model translations are not good enough
- Separate languages and tools via a clean interface (API)
- API should be simple: allow many different languages
- API should be rich: expose structure, enable algorithms
Interface based on a **Partitioned Next-State function**

- **State**: Fixed-size vector of integers
  \[
  \langle 3, 5, 5, 4, 1, 3 \rangle_{IV}
  \]

- **Partitioned transition relation**
  \[
  \rightarrow = \bigcup_{i} \rightarrow_{i}
  \]
PINS in a Nutshell

Interface based on a Partitioned Next-State function

- State: Fixed-size vector of integers
- Partitioned transition relation
- Dependency Matrix $[D]_{N \times K}$:
  - Language module guarantees: if $D_{i,j} = -1$, then transition group $\rightarrow_i$ is independent of state slot $j$.
  - Matrix: statically known (currently)
  - Language module may over-approximate dependencies

\[
\langle 3, 5, 5, 4, 1, 3 \rangle_{IV} \quad \rightarrow = \bigcup_{i} \rightarrow_i
\]

\[
\begin{bmatrix}
X_1 & \cdots & X_j & \cdots & X_N \\
\rightarrow_1 & + & \cdots & - \\
\vdots & - & \cdots & + \\
\rightarrow_K & + & \cdots & +
\end{bmatrix}
\]
Example Dependency Matrix

```c
int x=7;
process p1() {
  do
  ::{x>0 => x--;y++}
  ::{x>0 => x--;z++}
  od }
```

Default Matrix

\[
\begin{bmatrix}
  x & y & z \\
  p_1 & + & + \\
  p_2 & + & + \\
  p_3 & + & + \\
\end{bmatrix}
\]

Better Matrix

\[
\begin{bmatrix}
  x & y & z \\
  p_1 & .1 & + \\
  p_1 & .2 & + \\
  p_2 & .1 & + \\
  p_2 & .2 & + \\
  p_3 & .1 & - \\
  p_3 & .2 & - \\
\end{bmatrix}
\]

state = \langle 7, 3, 9 \rangle

IV

\langle 7, 3, 9 \rangle IV

p_1 .1 \rightarrow \langle 6, 4, 9 \rangle IV

\langle 7, 3, 9 \rangle IV

p_1 .2 \rightarrow \langle 6, 4, 9 \rangle IV

\langle 7, 3, 9 \rangle IV

p_3 .2 \rightarrow \langle 7, 4, 8 \rangle IV

\langle 7, 3, 9 \rangle IV

p_3 .2 \rightarrow \langle 7, 4, 8 \rangle IV

\langle 7, 3, 9 \rangle IV
### Example Dependency Matrix

```plaintext
int x=7;
process p1() {
  do
  ::{x>0 => x--;y++}
  ::{x>0 => x--;z++}
  od }

int y=3;
process p2() {
  do
  ::{y>0 => y--;x++}
  ::{y>0 => y--;z++}
  od }

int z=9;
process p3() {
  do
  ::{z>0 => z--;x++}
  ::{z>0 => z--;y++}
  od }
```
Example Dependency Matrix

```
int x=7;
process p1() {
do
  ::{x>0 => x--;y++}
  ::{x>0 => x--;z++}
od }

int y=3;
process p2() {
do
  ::{y>0 => y--;x++}
  ::{y>0 => y--;z++}
od }

int z=9;
process p3() {
do
  ::{z>0 => z--;x++}
  ::{z>0 => z--;y++}
od }
```

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>p2</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>p3</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>
### Example Dependency Matrix

**Default Matrix**

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>p1</strong></td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td><strong>p2</strong></td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td><strong>p3</strong></td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>

