Nitpick:
A counterexample generator for higher-order logic based on a relational model finder

Jasmin Blanchette & Tobias Nipkow
TU München

Isabelle/HOL

small-kernel interactive theorem prover
for higher-order logic

Isabelle/HOL

small-kernel interactive theorem prover
for higher-order logic

Quickcheck

random generation of variable assignments

+ fast − requires exec.

Isabelle/HOL

small-kernel interactive theorem prover
for higher-order logic

Quickcheck

random generation of variable assignments

+ fast − requires exec.

Refute

finite model finding using SAT

+ general − slow

Nitpick

finite model finding using Kodkod

+ general − slow

Nitpick

finite model finding using Kodkod

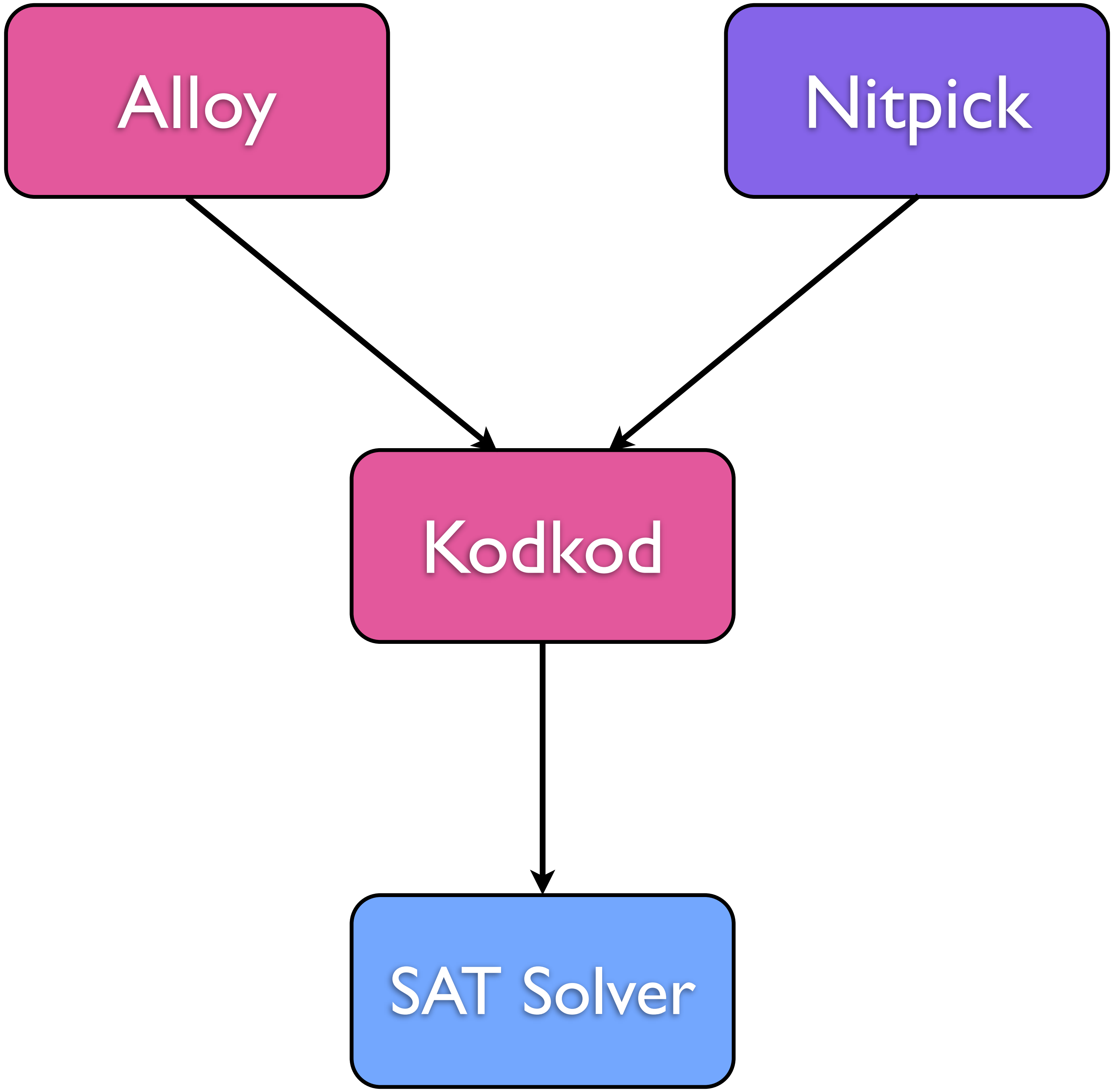
+ general − slow

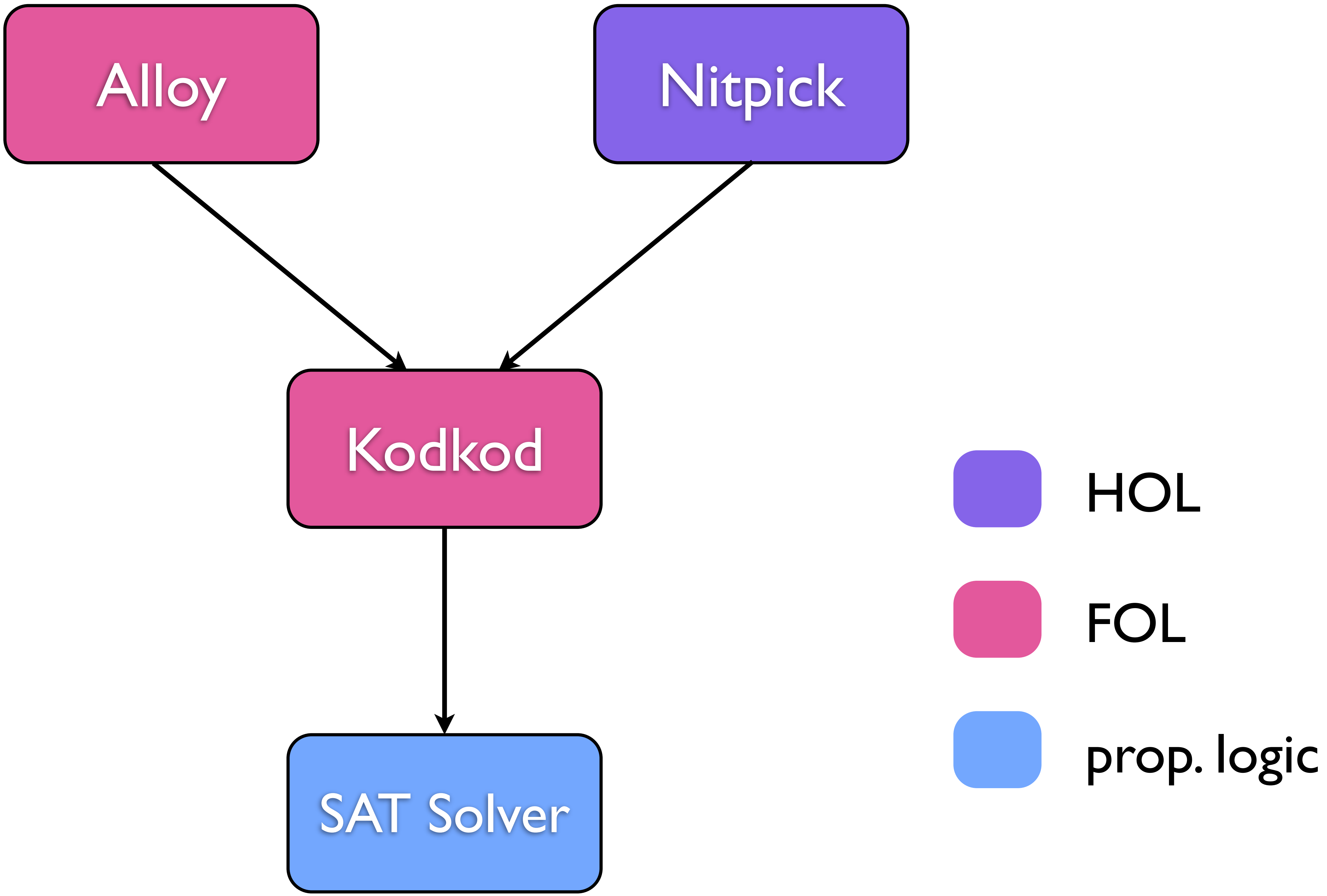
Kodkod

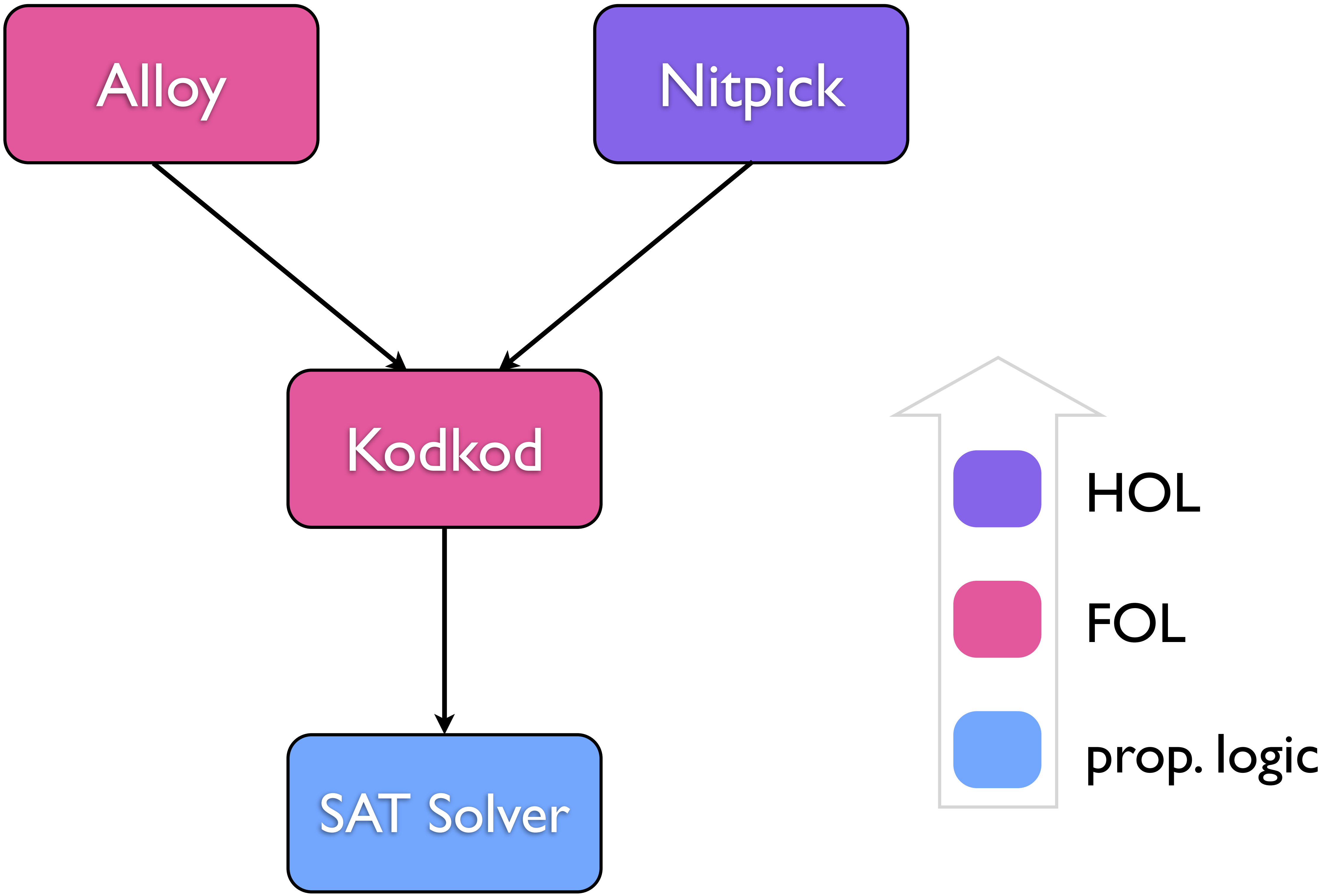
finite model finder for FOL w/ relational calc.

