

*A formal security  
policy for Xenon*

*John McDermott and Leo Freitas  
Naval Research Laboratory and University of York  
October 2008 @ FMSE*

## Hoare's Verification Grand Challenge

**A mature scientific discipline should set its own agenda and pursue ideals of purity, generality, and accuracy far beyond current needs**

- **science** *explains why things work in full generality by means of calculation and experiment*
- **an engineering discipline** *exploits scientific principles in the study of the specification, design, construction, and production of working artifacts, and improvements to both process and design*
- **the verification challenge** *is to achieve a significant body of verified programs that have precise external specifications, complete internal specifications, machine-checked proofs of correctness with respect to a sound theory of programming*

## Deliverables

1. **a comprehensive theory of programming**
  - *covering the features needed to build practical and reliable programs*
2. **a coherent toolset**
  - *automating the theory and scaling up to large codes*
3. **a collection of verified programs**
  - *replacing existing unverified ones*
  - *continuing to evolve as verified code*
  - **a repository**

**You cant say any more it cant be done!  
Here, we've done it!**

## Circus tools architecture overview

- *Circus AST, XML interchange format (CZT project)*
- *parsers, pretty printing and IDE integration: jEdit, Eclipse*
- *typechecking (**under development**) (MSc thesis)*
- *complete operational semantics specification (PhD thesis)*
- *compiler (v2): new operational semantics (FACJ article)*
- *plugable theorem proving architecture (PhD thesis)*
  - *Z/Eves—Java integration*
  - *various Z toolkit laws and MC theories*
  - *studies for PVS and SAT solvers integration*
  - *trivial symbolic evaluator: arithmetic and propositional calculus*

## Circus

- $Circus = Z + CSP + ZRC$
- associated refinement theory and calculus based on UTP
- correctness by construction
- *Circus* programs
  - sequence of paragraphs:  $Z + channels + processes$
  - channels
    - \* strongly-typed
    - \* allows generic actuals
  - processes
    - \* encapsulated state:  $Z$
    - \* behaviour:  $Z + CSP + \dots$

