

11. Das Typsystem von PVS (II)

- Judgements
- Typ-Konversionen (engl. *Conversions*)

Typen in PVS: Judgements

Judgements dienen dazu, Typinformation, insbesondere diejenige, die mit prädikativen Untertypen verbunden ist (semantische Typinformation) in einer Form zur Verfügung zu stellen, die der Typechecker ausnutzen kann (“Hilfestellung” für den Typechecker).

~> Anzahl der zu generierenden TCCs kann (im allgemeinen) reduziert werden

Judgements sind formalisierte Typ-Aussagen

- führen in der Regel zu TCCs, die nachgewiesen werden müssen

Formen von Judgements

- *Constant Judgements*: HAS_TYPE
- *Subtype Judgements*: SUBTYPE_OF

Judgements – Beispiele (1)

```
mod: THEORY
  i,k: VAR int
  m: VAR posnat
  n: VAR nat
  j: VAR nonzero_int

  mod(i,j): { k | abs(k) < abs(j) } = i - j+floor(i,j)

  mod_below: JUDGEMENT mod(i,m) HAS_TYPE below(m)
  ...
END mod
```

Judgements – Beispiele (2)

`i, j, k: VAR nat`

`n: VAR posnat`

`upfrom_nat_is_nat: JUDGEMENT upfrom(i) SUBTYPE_OF nat`

`upfrom_posnat_is_posnat: JUDGEMENT upfrom(n) SUBTYPE_OF posnat`

`above_nat_is_posnat: JUDGEMENT above(i) SUBTYPE_OF posnat`

`subrange_nat_is_nat: JUDGEMENT subrange(i,j) SUBTYPE_OF nat`

`subrange_posnat_is_posnat: JUDGEMENT subrange(n,j) SUBTYPE_OF posnat`

Judgements – Beispiele (3)

`injective?(f): bool = (FORALL x1, x2: (f(x1) = f(x2) => (x1 = x2)))`

`surjective?(f): bool = (FORALL y: (EXISTS x: f(x) = y))`

`bijjective?(f): bool = injective?(f) & surjective?(f)`

`bij_is_inj: JUDGEMENT (bijjective?) SUBTYPE_OF (injective?)`

`bij_is_surj: JUDGEMENT (bijjective?) SUBTYPE_OF (surjective?)`

`bij_is_inj: JUDGEMENT (bijjective?) SUBTYPE_OF (injective?)`

Judgements – Beispiele (4)

```
orders [T: TYPE]: THEORY
```

```
BEGIN
```

```
  x, y: VAR T
```

```
  <=, < : VAR pred[[T, T]]
```

```
  p: VAR pred[T]
```

```
preorder?(<=): bool = reflexive?(<=) & transitive?(<=)
```

```
preorder_is_reflexive: JUDGEMENT (preorder?) SUBTYPE_OF (reflexive?[T])
```

```
preorder_is_transitive: JUDGEMENT (preorder?) SUBTYPE_OF (transitive?[T])
```

```
equiv_is_preorder:      JUDGEMENT (equivalence?[T]) SUBTYPE_OF (preorder?)
```

```
partial_order?(<=): bool = preorder?(<=) & antisymmetric?(<=)
```

```
dichotomous?(<=): bool = (FORALL x, y: (x <= y OR y <= x))
```

```
total_order?(<=): bool = partial_order?(<=) & dichotomous?(<=)
```

```
total_is_po:           JUDGEMENT (total_order?) SUBTYPE_OF (partial_order?)
```

Typen-Konversion in PVS

Typ-Konversion ist ähnlich einer *coercion* etwa einer ganzen Zahl in eine reelle Zahl (float).

Typ-Konversionen werden bei der Typüberprüfung “automatisch” eingefügt, wenn ohne sie Typkorrektheit nicht abgeleitet werden kann und das Einfügen zu vollständiger Typisierung führt.

Beispiel (aus Sprachbeschreibung):

```
c: [int -> bool]
CONVERSION c      % deklariert c als Konversionsfunktion
two: FORMULA 2

g: [int -> int]
F: [[nat -> int] -> bool]
F_app: FORMULA F(g)
```

F_app ist nicht typkorrekt.

Typen in PVS: Conversions (2)

Benötigt: eine Einschränkung einer Funktion auf einen Untertyp des Definitionstyps:

```
restrict [T: TYPE, S: TYPE FROM T, R: TYPE]: THEORY
  BEGIN
    f: VAR [T -> R]
    s: VAR S
    restrict(f)(s): R = f(s)
    CONVERSION restrict
  END restrict
```

Eine typkorrekte Form der Formel in F_{app} ist

$$F(\text{restrict}[\text{int}, \text{nat}, \text{int}](g))$$

Warnung: solche “automatischen” Veränderungen von Ausdrücken sollten nur mit großer Vorsicht benutzt werden!