Thoughts on Trusting RAHD Computations

G.O.Passmore and P.B.Jackson and F.Kirchner
LFCS, Edinburgh and INRIA/IRISA
TEITP 2010

the dream

$$\mathbb{P} = \{p_1, \ldots, p_m\} \subset \mathbb{Q}[x_1, \ldots, x_n],$$

$$\varphi = \mathbb{B}(p_1, \ldots, p_m)$$ with $\mathbb{B}$ a boolean function in relations $\{<, \leq, =, \geq, >\}$.

does $\langle \mathbb{R}, \ldots \rangle \models \exists x_1 \ldots \exists x_n \text{ s.t. } \varphi$?

we want our decision procedures to:

- scale to problems of realistic size (esp. many variables a.k.a. high dimensions),
- be customisable for classes of problems with similar structure,
- produce a form of proof trace when needed.

good news, bad news

- good news: *RCF decidable!*
- bad news: *RCF infeasible!*
- good news: $\exists$ *RCF has a theoretical exponential speed-up over RCF!*
- bad news: *speed-up not evident in practice!*

more good news

- there are many different RCF (semi-)decision methods
- most known RCF (semi-)decision methods have *sweet spots*
  - such sweet spots can often be *combined* to decide sentences out of the reach of individual decision methods when used in isolation

our approach: RAHD

build a tool (RAHD) with goal of providing

- robust implementations of many different RCF decision methods,
- automatic book-keeping for orchestrating their combination,
- ship with a number of novel combinations of decision methods,
- an interactive mode for performing manual proof and developing proof strategies (should include, in addition to powerful decision methods, techniques one would use by hand)
- user-extensible via
  - ` verified rulesets' (used for forward-chaining),
  - user-defined proof strategies

  especially helpful for targeting classes of similar problems.
- ability to generate ` proof traces' if needed

some methods in RAHD

- exact interval constraint propagation which can be parameterised with sign-deciding strategies (e.g., multivariate factorisation, SOS decomposition, ...) – this acts as the ` glue' between many disparate RAHD procedures via ` state'
- quantifier elimination by Muchnik sequences,
- quantifier elimination by *extended* partial FD CAD,
- quantifier elimination by quadratic virtual term substitution (not prime time),
- real nullstellensatz search by Tiwari GB method extended with ICP,
- positivstellensatz search by a number of different methods,
- many different simplification and degree and dimensional reduction techniques,
- ...

case manipulation functions

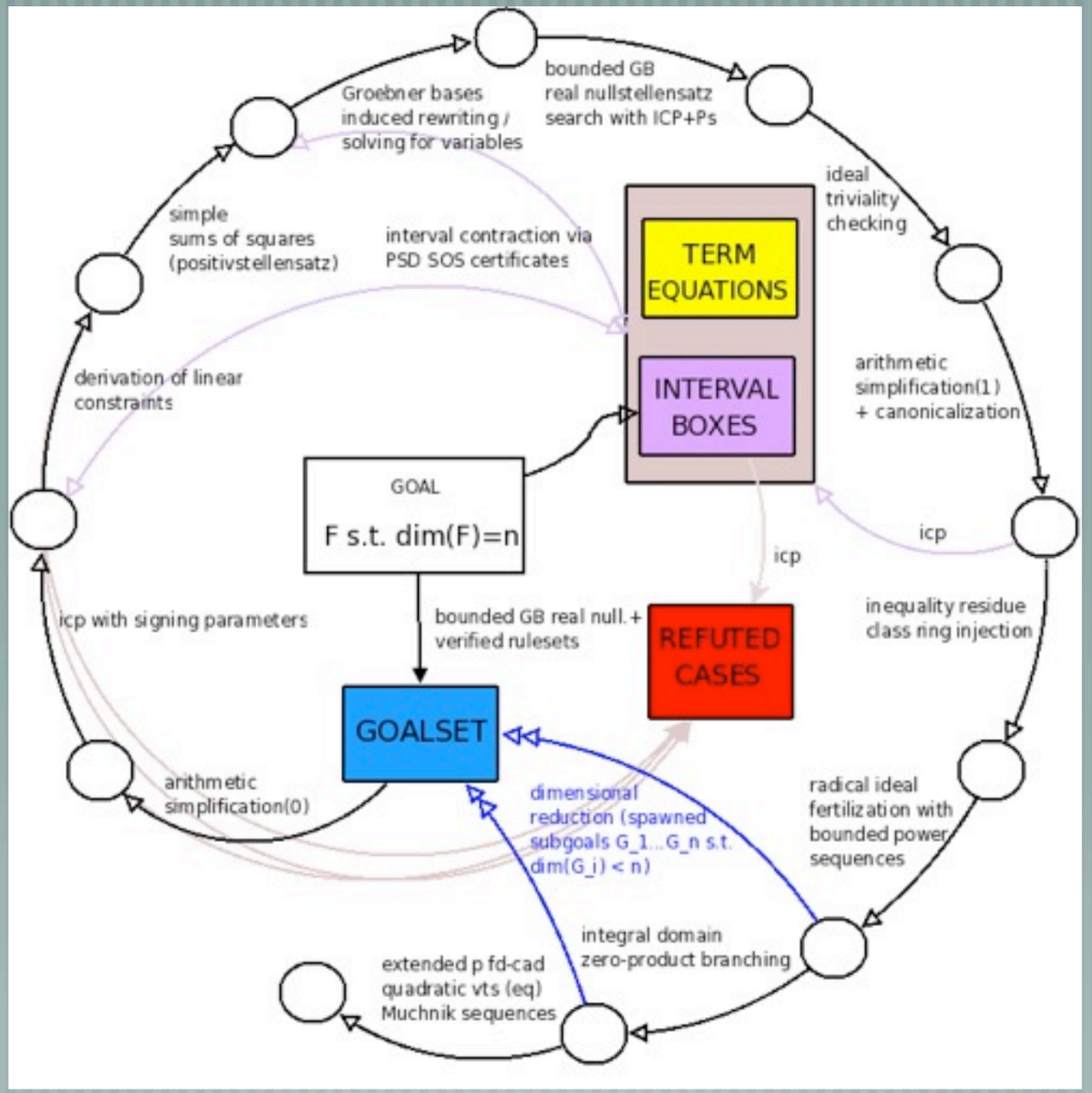
- atomic proof techniques in RAHD encapsulated in *cmf's*
- a cmf has the following shape:

$$\mathfrak{C} : \left(\bigwedge_{i=1}^{m} p_i \odot_i 0 \right) \times (\mathrm{Option}_1, \ldots, \mathrm{Option}_k) \rightarrow \left(\bigwedge_{j=1}^{w} \bigvee_{i=1}^{m'} (s_{i,j} \odot_{i,j} 0) \right)$$

- resulting formula must be equisatisfiable with original
- options may include RAHD proof strategies for subsidiary operations (will give example using extended FD CAD)
- cmf's implicitly take and may modify a ` proof context' or ` state' parameter, with an analogous equisatisfiability criteria

default strategy: waterfall

- RAHD ships with a default strategy: the *waterfall*
- main idea:
  - simple procedures before complex ones,
  - if decision not met, derive facts which could help later procedures (explicitly and via state),
  - work hard to derive simpler subproblems (esp. those in less variables than original),
  - if non-conjunctive subgoal derived, then waterfall calls itself recursively upon derived subgoals,
  - have complete procedure(s) sitting at bottom.



let's run RAHD on a few examples

all cmfs are not created equal i

- some RAHD cmfs give rise to easily checkable algebraic certificates which can be checked by a PA with minimal support for real algebra: real nullstellensatz, complex (weak) nullstellensatz, positivstellensatz, etc.
- these are very convenient! if only all were this way...

all cmfs are not created equal ii

- some RAHD cmfs, however, produce only proof traces which are at a much higher level: **epcad** proof traces, for instance, contain primitives such as:
  - real root isolation,
  - signed subresultant computation, and
  - liftable projection.

- mathematics underlying **epcad** is deep: while much progress has been made in Coq for instance (real root isolation, subresultants), verifying a liftable projection operator in a PA seems years away.

motivating trust questions i

- what are some good approaches to replaying RAHD proofs in fully-expansive proof assistants?
- we'll talk about work we've been doing with the Coq team (F. Kirchner) on this.
- this builds upon Shankar's idea(s) of the *Kernel of Truth* and the *Evidential Tool-bus*.

motivating trust questions ii

- imagine we knew proof assistant X could automatically replay RAHD proofs which only used cmfs C1, ..., Ck.
- would it be useful to be able to run RAHD in an *X-compatible* mode so that RAHD only searched for proofs which were currently automatically replayable in system X?
- should we develop *X-compatible* modes for each X?
- if so, what's the best way to go about this?

