THURSDAY, Session 2
Symmetry and Model Checking

1 Symmetry and Model Checking

- Some systems exhibit significant symmetry, for example
  - a network of many indistinguisahble processes
  - a ring symmetric under rotations
- Example: mutual exclusion protocol
  - the two processes p_1 and p_2 are symmetric, *e.g.*, consider the states

    $$(\{T_1, C_2\}, \text{turn} = 2) \qquad \text{and} \qquad (\{T_2, C_1\}, \text{turn} = 1)$$

    i.e., process 2 critical and process 1 trying, or the reverse. These states are symmetric by swapping the index values 1 and 2.

  Question: is it necessary to consider both states in checking temporal properties of the model?

1.1 Permutations

- A permutation π on a set S is a bijection (invertible function) from $S \to S$.
- A permutation is often represented by a vector.
  e.g., if $\pi = (1, 3, 2)$, then $\pi(1) = 1$, $\pi(2) = 3$, $\pi(3) = 2$.
- A permutation on states can be given indirectly by a permutation on values or indices.
  e.g., applying $\pi = (0, 2, 1)$ to the state

  $$(\{T_1, C_2\}, \text{turn} = 2)$$

  we get

  $$(\{T_2, C_1\}, \text{turn} = 1)$$

  and *vice versa*.

1.2 Permutation groups

– A permutation group G is a set of permutations that is closed under functional composition.

  That is, by functional compositon, we mean $\pi_1\pi_2$ is the permutation that results from applying first π_1 then π_2.
  For example, if $\pi = (3, 1, 2)$ (a left rotation), then

$$\pi^2 = (2, 3, 1)$$

  (a left rotation by 2).

– The group $G(\Pi)$ generated by a set of permutations Π is just the closure of Π under composition (*i.e.*, all permutations $\pi_1\pi_2 \dots \pi_n$ where $\pi_{1\dots n} \in \Pi$, $n = 0, 1, 2, \dots$.

  * example: $G\{(4, 1, 2, 3)\} = \mathrm{Rot}_4$, the rotations on $1 \dots 4$.
  * example:

$$G\{(2, 1, 3, 4), (1, 3, 2, 4), (1, 2, 4, 3)\} = \mathrm{Sym}_4$$

    the "symmetric group" (all permutations on $1 \dots 4$). Note, all permutations can be obtained by a sequences of nearest-neighbor swaps, as in "bubble sort".

1.3 Symmetry in Kripke Models

References: [1, 2]

– A permutation π on S is a symmetry of the Kripke model (S, R, L) when

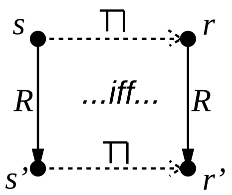
  * R (the transition relation) is invariant under π:

$$(\pi s_1)R(\pi s_2) \qquad \text{exactly when} \qquad s_1 R s_2$$

  * every atomic proposition a is invariant under π:

$$\pi s \models a \qquad \text{exactly when} \qquad s \models a$$

– Note: by definition, a symmetry of (S, R, L) is a special case of a bisimulation (*i.e.*, a bisimulation which happens to be bijection).



• Orbits

– A permutation group G defines an equivalence $\equiv_G$, where

$$s_1 \equiv_G s_2 \quad \text{when} \quad \pi s_1 = s_2 \text{ for some } \pi \in G$$

(note this relation is symmetric and transitive by group axioms)

– A equivalence class of $\equiv_G$ is called an *orbit*.

If Π is a set of symmetries, then every $\pi \in G(\Pi)$, the symmetry group generated by Π, is also a bisimulation, (since bisimulations are transitive).

Since bisimulations are closed under union, $\equiv_G$ $(= \cup G)$ is also a bisimulation.

$\Rightarrow$ *Every element of an orbit satisfies exactly the same CTL* formulas*

Denote by $[s]$ the orbit of a given state s.

- Quotients

  Suppose G is a symmetry group of the Kripke model $K = (S < R < L)$. Then the quotient $K/\equiv_G$ is a structure (S'R',L') with one state for every orbit of K, where

  – $[s_1]R[s_2]$ exactly when $s_1 R s_2$, and
  – $[s] \models a$ (an atomic prop) exactly when $s \models a$

  Note: the quotient is bisimular to K, hence preserves all CTL* properties.