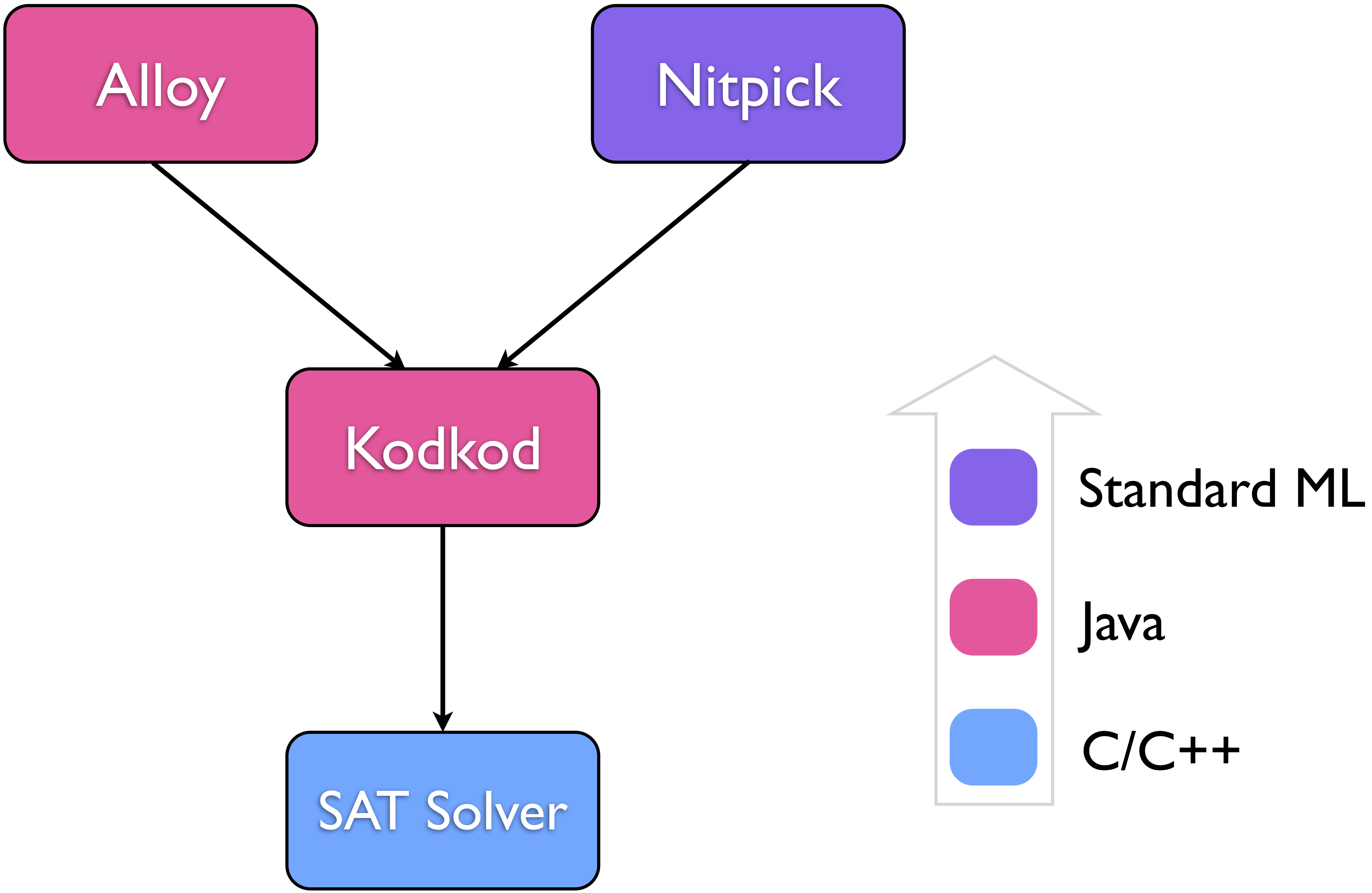
based on SAT

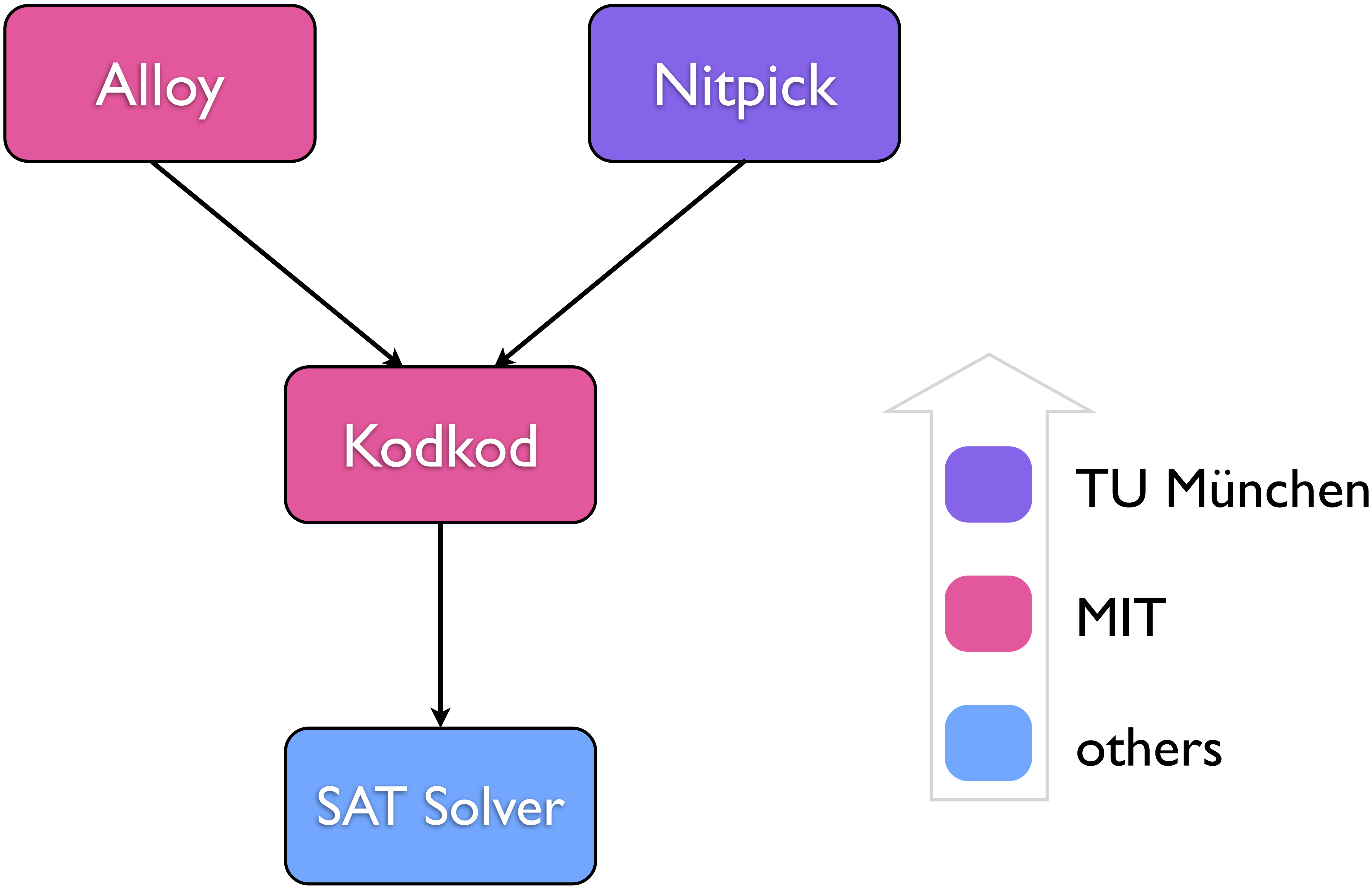
backend of the Alloy Analyzer


Alloy
Nitpick
Kodkod
SAT Solver


Alloy
Nitpick
Kodkod
SAT Solver
HOL
FOL
prop. logic


Alloy
Nitpick
Kodkod
SAT Solver
HOL
FOL
prop. logic


Alloy
Nitpick
Kodkod
SAT Solver
Standard ML
Java
C/C++



Alloy
Nitpick
Kodkod
SAT Solver
TU München
MIT
others

Nitpick in a nutshell

Nitpick in a nutshell

scope enumeration

Nitpick in a nutshell

scope enumeration

scope = dom. size spec.

