

# Data Abstraction

## The Common Wisdom:

To verify a reactive system  $S$ ,

- If it is finite state, model check it.
- Otherwise, prove it by temporal deduction, using a temporal deductive system such as [MP] or TLA, supported by theorem provers, such as STeP, TLP, or PVS.

Often, both approaches to verification can be simplified by using abstraction.

## AAV: Abstraction Aided Verification

An **Obvious** idea:

- **Abstract** system  $S$  into  $S_A$  – a simpler system, but admitting more behaviors.
- **Verify** property for the **abstracted system**  $S_A$ .
- **Conclude** that property holds for the **concrete system**.

Approach is particularly **impressive** when abstracting an **infinite-state** system into a **finite-state** one.

## Can Abstraction Replace Deduction?

An intriguing question is whether abstraction can completely replace the need for Temporal Deduction.

That is, is it the case that, for every (possibly infinite-state) system  $\mathcal{D}$  and a property  $\psi$  valid for  $\mathcal{D}$ , we can find an abstraction  $\alpha$  such that  $\mathcal{D}^\alpha$  is finite-state,  $\psi^\alpha$  is propositional, and  $\mathcal{D}^\alpha \models \psi^\alpha$ ?

This will relegate temporal reasoning to the regime of automatic model checking techniques.

### Possible Advantages:

- First-order temporal reasoning is more difficult to master than first-order state reasoning.
- People (engineers) find it easier to program, than to write logical formulas. An abstraction is easier to develop (program) than an invariant assertion.

## Finitary Abstraction

Based on the notion of **abstract interpretation** [CC77].

Let  $\Sigma$  denote the set of states of an FDS  $\mathcal{D}$  – the **concrete states**. Let  $\alpha : \Sigma \mapsto \Sigma_A$  be a mapping of concrete into **abstract states**.  $\alpha$  is **finitary** if  $\Sigma_A$  is finite.

We formulate the strategy of **Verification by finitary Abstraction**:

- **Define** a finitary **abstraction mapping**  $\alpha$  to abstract the (possibly infinite) concrete FDS  $\mathcal{D}$  into a finite-state **abstract FDS**  $\mathcal{D}^\alpha$ .
- **Abstract** the **temporal property**  $\psi$  into a **finitary abstract property**  $\psi^\alpha$ .
- **Model Check**  $\mathcal{D}^\alpha \models \psi^\alpha$ .
- **Conclude**  $\mathcal{D} \models \psi$ .

The question is how to define the abstractions  $\mathcal{D}^\alpha$  and  $\psi^\alpha$  such that  $\mathcal{D}^\alpha \models \psi^\alpha$  implies  $\mathcal{D} \models \psi$ ?

That is, how to ensure that the abstraction is **sound**.

## Example: Program ANY-Y

Consider the program

$x, y : \text{integer initially } x = y = 0$

$$P_1 :: \left[ \begin{array}{l} \ell_0 : \text{while } x = 0 \text{ do} \\ \quad [\ell_1 : y := y + 1] \\ \ell_2 : \end{array} \right] \quad || \quad P_2 :: \left[ \begin{array}{l} m_0 : x := 1 \\ m_1 : \end{array} \right]$$

Assume we wish to verify the property  $\square (y \geq 0)$  for system ANY-Y.

We introduce two abstract variables:

$X : \text{boolean}, \quad Y : \{-1, 0, +1\}$

The **abstraction mapping**  $\alpha$  is specified by the following list of defining expressions:

$\alpha : \quad [X = (x \neq 0), \quad Y = \text{sign}(y)]$

where  $\text{sign}(y)$  is defined to be  $-1$ ,  $0$ , or  $1$ , according to whether  $y$  is negative, zero, or positive, respectively.

## The Abstracted Version

With the mapping  $\alpha$ , we can obtain the abstract version of ANY-Y, called ANY-Y $^\alpha$ :

$X$ : **boolean**    **where**  $X = 0$   
 $Y$ :  $\{-1, 0, 1\}$     **where**  $Y = 0$

$$P_1 :: \left[ \begin{array}{l} l_0 : \text{while } X = 0 \text{ do} \\ \left[ \begin{array}{l} l_1 : Y := \left( \begin{array}{l} \text{if } Y = -1 \\ \text{then } \{-1, 0\} \\ \text{else } 1 \end{array} \right) \\ l_2 : \end{array} \right] \end{array} \right] \parallel P_2 :: \left[ \begin{array}{l} m_0 : X := 1 \\ m_1 : \end{array} \right]$$

The original invariance property  $\psi: \square (y \geq 0)$ , is abstracted into:

$$\psi^\alpha: \square (Y \in \{0, 1\}),$$

which can be **model-checked** over ANY-Y $^\alpha$ .

