

# ROSAEC workshop

Fifth session

## Reduced Product in ASTRÉE

Jérôme Feret

Laboratoire d'Informatique de l'École Normale Supérieure  
INRIA, ÉNS, CNRS

<http://www.di.ens.fr/~feret>

January, 2009

# Overview

1. Introduction
2. Network of domains
3. Domain cooperation
4. Extrapolation strategies
5. Conclusion

# Reduction

The ASTRÉE analyzer is a collection of abstract domains:

1. Main domains are used to express or prove properties of interest.
2. Other domains are used to prove more complex properties that are required in the properties of interest proof.

We need to make domains collaborate to achieve accuracy.

# Optimal V.S. reduced reduction

Let:

- $D, D^\#$  be two partial orders  
( $D^\#$  intends to be a composite abstract domain);
- $D \xrightleftharpoons[\alpha]{\gamma} D^\#$  is a Galois connexion;

A reduction  $\rho : D^\# \rightarrow D^\#$  is a unary operator such that:

$$\gamma(A) \subseteq \gamma(\rho(A)).$$

This is a soundness requirement, in practice we also expect  $\rho(A) \sqsubseteq A$ .

Reductions can be used during the analysis to refine abstract elements.

# Example of reductions

Examples:

1.  $\rho = \alpha \circ \gamma$  is always the most precise reduction, but it might be non-computable or too costly;
2. any  $\rho$  such that  $\alpha \circ \gamma \sqsubseteq \rho$  is a reduction operator;
3. when  $D^\# = \text{Interval} \times \text{Parity}$ ,

$$\rho : \begin{cases} (\emptyset, p) & \mapsto & (\emptyset, \emptyset), \\ (i, \emptyset) & \mapsto & (\emptyset, \emptyset), \\ ([2, 5], \{\text{even}\}) & \mapsto & ([2, 4].\{\text{even}\}), \\ (\{2\}, \{\text{even}, \text{odd}\}) & \mapsto & (\{2\}, \{\text{even}\}), \\ x & \mapsto & x \quad \text{otherwise;} \end{cases}$$

is a reduction operator.

# A reduction scheme for transition systems

## Concrete semantics

Let  $(\mathcal{C}, C_0, \rightarrow)$  be a transition system,

We restrict our study to its collecting semantics:

this is the set of the states that are reachable within a finite transition sequence.

$$\mathcal{S} = \{C \mid \exists i \in C_0, i \rightarrow^* C\}$$

It is also given by the least fixpoint of the following  $\cup$ -complete endomorphism

$\mathbb{F}$ :

$$\mathbb{F} = \begin{cases} \wp(\mathcal{C}) & \rightarrow \wp(\mathcal{C}) \\ X & \mapsto C_0 \cup \{C' \mid \exists C \in X, C \rightarrow C'\} \end{cases}$$

This fixpoint is **usually not computable automatically**.

# A reduction scheme for transition systems

## Abstract domain

We introduce an abstract domain of properties:

- properties of interest;
- more complex properties used in calculating them.

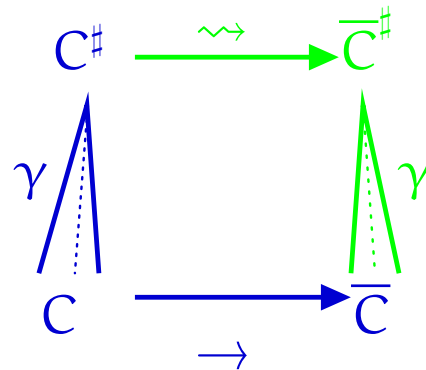
This domain is often a lattice:  $(\mathcal{D}^\#, \sqsubseteq, \sqcup, \perp, \sqcap, \top)$  and is related to the concrete domain  $\wp(\mathcal{C})$  by a monotonic concretization function  $\gamma$ .

$\forall A \in \mathcal{D}^\#, \gamma(A)$  is the set of the elements which satisfy the property  $A$ .

# A reduction scheme for transition systems

## Abstract transition system

Let  $C_0^\sharp$  be an abstraction of the initial states and  $\rightsquigarrow$  be an abstract transition relation, which satisfies  $C_0 \subseteq \gamma(C_0^\sharp)$  and the following diagram:



Then,  $\mathcal{S} \subseteq \bigcup_{n \in \mathbb{N}} \gamma(\mathbb{F}^{\sharp n}(C_0^\sharp))$ ,

where  $\mathbb{F}^\sharp(C^\sharp) = C_0^\sharp \sqcup C^\sharp \sqcup \left( \bigsqcup_{finite} \{\bar{C}^\sharp \mid C^\sharp \rightsquigarrow \bar{C}^\sharp\} \right)$ .

# A reduction scheme for transition systems

## Widening operator

We require a widening operator to ensure the convergence of the analysis:

$$\nabla : D^\# \times D^\# \rightarrow D^\#$$

such that:

- $\forall X_1^\#, X_2^\# \in D^\#, X_1^\# \sqcup X_2^\# \sqsubseteq X_1^\# \nabla X_2^\#$ ;
- for all increasing sequence  $(X_n^\#) \in (D^\#)^\mathbb{N}$ , the sequence  $(X_n^\nabla)$  defined as

$$\begin{cases} X_0^\nabla = X_0^\# \\ X_{n+1}^\nabla = X_n^\nabla \nabla X_{n+1}^\# \end{cases}$$

is ultimately stationary.

# A reduction scheme for transition systems

## Abstract iteration

The abstract iteration  $(C_n^\nabla)$  of  $F^\sharp$  defined as follows

$$\begin{cases} C_0^\nabla = C_0^\sharp \\ C_{n+1}^\nabla = \begin{cases} C_n^\nabla & \text{if } F^\sharp(C_n^\nabla) \sqsubseteq C_n^\nabla \\ C_n^\nabla \nabla F^\sharp(C_n^\nabla) & \text{otherwise} \end{cases} \end{cases}$$

is **ultimately stationary** and its limit  $C^\nabla$  satisfies  $lfp_\emptyset F \subseteq \gamma(C^\nabla)$ .