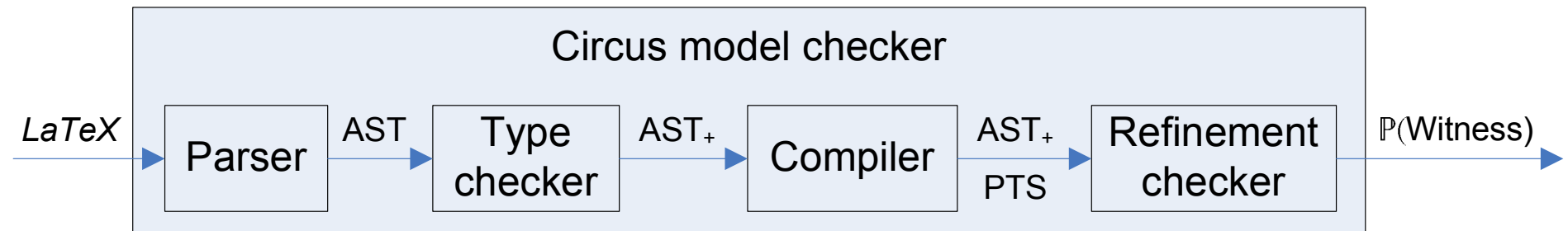
## Circus by example

### Accumulator process

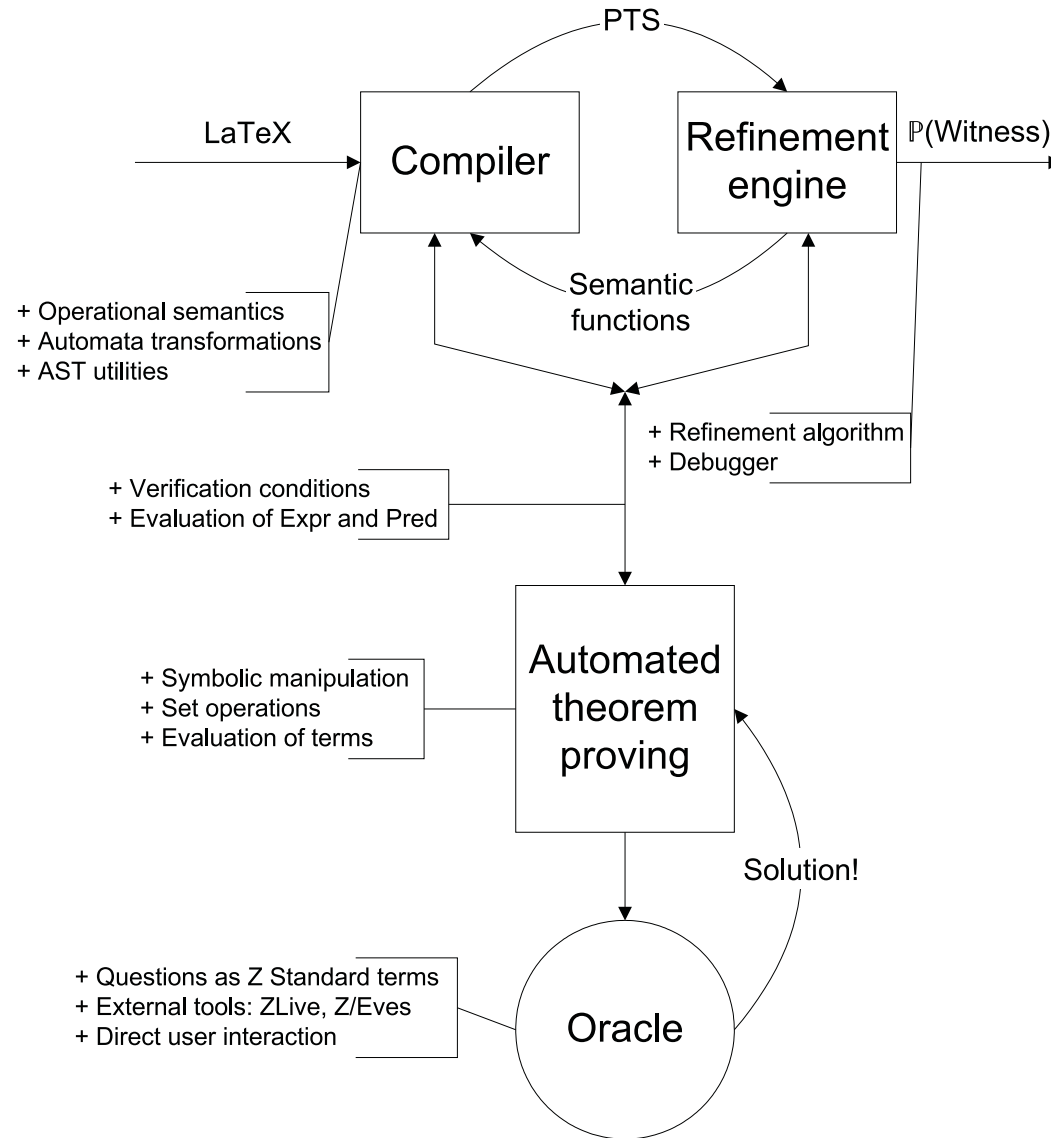
```
channel out :  $\mathbb{N}$   
process Accumulator1  $\hat{=}$   
begin  
  state  
  State  $\hat{=}$  [v :  $\mathbb{N}$ ]  
  AddOp  $\hat{=}$  [ $\Delta$  State | v' = v + 1]  
  • ( $\mu$  X • out!v  $\rightarrow$  AddOp; X)  
end
```

Notice the nondeterminism on the initial value of *v* in *out!**v*

## Circus Model checker top-level design



- $\text{\LaTeX}$  (and other) source(s) input  $\longrightarrow$  witnesses
- symbolic (on-the-fly) refinement model checking
- integrated theorem proving with extensible interfaces
  - stacked pipelining with external tools
  - application oriented theories as (source) modules
  - elegant OO-design: modular, flexible, configurable, etc.



## Circus tools summary

### Textual UI

- *SPM: parsing, pretty-printing, typechecking*
- *compiler for process behaviour (ProBE-like) exploration*
- *model checker for refinement (FDR-like) checking*
- *TPM: Expr, Pred transformation (and VC elimination)*

### Graphical UI

- *initial integration with jEdit and Eclipse via CZT*
- *stand-alone GUI in Java or Web-based (envisaged)*
- *networked version of TUI through sockets*

## Representing *Circus*: motivations and rationale

### *Why $\LaTeX$ and XML?*

- *CZT support for standard Z- $\LaTeX$ , as well as other Z tools*
- *interoperability with existing tools that use  $\LaTeX$*
- *seemingly description between reports/papers and actual code*