## Abstraction of Assertions

Assume that the **abstraction mapping** is given as  $V_A = \mathcal{E}^\alpha(V)$ . How to **lift**  $\alpha$  from **states** to **assertions**?

There are two (**dual**) ways to abstract an **assertion**  $p$ :

$$\begin{aligned} \alpha^+(p): \quad & \exists V (V_A = \mathcal{E}^\alpha(V) \quad \wedge \quad p(V)) && \text{and} \\ \alpha^-(p): \quad & \text{map}(V_A) \wedge \forall V (V_A = \mathcal{E}^\alpha(V) \rightarrow p(V)) && \text{where} \\ & \text{map}(V_A) : \quad \exists V : V_A = \mathcal{E}^\alpha(V) \end{aligned}$$

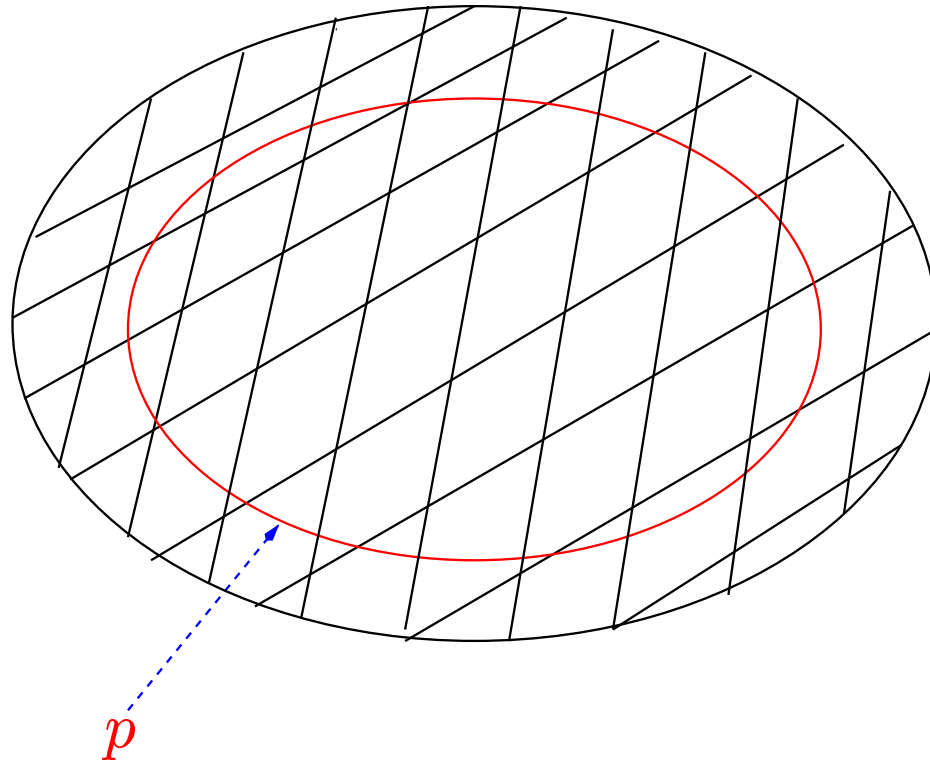
Observe:

- An **abstract** state  $S \in \Sigma_A$  satisfies  $\alpha^+(p)$  iff some **concrete** state  $s \in \alpha^{-1}(S)$  satisfies  $p$ .
- An **abstract** state  $S \in \Sigma_A$  satisfies  $\alpha^-(p)$  iff  $\alpha^{-1}(S) \neq \emptyset$  and **all** **concrete** states  $s \in \alpha^{-1}(S)$  satisfy  $p$ .