# A reduction scheme for transition systems

## Composing two abstractions

Given two abstractions  $(\mathcal{D}^\#, \gamma, C_0^\#, \rightsquigarrow, \nabla)$  and  $(\mathcal{D}^\#, \gamma, C_0^\#, \rightsquigarrow, \nabla)$ , and a reduction  $\rho : \mathcal{D}^\# \times \mathcal{D}^\# \rightarrow \mathcal{D}^\# \times \mathcal{D}^\#$  which satisfy:

$$\forall (A, A) \in \mathcal{D}^\# \times \mathcal{D}^\#, \gamma(A) \cap \gamma(A) \subseteq \gamma(a) \cap \gamma(a) \text{ where } (a, a) = \rho(A, A).$$

Then  $(\mathcal{D}^\#, \gamma, C_0^\#, \rightsquigarrow, \nabla)$  where:

- $\mathcal{D}^\# = \mathcal{D}^\# \times \mathcal{D}^\#$ ;
- $\nabla$  is pair-wisely defined;
- $\gamma(A, A) = \gamma(A) \cap \gamma(A)$ ;
- $C_0^\# = \rho(C_0^\#, C_0^\#)$ ;
- $(A, A) \rightsquigarrow \rho(C, C)$   
if  $B \rightsquigarrow C$  and  $B \rightsquigarrow C$  and  $(B, B) = \rho(A, A)$

is also an abstraction.

# A reduction scheme for compositional semantics

## Syntax

Let  $\mathcal{V}$  be a finite set of variables.

Let  $\mathcal{I}$  be the set of real intervals (including  $\mathbb{R}$ ).

Expressions  $\mathcal{E}$  are affine forms of variables  $\mathcal{V}$  with real interval coefficients:

$$E ::= I + \sum_{j \in J} I_j \cdot V_j$$

Programs are given by the following grammar:

```
P ::= skip
    | P;P
    | V := E
    | if (V ≥ 0) {P} else {P}
    | while (V ≥ 0) {P}
```

# A reduction scheme for compositional semantics

## Semantics

$$\llbracket P \rrbracket : (\mathcal{V} \rightarrow \mathbb{R}) \rightarrow \wp(\mathcal{V} \rightarrow \mathbb{R})$$

by induction over the syntax of P:

$$\llbracket \text{skip} \rrbracket(\sigma) = \{\sigma\},$$

$$\llbracket P_1; P_2 \rrbracket(\sigma) = \{\sigma'' \mid \exists \sigma' \in \llbracket P_1 \rrbracket(\sigma), \sigma'' \in \llbracket P_2 \rrbracket(\sigma')\},$$

$$\llbracket V := I + \sum_{j \in J} I_j \cdot V_j \rrbracket(\sigma) = \left\{ \sigma[V \mapsto i + \sum_{j \in J} i_j \cdot \sigma(V_j)] \mid i \in I, \forall j \in J, i_j \in I_j \right\},$$

$$\llbracket \text{if } (V \geq 0) \{P_1\} \text{ else } \{P_2\} \rrbracket(\sigma) = \begin{cases} \llbracket P_1 \rrbracket(\sigma) & \text{if } \sigma(V) \geq 0 \\ \llbracket P_2 \rrbracket(\sigma) & \text{otherwise,} \end{cases}$$

$$\llbracket \text{while } (V \geq 0) \{P\} \rrbracket(\sigma) = \{\sigma' \in \text{Inv} \mid \sigma'(V) < 0\}$$

$$\text{where } \text{Inv} = \text{lfp}(X \mapsto \{\sigma\} \cup \{\sigma'' \mid \exists \sigma' \in X, \sigma'(V) \geq 0 \text{ and } \sigma'' \in \llbracket P \rrbracket(\sigma')\}).$$

# A reduction scheme for compositional semantics

## Abstract domain

An abstract domain  $ENV^\#$  is a set of environment properties.

A concretization map  $\gamma$  relates each property to the set of its solutions:

$$\gamma : ENV^\# \rightarrow \wp(\mathcal{V} \rightarrow \mathbb{R}).$$

Some primitives simulate concrete computation steps in the abstract:

- an abstract control path merge  $\sqcup$ ;
- an abstract guard  $GUARD$  and an abstract assignment  $ASSIGN$ ;
- an abstract least fixpoint  $lfp^\#$  operator, which maps sound counterpart  $f^\#$  to monotonic function  $f$ , to an abstraction of the least fixpoint of  $f$ .

$lfp^\#$  is defined using extrapolation operators  $(\perp, \nabla, \Delta)$ .

Soundness follows from the monotony of the concrete semantics.