**Better Matrix**

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>p1.1</strong></td>
<td>+</td>
<td>+</td>
<td>−</td>
</tr>
<tr>
<td><strong>p1.2</strong></td>
<td>+</td>
<td>−</td>
<td>+</td>
</tr>
<tr>
<td><strong>p2.1</strong></td>
<td>+</td>
<td>+</td>
<td>−</td>
</tr>
<tr>
<td><strong>p2.2</strong></td>
<td>−</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td><strong>p3.1</strong></td>
<td>+</td>
<td>−</td>
<td>+</td>
</tr>
<tr>
<td><strong>p3.2</strong></td>
<td>−</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>

```plaintext
int x=7;
process p1() {
  do
    ::{x>0 => x--;y++}
    ::{x>0 => x--;z++}
  od }

int y=3;
process p2() {
  do
    ::{y>0 => y--;x++}
    ::{y>0 => y--;z++}
  od }

int z=9;
process p3() {
  do
    ::{z>0 => z--;x++}
    ::{z>0 => z--;y++}
  od }
```
### Example Dependency Matrix

```plaintext
int x=7;
process p1() {
  do
    ::{x>0 => x--;y++}
    ::{x>0 => x--;z++}
  od }

int y=3;
process p2() {
  do
    ::{y>0 => y--;x++}
    ::{y>0 => y--;z++}
  od }

int z=9;
process p3() {
  do
    ::{z>0 => z--;x++}
    ::{z>0 => z--;y++}
  od }
```

### Better Matrix

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1.1</td>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>p1.2</td>
<td>+</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>p2.1</td>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>p2.2</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>p3.1</td>
<td>+</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>p3.2</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>

State: \( \langle 7, 3, 9 \rangle_{IV} \)

<table>
<thead>
<tr>
<th>State</th>
<th>Transition</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \langle 7, 3, 9 \rangle_{IV} )</td>
<td>( p1.1 )</td>
<td>( \langle 6, 4, 9 \rangle_{IV} )</td>
</tr>
<tr>
<td>( \langle 7, 3, * \rangle_{IV} )</td>
<td>( p1.1 )</td>
<td>( \langle 6, 4, * \rangle_{IV} )</td>
</tr>
<tr>
<td>( \langle 7, 3, 9 \rangle_{IV} )</td>
<td>( p3.2 )</td>
<td>( \langle 7, 4, 8 \rangle_{IV} )</td>
</tr>
<tr>
<td>( \langle *, 3, 9 \rangle_{IV} )</td>
<td>( p3.2 )</td>
<td>( \langle *, 4, 8 \rangle_{IV} )</td>
</tr>
</tbody>
</table>
PINS / LTSmin

Input Language

mCRL2
Process algebra

Promela
SPIN / NIPS-vm

DVE (BEEM)

Transition Caching

Static Reordering

(P.O. Reduction)

Distributed Generation

Multi-core Reachability

Symbolic Reachability

PINS

Pins2pins Wrappers

Reachability Tools
Basic functions (algorithms call language modules)

- **GetMatrix**: returns the dependency matrix \([D]_{N \times K}\)
- **InitState()**: returns the initial state vector
- **NextState(i,s)**: successors of state \(s\) in transition group \(i\)
Table of Contents

1 Model Checking in a Nutshell
2 Planning Example: Sokoban
3 LTSmin Tool Architecture
4 Symbolic Algorithms
5 Multi-Core Algorithms
Local Transition Caching
Caching Local Transitions (1)

- Recall local transition in specification:
  \[ p3.2: \text{atomic} \{ z > 0 \implies z--; y++ \} \]

- Dependency matrix row: \[ p3.2 \begin{bmatrix} 0 & 1 & 1 \end{bmatrix} \]

- Define projection: \( \pi_{p3.2} \langle x, y, z \rangle = \langle y, z \rangle \)
Recall local transition in specification:

\[
p3.2: \text{atomic } \{ \ z > 0 \rightarrow z--; y++ \ \}
\]

\[
x \ y \ z
\]

Dependency matrix row: \[ p3.2 [0 \ 1 \ 1] \]

Define projection: \[ \pi_{p3.2}(x, y, z) = (y, z) \]

Next, consider two consecutive calls to \( p3.2 \):

- first call: \( (x, y, z) \)
- successor: \( (x, y', z') \)
- project and store in cache: \( (y, z) \rightarrow (y', z') \)
Caching Local Transitions (1)

- Recall local transition in specification:
  \[ p3.2: \text{atomic} \{ z>0 \rightarrow z--; y++ \} \]
  \[
  \begin{array}{ccc}
  x & y & z \\
  \end{array}
  \]

- Dependency matrix row: \[ p3.2 \begin{bmatrix} 0 & 1 & 1 \end{bmatrix} \]

- Define projection: \[ \pi_{p3.2} \langle x, y, z \rangle = \langle y, z \rangle \]

