

Mixing Finite Success and Finite Failure in an Automated Prover

Alwen Tiu¹, Gopalan Nadathur², and Dale Miller³

¹ INRIA Lorraine/LORIA

² Digital Technology Center and Dept of CS, University of Minnesota

³ INRIA & LIX, Ecole Polytechnique

Abstract. The operational semantics and typing judgements of modern programming and specification languages are often defined using relations and proof systems. In simple settings, logic programming languages can be used to provide rather direct and natural interpreters

programming system, not in a purely logical way at least, even though they

$\Sigma \sigma$

definitions is an extension of work by Schroeder-Heister [SH93], Eriksson [Eri91],

the following classes of formulas:

Level 0: $G := \neg \mid A \mid G \mid G \mid G \mid G \mid \exists x:G \mid \forall x:G$
 Level 1: $D := \neg \mid \neg \mid A \mid D \mid D \mid D \mid D \mid \exists x:D \mid \forall x:D \mid \exists x:D \mid G \mid \supset \mid D$
 atomic: $A := p \mid t_1 \mid \dots \mid t_n$

Here, atomic formulas A in level 0 j6264 Tf 19.9052 962648

Concrete syntax The concrete syntax for Level 0/1 prover follows the syntax of

more subtle. Consider the case where both eigenvariables and logic variables are present in a negative goal, for example, consider proving the goal

$\neg x:zy:($

z : p

in : $n \quad (n \quad p) \quad p$

```
onep (in X M) (dn X) M.Q           % bound input
one  (out X Y P) (up X Y) P.       % free output
one  (taup P) tau P. tau           % tau
one  (match X X P) A Q :=
```

We now define a notion of equivalence between processes, called bisimulation. It is formally defined as follows: a relation \approx is a bisimulation, if it is a symmetric relation such that for every $(P; Q) \in \approx$,

1. if $P \xrightarrow{x(z)} P^0$ and $(P; Q) \in \approx$ is a free action, then there is Q^0 such that $Q \xrightarrow{x(z)} Q^0$
2. if $Q \xrightarrow{x(z)} Q^0$ and $(P; Q) \in \approx$ then there is P^0 such that $P \xrightarrow{x(z)} P^0$ for every


```
sat P top.
sat P (and A B) := sat P A, sat P :.
sat P (or A B) := sat P A; sat P :.
sat P (boxMatch X Y A) := (X = Y) => sat P A.
sat P (diaMatch X Y A) := (X = Y), sat P A.
sat P (boxAct X A) := pi P1\ one P X P1 => sat P1 A.
sat P (diaAct X A) := sigma P1\ one P X P1, sat P1 A.
sat P (boxOut X A) := pi Q\ onep P (up X) Q => nabl a y\ sat (Q y)
```

an unevaluated expression, and its evaluation is

9 F

$\frac{\Sigma ; \sigma \triangleright \quad \Sigma ; \sigma \triangleright}{\Sigma ; \sigma \triangleright} \text{init} \quad \Sigma ; \cdot \vdash \mathcal{B} \quad \Sigma ; \mathcal{B} \vdash \cdot \mathcal{C}$



[MPW93] Robin Milner, Joachim