Theorem Proving with Vampire for Rigorous Systems Engineering

Reasoning in Theories

Giles Reger and Martin Suda

University of Manchester and TU Wein

What is a Theory?

Reminder: first-order signature and interpretations/models

A theory (in our presentation) is

- A signature of interpreted symbols
- A class of interpretations that fix some interpretation for that signature

For example

- a sort Int,functions $+, *, -$, predicate $<$ and constants $1, 2, 3, \ldots$
- The class of interpretations interpreting Int as $\mathbb{Z}$ and symbols in the single intended way

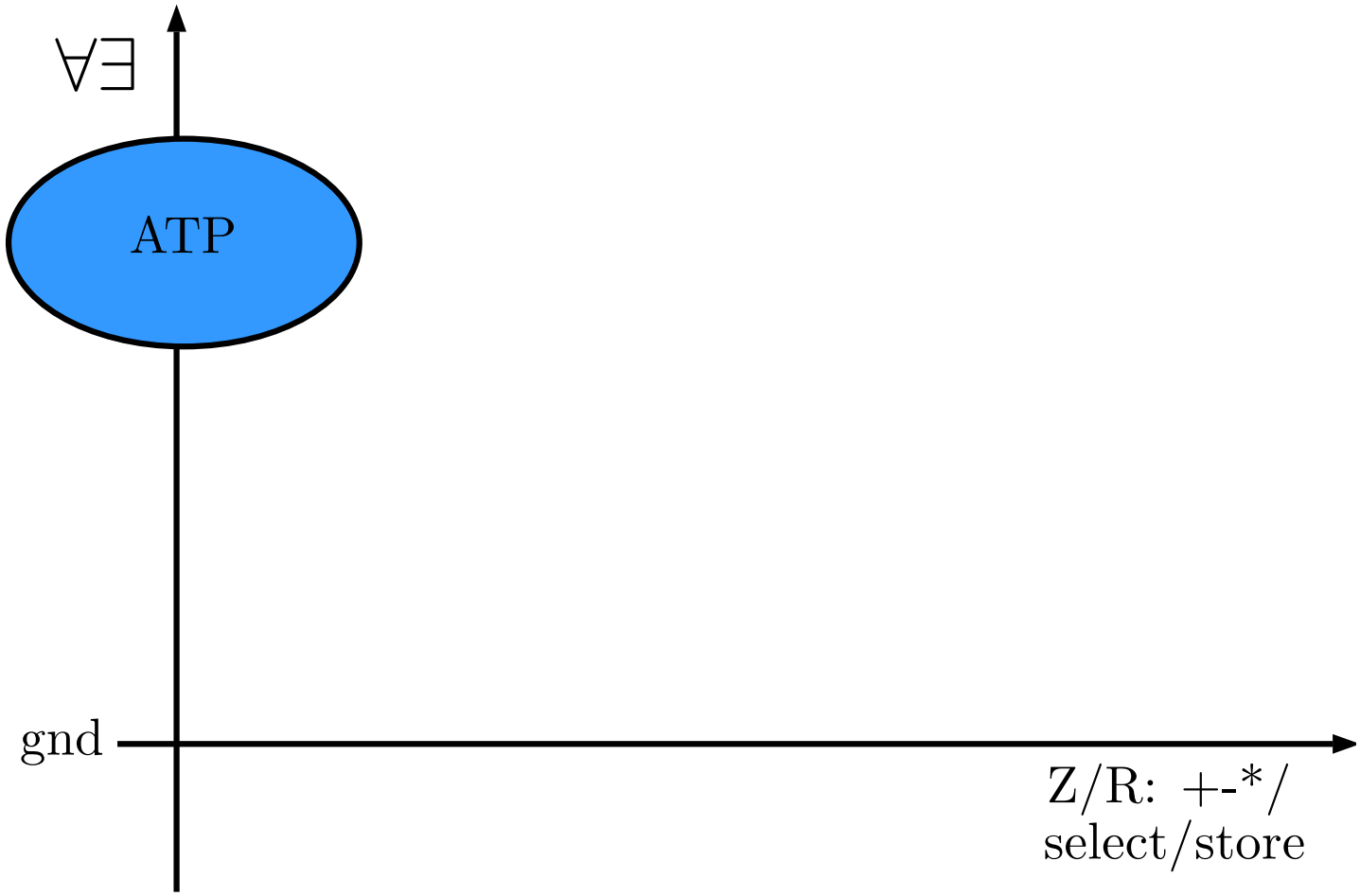
The class interpretation is particularly useful where we have under-defined functions e.g. division

Why is it hard?