### *Why not like $CSP_M$ ?*

- *Z standard is  $\LaTeX$  oriented*
- *extending available  $CSP_M$  parser for Z is difficult*
- *extending available CZT Z tools for  $CSP/Circus$  is easier*

## Representing Circus

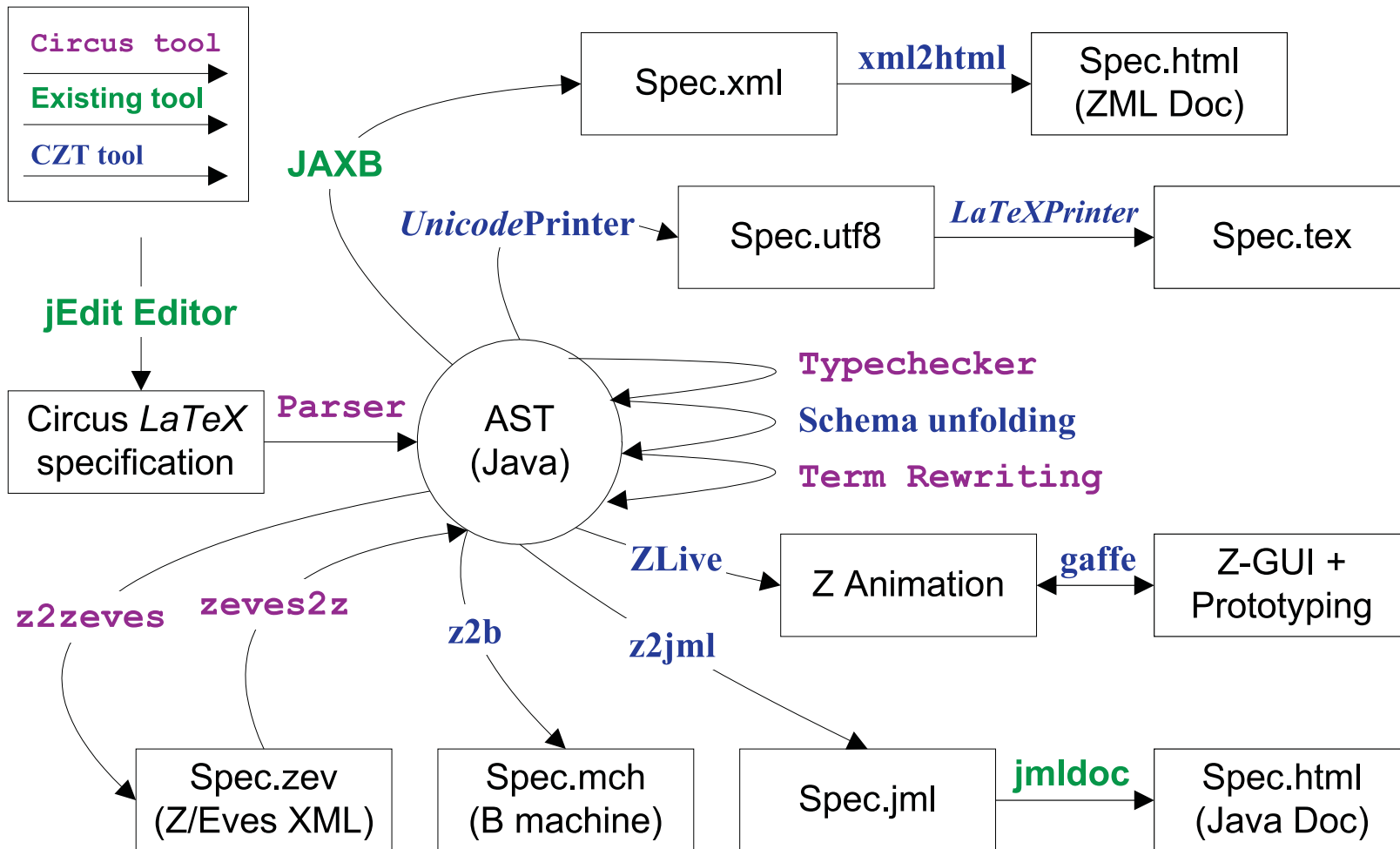
### *Circus XML format*

- *standard format enabling interoperability between Circus tools*
- *allows different programming languages to handle Circus*
- *useful in case the language is extended*

