Program BAKERY: The Bakery Algorithm.

```
loop forever do
  Non-critical
  await "#", "$", "#", "\", "#" \text{ Critical}
  I + ([N]i, \ldots, [1]i) \max = [i]i
  0 = [i]i \iff i \neq I \wedge \text{I + (}[N]i, \ldots, [1]i) \max = [i]i
  0 = [i]i \iff i \neq I \wedge \text{I + (}[N]i, \ldots, [1]i) \max = [i]i
```

We consider program BAKERY-2, but this time, we present the general parameterized form.

Computing Abstractions with TLV
To represent BAKERY as an SMV program we need to impose an upper bound on the values of the array of the program.

This leads to the following prefix of program:

```smv
MODULE main
DEFINE N:=3; M:=10;
VAR Y:array 1..N of 0..M;

PROCMP(i,y,yin,N,M)
PROCRED(i,y,N)

for(i=1;i<=N;i=i+1) {

  ! (N, Y', t) process Reduce : [\exists t \exists N' \exists t' \forall N \forall t' \forall Y' \forall Y \forall t' : N' = N \land t' = t' \land Y' = Y \land Y \land \forall t' : t' = t' 

  RED 

  for(i=1;i<=N;i=i+1) {

    ! (N, Y', t) process MP(i, y, yin, N, M)

    for(i=1;i<=N;i=i+1) {

      P[i]: process

    }

    Red[i]: process

  }

} 

Note the use of loop constructors in the last two lines. These declare the

sequence of processes P[1], P[2], ..., P[N] and Red[1], Red[2], ..., Red[N].

To represent BAKERY with TLV

An SMV Representation

A. Pnueli

Program Analysis, NYU, Spring, 2004
Module MP is defined by

```plaintext
MODULE MP(id,y,yin,N,M)
VAR loc:0..5;
DEFINE is_max:=&for(i=1;i<=N;i=i+1){y[i]<yin};
enter:=&for(i=1;i<=N;i=i+1){y[i]=0|y[id]<=y[i]};
ASSIGN init(y[id]):=0;
init(loc):=0;
next(loc):=caseloc in {0,4,5}:(loc+1)mod6;
loc=1:{1,2};
loc=2&is_max:3;
loc=3&enter:4;
loc=2&is_max:3;
next(loc):=caseloc=2&is_max:y[0];
loc=5:0;
next(y[id]):=caseloc=2&is_max:yin;
esac;