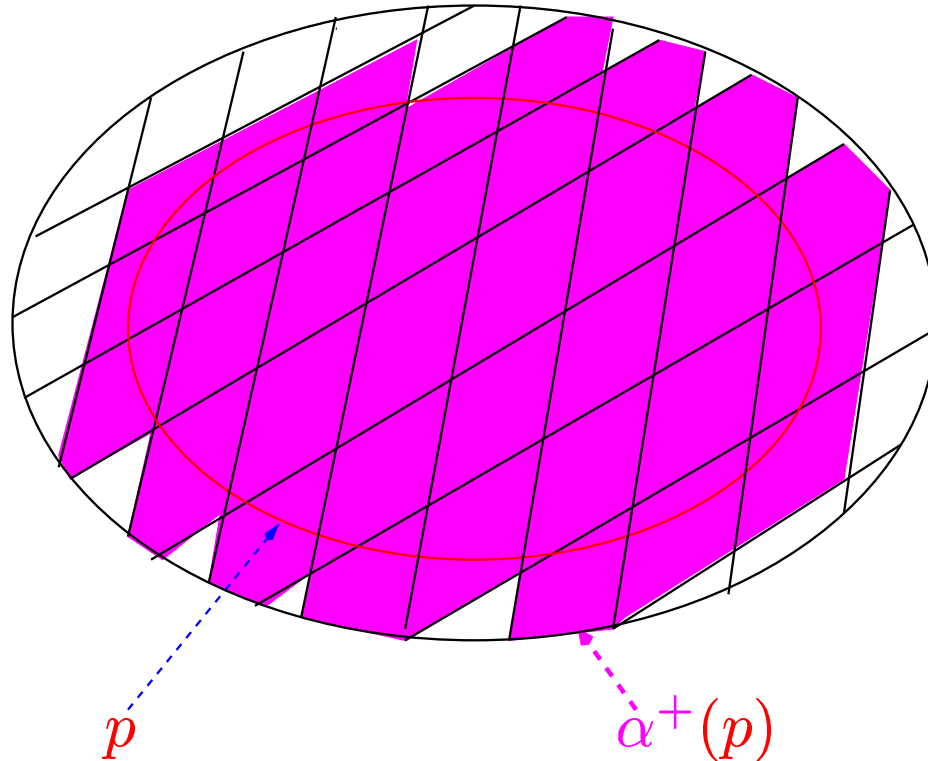
Equivalently:

$$\underbrace{\alpha^{-1}(\|\alpha^-(p)\|)}_{\text{contracting}} \subseteq \|p\| \subseteq \underbrace{\alpha^{-1}(\|\alpha^+(p)\|)}_{\text{expanding}}$$

# The Two Abstraction Transformers



# The Existential (expanding) Abstraction



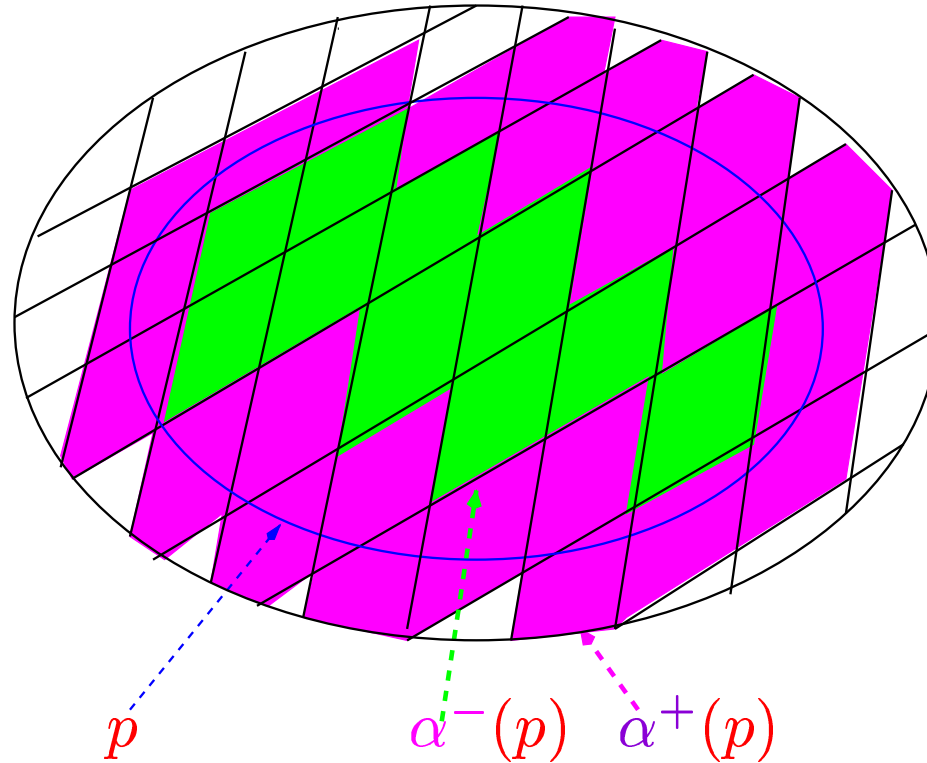
An abstract state  $S$  belongs to  $\alpha^+(p)$  if some concrete state  $\alpha$ -mapped into  $S$  satisfies  $p$ .