- Next, consider two consecutive calls to \( p3.2 \):
  
  first call: \( \langle x, y, z \rangle \)
  
  successor: \( \langle x, y', z' \rangle \)
  
  project and store in cache: \( \langle y, z \rangle \rightarrow \langle y', z' \rangle \)
  
  second call: \( \langle x'', y, z \rangle \)
  
  project: \( \langle y, z \rangle \)
  
  cache lookup: \( \rightarrow \langle y', z' \rangle \)
  
  expand: \( \langle x'', y', z' \rangle \)
Caching Local Transitions (1)

- Recall local transition in specification:
  
  \( p3.2: \text{atomic} \{ z>0 \rightarrow z--; y++ \} \)

- Dependency matrix row: \( p3.2 \begin{bmatrix} 0 & 1 & 1 \end{bmatrix} \)

- Define projection: \( \pi_{p3.2} \langle x, y, z \rangle = \langle y, z \rangle \)

- Next, consider two consecutive calls to \( p3.2 \):
  
  first call: \( \langle x, y, z \rangle \)  
  successor: \( \langle x, y', z' \rangle \)  
  project: \( \langle y, z \rangle \)  
  project and store in cache: \( \langle y', z' \rangle \)  
  cache lookup: \( \rightarrow \langle y', z' \rangle \)  
  expand: \( \langle x'', y', z' \rangle \)

- Maintain a memoization table \( cache[i] \) for each transition group \( i \)
Caching Local Transitions (2)

- Caching can save calls to the language module
- Still some work for every concrete state (cache lookup)

- Caching is useful especially:
  - For expressive/inefficient languages
  - When dependency matrices are sparse

- Always uses a bit more memory (tables)
Static Matrix Reordering

PINS Optimization

- Regrouping **similar** transition groups reduces overhead
- Reordering **state variables** reduces BDD sizes (a.o.)
## Static Matrix Reordering

### PINS Optimization

- Regrouping similar transition groups reduces overhead
- Reordering state variables reduces BDD sizes (a.o.)

```plaintext
Sokoban before and after regrouping

...(58 rows)....
```

```plaintext
Sokoban before and after regrouping

...(58 rows)....
```
Multi-Valued Decision Diagrams
Every path in the MDD represents a concrete state vector
Multi-Valued Decision Diagrams

- Every path in the MDD represents a concrete state vector
- Potential gain in memory saving: exponential (here: $54 \rightarrow 15$)
- Symbolic Reachability: explore sets of states stored as MDDs
Symbolic Reachability Algorithm

[ICTAC 2008]

- $L$, $V$: MDDs for sets of long state vectors (level, visited)
- $R_i$: MDDs to store transition relation $i$ on short vectors
- $L_i$, $V_i$: MDDs for sets of short state vectors (level, visited for $i$)
Symbolic Reachability Algorithm
[ICTAC 2008]

- $L, V$: MDDs for sets of long state vectors (level, visited)
- $R_i$: MDDs to store transition relation $i$ on short vectors
- $L_i, V_i$: MDDs for sets of short state vectors (level, visited for $i$)

symbolic-reachability()

1. $L := \{\text{INITSTATE()}\}$; $V := L$; all $R_i := \emptyset$; all $V_i := \emptyset$
2. while $L \neq \emptyset$ do
3.   for $i \in \text{groups}$ do /* enumerate short vectors */
4.     $L_i := \pi_i([D]_{N \times K}, L) \setminus V_i$; $V_i := V_i \cup L_i$
5.     $R_i := R_i \cup \{(s, s') | s \in L_i \land s' \in \text{NEXTSTATE}(i, s)\}$
6.     $L := \bigcup_i (R_i(L) \setminus V)$; $V := V \cup L$
7. return $V$
Symbolic Reachability

Symbolic Reachability with PINS

- Global set of reachable states is computed as fix point
- Stored as a multi-valued decision diagram (MDD)
- Learn symbolic sub-groups $R_i$ on-the-fly (via $\text{NextState}$)
Symbolic Reachability

Symbolic Reachability with PINS

- Global set of reachable states is computed as fix point
- Stored as a multi-valued decision diagram (MDD)
- Learn symbolic sub-groups $R_i$ on-the-fly (via `NextState`)