JUSTICE loc=0, loc=2, loc=3 & enter, loc=4, loc=5
```
Rather than computing directly the expression \( [p_1] \hat{y} > [i] \hat{y} \land 0 = [i] \hat{y} \) as required in statement \( g \) of \texttt{BAKERY}, we include the following construction in \texttt{BAKERY.smv}:

Rather than computing directly the expression \([N] \hat{y}, \ldots, [1] \hat{y}\) as required in statement \( g \), we define the assertion \texttt{enter} which is true if, for each \( i \), \[ N, \ldots, = I, \ldots, \]

We also define the assertion \texttt{enter} which is true if, for each \( i \),

\texttt{Wedeclarethevariable} \( y \) \texttt{to be of type} \( 0..M \). \texttt{Thefactthatnomodulemakesanexplicitassignmenttothisvariableidentifiesitasaninputvariable,implyingthat,ateachtransition,thisvariableisnondeterministicallyassignedavalueinitsrespective(range).}

\texttt{Then,withinmodule} \( MP \), \texttt{we define the assertion} \( \texttt{t} = \texttt{max} \) \texttt{which is true iff for each} \( i \), \( i \) is greater than each \( i \). On arriving to location \( g \), \texttt{we proceed to} \( Y \), \texttt{its respective range.}

\texttt{Thejusticerequirementsguaranteethatexecutiongetstuckonlyatlocation} \( g \), \texttt{or at location} \( g \) if the entity condition is false infinitely often.

\texttt{Thisjusticerewequirementsguaranteethatexecutiongetstuckonlyatlocation} \( g \).
Lecture 8: Computing Abstractions with TLV

A. Pnueli

Module

Reduce

is defined as:

\[ \text{MODULEReduce}(id, y, N) \]

\[ \text{DEFINE} \]

\[ \text{sub} := \{ i = 1 \text{ to } N \} : (y[i] + 1) \neq y[id] \];

\[ \text{ASSIGN} \]

\[ \text{next}(y[id]) := \begin{cases} y[id] - 1 \text{ if } \text{sub} \text{ and } y[id] > 1 \text{ and } \forall I < I \text{ for all } I, \exists J \text{ s.t. } [N]y[J] \neq [I]y \text{ and } I < [pI]y \text{ and } [pI]y \neq I \text{ otherwise} \; \text{satisfies the properties of mutual exclusion and accessibility} \} \]

We can model check \[ \text{bakerYN.smv} \] for various values of \[ M, N \] and find out that it satisfies the properties of mutual exclusion and accessibility.

Justice requirement attached to \( N \).

Frequent activation of this module will guarantee that the values of \( Y \), \( M \), and \( N \) will not grow too much or too quickly. Such frequent activation is imposed by the

\[ \forall i, j, \forall N \geq T, \forall I = T \text{ for all } I, \exists J \text{ s.t. } [N]y[J] \neq [I]y \text{ and } I < [pI]y \text{ and } [pI]y \neq I \text{ otherwise} \; \text{satisfies the properties of mutual exclusion and accessibility} \}

\[ \text{Essay!} \]

\[ \forall [pI]y \;

\[ [I]y \text{ s.t. } \exists J \text{ s.t. } [N]y[J] \neq [I]y \text{ and } I < [pI]y \text{ and } [pI]y \neq I \text{ otherwise} \; \text{satisfies the properties of mutual exclusion and accessibility} \]

\[ \text{ASSIGN} \]

\[ \text{next} = : ([pI]y) \text{ s.t. } \exists J \text{ s.t. } [N]y[J] \neq [I]y \text{ and } I < [pI]y \text{ and } [pI]y \neq I \text{ otherwise} \; \text{satisfies the properties of mutual exclusion and accessibility} \]

\[ \text{DEFINE} \]

\[ \text{MODULE Reduce(q, N)} \]

Module \( \text{Reduce} \) is defined as:

\[ \text{Reduce} \]
We then proceed to define the program abs-bakeryN.smv which will serve as the carrier file abs-bakeryN.smv. We then proceed to define the file abs-bakeryN.smv which will serve as the carrier file abs-bakeryN.smv.

For the case of $N = 2$ (i.e., BAKERY-2), we choose the following predicate base:

\[
\begin{align*}
0 & = [1]f & : 0 = [1]f B
\end{align*}
\]
Modules MP and Reduce are copied from file bakeryN.smv.