motivating trust questions iii

- returning to the difficult **epcad** cmf...
- imagine an interactive "proof review system" in which users could navigate RAHD proofs and "verify by cosimulation" claims such as "the signed subresultant prs for p, q is R" by automatically running the relevant computations across many different computer algebra systems.
- to what extent would this contribute to trust?
- to what extent could a structured combination of algorithmic cosimulation and deductive verification become a robust form of social review for RAHD proofs (if it could at all)?

an interesting combination

- one particularly interesting combined technique in RAHD is *extended partial full-dimensional CAD*
- this procedure is introduced and analysed in my ph.d. thesis
- main idea: extend partial CAD (Hong) with *d.m. parameters* to allow efficient ` short-circuiting' of CAD construction, all in the context of *full-dimensional* lifting (McCallum)
- d.m. parameters can affect both *projection* and *lifting*
- let's focus on its use for *cylindrical algebraic lifting*

cylindrical algebraic decomposition

a $\mathbb{P}-$ invariant cylindrical algebraic decomposition of $\mathbb{R}^n$

w.r.t. a set of polynomials

$$\mathbb{P} = \{p_1, \ldots, p_m\} \subset \mathbb{Q}[x_1, \ldots, x_n]$$

is a partitioning of $\mathbb{R}^n$ into finitely many connected components (each semi-algebraically described) s.t.

each polynomial p_i

is *sign-invariant* on each component.

* must also be *cylindrically arranged* but we're skipping a lot of technicalities...

cad yields easy SAT decision

- given such a CAD, deciding an existential sentence is conceptually very easy:
  just select a *sample point* in each connected component and *evaluate* the sentence upon it!
- ...sentence is SAT iff it is SAT at *some* such sample point!
- we see immediately one source of difficulty: *irrational points!*

cells: our connected components

what do our connected components look like?

they are *cells* (*cellularity* is defined by *induction on dimension*):

c is a cell in $\mathbb{R}$

iff $\begin{cases} c =]l, r[\text{ s.t. } l, r \in \mathbb{R}, \text{ or} \\ c \in \mathbb{R}. \end{cases}$

c is a cell in $\mathbb{R}^{k+1}$

iff $\begin{cases} c = \{\langle \vec{x}, y \rangle \mid \vec{x} \in \mathfrak{C} \ \wedge \ f(\vec{x}) < y < g(\vec{x})\}, \text{ or} \\ c = \{\langle \vec{x}, f(\vec{x}) \rangle \mid \vec{x} \in \mathfrak{C}\} \text{ where} \\ \quad \vec{x} \in \mathbb{R}^k, \ \mathfrak{C} \text{ is a cell in } \mathbb{R}^k, \\ \quad f, g \in \mathcal{C}(\mathfrak{C}, \mathbb{R}) \cup \{-\infty, +\infty\} \text{ (semialgebraic w.f.)} \\ \quad \text{s.t. } \forall \vec{r} \in \mathfrak{C}(f(\vec{r}) < g(\vec{r})). \end{cases}$

to be a cad: project... and lift!

- we will build a CAD also by induction on dimension:

$P \subset \mathbb{Q}[x_1, \ldots, x_n] \xrightarrow{\text{project}} \Pi_{x_n}(P) \subset \mathbb{Q}[x_1, \ldots, x_{n-1}]$

$\downarrow$ CAD$_n$ $\qquad\qquad$ $\downarrow$ CAD$_{n-1}$

CAD$(P) \subset (\mathbb{R}^n)^K \xleftarrow{\text{lift}(n-1 \to n)}$ CAD$(\Pi_{x_n}(P)) \subset (\mathbb{R}^{n-1})^K$

Moral: To construct a CAD for $\mathbb{R}^n$ w.r.t. P, we project P to a "proj. factor set" in one less variable, construct a CAD for it, then lift that CAD to a CAD for P.

our projection operator

$$\Pi_{x_n}(\mathbb{P}) = \mathcal{D} \cup \mathcal{S} \cup \mathcal{L},$$

where

$$\mathcal{D} = \{ sRes_{j=0}^{deg(R)-2}\left(R, \frac{\partial R}{\partial x_k}\right) \mid R \in Trunc(P) \wedge P \in \mathbb{P} \},$$

$$Trunc(P) = \begin{cases} \{P\}, \text{ if } deg_{x_n}(P) = 0, \\ \{P\} \cup Trunc(T(P)), \text{ oth.} \\ \text{where } T\left(\sum_{i=1}^{k} c_i x_n^i\right) = \sum_{i=1}^{k-1} c_i x_n^i. \end{cases}$$

$$Trunc(\mathbb{P} = \{P_1, \ldots P_m\}) = \bigcup Trunc(P_i).$$

$$\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2 \cup \mathcal{S}_3 \text{ where}$$