# A reduction scheme for compositional semantics

## Abstract semantics

$$\llbracket \text{skip} \rrbracket^\#(a) = a$$

$$\llbracket P_1; P_2 \rrbracket^\#(\sigma^\#) = \llbracket P_2 \rrbracket^\#(\llbracket P_1 \rrbracket^\#(\sigma^\#))$$

$$\llbracket V := E \rrbracket^\#(a) = \text{ASSIGN}(V, E, a)$$

$$\llbracket \text{if } (V \geq 0) \{P_1\} \text{ else } \{P_2\} \rrbracket^\#(a) = a_1 \sqcup a_2,$$

with  $\begin{cases} a_1 = \llbracket P_1 \rrbracket^\#(\text{GUARD}(V, [0; +\infty[, a)) \\ a_2 = \llbracket P_2 \rrbracket^\#(\text{GUARD}(V, ]-\infty; 0[, a)) \end{cases}$

$$\llbracket \text{while } (V \geq 0) \{P\} \rrbracket^\#(a) = \text{GUARD}(V, ]-\infty; 0[, \text{Inv}^\#)$$

where  $\text{Inv}^\# = \text{lfp}^\#(X \mapsto a \sqcup \llbracket P \rrbracket^\#(\text{GUARD}(V, [0; +\infty[, X)))$

# A reduction scheme for compositional semantics

## Soundness

We prove by induction over the syntax:

For any program  $P$ , environment  $\sigma$ , abstract element  $\alpha$ , we have:

$$\sigma \in \gamma(\alpha) \implies \llbracket P \rrbracket(\sigma) \subseteq \gamma(\llbracket P \rrbracket^\#(\alpha)).$$

# Reduction as a refinement of transfer functions

Given a reduction  $\rho \in \text{ENV}^\# \rightarrow \text{ENV}^\#$  such that  $\gamma(a) \sqsubseteq \gamma(\rho(a))$ ,

we can replace each abstract primitive

$$H \in J \times \text{ENV}^{\#n} \rightarrow \text{ENV}^\#,$$

with the following one:

$$H'(X, (a_k)) = \rho(H'(X, (\rho(a_k)))).$$

New abstract primitives also satisfy soundness criteria.

So **soundness is preserved**.

# Reduction and extrapolation

Take  $D^\# = \text{Zone}(\mathbb{R}^2) = \mathcal{I}^3$

with the reduction  $\rho(x, y, z) = (x \sqcap (z + y), y \sqcap (x - z), z \sqcap (x - y))$ ,

and consider the sequence  $n_i = ([-(i+1), i+1], [-(i+1), i+1], [-1, 1])$ .

Compute  $\rho(\sqcup m_i)$  where:

$$\begin{cases} m_0 = ([-0, 0], [-1, 1], [-1, 1]), \\ m_{i+1} = m_i \nabla n_i. \end{cases}$$

We have:

- $m_1 = (\overline{\infty}, \overline{1}, \overline{1})$ ;
- $m_i = (\overline{\infty}, \overline{\infty}, \overline{1})$ .

Then:

$$\rho(\sqcup m_i) = (\overline{\infty}, \overline{\infty}, \overline{1}).$$

Compute  $(\sqcup m_i)$  where:

$$\begin{cases} m_0 = ([-0, 0], [-1, 1], [-1, 1]), \\ m_{i+1} = \rho(m_i \nabla n_i). \end{cases}$$

We have:

- $m_1 = (\overline{2}, \overline{1}, \overline{1})$ ;
- $m_{2i+1} = (\overline{2i+2}, \overline{2i+1}, \overline{1})$ ;
- $m_{2i+2} = (\overline{2i+2}, \overline{2i+3}, \overline{1})$ .

Then  $m_i$  diverges.

# Drawbacks

This classical reduction framework has several drawbacks:

1. Blind homogeneous reduction VS smart heterogeneous reduction:

Reduction is driven by better communication between domains:

- a domain can ask for information that it needs;
- or tell other domains when it has computed crucial information.

2. No reduction after extrapolation steps

- might be the cause of irrecoverable loss of information  
no theoretic comparison since extrapolation operators are non-monotonic;
- requires a very strong reduction before applying other transformers which
  - scan the whole abstract environment;
  - or require costly annotation (of the invariant) to be driven.

# Our goal

To construct a reduction scheme that satisfies the following *contradicting* goals:

1. expressiveness:

it should allow powerful collaboration between domains;

2. freedom:

it should allow reduction wherever it is wished;

3. efficiency:

it should not damage analyzer scalability;

4. extensibility:

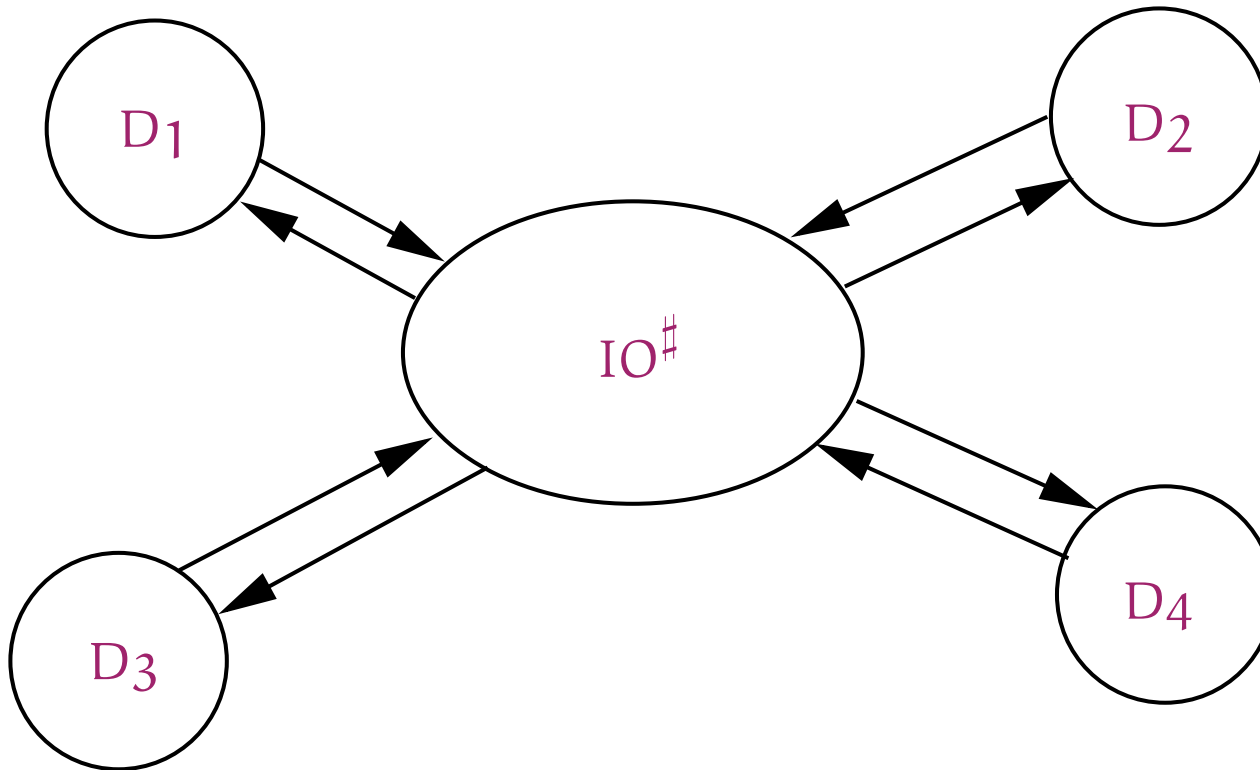
it should allow new *kind of* reductions easy addition.

