Programming with constraint solvers

code checking, angelic execution, debugging, synthesis and beyond

rastislav bodik & emina torlak

ExCAPE Kickoff Meeting · June 04, 2012
programming ...
programming ...

\[
P(x) \begin{cases} 
\ldots 
\end{cases}
\]

formula, input/output pairs, traces, another program, ...
programming ...

\begin{itemize}
  \item assume \texttt{pre(x)}
  \item assert \texttt{post(P(x))}
  \item \texttt{P(x) \{ \texttt{...} \}}
\end{itemize}
programming with a solver

assume \text{pre}(x)
P(x) \{
\ldots
\}
assert \text{post}(P(x))

translate(…)

SAT/SMT solver
programming with a solver: code checking

Is there a valid input $x$ for which $P(x)$ violates the spec?

```plaintext
assume $\text{pre}(x)$
$P(x) \{ 
\ldots 
\}$
```

```plaintext
assert $\text{post}(P(x))$
```

$$\exists x . \text{pre}(x) \land \neg \text{post}(P(x))$$

SAT/SMT solver

CBMC [Oxford], Dafny [MSR], Jahob [EPFL], Miniatur / MemSAT [IBM], etc.
programming with a solver: code checking

Is there a valid input $x$ for which $P(x)$ violates the spec?

\begin{verbatim}
assume pre(x)
P(x) {
    ...
}
assert post(P(x))
\end{verbatim}

\[ \exists x . \text{pre}(x) \land \neg \text{post}(P(x)) \]

SAT/SMT solver

$\exists x = 42$

counterexample

counterexample

model

\[ x = 42 \]

CBMC [Oxford], Dafny [MSR], Jahob [EPFL], Miniatur / MemSAT [IBM], etc.
programming with a solver: debugging

Given \( x \) and \( x' \), what subset of \( P \) is responsible for \( P(x) \neq x' \)?

```plaintext
assume pre(x)
P(x) {
    v = x + 2
    ...
}
assert post(P(x))
```

\[
\text{pre}(x) \land \text{post}(x') \land \\
x' = P(x)
\]

SAT/SMT solver
programming with a solver: debugging

Given \( x \) and \( x' \), what subset of \( P \) is responsible for \( P(x) \neq x' \)?

```c
assume \( \text{pre}(x) \)
\[
P(x) \{ 
    \quad v = x + 2 
    \quad \ldots 
\} 
assert \text{post}(P(x))
```

\[
\text{pre}(x) \land \text{post}(x') \land x' = P(x)
\]

repair candidates

SAT/SMT solver

MAXSAT/ MIN CORE

BugAssist [UCLA / MPI-SWS]
programming with a solver: angelic execution

Given $x$, choose $v$ at runtime so that $P(x, v)$ satisfies the spec.

```
assume pre(x)
P(x) {
  v = choose()
  ...
}
assert post(P(x))
```

$\exists v . \text{pre}(x) \land \text{post}(P(x, v))$

SAT/SMT solver

Kaplan [EPFL], PBnJ [UCLA], Skalch [Berkeley], Squander [MIT], etc.
programming with a solver: angelic execution

Given $x$, choose $v$ at runtime so that $P(x, v)$ satisfies the spec.

```plaintext
assume $\text{pre}(x)$
$P(x)$ {
  $v = \text{choose}()$
  …
}
assert $\text{post}(P(x))$
```

$v = 0, \ldots$

∃$v$. $\text{pre}(x) \land
\text{post}(P(x, v))$

SAT/SMT solver

Kaplan [EPFL], PBnJ [UCLA], Skalch [Berkeley], Squander [MIT], etc.
programming with a solver: synthesis

Replace ?? with expression e so that \( P_e(x) \) satisfies the spec on all valid inputs.

```
assumepre(x)
P(x) {
  v = ??
  ...
}
assert post(P(x))
```

\[ \exists e . \forall x . \text{pre}(x) \Rightarrow \text{post}(P_e(x)) \]

SAT/SMT solver
programming with a solver: synthesis

Replace ?? with expression e so that \( P_e(x) \) satisfies the spec on all valid inputs.

\[
\begin{align*}
\text{assume } & \text{pre}(x) \\
P(x) \{ & \quad v = x - 2 \\
& \quad \ldots \} \\
\text{assert } & \text{post}(P(x))
\end{align*}
\]

\[
\exists e . \forall x . \text{pre}(x) \Rightarrow \text{post}(P_e(x))
\]