Nitpick in a nutshell

scope enumeration

scope = dom. size spec.

for each scope:
convert $A \land \neg P$ to Kodkod

Nitpick in a nutshell

scope enumeration

main issues:
speed and precision

scope = dom. size spec.

for each scope:
convert $A \wedge \neg P$ to Kodkod

Nitpick in a nutshell

scope enumeration

scope = dom. size spec.

for each scope:
convert $A \wedge \neg P$ to Kodkod

main issues:
speed and precision

numbers,
datatypes,
ind. predicates,
rec. functions

Non-uniform encoding

Non-uniform encoding

- *n*-ary functions are coded as (n + 1)-ary relations

Non-uniform encoding

- *n*-ary functions are coded as (*n* + 1)-ary relations
- *but:* *n*-ary predicates
  → *n*-ary relations (sets of *n*-tuples)

Non-uniform encoding

- *n*-ary functions are coded as (*n* + 1)-ary relations
- *but:* *n*-ary predicates
  ➙ *n*-ary relations (sets of *n*-tuples)
- *but:* functions in higher-order constructs
  ➙ vectors of values

Non-uniform encoding

- *n*-ary functions are coded as (*n* + 1)-ary relations
- *but:* *n*-ary predicates
  ➙ *n*-ary relations (sets of *n*-tuples)
- *but:* functions in higher-order constructs
  ➙ vectors of values
- λ-abstractions ➙ set comprehensions

Infinite types

Infinite types

- are approximated by finite fragment; e.g., $\{0, 1, 2, \ldots, K\}$

Infinite types

- are approximated by finite fragment; e.g., $\{0, 1, 2, \ldots, K\}$
- unrep./unknown value: $\perp$

Infinite types

- are approximated by finite fragment; e.g., $\{0, 1, 2, \ldots, K\}$
- unrep./unknown value: $\bot$
- $\forall n{::}nat.\ P(n) \ \rightarrow\ (\forall n \leq K.\ P(n)) \wedge \mathrm{P}(\bot)$

Infinite types

- are approximated by finite fragment; e.g., $\{0, 1, 2, \ldots, K\}$
- unrep./unknown value: $\bot$
- $\forall n{::}nat.\ P(n) \;\rightarrow\; (\forall n \leq K.\ P(n)) \wedge \mathrm{P}(\bot)$
- Kleene logic

Infinite types

- are approximated by finite fragment; e.g., $\{0, 1, 2, \ldots, K\}$
- unrep./unknown value: $\bot$
- $\forall n{::}nat.\ P(n) \rightarrow (\forall n \leq K.\ P(n)) \wedge \mathrm{P}(\bot)$
- Kleene logic
- potential counterexamples

Inductive datatypes

Inductive datatypes

- are approximated by subterm-closed universes

Inductive datatypes

- are approximated by subterm-closed universes
- user can specify maxima on ctors

Inductive datatypes

- are approximated by subterm-closed universes
- user can specify maxima on ctors
- recursive functions: defined by their eq. spec.

Example: α *list*

scope: $|\alpha| = 2,\ |\alpha\ \mathit{list}| = 3$

ctors: $\mathit{Nil}^{\alpha\ \mathit{list}}$

$\mathit{Cons}^{\alpha \to \alpha\ \mathit{list} \to \alpha\ \mathit{list}}$

universes: $\{[], [a_1], [a_2]\}$

$\{[], [a_1], [a_1, a_1]\}$

$\{[], [a_1], [a_2, a_1]\}$

$\{[], [a_2], [a_1, a_2]\}$

$\{[], [a_2], [a_2, a_2]\}$

Example: α *list*

scope: $|\alpha| = 2$, $|\alpha\ list| = 3$

ctors: $Nil^{\alpha\ list}$

$Cons^{\alpha \to \alpha\ list \to \alpha\ list}$

universes: $\{[], [a_1], [a_2]\}$

$\{[], [a_1], [a_1, a_1]\}$

$\{[], [a_1], [a_2, a_1]\}$

~~$\{[], [a_2], [a_1, a_2]\}$~~

~~$\{[], [a_2], [a_2, a_2]\}$~~

Inductive predicates

Inductive predicates

- specified by introduction rules

Inductive predicates

- specified by introduction rules
- correspond to a lfp

Inductive predicates

- specified by introduction rules
- correspond to a lfp
- well-founded?