$$\mathcal{S}_1 = \{ sRes_1^*(R,S) \mid deg(R) > deg(S) \wedge R, S \in Trunc(\mathbb{T}) \},$$

$$\mathcal{S}_2 = \{ sRes_2^*(S,R) \mid deg(S) > deg(R) \wedge R, S \in Trunc(\mathbb{P}) \},$$

$$\mathcal{S}_3 = \{ sRes_3^*(S,\bar{R}) \mid deg(S) = deg(R) \wedge R, S \in Trunc(\mathbb{P}) \}$$

where $\bar{R} = lcoeff(S)R - lcoeff(R)S$,

$$sRes_1^* = sRes_{j=0}^{deg(S)-1}, \quad sRes_2^* = sRes_{j=0}^{deg(R)-1},$$

$$sRes_3^* = sRes_{j=0}^{deg(\bar{R})-1}, \quad deg = deg_{x_n},$$

$$\mathcal{L} = \{ lcoeff(R) \mid R \in Trunc(\mathbb{P}) \}.$$

all we need for Projection is
- $\frac{\partial P}{\partial x_n}$,
- Poly truncation and lcoeff's,
- Signed subresultant coefficients.

That's it !

lifting

lifting is conceptually very simple:

Given: $\langle c_1, \ldots, c_k \rangle$ a set of sample pts for a CAD of $\mathbb{R}^{k-1}$, and $\mathbb{P}^* \subset \mathbb{Q}[x_1, \ldots, x_k]$ a proj. factor set for $\mathbb{P}$ in dim. k.

We will construct a stack over each $\mathbb{R}^{k-1}$ cell sample pt c_i as follows:

(1) Let $\mathbb{P}^*(c_i)$ be the specialisation of $\mathbb{P}^*$ at $c_i \in \mathbb{R}^{k-1}$:
$\mathbb{P}^* = \{P_1^*, \ldots, P_m^*\} \subset \mathbb{Q}[x_1, \ldots, x_k] \Rightarrow \mathbb{P}^*(c_i) = \{P_1^*(c_i), \ldots, P_m^*(c_i)\} \subset \mathbb{Q}[x_k]$.

* Observe that $\mathbb{P}^*(c_i)$ is a univariate family!

(2) Isolate the real roots of $\mathbb{P}^*(c_i)$: this induces a CAD of $\mathbb{R}^1$.

(3) Select sample pts from the 1-dim CAD of $\mathbb{P}^*(c_i)$ and use them to build sample pts for k-cells:
$\{\langle c_i, s_1 \rangle, \ldots, \langle c_i, s_v \rangle\} \subset \mathbb{R}^k$.

So simple! But, ... what if sample pts are irrational?!

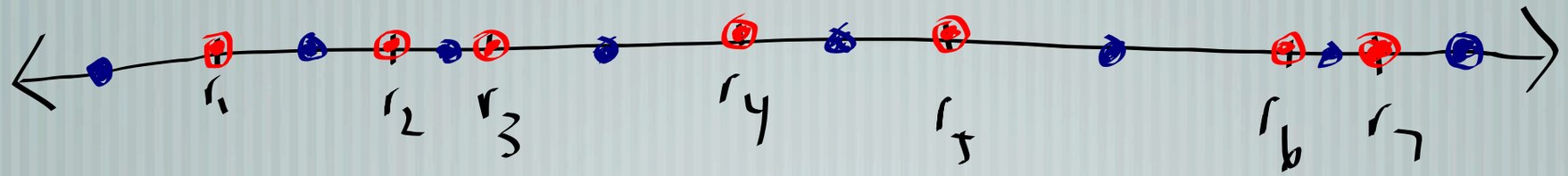
lifting illustration part i

Example: Let $P = \{P_1, \ldots, P_m\} \subset \mathbb{Q}[x_1, \ldots x_2]$.

$\widetilde{\Pi}_{x_2}(P) = \{P_1^*, \ldots, P_{m^*}^*\} \subset \mathbb{Q}[x_1]$.

We: (1) Construct a CAD for $\widetilde{\Pi}_{x_2}(P)$ since it is univariate. We do this by isolating the roots of $\left[\prod_{i=1}^{m^*} P_i^*\right]$. This gives a CAD of $\mathbb{R}^1$:

7 roots,
8 non-root sample pts
15 cells



Now, we must construct a stack over each cell:

*Notice: $0 \in \mathbb{Q}$ if we like (we do!).