For example,

$$\begin{aligned} \alpha^+(0 \leq y \leq 5) &= \exists y : (Y = \text{sign}(y) \wedge 0 \leq y \leq 5) \\ &\sim Y \in \{0, 1\} \end{aligned}$$

$$\alpha^{-1}(\|Y \in \{0, 1\}\|) = \{y \mid y \geq 0\} \supseteq \{y \mid 0 \leq y \leq 5\}$$

# The Universal (contracting) Abstraction



Abstract state  $S$  belongs to  $\alpha^-(p)$  if all concrete states  $\alpha$ -mapped into  $S$  satisfy  $p$ . For example,

$$\begin{aligned} \alpha^-(0 \leq y \leq 5) &= \\ &\exists y : (Y = \text{sign}(y)) \wedge \forall y : (Y = \text{sign}(y) \rightarrow 0 \leq y \leq 5) \\ &\sim Y = 0 \end{aligned}$$

$$\alpha^{-1}(\|Y = 0\|) = \{0\} \subseteq \{y \mid 0 \leq y \leq 5\}$$

## Which Should we use?

For abstracting the **transition relation**, we use  $\alpha^+$ , while for abstracting the **property**, we use  $\alpha^-$ .

An informal argument is that, in order to increase

$$\|\mathcal{D}\| \cap \|\neg\psi\| = \|\mathcal{D}\| \cap \overline{\|\psi\|},$$

we should expand  $\|\mathcal{D}\|$  while contracting  $\|\psi\|$ .

## Additional Evidence

Consider two assertions which are candidates for invariants of program  $\text{ANY-Y}$ , and their possible abstractions:

	$p$	$\alpha^+(p)$	$\alpha^-(p)$
$p_1 :$	$y \geq 0$	$Y \in \{0, 1\}$	$Y \in \{0, 1\}$
$p_2 :$	$0 \leq y \leq 5$	$Y \in \{0, 1\}$	$Y = 0$

It is not difficult to see that program  $\text{ANY-Y}^\alpha$  has essentially only two  $Y$ -observations which are given by

$\langle Y : 0 \rangle, \langle Y : 0 \rangle, \langle Y : 0 \rangle, \dots$  and  $\langle Y : 0 \rangle, \dots, \langle Y : 0 \rangle, \langle Y : 1 \rangle, \langle Y : 1 \rangle, \dots$

The following table indicates which of the assertions  $p_i, \alpha^+(p_i), \alpha^-(p_i)$ , is a valid invariant over its respective system.

	$p$	$\alpha^+(p)$	$\alpha^-(p)$
$p_1 :$	valid	valid	valid
$p_2 :$	invalid	valid	invalid

In a sound abstraction method  $\mathcal{D}^\alpha \models p^\alpha$  should imply  $\mathcal{D} \models p$ . Consequently, only  $\alpha^-(p)$  is a feasible candidate for the abstraction to be applied to the property  $p$ .

## Sound Temporal Abstraction

For a temporal formula  $\psi$ , let  $\psi^\alpha$  be obtained from  $\psi$  by replacing

- Every maximal  $p$ , a sub-assertion of  $\psi$  of positive polarity (enclosed within an even number of negations), by  $\alpha^-(p)$ , and
- Every maximal  $p$ , a sub-assertion of  $\psi$  of negative polarity (enclosed within an odd number of negations), by  $\alpha^+(p)$ .

Then

### Claim 12. [Soundness of an LTL Abstraction]

For a system  $\mathcal{D}$  and temporal property  $\psi$ ,

$$\models \psi^\alpha \quad \text{implies} \quad \models \psi.$$

This claim is based on the observation (provable by induction on the structure of  $\psi$ ) that, for every state sequence  $\sigma : s_0, s_1, \dots$ , and every position  $j \geq 0$ ,

$$(\alpha(\sigma), j) \models \psi^\alpha \quad \text{implies} \quad (\sigma, j) \models \psi.$$

Let us provide more details about the proof of Claim 12. With no loss of generality, assume that  $\psi$  uses the boolean operators  $\vee, \wedge, \neg$  and the temporal operators  $\bigcirc, \square, \diamond, \mathcal{U}, \mathcal{V}$ , where  $p \mathcal{V} q = \neg((\neg p) \mathcal{U} (\neg q))$ .