Extensions

- Multiple exploration strategies:
  - Breadth-first: $(T_1 + T_2 + \cdots + T_n)^*$
  - Chaining: $(T_1 \circ T_2 \circ \cdots \circ T_n)^*$
  - Saturation-like: $(T_1^* \circ T_2^* \circ \cdots \circ T_n^*)^*$
  - Full Saturation: $(((T_1^* T_2)^* T_3)^* \cdots)^* T_n)^*$
- Static variable reordering boosts performance
1. Model Checking in a Nutshell

2. Planning Example: Sokoban

3. LTSmin Tool Architecture

4. Symbolic Algorithms

5. Multi-Core Algorithms
Multi-Core Algorithms
Recent standard hardware (x86)

- Multiple cores per processor, multiple processors
- Typical big machine: 48–64 cores, 256 GB shared memory
- Communication via L1, L2 caches: cache-coherence protocols
Multi-Core “Crisis”

Intel Processor Clock Speed (MHz)

(Source: Smoothspan)
Moore’s Law: More Cache!

![Graph showing total cache size per chip (L2 + L3) vs. time. The graph displays a logarithmic increase in cache size from 1998 to 2009, with labels for different processor models and technologies like Pentium III, Pentium 4, Core 2 Duo, and Core 2 Quad.](Source: Anandtech)
Scalable Multi-Core Reachability
(cf. Holzmann, FMCAD 2006)

- Exploitable parallelism must double every 2 years
  (Corollary of Moore’s Law)

- (Graph) Reachability is basis of many verification problems

- Multi-Core Model Checking:
  state-of-the-art not very impressive
Scalable Multi-Core Reachability
(cf. Holzmann, FMCAD 2006)

▶ Exploitable parallelism must double every 2 years
(Corollary of Moore’s Law)

▶ (Graph) Reachability is basis of many verification problems

▶ Multi-Core Model Checking: state-of-the-art not very impressive

Having trouble with scaling simple (enumerative) reachability?

▶ Then what are the chances to parallelize:
Liveness, partial-order reduction, symbolic reachability, ...?
Multi-core reachability

Problems for Model Checking: Visited Set

- **Parallel** access to hash table: correctness and efficiency
- Parallel access requires synchronization ........ lock contention
- Graph traversal: Random memory access ........ cache misses
- Main problem with cache lines: ..................... false sharing
Experiments: SPIN 5.2.4 (NASA/JPL)
Experiments: LTSmin

![Graph showing speedup (X) for various benchmarks]

- Anderson.6
- at.5
- at.6
- Bakery.6
- Bakery.7
- Blocks.4
- Brp.5
- Cambridge.7
- Elevator_planning.2
- Firewire_link.5
- Fischer.6
- Frogs.4
- Frogs.5
- Hanoi.3
- Iproto.6
- Iproto.7
- Lamport.8
- Lann.6
- Lann.7
- Leader_filters.7
- Loyd.3
- Mcs.5
- Needham.4
- Peterson.7
- Phils.6
- Phils.8
- Production_cell.6
- Szymanski.5
- Telephony.4
- Telephony.7
- Gate.7
Study

- Multi-core reachability: (pseudo) breadth-first, depth-first, ...
- Load balancing

Where is efficiency lost?

- Lock Contention
- Lock Convoying
- Cache-Line Sharing
- Two-Step Dance
- Stampeding
- ...

![Graph showing decreasing trend over 8 steps]
Measure, Measure, Measure

- Analyze & distill thousands of measurements
- Guides decisions what to tackle next
Reachability Architectures

- **DiVinE 2.2:** Static Partitioning
  - BFS, high comm. cost, static load balancing

- **SPIN 5.2.4:** Shared Storage & Stack Slicing
  - DFS, multiple sync. points, integrated load balancing
Reachability Architectures II

Shared state storage as main sync point

- Flexible reachability algorithms
- Flexible load balancing

*Shared Hashtables in Parallel Model Checking* (Barnat, Ročkai 2007)

- “our shared hash tables do not scale beyond 8 cores”
- “could not investigate lockless hash table solution”
- “haven’t found the cause of the scalability issues with pre-sized tables”
Lockless Hash Table: Design
[FMCAD 2010]

1. Investigate Requirements on Shared Storage
2. Investigate Hardware Support
   fine-grained synchronization, caches, …
3. Exploit LTSmin Infrastructure
   incremental (Zobrist) hashing
Lockless Hash Table: Design