Reasoning with quantifiers and theories

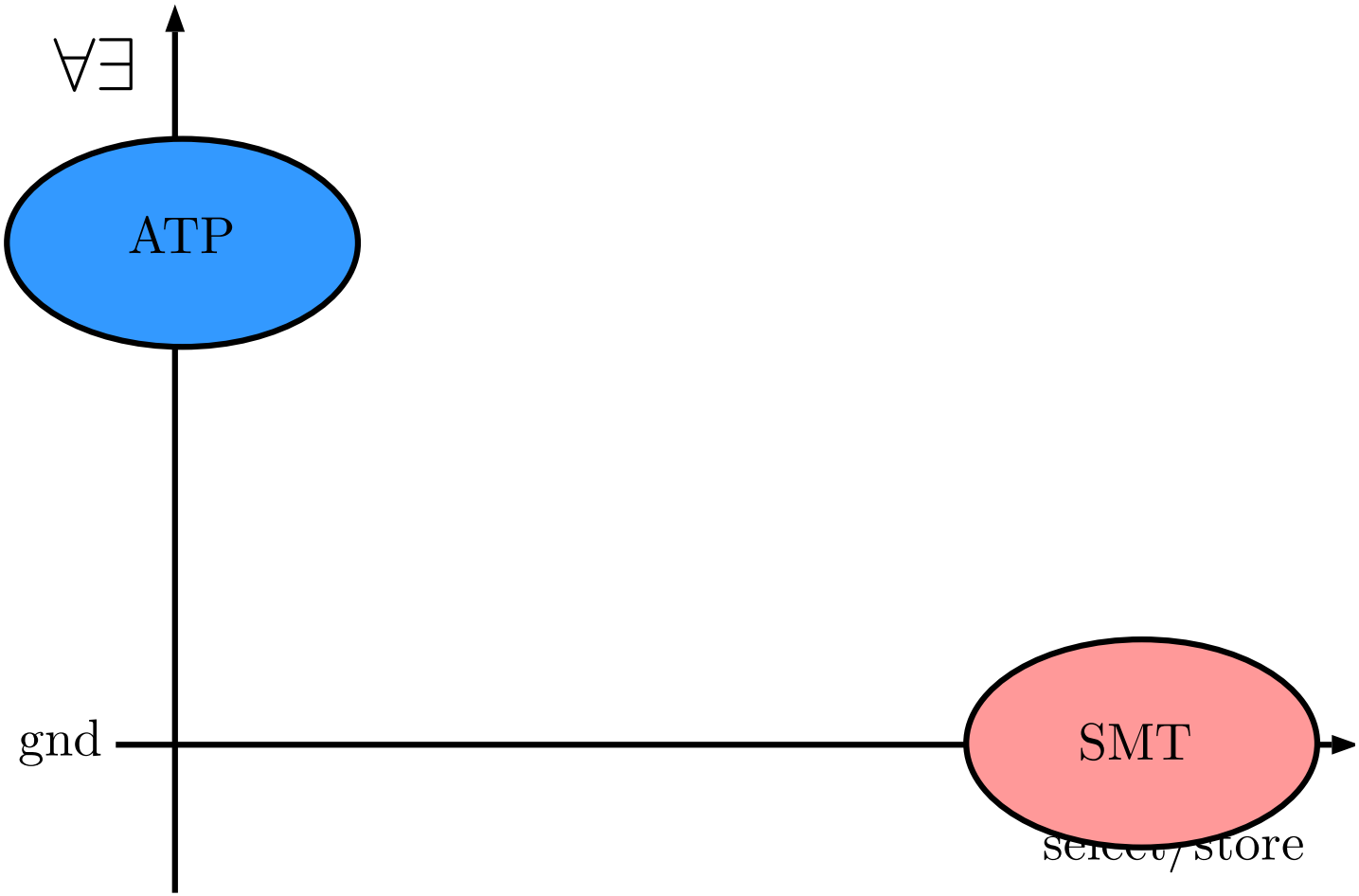
Why is it hard?