1.4 Reduced Models

We want to generate the quotient structure for K without building K itself. This involves two steps:

1. Verify that each generator is a symmetry *based on the program text.*
2. Choose a coninical representative for each orbit. Build the state graph as usual, but replacing each state by the canonical representative for its orbit.

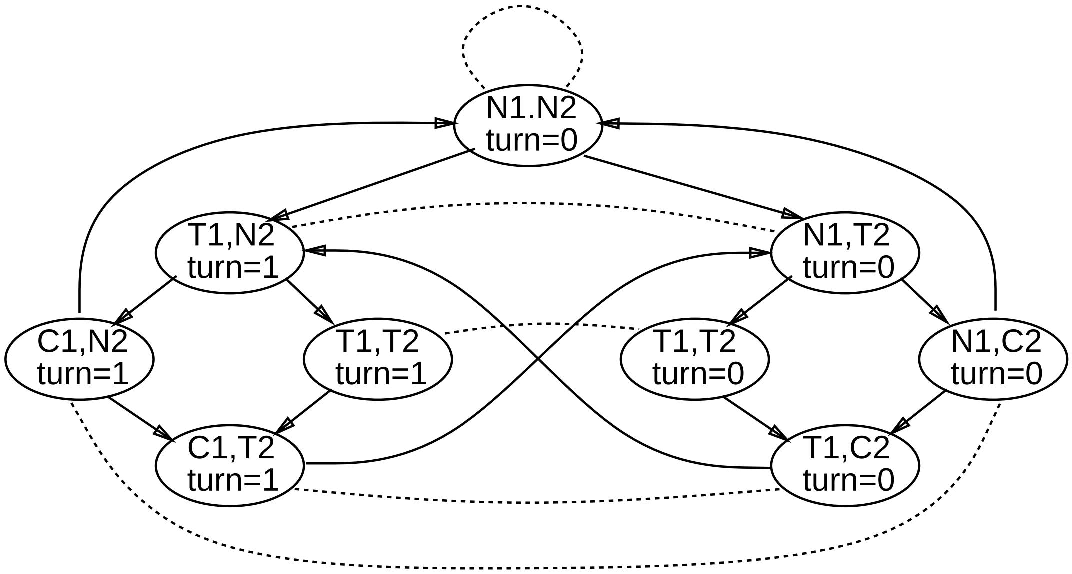
   For example, the canonical state could be the one that is lexically least in some representation.

- Example: Mutual exclusion protocol under the symmetry group

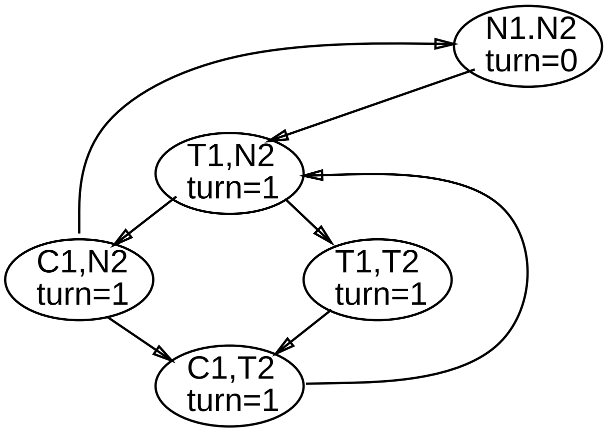
$$G\{(0, 2, 1)\}$$

  i.e., swapping process indices 1 and 2.

  Here is the original state graph, with orbits of the symmetry group marked as dashed lines:



Choosing as our representative of each orbit the leftmost state in the diagram, we obtain the following reduced graph:



- A note on canonical representatives:

  Suppose a state is a boolean vector

  $$(x_1, x_2, \ldots, x_n)$$

  and G is a group generated by some permutations on $1 \ldots n$ (*i.e.* by permuting the bits in the vector). For any two states x and x', determining if $x \equiv_G x'$ is as hard as the graph isomorphism problem. Therefore, so is finding a canonical representative for x.

