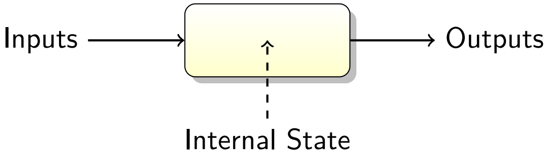
# Modular Abstractions of Reactive Nodes using Disjunctive Invariants

David Monniaux and Martin Bodin

December 5, 2011

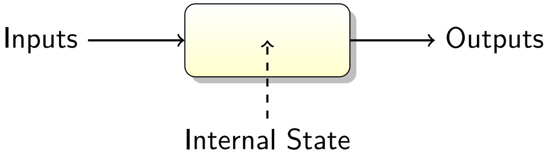
- A reactive node (LUSTRE, SCADE, SAO, SIMULINK) :




- It is an automaton.
- The internal state is usually given by some variables's values.

$\Longrightarrow$ Exponential number of state.

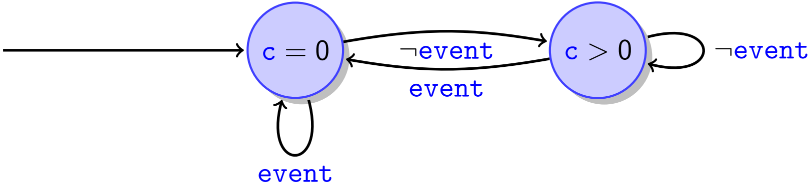
- A reactive node (LUSTRE, SCADE, SAO, SIMULINK) :




- It is an automaton.
- The internal state is usually given by some variables's values.

$\Longrightarrow$ Exponential number of state.

```
node timer (event : bool) returns (c : int);
let c = 0 ->
        (if event then 0
        else pre c + 1) ;
tel.
```



Goal:

- Abstract a reactive node $\longrightarrow$ an automaton.
- A bounded number of states.

Abstraction:

- It over-approximates the behavior of the node.
- We loose determinism.

Node Hierarchy:

- Modular analysis.
- Compositional analysis.

# Building an automaton

Process steps:

1. Decompose an over-approximation of the reachable states as a union of $n$ abstract states.
2. Compute which transitions are possible between those abstract states.

# What we need

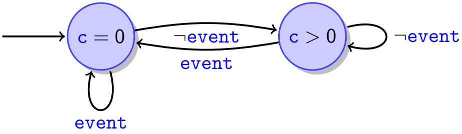
- **Reachable states** $\longleftarrow$ Solvers.
  SAT-solvers and SMT-solvers (*satisfiability modulo theory*).
  $\mathcal{NP}$-complete, but some tools (like YICES) try do do it efficiently.

- **Entry point** $\longrightarrow$ A finite set of predicates $\pi_1, \ldots, \pi_m$.
  They will be used to build the abstract states.
  $\rightarrow$ We more or less know what the invariant shall look like.

# What we need

- **Reachable states** $\longleftarrow$ Solvers.
  SAT-solvers and SMT-solvers (*satisfiability modulo theory*).
  $\mathcal{NP}$-complete, but some tools (like YICES) try do do it efficiently.

- **Entry point** $\longrightarrow$ A finite set of predicates $\pi_1, \ldots, \pi_m$.
  They will be used to build the abstract states.
  $\rightarrow$ We more or less know what the invariant shall look like.




$n = 2$

$\{\pi_1, \pi_2, \pi_3\} = \{c = 0, c < 0, c > 0\}$

$C_1 \equiv c = 0$

$C_2 \equiv c > 0$

- An abstract state is given by a conjonction $C_i$ of predicates.

$$C_i = \bigwedge_{j=1}^{m} (b_{i,j} \Rightarrow \pi_j)$$

- We seek an invariant of the form $\mathcal{T} = \bigvee_{i=1}^{n} C_i$. ($n$ is given.)
  $\Longrightarrow 2^{nm}$ possibilities for the Booleans $b_{i,j}$.

Some formulae used by the algorithm:

- The constraints $\forall\sigma, F$ over the variables (given). Typically, $F$ states (among other things) that $\mathcal{T}$ is an invariant.

The idea is to "discover" step by step what contraints $F$ yields for the state partition.

- A sequence of formulae $H_k$ (computed) that represents the discovered contraints over the Booleans $b_{i,j}$. Initially, $H_1 = \texttt{true}$.

A disjunctive inductive invariant, expressed by $B$.

$H :=$ `true`
**Loop:**
**match** $SAT(H)$ **with**
| UnSat $\rightarrow$ **return** NO_SOLUTION
| Sat($B_{i,j}$) $\rightarrow$
**match** $SMT(\neg F[B/b])$ **with**
| UnSat $\rightarrow$ **return** (SOLUTION $B$)
| Sat($\Sigma$) $\rightarrow$ $H := H \wedge F[\Sigma/\sigma]$

A disjunctive inductive invariant, expressed by $B$.

No set of states follows the constraints $H$.