Reasoning with quantifiers and theories



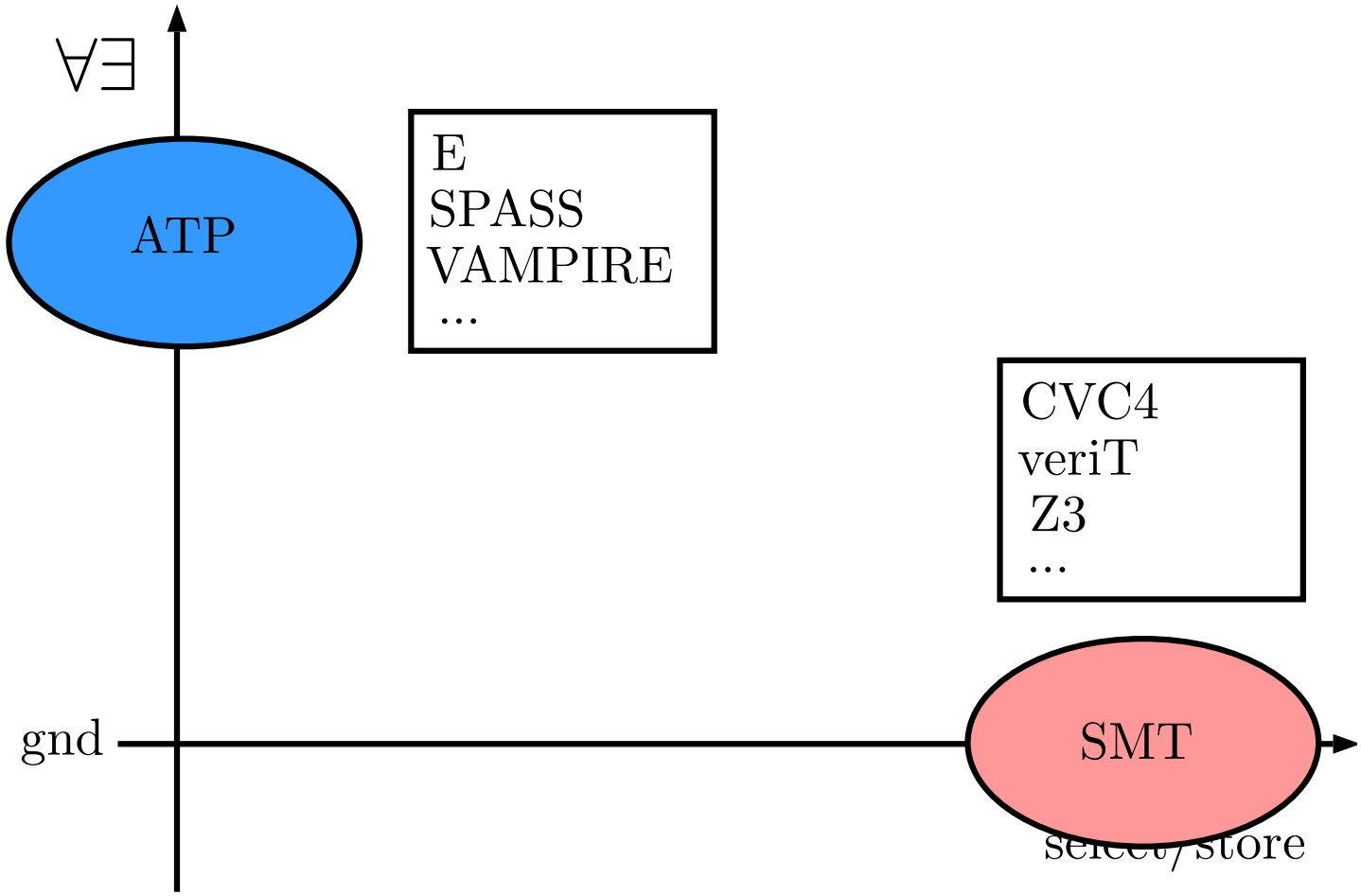
Why is it hard?