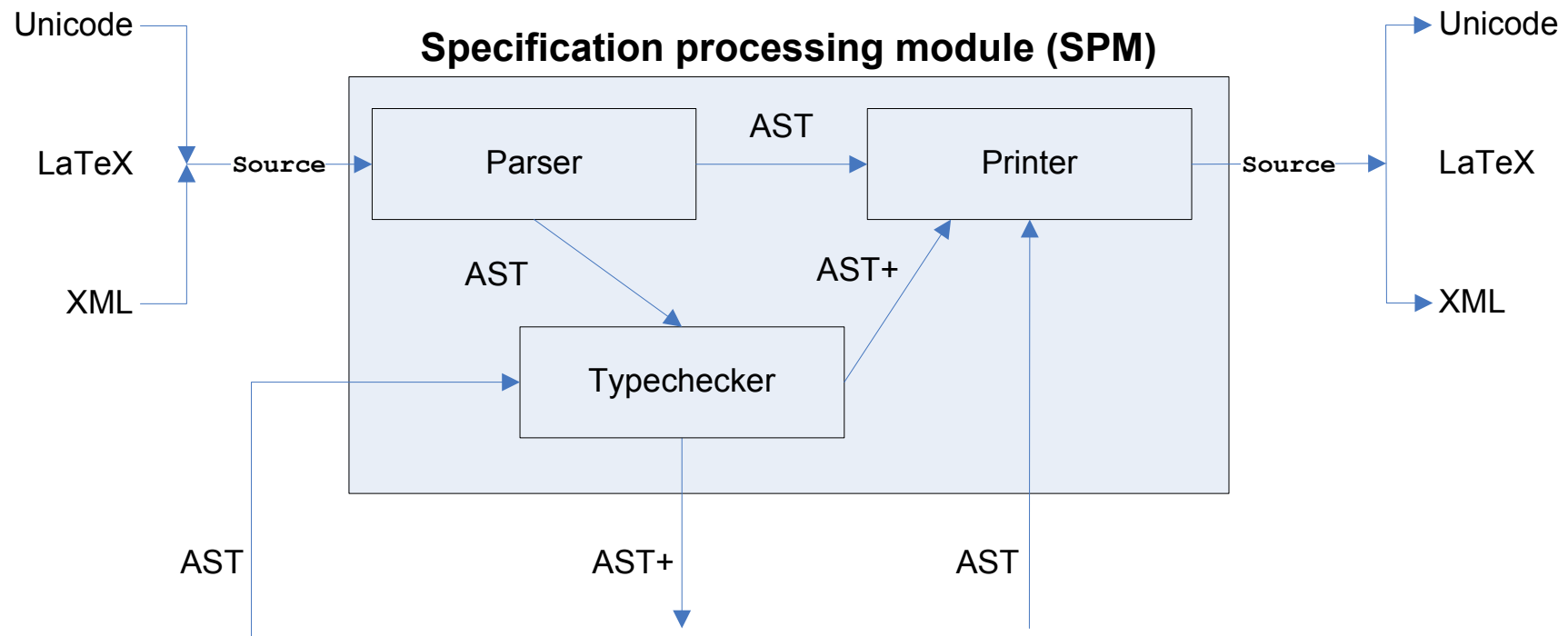
### *General Circus AST*

- *enables transformation between (3) different formats.*
- *Java (AST) code generation from XML (metadata) schemas*
- *some Circus tools based on this AST:*
  1. *typechecker*
  2. *model checker*
  3. *refinement calculator*
  4. *Circus  $\longrightarrow$  Java*