# Overview

1. Introduction
2. **Network of domains**
3. Domain cooperation
4. Extrapolation strategies
5. Conclusion

# A model of collaboration

Each abstract domain  $D_i$  is seen as an independent unit.  
Collaboration is ensured by an intermediate domain  $IO^\#$ .



# Communication domain

The intermediate domain is used to express the constraints that are communicated between domains.

More formally:

1.  $D$  is the concrete domain;
2.  $D_i^\#$  are abstract domains (with sound primitives);
3.  $\gamma_i : D_i^\# \rightarrow D$  are concretization functions;
4.  $IO^\#$  is a set of abstract properties;
5.  $\gamma_{IO^\#} : IO^\# \rightarrow D$  is a concretization function.

The communication domain requires no abstract primitives, since it is not made for (abstract) computations.

# Reduction primitives

1. Abstract domains can weaken their elements to produce constraints:

$$\text{EXTRACT}_i : D_i^\# \times IO^\# \rightarrow IO^\#,$$

such that:

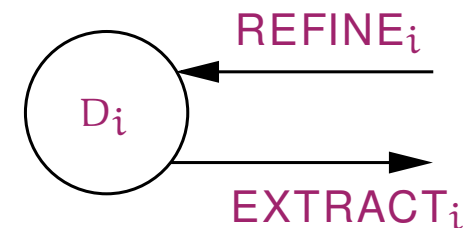
$$\gamma_i(c) \cap \gamma_{IO^\#}(io) \subseteq \gamma_{IO^\#}(\text{EXTRACT}_i(c, io)).$$

2. Abstract domains can weaken communicated constraints to refine their elements:

$$\text{REFINE}_i : D_i^\# \times IO^\# \rightarrow D_i^\#,$$

such that:

$$\gamma_i(c) \cap \gamma_{IO^\#}(io) \subseteq \gamma_i(\text{REFINE}_i(c, io)).$$



# Input channel

Input channels provide information on both:

1. the post-condition being computed;
2. and the precondition computed in the last computation step.

Input channels contain elements in  $IO^\#$ .

A domain  $D_i^\#$  may:

1. read the contents of the channel (using  $REFINE_i$ );
2. update the contents of the post-condition channel at the end of its own computation (using  $EXTRACT_i$ ).

# Output channels

Output channels are used when a domain wants to send a message to others.

A domain may:

- refine previous computation in other domains;
- refine not yet done computations (more costly).

Output channels contain elements in  $IO^\#$ .

The domain  $D_i$  that initiates the refinement uses the primitive  $EXTRACT_i$ .  
Targeted domains use the primitive  $REFINE_i$ .

# Implementation issues

## Abstract computation

### 1. Input channels:

- reduction is initiated by the domain that needs the refinement;
- they are implemented by using *product types*;
- refining domains overload method definitions;
- refined domains use these methods.