$H := \texttt{true}$

**Loop:**

**match** $SAT(H)$ **with**

| UnSat → **return** NO_SOLUTION

| Sat($B_{i,j}$) →

**match** $SMT(\neg F[B/b])$ **with**

| UnSat → **return** (SOLUTION $B$)

| Sat(Σ) → $H := H \wedge F[\Sigma/\sigma]$

A disjunctive inductive invariant, expressed by $B$.

No set of states follows the constraints $H$.

$F$ is thus always true with this state partitioning.

$H := \texttt{true}$

**Loop:**

**match** $SAT(H)$ **with**

| UnSat $\rightarrow$ **return** $\textsc{No\_Solution}$

| Sat($B_{i,j}$) $\rightarrow$

**match** $SMT(\neg F[B/b])$ **with**

| UnSat $\rightarrow$ **return** ($\textsc{Solution}$ $B$)

| Sat($\Sigma$) $\rightarrow$ $H := H \wedge F[\Sigma/\sigma]$

A disjunctive inductive invariant, expressed by $B$.

No set of states follows the constraints $H$.

$H :=$ `true`
**Loop:**
**match** $SAT(H)$ **with**
| UnSat $\rightarrow$ **return** NO_SOLUTION
| Sat($B_{i,j}$) $\rightarrow$
**match** $SMT(\neg F[B/b])$ **with**
| UnSat $\rightarrow$ **return** (SOLUTION $B$)
| Sat($\Sigma$) $\rightarrow$ $H := H \wedge F[\Sigma/\sigma]$

$F$ is thus always true with this state partitioning.

Add the new discovered contraint and retry.

- This loop computes an invariant $\rightarrow$ `true` is an invariant!
- We need a (*locally*) minimal invariant.
- After getting an invariant $B^{(0)}$, restart the algorithm with a new constraint: the new invariant $B^{(1)} \subsetneq B^{(0)}$.

- This loop computes an invariant $\rightarrow$ `true` is an invariant!
- We need a (*locally*) minimal invariant.

- After getting an invariant $B^{(0)}$, restart the algorithm with a new constraint: the new invariant $B^{(1)} \subsetneq B^{(0)}$.

```
bool b;
int i = 0, a ; /* precondition a > 0 */
while ( i < a ) {
    b = random ( ) ;
        if (b)
            i = i + 1;
}
```

$$
\begin{aligned}
F \triangleq \quad & ((\mathtt{i} = 0 \wedge \mathtt{a} \geqslant 1) \Longrightarrow \mathcal{T}(\mathtt{b}, \mathtt{i}, \mathtt{a})) \\
& \wedge ((\mathtt{b'} \wedge \mathtt{i'} = \mathtt{i} + 1) \vee (\neg \mathtt{b'} \wedge \mathtt{i'} = \mathtt{i})) \\
& \wedge (\mathcal{T}(\mathtt{b}, \mathtt{i}, \mathtt{a}) \Longrightarrow (\mathcal{T}(\mathtt{b'}, \mathtt{i'}, \mathtt{a'}) \vee \mathtt{i'} \geqslant \mathtt{a'}))
\end{aligned}
$$

$$
\begin{aligned}
F \triangleq\ & ((\texttt{i} = 0 \wedge \texttt{a} \geqslant 1) \Longrightarrow \mathcal{T}(\texttt{b}, \texttt{i}, \texttt{a})) \\
& \wedge ((\texttt{b'} \wedge \texttt{i'} = \texttt{i} + 1) \vee (\neg \texttt{b'} \wedge \texttt{i'} = \texttt{i})) \\
& \wedge (\mathcal{T}(\texttt{b}, \texttt{i}, \texttt{a}) \Longrightarrow (\mathcal{T}(\texttt{b'}, \texttt{i'}, \texttt{a'}) \vee \texttt{i'} \geqslant \texttt{a'}))
\end{aligned}
$$

We set $n = 2$ and the predicates $\{\pi_1, \dots, \pi_8\} \triangleq \{\texttt{i} = 0, \texttt{i} < 0, \texttt{i} > 0, \texttt{i} = \texttt{a}, \texttt{i} < \texttt{a}, \texttt{i} > \texttt{a}, \texttt{b}, \neg\texttt{b}\}$.

| $H$ | $SAT(H)$: invariant candidate | $SMT(\neg F[B/b])$ |
|---|---|---|
| true | $(\texttt{i} = 0 \wedge \texttt{i} = \texttt{a} \wedge \texttt{b}) \vee (\texttt{i} = 0 \wedge \texttt{i} = \texttt{a} \wedge \neg\texttt{b})$ | $\texttt{i} = 0, \texttt{a} = 1, \texttt{b} = \bot$ |
| $F(0, 1, \bot)$ | $(\texttt{i} = 0 \wedge \texttt{i} = \texttt{a} \wedge \texttt{b}) \vee (\texttt{i} = 0 \wedge \texttt{i} < \texttt{a} \wedge \texttt{b})$ | $\texttt{i} = 0, \texttt{a} = -1, \texttt{b} = \bot$ |
| ⋮ | ⋮ | ⋮ |
| $H_6$ | $(\texttt{i} = 0 \wedge \texttt{i} < \texttt{a}) \vee \texttt{i} > 0$ | *accepted*! |