## Proof of Soundness Continued

Consider first the case that  $\psi$  is a formula in a **positive form**, i.e., that negations can only appear within state sub-formulas. Let  $p_1, \dots, p_n$  be all the maximal sub-assertions of  $\psi$  which, by definition, have each a positive polarity. Writing  $\psi = \psi(p_1, \dots, p_m)$ , we have that  $\psi^\alpha = \psi(\alpha^-(p_1), \dots, \alpha^-(p_m))$ . We proceed to show by induction on the size of  $\psi$  that  $(\alpha(\sigma), j) \models \psi^\alpha$  implies  $(\sigma, j) \models \psi$ . If  $\sigma = s_0, s_1, \dots$ , denote  $\alpha(\sigma) = S_0, S_1, \dots$ , where  $S_j = \alpha(s_j)$ , for every  $j = 0, 1, \dots$

- For the case that  $\psi$  is an assertion, i.e.,  $\psi = p_i$ , then  $(\alpha(\sigma), j) \models \psi$  implies  $S_j \models \alpha^-(p_i)$ . Recall that this is the case if  $S_j$  has at least one  $\alpha$ -source and  $s \models p_i$  for every  $s$  such that  $\alpha(s) = S_j$ . Since  $\alpha(s_j) = S_j$ , we conclude that  $s_j \models p_i$  and, therefore  $(\sigma, j) \models p_i$ , leading to  $(\sigma, j) \models \psi$
- For the case that  $\psi = p \wedge q$ , then since  $(p \wedge q)^\alpha = p^\alpha \wedge q^\alpha$ ,  $(\alpha(\sigma), j) \models \psi^\alpha$  implies  $(\alpha(\sigma), j) \models p^\alpha$  and  $(\alpha(\sigma), j) \models q^\alpha$ . By the induction hypothesis, this implies that  $(\sigma, j) \models p$  and  $(\sigma, j) \models q$ , leading to  $(\sigma, j) \models p \wedge q$ , therefore,  $(\sigma, j) \models \psi$ . The case that  $\psi = p \vee q$  is treated in a similar way.
- Next, consider the case that  $\psi = \square p$ . Since  $(\square p)^\alpha = \square (p^\alpha)$ ,  $(\alpha(\sigma), j) \models \psi^\alpha$  implies  $(\alpha(\sigma), k) \models p^\alpha$ , for every  $k \geq j$ . By the induction hypothesis, it follows that  $(\sigma, k) \models p$ , for every  $k \geq j$  and, therefore,  $(\sigma, j) \models \square p = \psi$ . The case of the other temporal operators is treated similarly.

## Formulas not in Positive Form

Next, consider the case that  $\psi$  is not necessarily in positive form. Assume that  $\psi$  contains the maximal sub-assertions  $p_1, \dots, p_m$  under positive polarity and the maximal sub-assertions  $q_1, \dots, q_n$  under negative polarity. Let  $\varphi = \text{pos}(\psi)$  be the **positive-form** formula congruent to  $\psi$ . Formula  $\varphi$  can be obtained from  $\psi$  by repeatedly applying the following rewrite transformations until none is applicable anymore:

$$\begin{array}{lcl}
 \neg(p \vee q) & \longrightarrow & (\neg p) \wedge (\neg q) \\
 \neg(p \wedge q) & \longrightarrow & (\neg p) \vee (\neg q) \\
 \neg\neg p & \longrightarrow & p \\
 \neg\bigcirc p & \longrightarrow & \bigcirc\neg p \\
 \neg\Box p & \longrightarrow & \Diamond\neg p \\
 \neg\Diamond p & \longrightarrow & \Box\neg p \\
 \neg(p \mathcal{U} q) & \longrightarrow & (\neg p) \mathcal{V} (\neg q) \\
 \neg(p \mathcal{V} q) & \longrightarrow & (\neg p) \mathcal{U} (\neg q)
 \end{array}$$

It is not difficult to see that  $\varphi$  will have the form  $\varphi(p_1, \dots, p_m, \neg q_1, \dots, \neg q_n)$ , where  $p_1, \dots, p_m, q_1, \dots, q_n$  are the maximal sub-assertions occurring in  $\psi$ .

## Proof Continued

Obviously,

$$\psi(p_1, \dots, p_m, q_1, \dots, q_n) \approx \varphi(p_1, \dots, p_m, \neg q_1, \dots, \neg q_n)$$

Assume that  $(\alpha(\sigma), j) \models \psi^\alpha$ . Applying the recipe for abstracting a temporal property, we get  $\psi^\alpha = \psi(\alpha^-(p_1), \dots, \alpha^-(p_m), \alpha^+(q_1), \dots, \alpha^+(q_n))$ . In comparison, applying the same recipe to the positive form formula  $\varphi$  yields  $\varphi^\alpha = \varphi(\alpha^-(p_1), \dots, \alpha^-(p_m), \alpha^-(\neg q_1), \dots, \alpha^-(\neg q_n))$ .