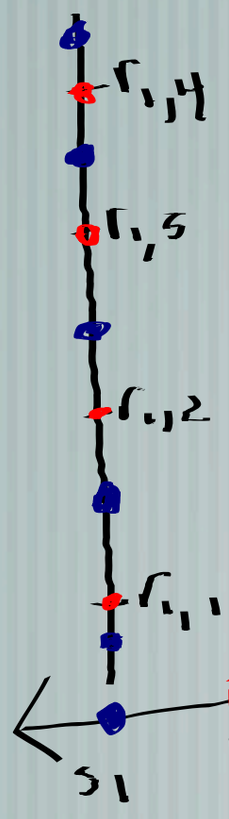
lifting illustration part ii

Example: Let $\mathbb{P} = \{P_1, \ldots, P_m\} \subset \mathbb{Q}[x_1, x_2]$.

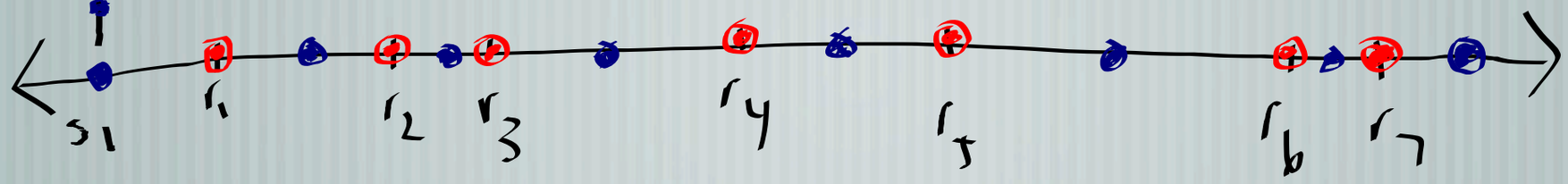
$\widetilde{\Pi}_{x_2}(\mathbb{P}) = \{P_1^*, \ldots, P_{m'}^*\} \subset \mathbb{Q}[x_1]$.

Now, we must construct a stack over each cell:

Let's begin with $S_1 = \frac{1}{2}$.



Now, we form a new univariate family by substituting the value of the sample pt S_1 in for x_1 in $\mathbb{P}$. Then, we isolate the real roots of $\mathbb{P}(S_1)$, inducing a CAD of $\mathbb{R}'$ which is then the stack of the cell whose sample pt is S_1. Repeat!