  However, it is not necessary for the representatives to be canonical. There may be several repesentatives for each orbit (in general, it is always sound to replace a state by any other bisimular state). Choosing good representatives while minimizing computation time is an area for heuristics.

There are some easy cases, however:

- The symmetric group acting on a vector: simply sort the vector in increasing order.
- The rotation group acting on a vector: simply rotate the least element to the front.

1.5 Checking CTL* formulas

- Note in the previous example, π is not really a symmetry, since it does not preserve the atomic propositions (*e.g.*, C_1 becomes C_2, *etc.*).

  However, suppose we want to verify

  $$AG\neg(C_1 \wedge C_2)$$

  We can treat $\neg(C_1 \wedge C_2)$ as an atomic proposition. Note $\neg(C_1 \wedge C_2)$ is invariant under $\pi = (0, 2, 1)$, since

  $$\neg(C_1 \wedge C_2) \equiv \neg(C_2 \wedge C_1)$$

  Therefore $AG\neg(C_1 \wedge C_2)$ can be checked on the reduced model (notice it's true in the reduced state graph).

- Notice, however, $AG(T_2 \Rightarrow AF\ C_2)$ has become *false* in the reduced model. Also,

  $$AG \wedge_{i=1}^{2} (T_i \Rightarrow AF\ C_i)$$

  though it appears symmetric, is also false (because of the way we chose representatives, C_2 never becomes true).

- In general, a CTL* formula is preserved on the reduced model if all the *maximal propositional subformulas* are invariant under the group generators.

  Also, on reduced graphs, we often cannot state fairness constraints. For a solution to this problem, see Emerson and Sistla, CAV95.

2 Murphi – a practical approach to symmetry reductions

To apply symmetry reductions, we must be able to

1. identify and verify symmetries on the program text.
2. efficiently find orbit representatives
3. ensure that checked properies are invariant under symmetries

In the Murphi system [**?**]

1. symmetries are induced by special symmetric data types called scalar sets.
2. a search process is used to find representatives (which may or may not be canonical).
3. only invariant (safety) properties and nondeadlock can be checked

2.1 The Murphi language

...is based on the UNITY model. A UNITY program has the form:

```
<global variable declarations>
[         condition → <compound statement>
    □     condition → <compound statement>
    □     condition → <compound statement>
    ...
]*
```

That is, a nondeterministic choice among guarded commands that repeats forever. The guarded commands execute *atomically* (without interruption). There are no message passing primitives as in CSP.

Example of a Murphi program – Mutual exclusion program:

```
Type pid: 1..2;
     label_t: Enum{N,T,C};

Var
  P: Array [ pid ] Of label_t;
  turn : pid;

Ruleset i: pid Do

  Rule P[i] = N  ==> Begin P[i] := T; End;

  Rule P[i] = T & ( (P[3-i] != T) | (turn != i) )
       ==> Begin P[i] := C; End;

  Rule P[i] = C ==> Begin turn := i; P[i] := N; End;

End;
```

Initial states are generated by special “rules”:

```
Startstate
Begin
  For i:pid Do P[i] := N; End;
  Undefine turn;
End;
```

Safety properties a given using “invariant” rules, that transition to an error state when the condition is false:

```
Invariant
  !((P[1] = C) & (P[2] = C));
```

2.2 Scalar sets

A "scalar set" type is declaraed as follows:

```
Type t : ScalarSet(n);
```

A variable of this type may take values $1 \ldots n$. Operations on scalar sets are resticted, however, so that these values are symmetric. In particular, scalar set variables can only be used in the following restricted ways:

- Assignments, equality tests, and array indices
- `Ruleset i : t do <rule> end`

  generates n identical rules for $i = 1 \ldots n$.
- `For i : t do <statement> end`

  Loop over scalar set. Loop body must be parallelizable (i.e., iterations must be executable in any order with same result). This can be checked at run time by determining that no iteration write a value that is read or written by another iteration.
- expression `Forall i : t do <boolean expression> end`

  logical "and" over all scalar set values. There is also an "Exists" quantifier

In particular, you cannot express constants of a scalar set type.