If we could establish that  $\alpha^-(\neg q_k)$  is congruent to  $\alpha^+(q_k)$ , for every  $k = 1, \dots, n$ , we would establish that  $\psi^\alpha$  is congruent to  $\varphi^\alpha$ . In general,  $\alpha^-(\neg q_k)$  is not congruent to  $\alpha^+(q_k)$ . However, the two are congruent over all abstract states  $S_i$  which have at least one  $\alpha$ -source. We conclude that  $\psi^\alpha$  is congruent to  $\varphi^\alpha$  over all sequences of the form  $\alpha(\sigma)$ .

It follows that  $((\alpha(\sigma), j) \models \psi^\alpha$  implies  $((\alpha(\sigma), j) \models \varphi^\alpha$ . Since  $\varphi$  is in positive form, this implies  $(\sigma, j) \models \varphi$ , from which we can conclude  $(\sigma, j) \models \psi$ .

## And Now to Systems

Given an FDS  $\mathcal{D} = \langle V, \mathcal{O}, W, \Theta, \rho, \mathcal{J}, \mathcal{C} \rangle$ , there exists a temporal formula  $Sem(\mathcal{D})$ , called the **temporal semantics** of  $\mathcal{D}$ , such that, for every infinite state sequence  $\sigma$ ,

$$\sigma \models Sem(\mathcal{D}) \quad \text{iff} \quad \sigma \text{ is a computation of } \mathcal{D}.$$

$Sem(\mathcal{D})$  is given by:

$$\Theta \wedge \square \rho(V, \bigcirc V) \wedge \bigwedge_{J \in \mathcal{J}} \square \diamond J \wedge \bigwedge_{(p,q) \in \mathcal{C}} (\square \diamond p \rightarrow \square \diamond q)$$

Given a verification problem  $\mathcal{D} \stackrel{?}{\models} \psi$ , we construct the temporal formula

$$Ver(\mathcal{D}, \psi): Sem(\mathcal{D}) \rightarrow \psi.$$

It is not difficult to establish that  $\mathcal{D} \models \psi$  iff  $Ver(\mathcal{D}, \psi)$  is **valid**.

## Sound Joint Abstraction

For an FDS  $\mathcal{D} = \langle V, \mathcal{O}, W, \Theta, \rho, \mathcal{J}, \mathcal{C} \rangle$ , we define the  $\alpha$ -abstracted version of  $\mathcal{D}$  to be the FDS  $\mathcal{D}^\alpha = \langle V_A, \mathcal{O}_A, W_A, \Theta^\alpha, \rho^\alpha, \mathcal{J}^\alpha, \mathcal{C}^\alpha \rangle$ , where

$$\begin{aligned} \Theta^\alpha &= \alpha^+(\Theta) \\ \rho^\alpha &= \alpha^{++}(\rho) \\ \mathcal{J}^\alpha &= \{\alpha^+(J) \mid J \in \mathcal{J}\} \\ \mathcal{C}^\alpha &= \{(\alpha^-(p), \alpha^+(q)) \mid (p, q) \in \mathcal{C}\} \end{aligned}$$

Where,

$$\alpha^{++}(\rho) = \exists V, V' \left( \begin{array}{l} V_A = \mathcal{E}^\alpha(V) \wedge V'_A = \mathcal{E}^\alpha(V') \\ \wedge \rho(V, V') \end{array} \right)$$

### Soundness:

If  $\mathcal{D}$  and  $\psi$  are abstracted according to the recipe presented above, then

$$\mathcal{D}^\alpha \models \psi^\alpha \quad \text{implies} \quad \mathcal{D} \models \psi.$$

## Predicate Abstraction

An important question is how to choose the abstraction mapping  $\alpha = \mathcal{E}_\alpha : \Sigma \mapsto \Sigma_A$  in a way that will lead to a correct proof. One partial answer is provided by the method of **predicate abstraction**.

Let  $p_1, p_2, \dots, p_k$  be the set of all atomic formulas referring to the **data** (non-control) variables appearing within conditions in the program **P** and within the temporal formula  $\psi$ .