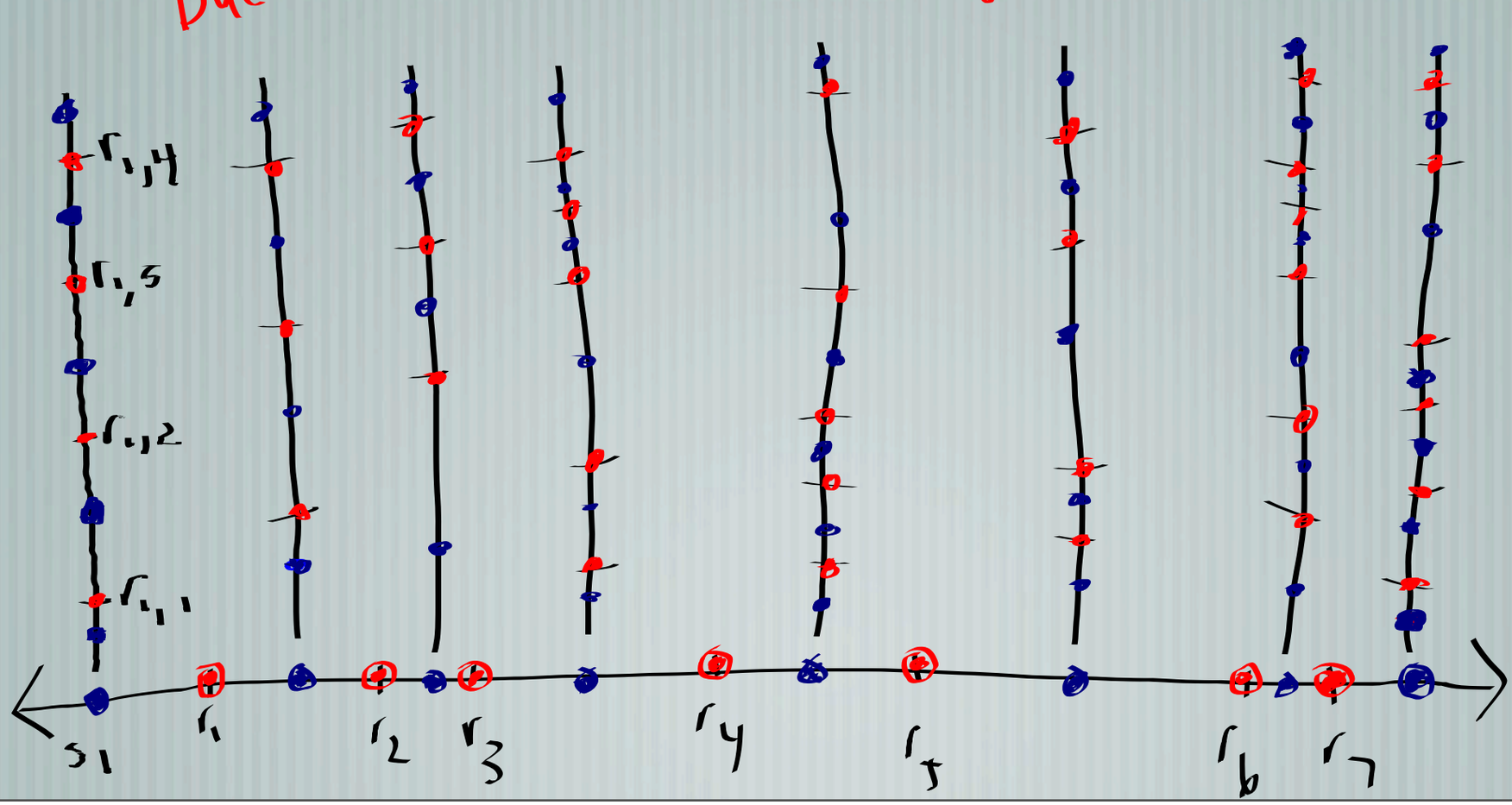
lifting illustration part iii

Example: Let $\mathbb{P} = \{p_1, \ldots, p_m\} \subset \mathbb{Q}[x_1, x_2]$.

$\widetilde{\pi}_{x_2}(\mathbb{P}) = \{p_1^*, \ldots, p_{m^*}^*\} \subset \mathbb{Q}[x_1]$.

Now, we must construct a stack over each cell.

But what about the roots? If they are irrational, how do we substitute them?



lifting over roots can be hard

- normal CAD requires we lift over *all* cells in *each* induced 1-dimensional CAD we use during stack construction
- but, this requires we lift over *roots*, which may be irrational algebraic numbers (they are their cell's only sample point)
- doing this is expensive: requires algebraic number computations
- it turns out these algebraic number computations are often the bottle-neck of CAD computation!

full-dimensional CAD

- full-dimensional CAD: only lift over full-dimensional cells!
- theorem: a cell over $\mathbb{R}^{n+1}$ is f.d. iff it was lifted from a f.d. cell over $\mathbb{R}^n$ (McCallum)
- but, the set of satisfying real vectors for polynomial *strict* inequalities in $\mathbb{R}^n$ is always *open and homeomorphic to* $\mathbb{R}^n$ … that is, *f.d.!*
- so, to decide SAT of strict inequalities, we can just use f.d. CAD! no lifting over irrational algebraic numbers needed.

RAHD and f.d. CAD

- we investigated a combination of f.d. CAD (via QEPCAD-B) and Groebner bases (see Calculemus'09 paper)
- now, we have written our own proof-trace producing version, and have extend Hong's notion of *partial cad* so as to take RAHD strategies as parameters
- these parameters can be used to *short-circuit* lifting
- let's sketch the basic idea...

extended partial f.d.cad

cad tree grows rapidly, so at each stack construction, let's ask a cheap RAHD strategy if the stack construction can be avoided!

ouch!

$\varphi = \bigwedge (P_i \Theta_i 0)$, $\mathbb{P} = \{P_1, \ldots, P_k\} \subset \mathbb{Q}[\vec{x}]$.

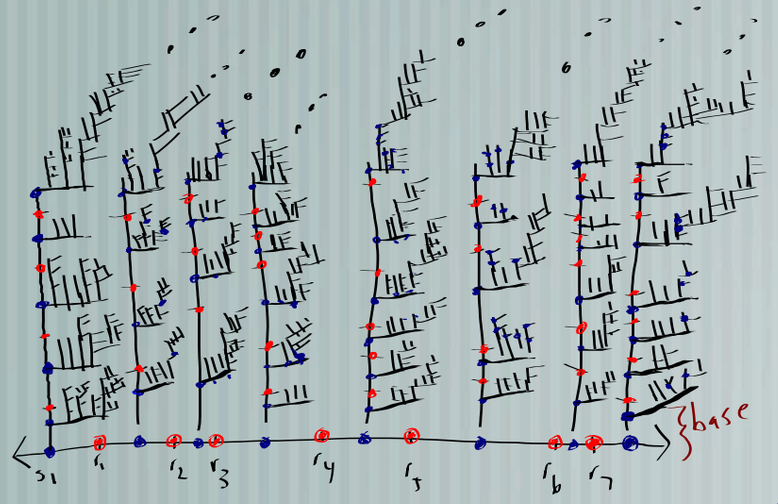
At dimension k, ask RAHD-STRAT(k):