## Basic tools around AST for *Circus*

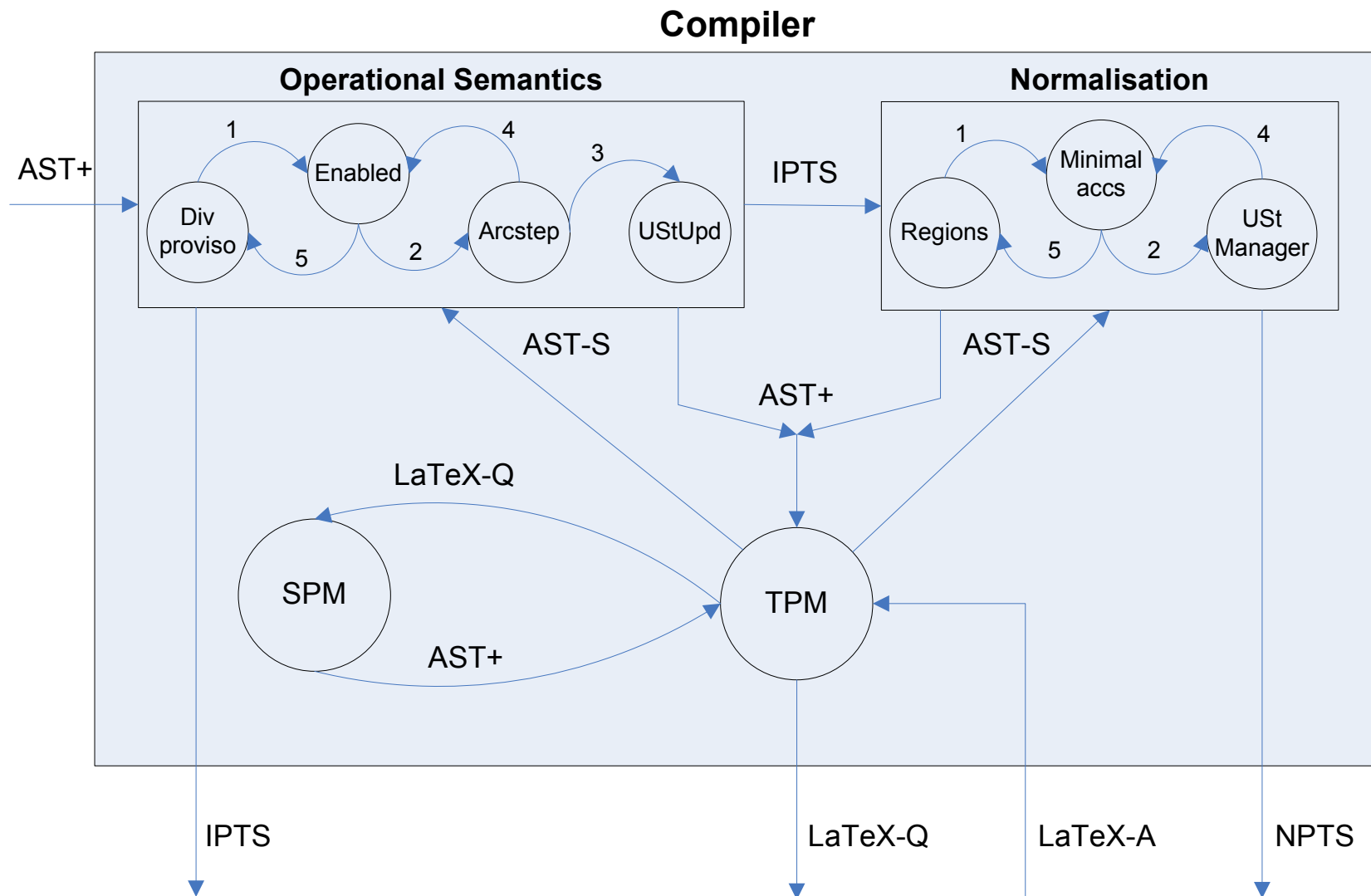


## Circus tools front end



processing from/to various sources:  $\text{\LaTeX}$ , UNICODE, XML

# Compiler architecture overview



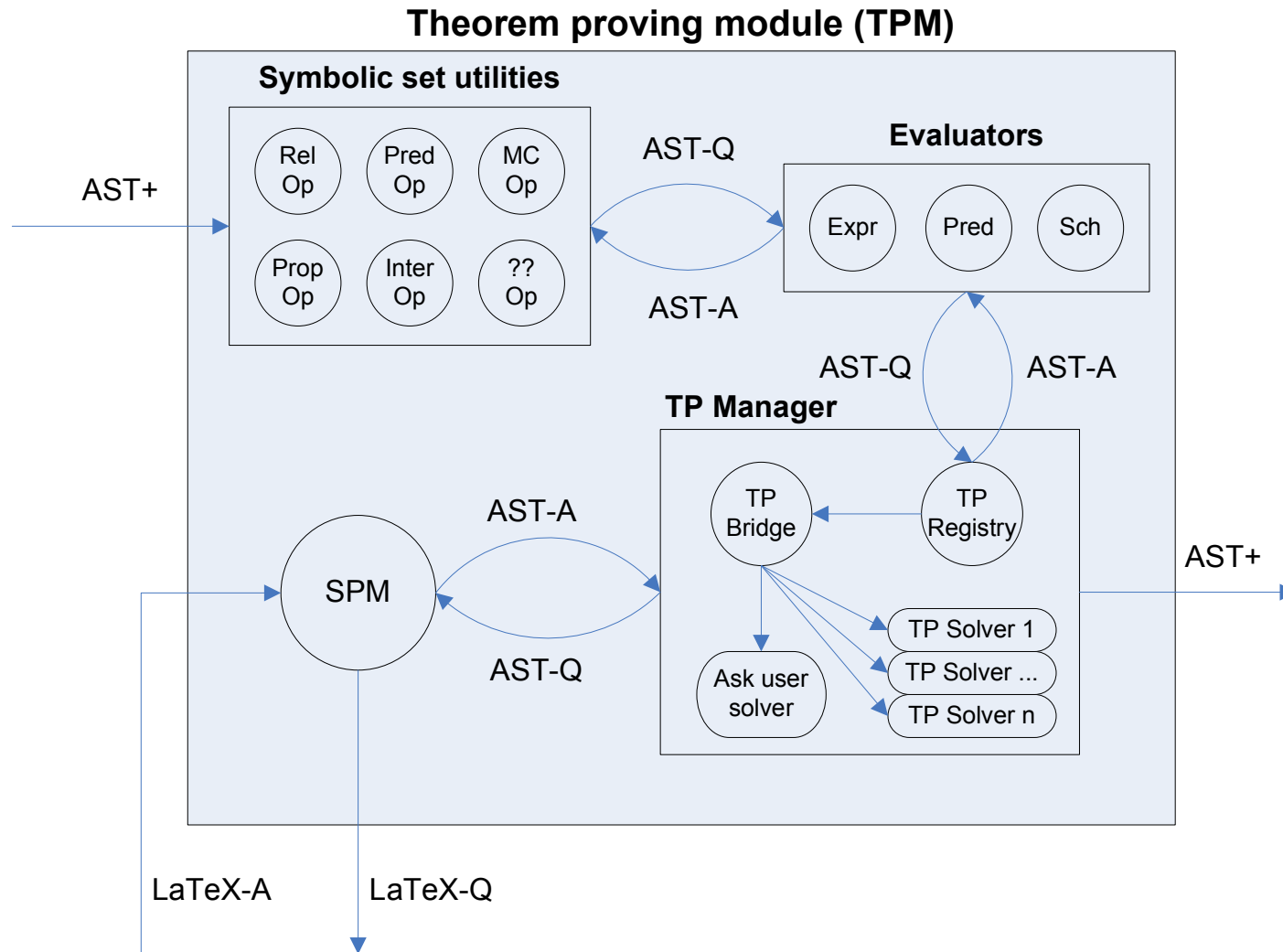
## Plugable TP architecture

- *symbolic set utilities: various operations over Expr & Pred*
- *symbolic evaluators: predicate, expression, and sch-text transformers*
- *theorem proving stacked solvers: external tools integration*
- *integration with SPM for interactive (AskUser) mode*

## *Trivial symbolic evaluator (i.e., automatic TP)*

- *simple arithmetic:  $+$ ,  $-$ ,  $*$ ,  $\text{div}$ ,  $\text{mod}$ ,  $>$ ,  $\geq$ ,  $<$ ,  $\leq$ , etc.*
- *simple predicates:  $\wedge$ ,  $\vee$ ,  $\Rightarrow$ ,  $\Leftrightarrow$ ,  $\neg$ ,  $x$ ,  $c$ ,  $\text{num}$ , etc.*
- *finite predicate calculus:  $\forall$ ,  $\exists$ , etc.*
- *most Z toolkit laws for sets and relations*
- *MC (regions theory) laws for normalisation*

# Theorem proving architecture overview



## Plugable TP architecture (cont.)

### *TP stacked solvers: pre*

- *integration with at least one available format*
- *Java interface to call external tools APIs*
- *recognition of set theoretical Expr and Pred*

### *TP stacked solvers: post*

- *become part of the VC discharging chain*
- *undischarged VC's are passed along the chain*
- *default evaluation of tagged VC's for batch execution*
- *rationale of use among solvers rather than simply stacked*