Following [BBM95], define **abstract boolean variables**  $B_{p_1}, B_{p_2}, \dots, B_{p_k}$ , one for each atomic data formula. The abstraction mapping  $\alpha$  is defined by

$$\alpha: \{B_{p_1} = p_1, B_{p_2} = p_2, \dots, B_{p_k} = p_k\}$$

## Example: Program BAKERY-2

**local**  $y_1, y_2$  : **natural** **initially**  $y_1 = y_2 = 0$

$$P_1 :: \left[ \begin{array}{l} l_0 : \text{loop forever do} \\ \left[ \begin{array}{l} l_1 : \text{Non-Critical} \\ l_2 : y_1 := y_2 + 1 \\ l_3 : \text{await } y_2 = 0 \vee y_1 < y_2 \\ l_4 : \text{Critical} \\ l_5 : y_1 := 0 \end{array} \right] \end{array} \right]$$

||

$$P_2 :: \left[ \begin{array}{l} m_0 : \text{loop forever do} \\ \left[ \begin{array}{l} m_1 : \text{Non-Critical} \\ m_2 : y_2 := y_1 + 1 \\ m_3 : \text{await } y_1 = 0 \vee y_2 \leq y_1 \\ m_4 : \text{Critical} \\ m_5 : y_2 := 0 \end{array} \right] \end{array} \right]$$

The temporal properties for program BAKERY-2 are

$$\begin{aligned}
 \psi_{exc} & : \square \neg (at\_l_4 \wedge at\_m_4) \\
 \psi_{acc} & : \square (at\_l_2 \rightarrow \lozenge at\_l_4),
 \end{aligned}$$

## Abstracting Program BAKERY-2

Define abstract variables  $B_{y_1=0}$ ,  $B_{y_2=0}$ , and  $B_{y_1 < y_2}$ .

**local**  $B_{y_1=0}, B_{y_2=0}, B_{y_1 < y_2} : \text{boolean}$   
**initially**  $B_{y_1=0} = B_{y_2=0} = 1, B_{y_1 < y_2} = 0$

$P_1 :: \left[ \begin{array}{l} \ell_0 : \text{loop forever do} \\ \left[ \begin{array}{l} \ell_1 : \text{Non-Critical} \\ \ell_2 : (B_{y_1=0}, B_{y_1 < y_2}) := (0, 0) \\ \ell_3 : \text{await } B_{y_2=0} \vee B_{y_1 < y_2} \\ \ell_4 : \text{Critical} \\ \ell_5 : (B_{y_1=0}, B_{y_1 < y_2}) := (1, \neg B_{y_2=0}) \end{array} \right] \end{array} \right]$

$\parallel$

$P_2 :: \left[ \begin{array}{l} m_0 : \text{loop forever do} \\ \left[ \begin{array}{l} m_1 : \text{Non-Critical} \\ m_2 : (B_{y_2=0}, B_{y_1 < y_2}) := (0, 1) \\ m_3 : \text{await } B_{y_1=0} \vee \neg B_{y_1 < y_2} \\ m_4 : \text{Critical} \\ m_5 : (B_{y_2=0}, B_{y_1 < y_2}) := (1, 0) \end{array} \right] \end{array} \right]$

The abstracted properties can now be **model-checked**.

## Abstraction Alone is Insufficient

Not all properties can be proven by pure finitary abstraction.

Consider the program **LOOP**.

$  \begin{array}{l}  y: \text{ natural} \\  l_0: \text{ while } y > 0 \text{ do} \\  \quad \left[ \begin{array}{l}  l_1: y := y - 1 \\  l_2: \text{ skip}  \end{array} \right] \\  l_3:  \end{array}  $
---

**Termination** of this program cannot be proven by pure finitary abstraction. For example, the abstraction  $\alpha: \mathbb{N} \mapsto \{0, 1\}$  leads to the abstracted program

$  \begin{array}{l}  Y: \{0, 1\} \\  l_0: \text{ while } Y = 1 \text{ do} \\  \quad \left[ \begin{array}{l}  l_1: Y := \text{sub1}(Y) \\  l_2: \text{ skip}  \end{array} \right] \\  l_3:  \end{array}  $
---

where

$$\text{sub1}(Y) = \text{if } Y = 1 \text{ then } \{0, 1\} \text{ else } 0$$

This abstracted program may **diverge**!

# The Solution: Augment the Program with a Non-Constraining Progress Monitor

$$\begin{array}{c}
 y: \text{ natural} \\
 \left[ \begin{array}{l}
 l_0 : \text{ while } y > 0 \text{ do} \\
 \quad \left[ \begin{array}{l}
 l_1 : y := y - 1 \\
 l_2 : \text{ skip}
 \end{array} \right] \\
 l_3 :
 \end{array} \right] \quad ||| \quad \left[ \begin{array}{l}
 inc : \{-1, 0, 1\} \\
 \text{compassion} \\
 \quad (inc < 0, inc > 0) \\
 \text{always do} \\
 m_0 : inc := \text{sign}(y' - y)
 \end{array} \right] \\
 \text{--- LOOP ---} \qquad \qquad \qquad \text{--- MONITOR } M_y \text{ ---}
 \end{array}$$