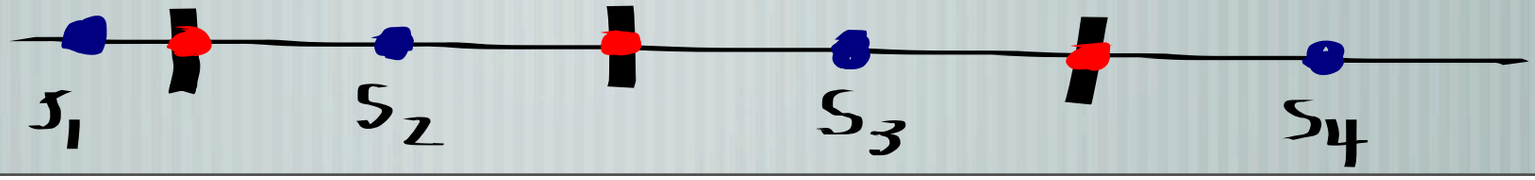
Is $\varphi \wedge (x = s_1)$ SAT?

$\varphi \wedge (x = s_2)$ SAT?

$\varphi \wedge (x = s_3)$ SAT?

$\varphi \wedge (x = s_4)$ SAT?





trusting RAHD proofs

- skeptical approach to RAHD+PA integration: PA delegates proof search to RAHD, then PA performs checking upon the answer, using a *proof trace* constructed by RAHD

- RAHD tries to generate *proof traces* consisting of "proof milestones" – only enough information required for the PA to reconstruct the proof, not more: Note, this is PA specific

- verbosity required for proof milestones may change over time, as PA develops more robust library of tools for real algebra

RAHD+ECDB: Coq integration

With Florent Kirchner, we've performed a preliminary integration of RAHD with Coq.