Reasoning with quantifiers and theories



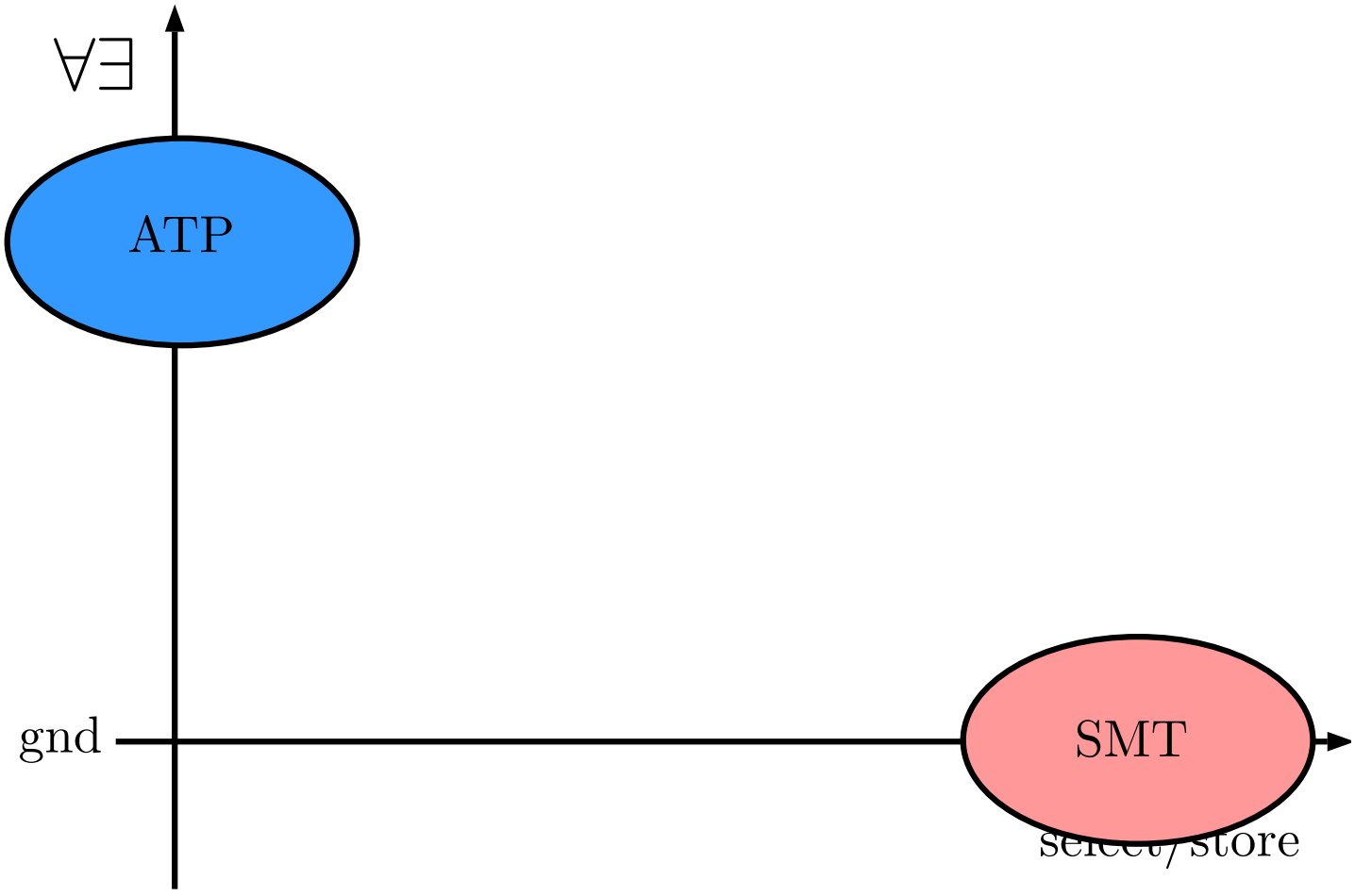
Why is it hard?