[FMCAD 2010]

1. Investigate Requirements on Shared Storage
2. Investigate Hardware Support
   fine-grained synchronization, caches, …
3. Exploit LTSmin Infrastructure
   incremental (Zobrist) hashing

Hash Table Designed for Model Checking:

- **FindOrPut** operation only
- Don’t store pointers, no allocation
- No resizing!
- Fixed-size keys (state vectors)
- Low memory working set improves scalability
Lockless Hash Table: Layout
[FMCAD 2010]

- Open Addressing
- Hash Memoization
- Separate Data
- Walking the Line
- Lockless (CAS + write bit)

See also: Cliff Click (JavaOne 2007 presentation)
Incremental Hashing

\[ H_x \quad (H_x \oplus Z_{g,1}) \oplus Z_{f,3} = H_y \]

Incremental Hashing For State Vectors

Input: transition \( s \rightarrow_i s' \)

Input: hash value \( h \) of \( s \)

Output: hash value \( h' \) of \( s' \)

\[
s = \langle s_1, \ldots, s_N \rangle
\]

\[
s' = \langle s'_1, \ldots, s'_N \rangle
\]

\[
h' \leftarrow h
\]

\[
\text{for } j \in \{ j \mid D_{i,j}^W = 1 \} \text{ do }
\]

\[
h' \leftarrow h' \oplus Z[j][s_j \mod L]
\]

\[
h' \leftarrow h' \oplus Z[j][s'_j \mod L]
\]

- Zobrist matrix \( Z \):
  filled with random integers

- Dependency Matrix:
  find modified vector elements

- Difference to Chess:
  possible values not statically known

- Limit size of \( Z \) (e.g., \( L = 2^6 \)),
  domain of each vector slot is usually small
Experiments
Experimental Setup

- 16-core AMD Opteron 8356, 64 GB RAM
- Linux 2.6.32+patch
- BEEM model database (250+ models)
  - extensive collection of models for enumerative model checkers
  - case studies, protocols, games, planning, synthetic models, ...
  - http://anna.fi.muni.cz/models/
- Statically sized hash tables (no resizing)
- Speedups: relative to best sequential(!) tool
Experiments: Summary

![Graphs showing runtime and speedups of Spin, Divine 2, and LTSmin-mc.](image)

- **Green** LTSmin (UTwente)
- **Red** DiVinE 2.2
- **Blue** SPIN 5.2.4
Experiments: LTSmin (Detailed)
Effects of Load Balancing

Static Load Balancing (SLB)
- Feasible for many models
- Threads can run out of work

Synchronous Random Polling (Sanders ’97)
- Work stealing/hand-off
- Almost no overhead vs. SLB
- Additional Improvements
  - Shared-Memory Multi-Core
  - Informed Polling
  - Scare off stampeding threads
Observations

Model Checking Limitations:

- **Old Days: Memory**
- **Availability of large RAM:**
  Time-Outs ("Patience-Out")
- **Multi-Core Reachability**
  - Analyzing 10 Million states/sec on 16-core AMD
  - Allocation rate: 1 GB/sec

Memory is again Bottleneck
State Vector Compression
Tree Compression
[PDf2 2007, JLC 2009]

- State vectors highly similar
- Compression via tree of tables, recursive version of SPIN’s COLLAPSE
- Multi-Core version: Lockless hash table as building block!
- Increases *arithmetic intensity*: Super-linear speedups!
- “For Free”! (pays its own way)
Folded vector $\langle 2, 1 \rangle_{FV}$ represents $\langle 0, 0, 1, 0, 1, 0 \rangle_{IV}$ resp. $\langle O, O, X, O, X, O \rangle_{SV}$ (with $O \leftrightarrow 0, X \leftrightarrow 1$)

Selected tree fringe of grey boxes corresponds to state vector
Tree Compression

[PDMC 2007, JLC 2009]

Folded vector $\langle 2, 1 \rangle_{FV}$ represents $\langle 0, 0, 1, 0, 1, 0 \rangle_{IV}$ resp. $\langle O, O, X, O, X, O \rangle_{SV}$ (with $O \leftrightarrow 0$, $X \leftrightarrow 1$)