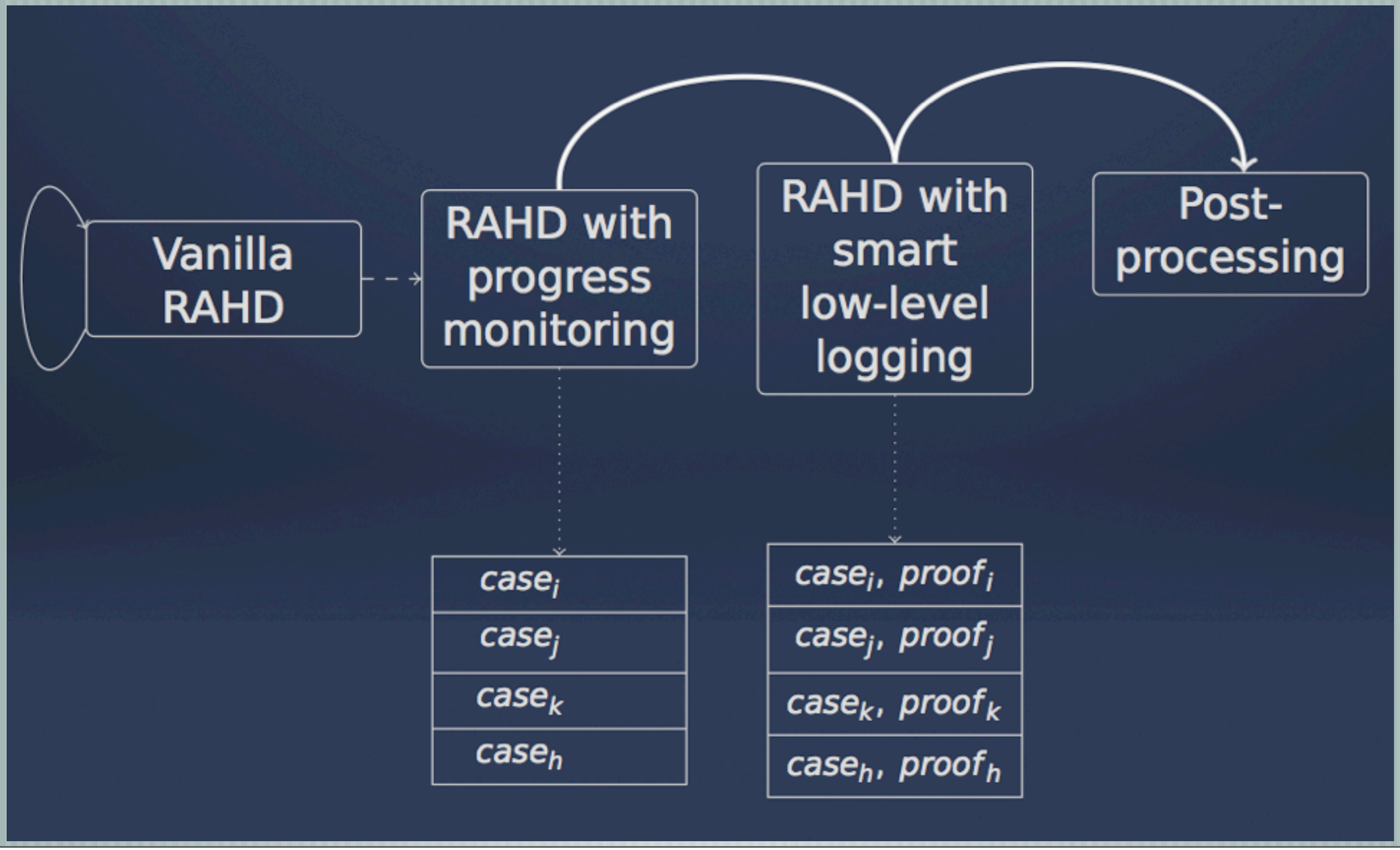
RAHD+ECDB Cont'd

Example ECDB table

case-id	goal-key	case	cmf	cert	status
0	0	$A \wedge B \wedge \cdots$	simp-real-null	psatz	:UNKNOWN
1	0	$C \wedge D \wedge \cdots$	rcr-svars	nil	(:UNSAT :BY-SG)
0	0.1	$D' \wedge \cdots$	simp-tvs	auto	:UNSAT
2	0	$E \wedge F \wedge \cdots$	quick-sat	model	:SAT

persistent certificate store,
state-preserving shutdown and restart,
database queries using simple equational constraints,
in-place certificate post-processing,
on-the-fly translation of cases into PA syntax.

Staggered proof refinement



Proof command language

```
1     Init(poly1, P1)
2     Init(poly2, P2)
3     Init(poly3, P3)
4     Label(Ideal(P1,P2,P3),I)
5     Label(SPol(P1,P2),P4)
6     Label(SPol(P2,P3),P5)
7     ...
8     Label(SPol(Pj,Pk),Pn)
9     Infer(UNSAT-BY-NULLSTELLENSATZ,I,Pn)
```

best way forward?

returning to the three discussion questions:

1. what are best approaches for skeptical RAHD+PA integration?

2. should we develop *X-compatible* modes for different PA's X?

3. what about cmfs such as **epcad** which produce traces which are very difficult to verify? how can we in the mean time increase our trust in RAHD's implementations of such procedures? does *verification by cosimulation* by CAS's, for instance, enhance our trust in RAHD's results? Shankar thinks no for some interesting reasons...

moral of the story

- there is no single RCF decision method which is suitable for all uses
- there are many approaches available: let's take advantage of this! what a wonderful problem to have!
- we dream of providing a playground for combining robust, original, proof trace producing implementations of all RCF methods which might ever be useful, together with powerful default proof strategies effective for large classes of practical problems
- we want to make it easy for users to tailor their own combinations, so as to easily develop strategies for solving classes of problems they encounter in practice
- many questions remain as to how best to improve trust in RAHD's results: some techniques are easy to check, some are not. how do we best proceed?
- thank you!

references

- *Combined Decision Techniques for the Existential Theory of the Reals* - Passmore, Jackson - Calculemus'2009
- *Thinking Outside the Arithmetic Box: Certifying RAHD Computations* - Kirchner, Passmore - LfSA'2010
- *On Locally Minimal Nullstellensatz Proofs* - De Moura, Passmore - SMT'2009
- *Superfluous S-Polynomials in Strategy-Independent Groebner Bases* - Passmore, De Moura - SYNASC'09
- *Groebner Basis Construction Algorithms Based on Theorem Proving Saturation Loops* - Passmore, De Moura, Jackson - Dagstuhl Proceedings 2010