## Integrated stacked solvers

### *internal tools*

- trivial symbolic evaluator (*done*)
- CZT Z schema unfold/normaliser (*partially*)
- CZT theory-parameterised rewriting engine (*partially*)

### *external tools*

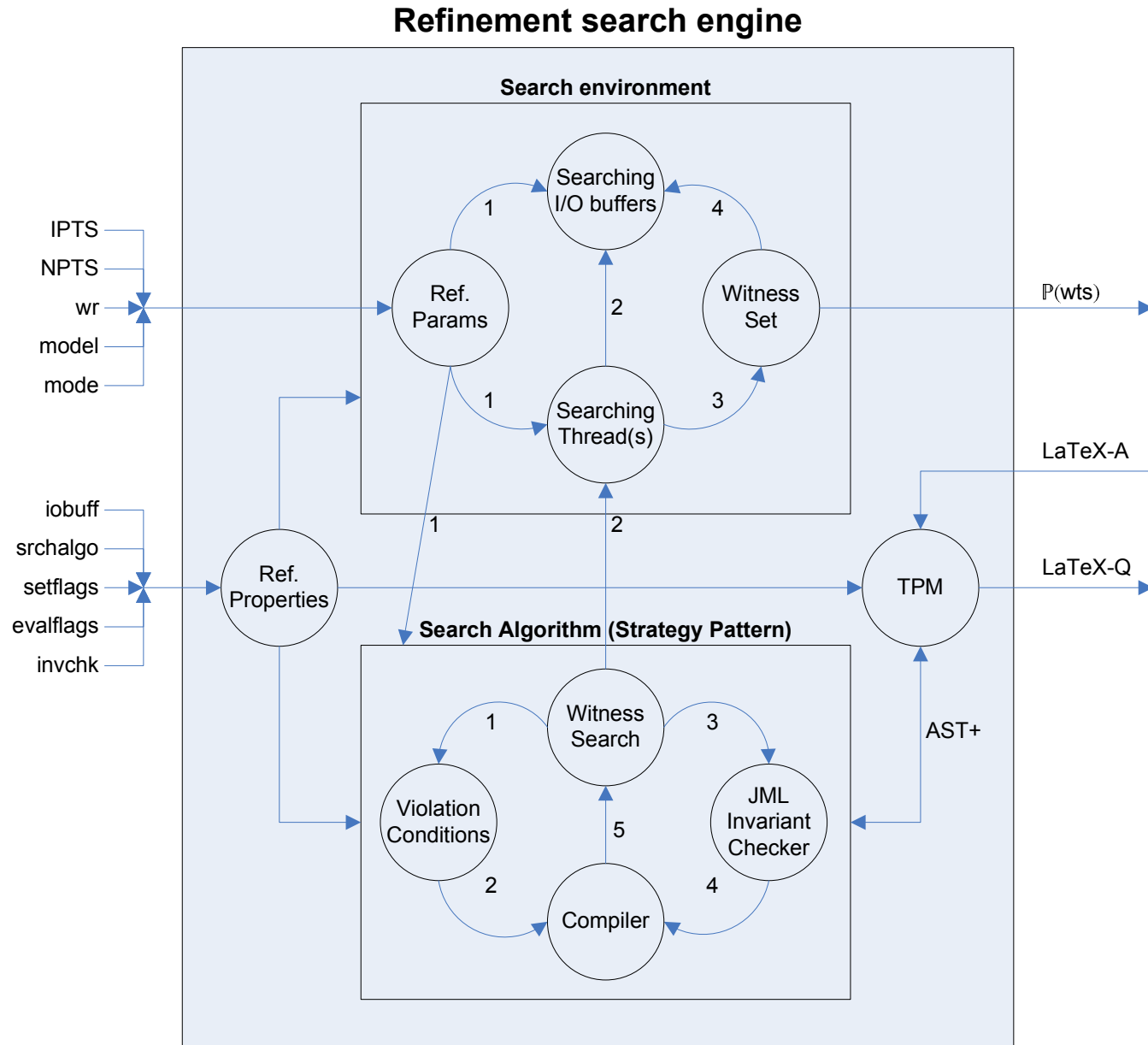
- CZT **ZLive**: Z animator (*done*)
- Z theorem provers: **Z/Eves** (*done*), **ProofPower-Z** (*TODO*)
- preliminary feasibility study: **PVS** (*done*), **SAT** solvers (*TODO*)