Selected tree fringe of grey boxes corresponds to state vector

Potential gain (here $54 \rightarrow 42$ entries):

- main table of size $N$ is only two integers wide
- small tables of size $O(\sqrt{N})$ only
Multi-Core Tree Compression
[NASA FM 2011, SPIN 2011]

- State vectors highly similar: Exploit combinatorial structure
- Extreme example: Memory/Time usage for firewire_tree.5:
  Uncompressed: 14 GB
  Tree Compression: 96 MB
- Near 100 % efficiency!
- Dependency Matrix: Incremental Tree Compression
Observations

- Centralized state storage scales at least as well as static state space partitioning
- Shared state storage orthogonal to: search strategy, load balancing, . . .
- Simpler architecture
- Arithmetic Intensity (hide memory latency)
- Data layout important (caches), sadly no language support
- Designed from the ground up for Scalability
- Establishing correctness of implementation is a pain . . .
Algorithm Engineering

- > 10× improvement over sequential algorithms
- Beats state-of-the-art tools, reopens earlier conclusions
- Research questions guided by experiments
- Engineering approach, repeatable benchmark results
- Hash table as building block for multi-core xDDs, liveness checking, ...
Conclusion
Adoption obstacles

- Modeling effort
  - many languages
  - avoid modeling?
- Scalability
  - parallel components
  - data, buffers, ...
- Complex tools
  - algorithms, heuristics
  - low-level details
### Adoption obstacles

- **Modeling effort**
  - many languages
  - avoid modeling?
- **Scalability**
  - parallel components
  - data, buffers, ... 
- **Complex tools**
  - algorithms, heuristics
  - low-level details

### Algorithmic solutions (combinatorics, locality)

- on-the-fly model checking
- symbolic model checking
- bounded model checking
- parallel model checking
- partial-order reduction
- symmetry reduction
Model Checking for Planning?

Adoption obstacles

- Modeling effort
  - many languages
  - avoid modeling?
- Scalability
  - parallel components
  - data, buffers, ...
- Complex tools
  - algorithms, heuristics
  - low-level details

Algorithmic solutions (combinatorics, locality)

- on-the-fly model checking
- symbolic model checking
- bounded model checking
- parallel model checking
- partial-order reduction
- symmetry reduction

Problem: Algorithms tied to specification languages

- No particular technique suits all applications / models
- Users need to rewrite model in different languages
Silver Bullets?

- No clear winner
- BUT: also depends on modelling
- PINS matrix can be used as predictor
LTSmin Verification Capabilities

Input Language
- mCRL2 (Process algebra)
- Promela (SPIN / NIPS–vm)
- DVE (BEEM)

PINS
- Transition Caching
- Static Reordering
- (P.O. Reduction)

Pins2pins Wrappers
- Distributed Generation
- Multi–core Reachability
- Symbolic Reachability

Reachability Tools

Verification
- On–the–fly:
  - deadlocks
  - goal/error actions
  - accepting cycle (LTL)
  (produces error traces)
- Symbolic:
  - (mu–calculus)
  - (CTL*)
Links to External Tools

- **Input Language**
  - mCRL2 (Process algebra)
  - Promela (SPIN / NIPS-vm)
  - DVE (BEEM)

- **PINS**
  - Transition Caching
  - Static Reordering

- **Pins2pins Wrappers**
  - Distributed Generation
  - Multi-core Reachability
  - Symbolic Reachability

- **Reachability Tools**
  - Distributed Minimization
  - External Model Checkers

- **LTS level Kripke structure**
  - PINS

- **Tools**
  - (nuSMV)
  - DiVinE
  - CADP
  - pins_open
  - bcg
## Portfolio of Language Modules

<table>
<thead>
<tr>
<th>Language</th>
<th>Description</th>
<th>Institution/Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mu$CRL</td>
<td>process algebra</td>
<td>CWI</td>
</tr>
<tr>
<td>mCRL2</td>
<td>process algebra, INESS</td>
<td>TU/e</td>
</tr>
<tr>
<td>PROMELA</td>
<td>SPIN</td>
<td>NASA/JPL</td>
</tr>
<tr>
<td>PROMELA</td>
<td>SpinJa</td>
<td>UTwente</td>
</tr>
<tr>
<td>DVE</td>
<td>DiVinE-cluster</td>
<td>MU Brno</td>
</tr>
<tr>
<td>DVE2</td>
<td>DiVinE model checking toolset</td>
<td>MU Brno</td>
</tr>
<tr>
<td>ETF</td>
<td>Enumerated Table Format</td>
<td>LTSmin</td>
</tr>
<tr>
<td>GNA</td>
<td>Genetic Network Analyzer</td>
<td>INRIA</td>
</tr>
<tr>
<td>ODE</td>
<td>Maple</td>
<td>EC-MOAN</td>
</tr>
</tbody>
</table>