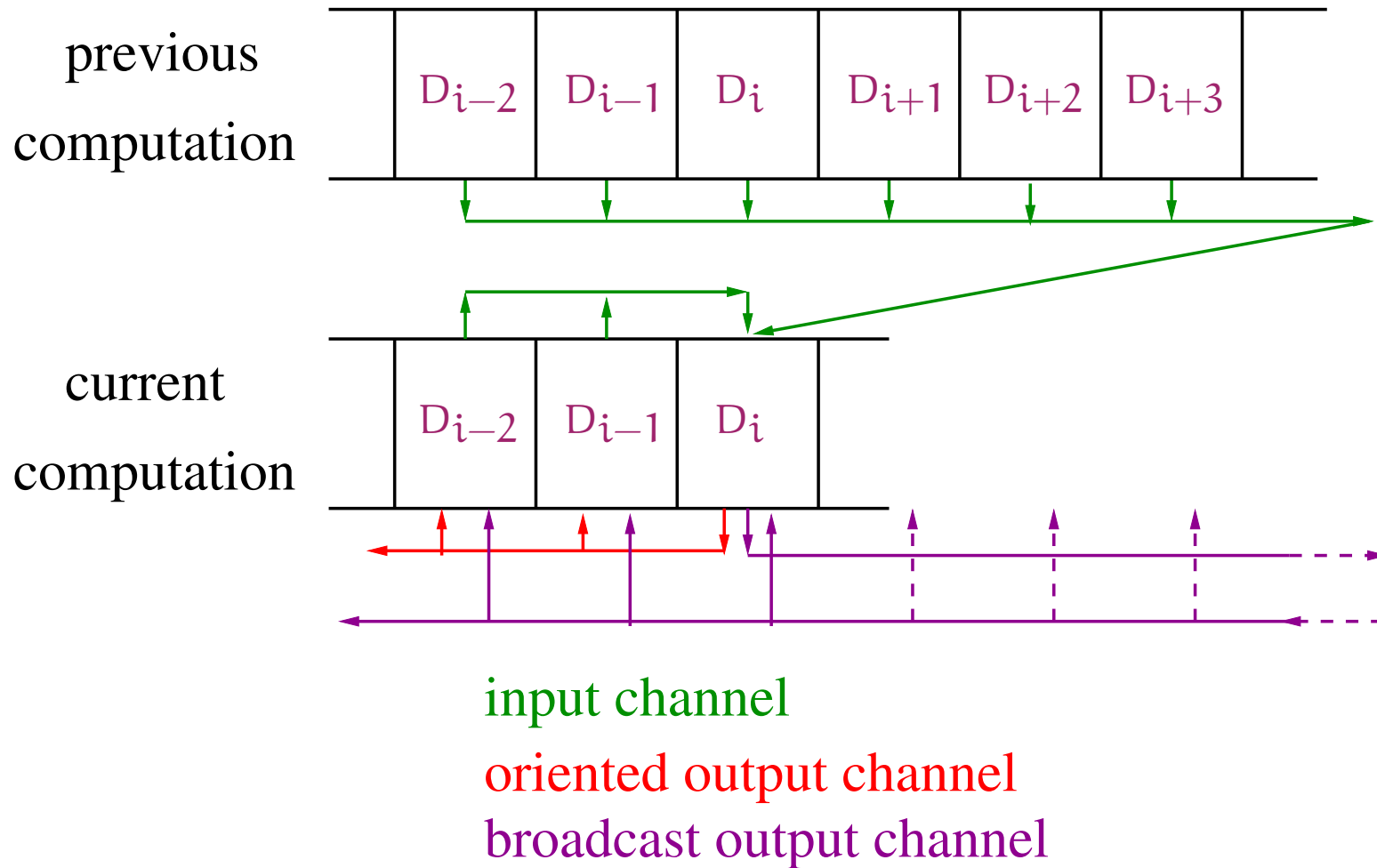
### 2. Output channels:

- reduction is initiated by the domain that has the information;
- they are implemented by using lists of constraints (expressed with *sum types*);
- refining domains add constraints in the list;
- refined domains use pattern matching.

# Overview

1. Introduction
2. Network of domains
3. **Domain cooperation**
4. Extrapolation strategies
5. Conclusion

# Reductions are constrained by computation order



# Formalization of the input channel

Abstract computations are made under assumptions about pre/post-conditions, and refine information about post-conditions.

For any  $n$ -ary concrete transfer function:  $\mathbb{F} \in \mathcal{D}^n \rightarrow \mathcal{D}$ ,  
we associate an abstract counterpart  $\mathbb{F}^\# \in (\mathcal{D}_i^\# \times \mathcal{IO}^\#)^n \times \mathcal{IO}^\# \rightarrow \mathcal{D}_i^\# \times \mathcal{IO}^\#$ ,  
such that:

- $c_i \in \gamma_{\mathcal{D}^\#}(a_i)$ , for  $1 \leq i \leq n$ ,
- $c_i \in \gamma_{\mathcal{IO}^\#}(io_i)$ , for  $1 \leq i \leq n$ ,
- $c_0 \in \gamma_{\mathcal{IO}^\#}(io_0)$ ,
- $c_0 \in \mathbb{F}((c_i)_{1 \leq i \leq n})$ .

implies that:

- $c_0 \in \gamma_i(\text{fst}(\mathbb{F}^\#((a_i, io_i)_{1 \leq i \leq n}, io_0)))$
- $c_0 \in \gamma_{\mathcal{IO}^\#}(\text{snd}(\mathbb{F}^\#((a_i, io_i)_{1 \leq i \leq n}, io_0)))$ .

# Formalization of the output channel

Refinements via the output channel can be modeled as applications of reduction operators  $\rho$ ,

- either on all domains (broadcast);
- or on domains that have already done their computation (oriented).

Each refinement step is:

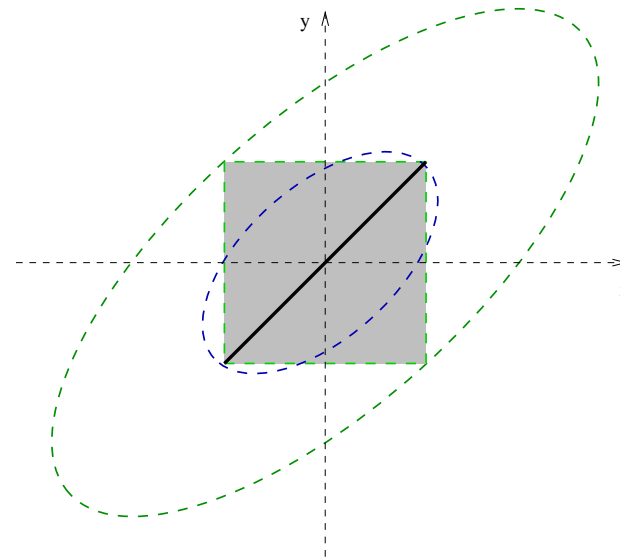
- triggered by the abstract domain having the information;
- depending on:
  - the domain who triggers the refinement;
  - the primitive that triggers the refinement.

# Precondition refinement

Some domains use input channels in order to refine abstract states before they are fed to their abstract transformers.

For instance, filters domains use interval and equality constraints to synthesize predicates:

- before computing assignments;
- before computing binary operators.