Thus $I_1 = (\texttt{i} = 0 \wedge \texttt{i} < \texttt{a}) \vee \texttt{i} > 0$ is an invariant, but we want a minimal one. We thus restart the algorithm.

$$\begin{aligned} F \triangleq \quad & ((\mathtt{i} = 0 \land \mathtt{a} \geqslant 1) \implies \mathcal{T}(\mathtt{b}, \mathtt{i}, \mathtt{a})) \\ & \land ((\mathtt{b'} \land \mathtt{i'} = \mathtt{i} + 1) \lor (\neg \mathtt{b'} \land \mathtt{i'} = \mathtt{i})) \\ & \land (\mathcal{T}(\mathtt{b}, \mathtt{i}, \mathtt{a}) \implies (\mathcal{T}(\mathtt{b'}, \mathtt{i'}, \mathtt{a'}) \lor \mathtt{i'} \geqslant \mathtt{a'})) \end{aligned}$$

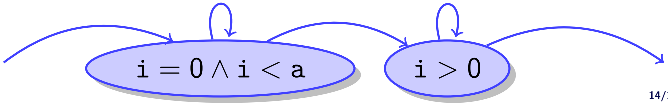
We set $n = 2$ and the predicates $\{\pi_1, \ldots, \pi_8\} \triangleq \{\mathtt{i} = 0, \mathtt{i} < 0, \mathtt{i} > 0, \mathtt{i} = \mathtt{a}, \mathtt{i} < \mathtt{a}, \mathtt{i} > \mathtt{a}, \mathtt{b}, \neg\mathtt{b}\}$.

| $H$ | $SAT(H)$: invariant candidate | $SMT(\neg F[B/b])$ |
|---|---|---|
| true | $(\mathtt{i} = 0 \land \mathtt{i} = \mathtt{a} \land \mathtt{b}) \lor (\mathtt{i} = 0 \land \mathtt{i} = \mathtt{a} \land \neg\mathtt{b})$ | $\mathtt{i} = 0, \mathtt{a} = 1, \mathtt{b} = \bot$ |
| $F(0, 1, \bot)$ | $(\mathtt{i} = 0 \land \mathtt{i} = \mathtt{a} \land \mathtt{b}) \lor (\mathtt{i} = 0 \land \mathtt{i} < \mathtt{a} \land \mathtt{b})$ | $\mathtt{i} = 0, \mathtt{a} = -1, \mathtt{b} = \bot$ |
| ⋮ | | |
| $H_6$ | $(\mathtt{i} = 0 \land \mathtt{i} < \mathtt{a}) \lor \mathtt{i} > 0$ | *accepted*! |

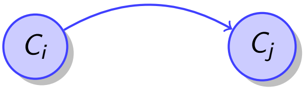
Thus $I_1 = (\mathtt{i} = 0 \land \mathtt{i} < \mathtt{a}) \lor \mathtt{i} > 0$ is an invariant, but we want a minimal one. We thus restart the algorithm.

- When adding the new condition $I_2 \subsetneq I_1$, we get within two steps that no new solution exists.
- Thus $I_1 = (\mathtt{i} = 0 \wedge \mathtt{i} < \mathtt{a}) \vee \mathtt{i} > 0$ was already a minimal one.

```
bool b;
int i = 0, a ; /* precondition a > 0 */
while ( i < a ) {
    b = random ( ) ;
        if (b)
            i = i + 1;
}
```



A state: a conjonction $C_i$ of predicates.




There exists a transition from the state $i$ to the state $j$ iff there exists variables following $C_i$ whose next variables follows $C_j$.

Using *quantifier elimination* on $\exists \sigma, \sigma', C_i(\sigma) \wedge C_j(\sigma') \wedge T(\sigma, \sigma')$, we can precise the transition by a condition on inputs.

$\rightarrow$ MJOLLNIR tool.

**Idea:** modifying $F \longrightarrow$ add any constraint we want.

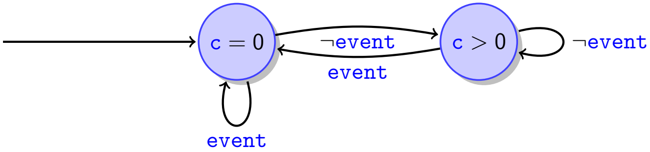
$\rightarrow$ implement different languages (adding precondition, postconditions, ...).

$\rightarrow$ Used to decrease the number of time SMT-solvers are called (*but this increases their computation time*).

- **Removal of Permutations**
  We add a canonical ordering for the disjunction $C_1 \wedge \ldots \wedge C_n$.
- **Satisfiability of Conjonctions**
  We add the condition that each $C_i$ is satisfiable. (require a SMT-solver.)
- **Removal of Subsumed Disjuncts**
  We require that no $C_i$ is subsumed by an other $C_j$.

Reactive node + Set of predicates $\longrightarrow$ invariant

$\longrightarrow$ inductive disjunctive invariant $\Rightarrow$ decomposition in states
$\longrightarrow$ labelling of transitions by quantifier elimination
$\longrightarrow$ automaton:




$\longrightarrow$ easy to change and optimize.