*Prototype*
Portfolio of Tools

\(\langle \text{spec} \rangle 2\text{lts-gsea}\) Depth/Breadth-First Enumerative Reachability
\(\langle \text{spec} \rangle \text{-reach}\) MDD-based symbolic reachability
\(\langle \text{spec} \rangle 2\text{lts-mpi}\) Distributed state-space generation
\(\langle \text{spec} \rangle 2\text{lts-mc}\) Multi-Core Enumerative Reachability
\(\langle \text{spec} \rangle 2\text{torx}\) TorX tester RPC interface
\(\text{pins\_open}\) Connection to CADP toolset (VASY/INRIA)
\(\text{ltsmin-mpi}\) Signature-based distributed minimization
\(\text{ce-mpi}\) Orzan’s distributed cycle elimination
\(\text{ltsmin-tracepp}\) (Error) trace pretty printer

Next: Partial-Order Reduction (POR), Linear Temporal Logic (LTL), \(\mu\)-calculus Saturation,
Who benefits from LTSmin?

Useful for end users: **larger case studies**

- Model in a suitable (your favourite) specification language
- Decide later what model checking algorithm to use
### Who benefits from LTSmin?

#### Useful for end users: **larger case studies**
- Model in a suitable (your favourite) specification language
- Decide later what model checking algorithm to use

#### Useful for tool providers: **algorithms for free**
- Your Domain Specific Specification Language can get HPMC
- Ideally, LTSmin can be viewed as a library
Who benefits from LTSmin?

**Useful for end users: larger case studies**

- Model in a suitable (your favourite) specification language
- Decide later what model checking algorithm to use

**Useful for tool providers: algorithms for free**

- Your Domain Specific Specification Language can get HPMC
- Ideally, LTSmin can be viewed as a library

**Useful for researchers: rigorous benchmarking**

- LTSmin allows benchmarking algorithms on one model
- LTSmin allows comparing languages on one algorithm

- Alfons Laarman, Jaco van de Pol, Michael Weber, *Parallel Recursive State Compression for Free* .............. (SPIN 2011)
- Alfons Laarman, Jaco van de Pol, Michael Weber, *Multi-Core LTSmin: Marrying Modularity and Scalability* (NFM2011, tool)
- Stefan Blom, Jaco van de Pol, Michael Weber, *LTSmin: Distributed and Symbolic Reachability* ............. (CAV 2010, tool)
- Alfons Laarman, Jaco van de Pol and Michael Weber, ... (FMCAD 2010) *Boosting Multi-Core Reachability Performance with Shared Hash Tables*
- Stefan Blom and Jaco van de Pol, *Symbolic Reachability for Process Algebras with Recursive Data Types* ............. (ICTAC'08)
- Stefan Blom, Bert Lisser, Jaco van de Pol and Michael Weber, *A Database Approach to Distributed State-Space Generation* (JLC 2009)
- Stefan Blom and Jaco van de Pol, *Distributed Branching Bisimulation Minimization by Inductive Signatures* ............ (PDMC’09; EPTCS 14)
Model Checking (and related) Venues

- **CAV** Computer Aided Verification
- **TACAS** Tools and Algorithms for the Construction and Analysis of Systems
- **VMCAI** Verification, Model Checking, and Abstract Interpretation
- **ATVA** Automated Technology for Verification and Analysis
- **FMCAD** Formal Methods in Computer Aided Design
- **SPIN** SPIN Workshop on Model Checking Software
- **PDMC** Parallel and Distributed Methods in Verification
- **MoChArt** Model Checking and Artificial Intelligence
- **QEST** Quantitative Evaluation of Systems
- **FORMATS** Formal Modelling and Analysis of Timed Systems