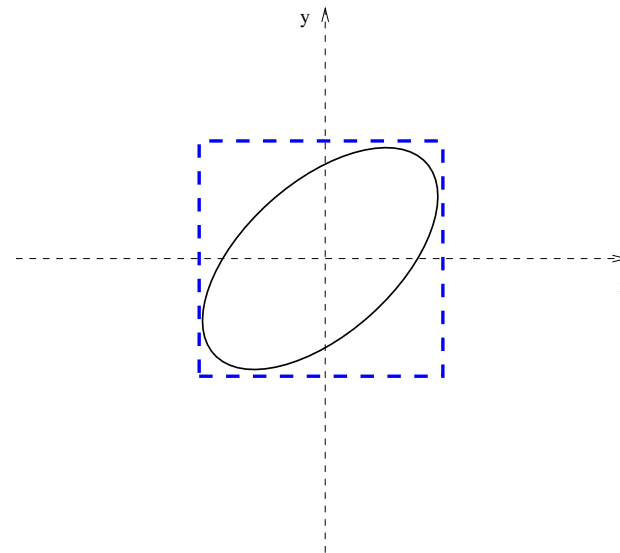
# Post-condition refinement

## When a new constraint is computed

When a domain synthesizes very useful information, it can propagate such information to the underlying domains by using the oriented output channel:

For instance, domain filters deal more accurately with filter iterations than the interval domain.

They communicate a precise bound on filter output to the interval domain after filter iteration.



# Post-condition refinement

## When a domain cannot make a precise computation

Some domains use the input channel in order to refine the information they compute on the post-condition.

Consider, for instance, the assignment  $X=Y$  on an octagon containing  $X$  but not  $Y$ . In that case, the octagon domain relies on the interval domain to compute the actual range of  $X$  in the post-condition.

# Post-condition refinement

## When a domain cannot make a precise computation

Alternatively, a domain may use the broadcast output channel to request further computation to refine its information:

Consider widening in interval domains:

if  $\rho(X) \sqcup \rho'(X) \sqsubset \rho(X) \nabla \rho'(X)$ ,

1. a broadcast message is sent to the output channel;
2. each domain computes a range for  $X$ ;
3. these ranges are communicated to the interval domain thanks to the oriented output channel.

# **Abstract transformer refinement**

## **To provide the appropriate level of abstraction**

Each domain requires an appropriate level of abstraction for expressions:

- Memory domain deals with C expressions;
- Intervals deal with arithmetic expressions over floating-point numbers;
- Relational numeric domains deal with arithmetic expressions over real numbers;
- Expressions can be more or less transformed (symbolic propagation, abstract pattern matching).

Domains use the input channel of preconditions to provide the appropriate level of abstraction to the predicate transformer.

# Abstract transformer refinement

## To drive strategies

Some domains may drive strategy choices in other domains.

When interpreting  $X=Y+Z$ , the arithmetic-geometric progression domain may:

1. replace the variable  $Y$  or the variable  $Z$  with its range;
2. or deal uniformly with  $Y$  and  $Z$   
(interpret  $Y+Z$  as twice the mean between  $X$  and  $Y$ ).

The choice is driven by:

- **dependencies information**: variables not occurring in cycles are replaced with ranges;
- **interval information**: to estimate which strategy gives better results (ultimately).

that are read in the input channel.

# Overview

1. Introduction
2. Network of domains
3. Domain cooperation
4. **Extrapolation strategies**
5. Conclusion

# Motivation

For **efficiency** and/or **accuracy**, we specialize the framework of Abstract Interpretation in order to get **tunable extrapolation strategies**.

We want to:

1. reduce the output of widening steps;
2. use union steps between widening steps.

We strengthen usual definitions (abstract domains, reductions, extrapolation operators)

in order to ensure **analyzer termination**.

# Usual framework: Iterative fixpoint approximation

Let:

- $\mathcal{D}$  be a chain-complete partial order;
- $\mathcal{D}^\#$  be a partial order;
- $\alpha \xleftrightarrow[\mathcal{D}]{\mathcal{D}^\#} \gamma$  be a Galois connexion;
- $\sqcup \in \mathcal{D}^\# \times \mathcal{D}^\# \rightarrow \mathcal{D}^\#$  that satisfies:  $a \sqsubseteq a \sqcup b$  and  $b \sqsubseteq a \sqcup b$ ;
- $\sqcap \in \mathcal{D}^\# \times \mathcal{D}^\# \rightarrow \mathcal{D}^\#$  satisfies  $\gamma(a) \cap \gamma(b) \subseteq \gamma(a \sqcap b)$ ;
- $\mathbb{F} \in \mathcal{D} \rightarrow \mathcal{D}$  be an endomorphism;
- $\mathbb{F}^\#$  be a sound counterpart to  $\mathbb{F}$  (i.e.  $\mathbb{F} \circ \gamma \subseteq \gamma \circ \mathbb{F}^\#$ ).

We know that, for  $x_0 \in \mathcal{D}$ :

- $\text{lfp}_{x_0} \mathbb{F}$  exists and  $\text{lfp}_{x_0} \mathbb{F} = \bigcup \{\mathbb{F}^n(x_0)\}$ ;
- For any  $n \in \mathbb{N}$ ,  $\mathbb{F}^n(x_0) \subseteq \gamma(\mathbb{F}^{\#n}(\alpha(x_0)))$ .

# Usual framework: Widening

A widening  $\nabla \in D^\# \times D^\# \rightarrow D^\#$  is an operator such that:

1.  $\gamma(a) \subseteq \gamma(a \nabla b)$ ;
2.  $\gamma(b) \subseteq \gamma(a \nabla b)$ ;
3. for any increasing sequence  $(a_n)$ , the sequence defined as
  - $a_0^\nabla = a_0$ ,
  - $a_{n+1}^\nabla = a_n^\nabla \nabla a_{n+1}$ ,

is ultimately stationary.