Reasoning with quantifiers and theories



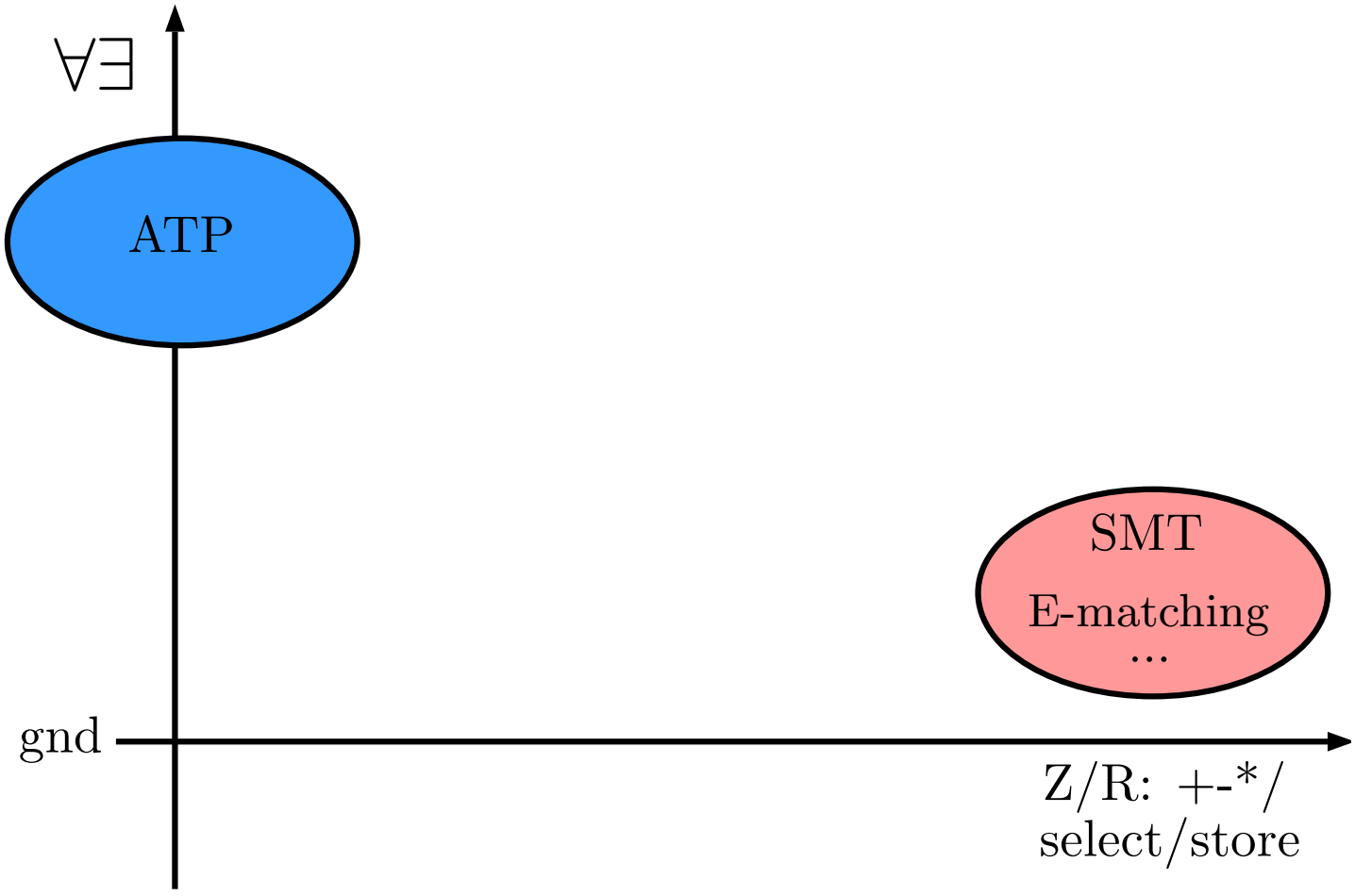
Why is it hard?