Result: *All permutations of scalar set type are symmetries of a Murphi program* (*i.e.* leave the transition relation invariant).

In particular, all expressible invariants are preserved in the reduced graph. This is complicated to prove, but need only be proved once by the language designer.

Example using scalar sets

Here is a version of the mutual exclusion algorithm using a scalar set to index the processes. Note that some conditions had to be re-expressed in order to use symmetrical constructs.

```
Type pid: scalarset (2);
     label_t: Enum{N,T,C};

Var
  P: Array [ pid ] Of label_t;
  turn : pid;

Ruleset i: pid Do

  Rule P[i] = N  ==> Begin P[i] := T; End;

  Rule P[i] = T & ( Forall k: pid Do ( k!=i ) -> !(P[k] = T) End
          | (turn != i ))
       ==> Begin P[i] := C; End;

  Rule P[i] = C ==> Begin turn := i; P[i] := N; End;

End;


Startstate
Begin
  For i:pid Do P[i] := N; End;
  Undefine turn;
End;
```

Notice the invariant has to be restated in a rather more complicated way:

```
Invariant
  ! Exists i1: pid Do
      Exists i2: pid Do
        ( i1 != i2 & P[i1] = C & P[i2] = C )
      End
    End;
```

Murphi's reduction algorithm

- Canonicalization

  generates all permutations of scalar set types and chooses lexically least.

- Normalization

  Lexically minimizes only a subset of the state variables. May result in multiple states within a given orbit.

2.3 Cache protocol example

Murphi was originally designed at Stanford for modeling cache coherence protocols that are use in shared memory multiplrocessors. In a typical such protocol:

- "lines" from main memory can be stored in the local caches of zero or more processes.
- Each cached line can be in a shared (nonmodifiable) or exclusive (modifiable) state
- each line in main memory has a "directory" that locates all cached copies
- Management of the cached copies is done by sending messages over a network

The figure shows a diagram of the state information associated with particular protocol modeled im Murphi.

Note there are several types that are symmetric here:

- processor id's within a "system"
- system id's
- memory line addresses
- data values

All of these can be declared as scalar set types.