SAT/SMT solver

Comfusy [EPFL], Sketch [Berkeley / MIT]
programming with a solver: synthesis

Replace ?? with expression e so that $P_e(x)$ satisfies the spec on all valid inputs.

assume $\text{pre}(x)$
$P(x) \{$
$v = x - 2$
...$
}$
assert $\text{post}(P(x))$

$\exists e . \bigwedge_i \text{pre}(x_i) \land \text{post}(P_e(x_i))$

SAT/SMT solver

expressions

model

Comfusy [EPFL], Sketch [Berkeley / MIT]
state of the art: fundamental challenges

decidability, tractability and scalability

- work in a **small finite universe**
- rich properties, incomplete reasoning
- work with (mostly) **decidable logics**
- complete reasoning, limited properties

```
assume pre(x)
P(x) {
  ...
}
assert post(P(x))
```
state of the art: fundamental challenges

decidability, tractability and scalability

- work in a small finite universe
- rich properties, incomplete reasoning
- work with (mostly) decidable logics
- complete reasoning, limited properties

```plaintext
assume pre(x)
P(x) {
  ...
}
assert post(P(x))
```

translate(…)

SAT/SMT solver
state of the art: fundamental challenges

decidability, tractability and scalability

• work in a \textit{small finite universe}
• rich properties, incomplete reasoning
• work with (mostly) \textit{decidable logics}
• complete reasoning, limited properties

\begin{verbatim}
assume pre(x)
P(x) {
  ...
}
assert post(P(x))
\end{verbatim}
state of the art: engineering challenges

building a new tool \( \approx \) Ph.D.

- pre-requisites
- program analysis, formal methods, logic
- deep knowledge of the solver
- fragmented infrastructure
- hard to reuse encodings, analyses

\[
\begin{align*}
\text{assume pre}(x) \\
P(x) \{ \\
\quad \ldots \\
\} \\
\text{assert post}(P(x))
\end{align*}
\]
state of the art: engineering challenges

building a new tool ≈ Ph.D.

- pre-requisites
  - program analysis, formal methods, logic
  - deep knowledge of the solver
- fragmented infrastructure
  - hard to reuse encodings, analyses

\[
\text{assume } \text{pre}(x) \\
P(x) \{ \\
\quad \ldots \\
\} \\
\text{assert } \text{post}(P(x))
\]
state of the art: engineering challenges

building a new tool ≈ Ph.D.

- pre-requisites
  - program analysis, formal methods, logic
  - deep knowledge of the solver
- fragmented infrastructure
  - hard to reuse encodings, analyses

```
assume pre(x)
P(x) {
  ...
}
assert post(P(x))
```
state of the art: design & usability challenges

using an existing tool $\simeq$ ABD

- it’s hard to keep the system in the loop
- need abstractions for communicating domain knowledge to the solver
- it’s hard to keep the user in the loop
- need abstractions for communicating failures to the user

```
assume pre(x)
P(x) {
    ...
}
assert post(P(x))
```

```
translate(…)
```

SAT/SMT solver
state of the art: design & usability challenges

using an existing tool $\approx$ ABD

- it’s hard to keep the system in the loop
  - need abstractions for communicating domain knowledge to the solver
- it’s hard to keep the user in the loop
  - need abstractions for communicating failures to the user

assume pre(x)
P(x) {
  ...
}
assert post(P(x))

translate(…)

SAT/SMT solver
state of the art: design & usability challenges

- using an existing tool $\approx$ ABD
  - it’s hard to keep the system in the loop
    - need abstractions for communicating domain knowledge to the solver
  - it’s hard to keep the user in the loop
    - need abstractions for communicating failures to the user

```plaintext
assume pre(x)
P(x) {
    ...
}
assert post(P(x))

translate(…)
```

SAT/SMT solver
vision: programming with solvers in ?? years

building a tool $\approx$ embedding a DSL into Scala or Racket

using a tool $\approx$ using a DSL
our new tool: rosette

assume \text{pre}(x)
\text{P}(x) \{ 
\ldots 
\}
assert \text{post}(\text{P}(x))

code checking
 angelic execution
 debugging
 synthesis
 for high performance computing
our new tool: rosette

if you need to prototype a synthesizer for your programming model or DSL

- define the language by writing an interpreter
- use angelic execution to simplify this task
- define what “holes” look like in the language
- Rosette will do the rest
- automatic translation to formulas
- thus enabling checking, angelic execution, debugging and synthesis for your language

if you need to a more scalable synthesizer

- Rosette will help you develop an efficient translator to formulas
- debug and synthesize an efficient encoding
- by ensuring it is semantically equivalent to the one obtained from the interpreter