```plaintext
 MODULE MP and Reduce are copied from file bakeryN.smv.

```
Lecture 8: Computing Abstractions with TLV

For each system, the TLV compiler builds a (fair) transition system (TS) also known as an FDS. This is a structure which contains all the relevant components for the system. Thus, for system bakery\text{N} within abs-bakery\text{N}.smv, the compiler constructs a separate transition relation for each process within the system. Typically, TLV constructs a separate transition system for each process corresponding to a process in the FDS. We refer to different TS's by their index. For TS number \( i \), the TS structure contains the following:

- The number of transitions whose disjunction forms the complete transition relation for TS
- The initial condition (assertion) for TS
- The set of system variables for TS
- The number of compassion requirements (\( s[i] . c[p][1] \), \( s[i] . c[q][1] \), ..., \( s[i] . c[p][c[p]] \), \( s[i] . c[q][c[q]] \))
- The number of justice requirements (\( s[i] . j[1] \), ..., \( s[i] . j[j] \))
- The number of justicerequirements
- The individual transitions

\( [i] . t_1, ..., [i] . t_{n} \) — The individual transitions

\( [i] . v \) — The set of system variables for TS

\( [i] . c[p] \) — The compassion requirements

\( [i] . j \) — The justice requirements

\( [i] . n \) — The number of transitions whose disjunction forms the complete transition relation for TS

\( [i] . i \) — The initial condition (assertion) for TS

\( [i] \) — The index of the process
On reading file abs-bakeryN.smv, the TLV compiler constructs two TS's. FDS TS[2][Z] represents system abs-bakery. Its variables are AS·Y2egal, AS·Y2egalZ, AS·Y2egalZ, AS·Y2egalZ, AS·Y2egalZ, AS·Y2egalZ, and AS·P[1]. It has 0 transitions and 0 justice requirements and a false initial condition.

FDS TS[1][Z] represents system abs-bakeryN. Its variables are CS·Y[1], CS·Y[2], CS·P[1], CS·P[2], and CS·loc. It has 4 transitions, and 10 justice requirements. CS·Y[1], CS·Y[2], CS·P[1], CS·P[2], and CS·loc. It has 4 transitions, and 10 justice requirements.
Abstracting Assertions and Transition Relations

Let us recall the formulas expressing abstractions of assertions and transition relations. Assume that the abstraction mapping is given by

\[
\begin{align*}
((\Lambda, \Lambda)^d \vee (\Lambda)^{\sigma} = \nu \Lambda) \land (\Lambda)^{\sigma} = \nu \Lambda & : (\nu \Lambda, \nu \Lambda)(d) \sigma \\
((\Lambda)^d \vee (\Lambda)^{\sigma} = \nu \Lambda) \land \sigma & : (\nu \Lambda)(d) \sigma
\end{align*}
\]

Next, consider a transition relation \( A \). The \( \sigma \)-abstraction of \( A \) is given by:

\[
((\Lambda)^d \vee (\Lambda)^{\sigma} = \nu \Lambda) \land \sigma : (\nu \Lambda)(d) \sigma
\]

by

\[
\begin{align*}
((\Lambda)^d \vee (\Lambda)^{\sigma} = \nu \Lambda) & : (\nu \Lambda)(d) \sigma
\end{align*}
\]

be an assertion over the concrete variables. The \( \sigma \)-abstraction of \( \Lambda \) is given by

\[
((\Lambda)^d \vee (\Lambda)^{\sigma} = \nu \Lambda) \land \sigma : (\nu \Lambda)^{\sigma}
\]

Let us recall the formulas expressing abstractions of assertions and transition relations.
Computing Abstractions with TLV

A. Pnueli

We consider the file `abs-bakeryN.pf`. First in this file are two functions which implement assertion abstraction and relation abstraction.

Implement assertion abstraction and relation abstraction.

Computing the Abstraction by TLV
The next procedure abstracts entire system $TS_1$ into system $TS_2$:

Abstracting One System Into Another

It abstracts the initial condition, each of the transitions, and each of the justice requirements.

To abs-sys;

Let $\_s[1].i := \text{abs-assert}(\text{conc}_1)$;

Let $\_s[2].i := \text{abs-assert}(\text{conc}_2)$;

Let $\_s[1].\text{tn} := \_s[1].\text{tn}$;

For $i$ in 1...$\_s[1].\text{tn}$

Let $\_s[1].t[i] := \text{abs-trans}(\text{conc}_1)$;

End--For $i$ in 1...$\_s[1].\text{tn}$

Let $\_s[2].\text{tn} := \_s[2].\text{tn}$;

Let $\_s[2].jn := \_s[1].jn$;

For $i$ in 1...$\_s[1].jn$

Let $\_s[1].j[i] := \text{abs-assert}(\text{conc}_1)$;

End--For $i$ in 1...$\_s[1].jn$

End--To abs-sys!

End -- for $\_s$ in 1...$\_s.\text{tn}$

(End -- for $\_s$ in 1...$\_s.\text{tn}$

Let $\_s[2].t := \text{abs-trans}(\text{conc}_2)$;

Let $\_s[2].\text{tn} := \_s[2].\text{tn}$;

Let $\_s[2].jn := \_s[2].jn$;

End--For $\_s$ in 1...$\_s.\text{tn}$

The next procedure abstracts entire system $TS_1$ into system $TS_2$:

Abstracting One System Into Another

By A. Pnueli
The preparation includes computation of the abstraction mapping:

```
let abstep = prime(abst)

let abst = As.Pi1 = CS.P[1].loc &
           As.Pi2 = CS.P[2].loc &
           (As.y1eq0 <-> (CS.y[1] = 0)) &
           (As.y2eq0 <-> (CS.y[2] = 0)) &
           (As.y1ley2 <-> (CS.y[1] <= CS.y[2])) &
           (As.y2ley1 <-> (CS.y[2] <= CS.y[1]));
```

Let vars = set-union(vars1,prime(vars1));
Finally, we test that the abstract system satisfies the expected properties of mutual exclusion and accessibility:

```
!call Temp-Entail(AS.Pi2=2, AS.Pi2=4, 2); 
print "check accessibility for Process 2\n"; 
!call Temp-Entail(AS.Pi1=2, AS.Pi1=4, 2); 
print "check accessibility for Process 1\n"; 
!call Invariance((AS.Pi1=4 & AS.Pi2=4), 2); 
print "check mutual exclusion of abstract system\n"; 
```

The parameter $n$ appearing last in each of the model checking requests, refers to the system number.
Why (and when) Can We Use a Truncated Version of the Concrete Systems?

In the previous examples, we used TLV to represent infinite-state systems which were truncated to a finite-state systems. What is the justification for this?

First, we observe that for the case that the abstract variables range over a finite domain, a computation of the abstraction of an assertion can be decomposed into solving finitely many satisfiability problems. Assume that the set of infinitely many (combination of) values that can be assumed by the abstract system variables is given by $\{\varphi \in \mathcal{D} : \text{sat}_A(L)\}$. For $i = 0, \ldots, \ell$, we denote by $\text{sat}_A(L)$ the fact that the assertion $\varphi$ is satisfiable by the abstract system variables.

$$\varphi = \bigwedge_{i=0}^{\ell} \varphi_i = \varphi_0 \lor \ldots \lor \varphi_\ell$$

$$\varphi_i$$

$$\varphi$$

Similar decomposition can be applied to the abstraction of a transition relation.

$$\varphi = \bigwedge_{i=0}^{\ell} \varphi_i = \varphi_0 \lor \ldots \lor \varphi_\ell$$

$$(\Lambda \varphi_i) \Rightarrow (\varphi_i \circ d) \Rightarrow (\Lambda \varphi_i) \Rightarrow \bigwedge_{i=0}^{\ell} (\Lambda \varphi_i) \Rightarrow (\Lambda \varphi)$$

Now, the abstraction of $\varphi$ is given by

$$\varphi = \bigwedge_{i=0}^{\ell} \varphi_i = \varphi_0 \lor \ldots \lor \varphi_\ell$$

By definition of $\text{sat}_A(L)$, for $i = 0, \ldots, \ell$, $\varphi_i$ is satisfiable by the abstract variables. Therefore, the abstraction of $\varphi$ is given by

$$\varphi = \bigwedge_{i=0}^{\ell} \varphi_i = \varphi_0 \lor \ldots \lor \varphi_\ell$$

$$\varphi$$

Where $\varphi_i$ is the assertion that the abstract variables are in the set $\mathcal{D}$. Then, the abstraction of $\varphi$ is given by

$$\varphi = \bigwedge_{i=0}^{\ell} \varphi_i = \varphi_0 \lor \ldots \lor \varphi_\ell$$

$$\varphi$$

In the previous examples, we used TLV to represent infinite-state systems which were truncated to a finite-state systems. What is the justification for this?
Consider for example program BAKERY. It refers to variables (and constants) \( M \). Assume that the variables \( x \) are

\[
\forall \alpha = (\forall x \hat{M}) \hat{W} \text{ where } \beta = (\forall x \hat{W})
\]

values to variables is given by:

The assignment of model in which the variables range over the domain \([1, \ldots, m] \). The assignment \( W \) be \( u \).

\( \forall x \). Observe, obviously, \( u \). We can construct a small

Let the distinct values assumed by \( W \).

\( \forall x \). Assumptions that \( x \) is satisfiable in some model \( W \).

\( \exists x \). \ldots \exists x \). Assumptions are \( \alpha \). Assumptions that the variables appearing in \( \alpha \) are

Assume that the form \( x \). Assume that the variables appearing in \( \alpha \) are

\( \exists x \). \ldots \exists x \).

Formulas are of the form \( x \). Assume that the variables appearing in \( \alpha \) are

Assumptions of Restricted Assertions

A. Pnueli

Program Analysis, NYU, Spring 2004

Lecture 8: Computing Abstractions with TLV
Extending the Atomic Formulas

Claim 8 is valid also for the case that we allow the additional atomic formulas of

\[\text{auxilliary variables in some cases.} \]

This may require the addition of expressions such as \( \bar{f} \) and \( f \) appearing \( f \) and \( f \), provided these are the only instances in which the forms \( f = x \) and \( f = x \) appear.