Reasoning with quantifiers and theories



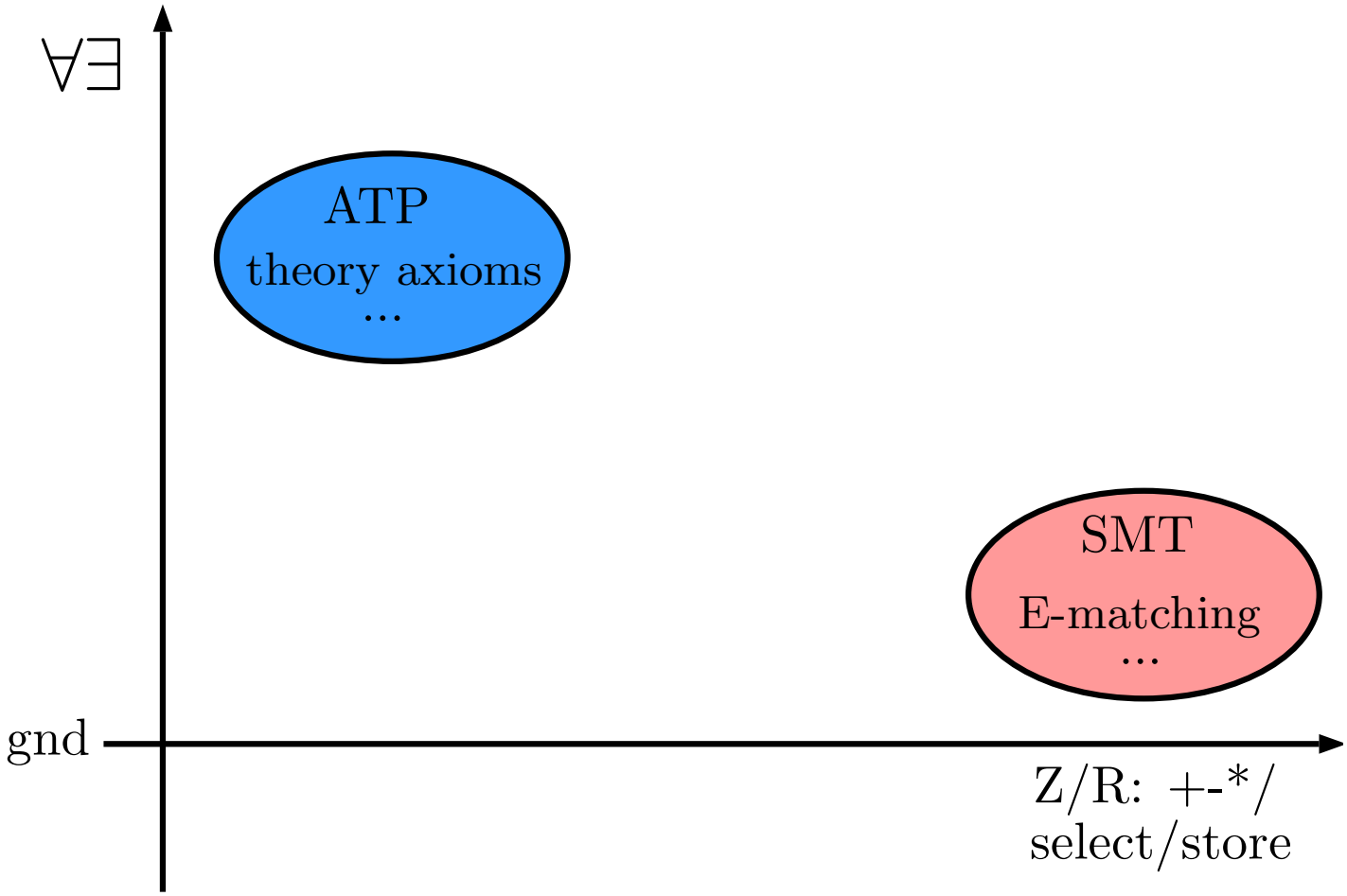
Why is it hard?

Reasoning with quantifiers and theories



Why is it hard?

Reasoning with quantifiers and theories



Theories of Interest

Supported by Vampire

- Arithmetic (integer/real linear/non-linear)
- Arrays
- Datatypes
- Bitvectors (not public yet)

Not Supported by Vampire (yet)

- Strings
- Floating Point

Handling Theories in TPTP and SMT-LIB

TPTP

- Special reserved symbols e.g. `$sum`
- No clear semantics i.e. they expected one
- Some rather odd symbols e.g. `$quotient_t`

SMT-LIB

- A pluggable notion of a theory that declares signature
- A bit better described
- see `http://smtlib.cs.uiowa.edu/theories.shtml`

Thankfully, they coincide on defined theories

Proof Search and Theories

- Evaluation
- Theory Axioms
- AVATAR modulo Theories
- Unification with Abstraction
- Theory Instantiation

Evaluation

Currently only for arithmetic

Normalise input to remove binary $-$, and $>, \geq, \leq$.

Evaluate ground expressions e.g.

- $p(2+1)$ becomes $p(3)$
- $4 > 2$ becomes *true*

Balance any equality containing a single uninterpreted sub-expression e.g. $2 * f(x) = 4$ becomes $f(x) = 2$.

Surprisingly (embarrassingly) effective - a large percentage of SMTLIB problems can be solved purely by first-order reasoning and evaluation.

Theory Axioms

Add axioms of the theory (statements true in the theory). Most theories of interest are not finitely axiomatisable.