Inductive predicates

- specified by introduction rules
- correspond to a lfp
- well-founded?
  - if yes, lfp = gfp

Inductive predicates

- specified by introduction rules
- correspond to a lfp
- well-founded?
  - if yes, lfp = gfp
  - otherwise, unroll

Inductive predicates

- specified by introduction rules
- correspond to a lfp
- well-founded?
  - if yes, lfp = gfp
  - otherwise, unroll

{ lexicographic_order
sizechange

Example: *even*

intro. rules: $even(0)$

$even(n) \Rightarrow even(n + 2)$

Example: *even*

intro. rules: $even(0)$

$even(n) \Rightarrow even(n + 2)$

fixpoint eq.: $even(n) =$

$(n = 0 \lor (\exists m.\ n = m + 2 \land even(m)))$

Example: *even*

intro. rules: $even(0)$

$even(n) \Rightarrow even(n + 2)$

fixpoint eq.: $even(n) =$

$(n = 0 \vee (\exists m.\, n = m + 2 \wedge even(m)))$

unrolled eq.: $even_{\mathbf{0}}(n) = \bot$

$even_{\mathbf{k+1}}(n) =$

$(n = 0 \vee (\exists m.\, n = m + 2 \wedge even_{\mathbf{k}}(m)))$

Function specialization

$$map\ f\ [\,] = [\,]$$
$$map\ f\ (x \cdot xs) = f\ x \cdot map\ f\ xs$$

Function specialization

$$map\ f\ []\ =\ []$$
$$map\ f\ (x\cdot xs)\ =\ f\ x\cdot map\ f\ xs$$

$$map\ (\lambda n.\ n+2)\ ks$$

Function specialization

$$\mathit{map}\ f\ [\,]\ =\ [\,]$$
$$\mathit{map}\ f\ (x \cdot \mathit{xs})\ =\ f\ x \cdot \mathit{map}\ f\ \mathit{xs}$$

$$\mathit{map}\ (\lambda n.\, n + 2)\ \mathit{ks}$$

$$\mathit{map}'\ [\,]\ =\ [\,]$$
$$\mathit{map}'\ (x \cdot \mathit{xs})\ =\ (x + 2) \cdot \mathit{map}'\ \mathit{xs}$$

$$\mathit{map}'\ \mathit{ks}$$

Evaluation

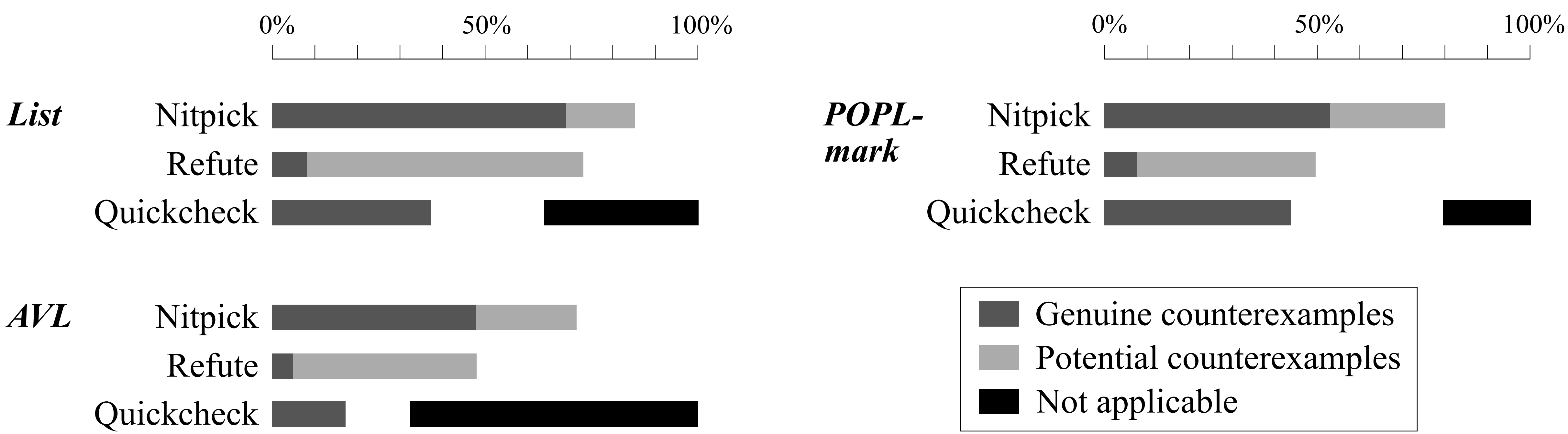


Fig. 1. Success rates of the counterexample generators on three theories

Status

- first public release two weeks ago
- a few users + TPTP
- found two bugs in TPS prover!

Status

- first public release two weeks ago
- a few users + TPTP
- found two bugs in TPS prover!

Future work

- more optimizations (speed and precision)
- evaluations