# Usual framework: Widened iterates

Then, the sequence defined as:

- $y_0^\nabla = \alpha(x_0)$ ,
- $y_{n+1}^\nabla = y_n^\nabla \nabla F^\#(y_{n+1}^\nabla)$ ,

is ultimately stationary.

Moreover, the limit  $l$  satisfies:

- $\gamma(F^{\#n}(\alpha(x_0))) \subseteq \gamma(l)$ ;
- $\gamma(F^\#(l)) \subseteq \gamma(l)$ ;
- $lfp_{x_0} F \subseteq \gamma(l)$ .

# Usual framework: Decreasing iterates

When  $F^\sharp$  is monotonic:

- we have:

$$\gamma(F^{\sharp n+1}(L)) \subseteq \gamma(F^{\sharp n}(L));$$

- which ensures that:

1.  $\gamma(F^{\sharp n}(L)) \subseteq \gamma(L)$ ;
2.  $\text{lfp}_{x_0} F \subseteq \gamma(F^{\sharp n}(L))$ .

So we may use a theorem prover to certify (externally) that the result of the analysis is sound.

# Usual framework: Decreasing(?) iterates

If  $D^\sharp$  is not monotonic,

- we still have:

$$\text{lfp}_{x_0} \mathbb{F} \subseteq \gamma \left( \mathbb{F}^{\sharp n}(\perp) \right)$$

(so the analysis is correct by construction);

- but we may have:

$$\gamma \left( \mathbb{F}^{\sharp n+1}(\perp) \right) \not\subseteq \gamma \left( \mathbb{F}^{\sharp n} \right).$$

(in such a case we cannot prove the analysis correction externally).

# Why not widening at each iterate?

The following program:

```
typedef enum {F=0,T=1} BOOL;
float X, Y;
int main () {
    X = 0 ; Y = 0 ;
    while (1) {
        BOOL B;
        if (B) {X = Y + 2;};
        Y = 0.99 * X + 3;}
}
```

does not overflow.

Indeed  $X \subseteq [0, 500] \wedge Y \subseteq [0, 498]$  is an inductive invariant.

# Why not widening at each iterate?

Widening after each iteration may lose information:

```
typedef enum {F=0,T=1} BOOL;
float X, Y;
int main () {
    X = 0 ; Y = 0 ;
    while (1) {
        BOOL B;
        if (B) {X = Y + 2;};
        Y = 0.99 * X + 3;
    }
```

iterate	X	Y
0	$[0, 0]$	$[0, 0]$
1	$[0, 10]$	$[0, 10]$
2	$[0, 10^2]$	$[0, 10^2]$
3	$[0, 10^3]$	$[0, 10^3]$
4	$[0, 10^4]$	$[0, 10^3]$
5	$[0, 10^4]$	$[0, 10^3]$
6	$[0, 10^5]$	$[0, 10^4]$
⋮	⋮	⋮
$(2 \cdot n) + 4$	$[0, 10^{4+n}]$	$[0, 10^{3+n}]$
$(2 \cdot n + 1) + 4$	$[0, 10^{4+n}]$	$[0, 10^{4+n}]$
⋮	⋮	⋮
	$[0, +\infty[$	$[0, +\infty[$

# Using union between widening iterates

We can skip widening steps when one variable becomes stable:

```
typedef enum {F=0,T=1} B00;  
float X, Y;  
int main () {  
    X = 0 ; Y = 0 ;  
    while (1) {  
        B00 B;  
        if (B) {X = Y + 2;};  
        Y = 0.99 * X + 3;}  
}
```

iterate	X	Y
0	[0, 0]	[0, 0]
1	[0, 10]	[0, 10]
2	[0, 100]	[0, 100]
3	[0, 1000]	[0, 1000]
4	[0, 1002]	[0, 1000]
5	[0, 1002]	[0, 1000]
6	[0, 1002]	[0, 1000]

# Why using union steps, between widening steps ?

We can use freshness counter-based widening strategies:

```
typedef enum {F=0,T=1} B00;  
float X, Y;  
int main () {  
    X = 0 ; Y = 0 ;  
    while (1) {  
        B00 B;  
        if (B) {X = Y + 2;};  
        Y = 0.99 * X + 3;}  
}
```

iterate	X		Y	
	range	freshness	range	freshness
0	[0, 0]	2	[0, 0]	2
1	[0, 2]	1	[0, 4.98]	1
2	[0, 10]	2	[0, 10]	2
3	[0, 12]	1	[0, 14.88]	1
4	[0, 100]	2	[0, 100]	2
5	[0, 102]	1	[0, 103.98]	1
6	[0, 1000]	2	[0, 1000]	2
7	[0, 1002]	1	[0, 1000]	2
8	[0, 1002]	1	[0, 1000]	2

# Case study

- $\mathcal{D}^\# = \{X \subseteq [0, 1] \mid \{0, 1\} \in X\}$ ;
- $\mathbb{F}^\# = \left[ S \mapsto S \cup \left\{ \frac{1}{2^{n+1}} \mid \frac{1}{2^n} \in S \right\} \right]$ ;
- $\cap$  and  $\cup$  are classical set operators;
- $\nabla$ :  $a \nabla b$  is computed in two steps:
  1. first we compute  $c = a \cup b$ ;
  2. then we extend the connected component that contains 1 until the number of connected components is strictly less than both 5 and the number of connected components of  $a$ .

We notice that:

- not widening at each step may cause divergence!
- intersecting with stable constraints may cause divergence!