Forming the cross product, we obtain:

$$\begin{array}{l}
 y \quad : \text{ natural} \\
 inc \quad : \{-1, 0, 1\} \\
 \text{compassion } (inc < 0, inc > 0) \\
 l_0 : \text{ while } y > 0 \text{ do} \\
 \quad \left[ \begin{array}{l}
 l_1 : (y, inc) := (y - 1, \text{sign}(y' - y)) \\
 l_2 : \quad \quad inc := \quad \quad \quad \text{sign}(y' - y)
 \end{array} \right] \\
 l_3 :
 \end{array}$$

# Abstracting the Augmented System

We obtain the program

$Y$	:	$\{0, 1\}$																					
$inc$	:	$\{-1, 0, 1\}$																					
<b>compassion</b>		$(inc < 0, inc > 0)$																					
$l_0$ :	<b>while</b>	$Y = 1$ <b>do</b>																					
	[	<table style="border: none;"> <tr> <td style="padding-right: 10px;"><math>l_1</math> :</td> <td style="padding-right: 10px;"><math>(Y, inc)</math></td> <td style="padding-right: 10px;">:=</td> <td style="padding-right: 10px;">(</td> <td style="padding-right: 10px;"><b>if</b></td> <td style="padding-right: 10px;"><math>Y = 1</math></td> <td style="padding-right: 10px;">)</td> </tr> <tr> <td></td> <td></td> <td></td> <td></td> <td style="padding-right: 10px;"><b>then</b></td> <td style="padding-right: 10px;"><math>(\{1, 0\}, -1)</math></td> <td></td> </tr> <tr> <td></td> <td></td> <td></td> <td></td> <td style="padding-right: 10px;"><b>else</b></td> <td style="padding-right: 10px;"><math>(0, 0)</math></td> <td></td> </tr> </table>	$l_1$ :	$(Y, inc)$	:=	(	<b>if</b>	$Y = 1$	)					<b>then</b>	$(\{1, 0\}, -1)$						<b>else</b>	$(0, 0)$	
$l_1$ :	$(Y, inc)$	:=	(	<b>if</b>	$Y = 1$	)																	
				<b>then</b>	$(\{1, 0\}, -1)$																		
				<b>else</b>	$(0, 0)$																		
	]																						
	$l_2$ :	$inc := 0$																					
$l_3$ :																							

Which **always terminate**.

## A More Complicated Case

Sometimes we need a more complex progress measure:

```

      y: natural

l0: while y > 1 do
      [
        l1: y := y - 2
        l2: y := {y + 1, y}
        l3: skip
      ]
l4:
  
```

To prove termination of this program we augment it by the monitor:

```

define       $\delta = y + at\_l_2$ 
inc         : {-1, 0, 1}
compassion (inc < 0, inc > 0)

m0: always do
      inc := sign( $\delta'$  -  $\delta$ )
  
```

## Complicated Case Continued

Augmenting and abstracting, we get:

$Y$  :  $\{0, one, large\}$   
 $inc$  :  $\{-1, 0, 1\}$   
**compassion** ( $inc < 0, inc > 0$ )

$l_0$  : **while**  $Y = large$  **do**

$$\left[ \begin{array}{l} l_1 : (Y, inc) := (sub2(Y), -1) \\ l_2 : (Y, inc) := (\{add1(Y), Y\}, \{0, -1\}) \\ l_3 : \quad inc := 0 \end{array} \right]$$

$l_4$  :

where,

$sub2(Y) =$   
**if**  $Y \in \{0, one\}$  **then** 0 **else**  $\{0, one, large\}$

$add1(Y) =$  **if**  $Y = 0$  **then** one **else** large

This program **always terminates**

# Verification by Augmented Finitary Abstraction - The VAA Method

To verify that  $\psi$  is  $\mathcal{D}$ -valid,

- Optionally choose a non-constraining progress monitor FDS  $M$  and let  $\mathcal{A} = \mathcal{D} \parallel M$ . In case this step is skipped, we let  $\mathcal{A} = \mathcal{D}$ .
- Choose a finitary state abstraction mapping  $\alpha$  and calculate  $\mathcal{A}^\alpha$  and  $\psi^\alpha$  according to the sound recipes.
- Model check  $\mathcal{A}^\alpha \models \psi^\alpha$ .
- Infer  $\mathcal{D} \models \psi$ .

**Claim 13.** *The VAA method is complete, relative to deductive verification.*

That is, whenever there exists a deductive proof of  $\mathcal{D} \models \psi$ , we can find a finitary abstraction mapping  $\alpha$  and a non-constraining progress monitor  $M$ , such that  $\mathcal{A}^\alpha \models \psi^\alpha$ .