Sometimes only add the 'a little' i.e. use ideas from set of support to reduce their impact on search space

Used for all theories.
You can also consider adding your own if you think it will help.

Again, surprisingly effective

AVATAR modulo theories

AVATAR where we replace the SAT solver by an SMT solver

Subproblems explored are now consistent in their ground theory part. In some cases there are much fewer.

The details are not that interesting

- Have to map our terms to z3 terms
- Need to handle case where z3 says unknown

Unification with Abstraction by Example

We cannot derive $p(7)$ from

$$r(14y) \qquad \neg r(x^2 + 49) \vee p(x),$$

as we cannot unify $14y$ and $x^2 + 49$.

General Solution: abstract i.e. rewrite to give

$$r(u) \vee u \neq 14y \qquad \neg r(v) \vee v \neq x^2 + 49 \vee p(x),$$

However, full abstraction like this destroys nice search space properties.

Our Solution: do abstraction during unification e.g. infer

$$14y \neq x^2 + 49 \vee p(x)$$

directly

Theory Instantiation

Now given

$$14y \neq x^2 + 49 \vee p(x)$$

what do we do with it?

Notice that there is a single theory consistent instance $(x = 7)$

Our solution: use an SMT solver to find instances

In practice, we have an inference rule

$$\frac{P \vee D}{D\theta}$$

where P is pure theory and $\neg P\theta$ is theory-valid, which for complete theories means that θ can be extracted from a model of P produced by z3.

Polymorphic theory of Arrays

Polymorphic means that we instantiate the theory for various range and value types e.g. array(Int, Int), array(Int, array(Int, Int)).

Standard axioms

$$(i = j \rightarrow select(store(a, i, v), j) = v)$$
$$(i \neq j \rightarrow select(store(a, i, v), j) = select(a, j))$$
$$((select(a, i) = select(b, i)) \rightarrow a = b)$$

Extensionality resolution

$$\frac{x = y \vee C \quad \underline{s \neq t} \vee D}{C\{x \mapsto s, y \mapsto t\} \vee D},$$

where $x = y \vee C$ is identified as an extensionality clause.

Datatypes (or finite term algebras)

A datatype has constructors and destructors.
Currently we only have them in the SMTLIB language e.g.:

- Natural numbers

```
(declare-datatypes ((Nat 0)) (((succ (pred Nat))
(zero))
```

- Lists

```
(declare-datatypes ((Lst 0)) (((cons (head Nat)
(tail Lst)) (nil))
```

- Trees

```
(declare-datatypes ((Tree 0)) (((node (data Nat)
(left Tree) (right Tree)) (leaf))
```

- Pairs

```
(declare-datatypes ((Pair 0)(ZLst 0)) (((mkpair
(first Nat) (second Nat)))
```

Datatypes, theory

Datatypes can be axiomatised:

- Domain closure
$$\bigvee_{f \in \Sigma} \exists \overline{y}(x = f(\overline{y}))$$
- Distinctness
$$f(\overline{x}) \neq g(\overline{y})$$
- Injectivity
$$f(\overline{x}) = g(\overline{x}) \rightarrow \overline{x} = \overline{y}$$
- Acyclicity (infinite)
$$t[x] \neq x$$

Datatypes, reasoning

We care about mixing datatypes with uninterpreted functions and quantifiers (e.g. known decision procedures cannot be used).

With axioms (gets us quite far, if we don't need acyclicity)

For acyclicity we either

- Introduce a subterm relation
- Extend the calculus with a special rule (also for the other axioms)

Bitvectors

(not currently supported, but coming soon)

Also known as bitstrings. A fixed length string of bits.

Useful for representing machine integers.

Types are parameterised by length i.e. a family of types

Lots of fun operators on bitvectors

Our implementation extends above things (AVATAR modulo theories, theory instantiation, evaluation etc). So nothing new, although gives extra power than SMT solvers.

Booleans

It might sound odd to talk of a theory of booleans but it would be nice to have functions returning boolean results and to mix terms and formulas.

SMTLIB does this already, but we extended TPTP to have it also

The theory introduces constants *true* and *false* and axioms

$$true \neq false \qquad (\forall x : bool)(x = true \vee x = false)$$

but that second axiom is horrible so we add a special inference rule

$$\frac{C[s]}{C[true] \vee s = false}$$