## Refinement engine

- *“skilful” driver of the compiler*
- *VC generation for MC algorithm (i.e., regions)*
- *searching algorithm code derived through refinement calculus*

## *Interacting with the model checker*

- *composition of SPM, Compiler, and TPM*
- *various stacked solvers to discharge VCs*
- *external tools plumbed through XML or Java interfaces*



## Compiler demo: GCD Euclidian algorithm examples

### *Versions from wikipedia GCD*

- *GCDA* — abstract recursive version (*AbsGCD*)
- *GCDB* — assignment base loop implementation (*ImplGCD*)
- *GCDC* — iterative Euclidian algorithm (*EuclidianGCD*)
- *GCDD* — efficient version of Euclidian algorithm (*EfficientGCD*)

### *Let's see them in Circus*

- *while loops as tail recursive actions*
- *i.e., (while b do P) encoded as:*

$(\mu X \bullet (\mathbf{if} \ b \rightarrow (P ; X) \parallel \neg b \rightarrow \mathbf{Skip} \ \mathbf{fi}))$

## Recursive GCD Euclidian algorithm in *Circus*

**channel**  $in, out : \mathbb{N}$

**process**  $AbsGCD \hat{=}$

**begin**

$GCDA \hat{=} in?p : (p > 0) \rightarrow in?q : (p > q) \rightarrow GCDA1(p, q)$

$GCDA1 \hat{=} (a, b : \mathbb{N}) \bullet$

**(if**  $(b = 0) \rightarrow out!a \rightarrow Skip$

$\parallel (\neg b = 0) \rightarrow GCDA1(b, a \bmod b)$

**fi)**

  •  $GCDA$

**end**

## Imperative GCD Euclidian algorithm in *Circus*

```

process ImplGCD  $\hat{=}$ 
begin
  GCDB  $\hat{=}$  in?p : (p > 0)  $\rightarrow$  in?q : (p > q)  $\rightarrow$  GCDB1(p, q)
  GCDB1  $\hat{=}$  (a, b :  $\mathbb{N}$ ) •
    ( $\mu$  X •
      (if (b = 0)  $\rightarrow$  Skip
        | ( $\neg$  b = 0)  $\rightarrow$ 
          (var t :  $\mathbb{N}$  • t := b ; b := a mod b ; a := t) ; X
        fi)
      ) ; out!a  $\rightarrow$  Skip
    • GCDB
end

```

## Iterative GCD Euclidian algorithm in *Circus*

```

process EuclidGCD  $\hat{=}$ 
begin
  GCDC  $\hat{=}$  in?p : (p > 0) → in?q : (p > q) → GCDC1(p, q)
  GCDC1  $\hat{=}$   $(a, b : \mathbb{N}) \bullet$ 
     $(\mu X \bullet$ 
      (if  $(a = b) \rightarrow \textit{Skip}$ 
         $\parallel (\neg a = b) \rightarrow$ 
          (if  $(a > b) \rightarrow a := a - b$ 
             $\parallel (a < b) \rightarrow b := b - a$ 
          fi) ; X
        fi)
      ) ; out!a → Skip
     $\bullet$  GCDC
end

```

## Efficient/factored GCD Euclidian algorithm in *Circus*

```

process EfficientGCD  $\hat{=}$ 
begin
  GCDD  $\hat{=}$  in?p : (p > 0) → in?q : (p > q) → GCDD1(p, q)
  GCDD1  $\hat{=}$   $(a, b : \mathbb{N}) \bullet (\mathbf{var} \ x, y, x0, y0, t, q : \mathbb{N} \bullet$ 
     $x, y, x0, y0 := 0, 1, 1, 0 ;$ 
     $(\mu \ X \bullet$ 
       $(\mathbf{if} \ (b = 0) \ \rightarrow \mathit{Skip}$ 
         $\parallel (\neg b = 0) \rightarrow$ 
           $t := b ;$ 
           $q := a \mathit{div} \ b ;$ 
           $b := a \mathit{mod} \ b ;$ 
           $a := t ; t := x ;$ 
           $x := x0 - (q * x) ;$ 
           $x0 := t ; t := y ;$ 
           $y := y0 - (q * y) ;$ 
           $y0 := t ; X$ 
         $\mathbf{fi}) ; \mathit{out!}a \rightarrow \mathit{Skip}$ 
       $\bullet \ \mathit{GCDD}$ 
     $\bullet \ \mathit{GCDD}$ 
  end

```