# Usual widening sequences

Let us compute the upward extrapolated iteration starting with the following element:

$$\left\{0, \frac{1}{2}, 1\right\}.$$

We get:

- $X_0 = \{0, \frac{1}{2}, 1\}$
- $\mathbb{F}^\sharp(X_0) = \{0, \frac{1}{4}, \frac{1}{2}, 1\}$
- $X_1 \stackrel{\Delta}{=} X_0 \nabla \mathbb{F}^\sharp(X_0) = \{0\} \cup [\frac{1}{4}, 1]$
- $\mathbb{F}^\sharp(X_1) = \{0, \frac{1}{8}\} \cup [\frac{1}{4}, 1]$
- $X_2 \stackrel{\Delta}{=} X_1 \nabla \mathbb{F}^\sharp(X_1) = [0, 1]$

# Not widening at each step may cause non termination!

Let us define the following sequence:

$$\begin{cases} X_0 = \{0, \frac{1}{2}, 1\}, \\ X_{2n+1} = X_{2n} \cup \mathbb{F}^\#(X_{2n}), \\ X_{2n+2} = X_{2n+1} \nabla \mathbb{F}^\#(X_{2n+1}). \end{cases}$$

We get:

- $X_0 = \{0, \frac{1}{2}, 1\}$
- $X_1 \triangleq X_0 \cup \mathbb{F}^\#(X_0) = \{0, \frac{1}{4}, \frac{1}{2}, 1\}$
- $X_2 \triangleq X_1 \nabla \mathbb{F}^\#(X_1) = \{0, \frac{1}{8}\} \cup [\frac{1}{4}, 1]$
- ...
- $X_{2n} = \{0, \frac{1}{2^{2n+1}}\} \cup [\frac{1}{2^{2n}}, 1]$
- $X_{2n+1} = \{0, \frac{1}{2^{2n+2}}, \frac{1}{2^{2n+1}}\} \cup [\frac{1}{2^{2n}}, 1]$

# Intersecting with a stable constraint may cause non termination!

Let us define the following sequence:

$$\begin{cases} X_0 = \{0, \frac{1}{2}, 1\}, \\ Y_{n+1} = X_n \nabla \mathbb{F}^\#(X_n), \\ X_{n+1} = Y_{n+1} \cap \{\frac{1}{2^n} \mid n \in \mathbb{N}\}. \end{cases}$$

We get:

- $X_0 = \{0, \frac{1}{2}, 1\}$
- $Y_1 = \{0\} \cup [\frac{1}{4}, \frac{1}{2}, 1]$
- $X_1 = \{0, \frac{1}{4}, \frac{1}{2}, 1\}$
- ...
- $X_n = \{0\} \cup \{\frac{1}{2^k} \mid k \in [0, n+1]\}$
- $Y_{n+1} = \{0, \frac{1}{2^{n+2}}, \frac{1}{2^{n+1}}\} \cup [\frac{1}{2^n}, 1]$

# Alternative equivalent definition for the widening

A widening  $\nabla \in D^\# \times D^\# \rightarrow D^\#$  is an operator such that:

1.  $\gamma(a) \subseteq \gamma(a \nabla b)$ ;
2.  $\gamma(b) \subseteq \gamma(a \nabla b)$ ;
3. the relation  $a \rightarrow c$  defined as:

$$a \rightarrow c \stackrel{\Delta}{\iff} \exists b, a \nabla b = c$$

is well-founded.

# Stronger definitions

Replace the last condition with:

1. delayed widening: the following relation  $\rightarrow$ :

$$a \rightarrow d \stackrel{\Delta}{\iff} \exists c, d, a \sqsubseteq b \text{ and } b \nabla c = d$$

is well-founded;

2. intersection with a constant: For any  $\rho$ , the relation  $\rightarrow_\rho$ :

$$a \rightarrow_\rho c \cap \rho \stackrel{\Delta}{\iff} \exists b, a \nabla b = c$$

is well-founded;

3. both delayed and intersected: For any  $\rho$ , the relation  $\rightarrow_\rho$ :

$$a \rightarrow_\rho d \cap \rho \stackrel{\Delta}{\iff} \exists c, d, a \sqsubseteq b \text{ and } b \nabla c = d$$

is well-founded;

This ensures termination for the corresponding extrapolation strategies.

# Do such operators exist?

We focus on the last (strongest) version

1. Take:

(a)  $\mathcal{D}^\#$  a totally ordered set;

(b)  $\sqcup = \max$ ;

(c)  $\nabla_L = a \nabla b = a \sqcup \min\{l \in L \mid l \sqsubseteq b\}$  for  $L$  a finite set  
(i.e. widening with thresholds)

2. Our requirements are preserved by Cartesian composition.

Examples of solution:

- a set of intervals (seen as pairs of bounds);
- an octagon (without normalization);
- ...

# Back to the reduced product

In order to ensure termination when:

1. we (fairly and heterogeneously) replace some widening steps with union steps;
2. we do reduction after widening;

we require that:

- the main abstract domain is a finite product of total orders

$$D = \prod_{i \in I} D_i;$$

- operators ( $\sqcup$ ,  $\sqcap$ ,  $\nabla$ ) are computed component-wise;
- a partial order  $<$  relates basic domains (hierarchy of domains);
- reduction may only flow from lower basic domains to higher basic domains.

$$\rho(A_i)_{i \in I} = \left( A_i \cap \rho_i(A_j)_{\{j \mid D_j < D_i\}} \right)_{i \in I}.$$

Termination is proved recursively, for any domain lower than  $D_j$ .

# Overview

1. Introduction
2. Network of domains
3. Domain cooperation
4. Extrapolation strategies
5. Conclusion

# Conclusion

We have proposed a reduction scheme that is

- efficient enough to build scalable analyzers;
- flexible enough to refine:
  - preconditions,
  - post-conditions,
  - abstract transformers,
  - abstract elements after extrapolation steps;

This scheme is compatible with any refinement used so far (previously hard-coded);

- easy using:
  - a reduction between two models only requires modifying these two models,
  - a 1-to-n or a n-to-1 reduction only requires modifying these n+1 domains.

To achieve this goal, we have strengthened usual definition.