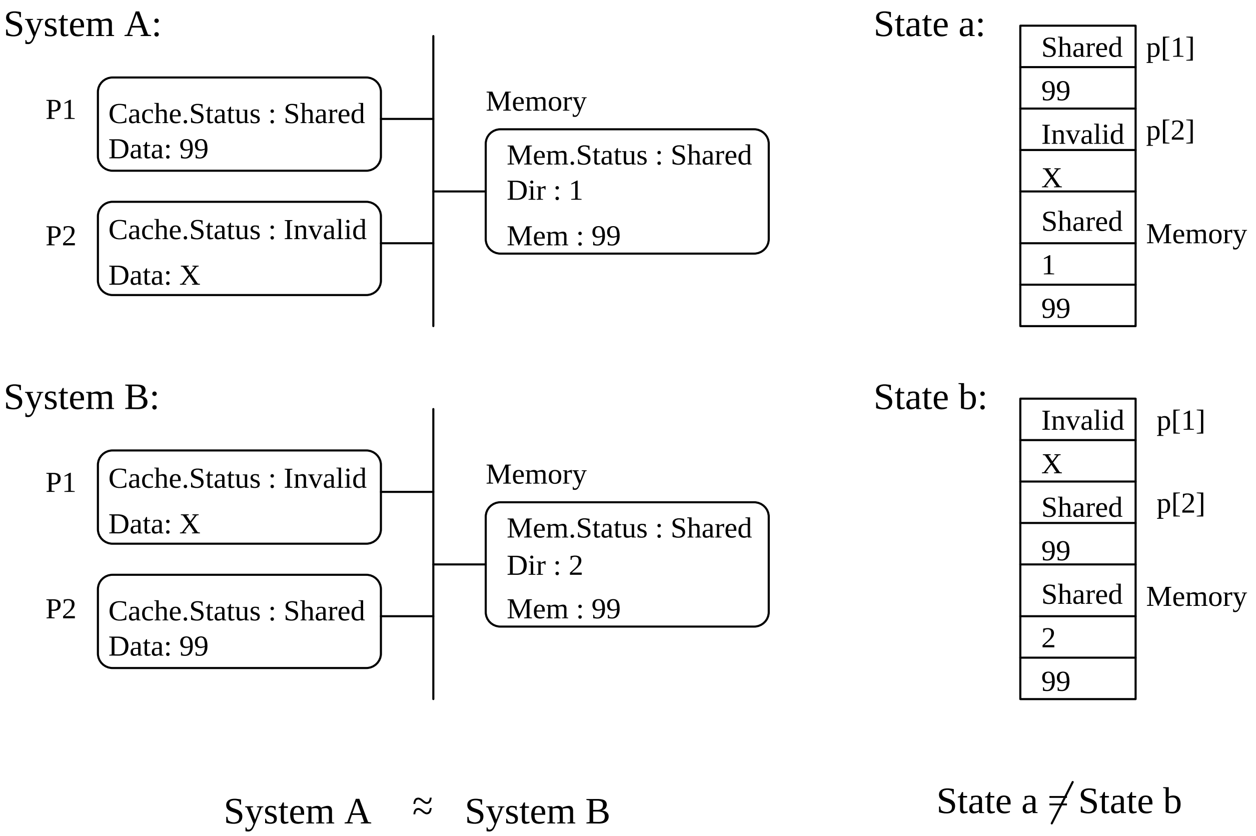


Figure 1: Cache protocol state

Here is an extract of the Murphi code declaring the state variables of the protocol:

```
Type
        pid:        ScalarSet(NumProcessor);
        mid:        ScalarSet(NumProcessor);
        address:    ScalarSet(NumAddress);
        value:      ScalarSet(NumDataValues);

Var
        P:          Array [pid] of
                        Record
                            State : enum {Invalid,Shared,Master};
                            Value : value;
                        End;
        M:          Array [mid] of
                        Array [address] of
                            Record
                                State : enum {Uncached,SharedRemote,DirtyRemote};
                                Dir : Array [1..dirsize] of pid;
                                Mem : value;
                            End;
        Net:        ... array of records representing messages ...
```

In the reduction algorithm, for example, when permuting process id's 1 and 2, first all pid's 1 and 2 stored in the directory and in messages are swapped. The the correspond array elements in the processor array are swapped.

Verification time and space results

Nodes	Algorithm	States	Time	% reduction
2	Unreduced	1,694	12s	0%
	Canonicalized	425	48s	75%
	Normalized	429	7s	75%
3	Unreduced	91,254	23min	0%
	Canonicalized	7,741	4.5hr	92%
	Normalized	9,002	13min	90%
4	Unreduced	>80Mb		
	Canonicalized		>36hr	
	Normalized	206,169	36hr	

2.4 Data Saturation

- Suppose a given scalar set type is never used as an index in the program, and the program has n variables of that type.

  Thm: Any orbit has a representative using only the values $1 \ldots n$ for variables of type t, independent of the scalar set size.

- This is known as "data saturation". When data saturation occurs, we can effectively verify a system with an arbitrary number of data values by verifying a system with n values.

  Note, this is a variation on the "data independence" idea discussed earlier.

3 Summary

1. A permutation on the state space is a symmetry when it leaves R and L invariant.
2. Symmetry groups can be generated by, for example, permutions on variable values or array indices.
3. A symmetry group induces a bisimulation equivalence on the state space. An equivalance class is called an orbit.
4. The quotient model is obtained by finding a canonical representative for each orbit.
5. The reduced graph preserved all CTL* formulas whose "maximal propositional subformlas" are left invariant.
6. The Murphi system:
   - Atomic guarded commands
   - Scalar sets with operations restricted to insure symmetry
   - verifies invariants and nondeadlock
   - uses heuristically effective "normalization" of states

References

[1] S. Jha E. Clarke, T. Filkhorn. Exploiting symmetry in temporal logic model checking. In *5th CAV*, 1993.

[2] E. Emerson and A. Sistla. Symmetry and model checking. In *5th CAV*, 1993.

[3] C.N. Ip and D.L. Dill. Better verification through symmetry. *Formal Methods in System Design*, 9(1-2):41–75, Aug. 1996.